People's Democratic Republic of Algeria

Ministry of Higher Education and Scientific Research

University M'Hamed BOUGARA – Boumerdes

Institute of Electrical and Electronic Engineering

Department of Power and Control

Final Year Project Report Presented in Partial Fulfilment of
the Requirements for the Degree of the:

'MASTER'

In :

Control Engineering
Computer Engineering

Title:

Indoor Obstacle Avoidance System Design And Evaluation Using Deep Learning And SLAM based Approaches

Presented by:

- **BENBEKHMA Abdelwadoud**
- **TAIBI Houssam Eddine**

Supervisor:

Dr. BENZAOUI Messaouda

Registration Number:………/2023

Dedication 1

With profound gratitude and utmost appreciation, I humbly dedicate this remarkable achievement to my esteemed **father**. Your unwavering love, invaluable guidance, and selfless sacrifices have been the driving forces behind my resounding success. I am eternally grateful for the invaluable virtues you have instilled within me.

To my beloved **mother**, the epitome of love, strength, and boundless selflessness, I extend my heartfelt gratitude. Your unwavering faith in my potential, nurturing presence, and unconditional love have served as the bedrock of my journey. Your countless sacrifices, unwavering encouragement, and heartfelt prayers have propelled me forward. May **Allah** shower His blessings upon you, dear **mother**.

To my beloved sisters, **Ghofrane**, **Iman**, and **Roumaissa**, I want to express my heartfelt appreciation for the immense joy and blessings your presence brings to my life. The moments we have cherished together are etched in the depths of my heart for eternity. To my dear brother **Ayoub**, your presence fills my life with immeasurable happiness, even in the moments of silence we share. The unspoken connection that binds us transcends words, and your mere existence brings me profound joy and gratitude.

To my esteemed friends **Bilel**, **Houssam**, **Zakaria**, and all my other companions who have journeyed alongside me. Your unwavering presence and unwavering camaraderie have made the challenges more conquerable and the triumphs more meaningful.

In remembrance of our dear friend **Salah Chelbi**, who departed from this ephemeral world far too soon, I offer my heartfelt prayers. May **Allah**, the Most Merciful, grant him forgiveness and mercy, and may He grant him an abode in **Jannah**, the eternal paradise.

May **Allah** bestow His blessings upon all my cherished loved ones, safeguarding them with unending happiness, robust health, and resounding success. May He keep them in His divine embrace, guiding and protecting them under His infinite grace.

Dedication 2

To my beloved **father** and **mother**,

With utmost reverence, deep gratitude, and boundless love, I dedicate this project to you. Your unwavering love, steadfast support, and countless sacrifices have been the bedrock of my journey. Your guidance, encouragement, and unwavering belief in my abilities have been a constant source of inspiration. Without your enduring love and unwavering support, this significant milestone would have remained beyond reach.

To my dear brothers, **Abdelmounaim**, **Nadjib**, **lil Iyad**, and my little sister, **Marwa**, I offer this heartfelt tribute. You have been my constant companions, confidants, and sources of joy. Together, we have shared laughter, tears, and countless unforgettable moments.

To my childhood friends, **Mimou**, **Azeddin**, and **Mahdi**, I express my heartfelt gratitude. Through thick and thin, you have stood as pillars of strength, unwavering in your support, and a source of invaluable camaraderie. Your presence has indelibly shaped my academic journey.

I extend my sincere appreciation to my esteemed friends, **Doudou**, **Zaki**, **Nasro**, **Bilel**, and **Adam**. You have become an extended family to me. Your unwavering support and camaraderie have served as constant sources of motivation and encouragement throughout my endeavors. This dedication reflects my profound gratitude and acknowledges the indelible influence they have had on my journey.
In loving memory of **Salah Chelbi**, whose spirit continues to inspire, this thesis is dedicated to his lasting impact on our lives.

May **Allah** bestow His blessings upon all my cherished loved ones, safeguarding them with unending happiness, robust health, and resounding success. May He keep them in His divine embrace, guiding and protecting them under His infinite grace.

Acknowledgments

In the pursuit of knowledge and the completion of this project, We would like to express our deep gratitude and thanks to Allah, the Most Gracious and Most Merciful. His guidance, blessings, and unwavering support have been the driving force behind every step of this journey.
We are profoundly grateful for the wisdom and enlightenment bestowed upon us throughout this project. We acknowledge that every accomplishment is a reflection of Allah's infinite grace and mercy.
We extend our heartfelt appreciation to our supervisor Mrs Benzaoui Messaouda for her invaluable guidance, unwavering support, and expert advice throughout the duration of this project. Her extensive knowledge and encouragement have been instrumental in shaping and refining the direction of our research.
We are also deeply grateful to the faculty and staff of The Institute Of Electrical And Electronics Engineering for providing the necessary resources, facilities, and opportunities that have facilitated the progress of this project. Their commitment to academic excellence has created an environment conducive to learning and exploration.
We extend our thanks to our colleagues and classmates for their collaboration, stimulating discussions, and willingness to share their knowledge. Their diverse perspectives and contributions have enriched our understanding and enhanced the quality of this project.
We are indebted to the participants who willingly volunteered their time and effort to contribute to the data collection process. Their cooperation and enthusiasm have been crucial in generating the empirical evidence necessary for this study.
Finally, we would like to acknowledge the support and encouragement from our family and friends. Their unwavering belief in our abilities and their continuous encouragement have been a source of inspiration throughout this project.
Although it is not possible to mention everyone individually, please accept our heartfelt appreciation for the assistance and support provided by each and every person who has contributed to this project.
Thank you all for your invaluable contributions.

Abstract

This project aims to contribute to the vibrant field of obstacle detection and safe autonomous navigation by designing a robust and cost-efficient system for indoor mobile robot obstacle avoidance. The system combines 2D LiDAR-based SLAM with the state-of-the-art RRT algorithm for effective path planning. In addition, a pioneering deep learning approach addresses challenges in SLAM-RRT-based obstacle avoidance, including uncertain sensor measurements, complex environments, generalization, planning efficiency, and non-geometric information. The deep learning model is trained using data from a simulated environment with a 2D LiDAR sensor, serving both SLAM and data acquisition purposes. Comparative analysis between odometry-based and SLAM-based pose computation methods provides insights into successful deep learning-based obstacle avoidance. Implemented within ROS2, this project represents a significant stride in exploring cutting-edge techniques for robust and cost-efficient indoor mobile robot obstacle avoidance.

List of Abbreviations and Terms

AI	Artificial Intelligence
IoT	Internet of Things
AMRs	Autonomous Mobile Robots
Industry 4.0	Fourth Industrial Revolution
SLAM	Simultaneous Localization and Mapping
2D	Two-Dimensional
ROS2	Robot Operating System 2
RRT	Rapidly Exploring Random Trees.
SLAM	Simultaneous Localization and Mapping
LiDAR	Light Detection and Ranging
GPS	Global Positioning System
IMU	Inertial Measurement Unit
API	Application Programming Interface
GUI	Graphical User Interface
PID	Proportional-Integral-Derivative
CPU	Central Processing Unit
GPU	Graphics Processing Unit
RGB-D	Red-Green-Blue Depth
CNN	Convolutional Neural Network
RNN	Recurrent Neural Network
Lidar	Light Detection and Ranging
3D	Three-Dimensional
RBPF	Rao-Blackwellized Particle Filter
ROS	Robot Operating System
Rviz	ROS visualization
LiDAR	Light Detection and Ranging
XML	Extensible Markup Language
RPLidar	Rotating Planar Lidar
Hz	Hertz

deg	degrees
URDF	Unified Robot Description Format
RNN	Recurrent Neural Networks
LSTM	Long Short-Term Memory
GAN	Generative Adversarial Networks
RL	Reinforcement Learning
FNN	Feedforward Neural Networks
SGD	Stochastic Gradient Descent
OHE	One hot encoding

Contents

List of Figures

List of Tables

General introduction

The increasing demand for autonomous mobile robots across various sectors, such as hospitals, institutions, agriculture, companies, and homes, has led to significant advancements in their capabilities [1]. These robots offer a promising solution for automation and efficiency in diverse environments, both indoors and outdoors. The emergence of Industry 4.0, with its focus on creating "smart factories" heavily reliant on automation and system interconnection [2, 3], has further accelerated the adoption of autonomous mobile robots.

However, this shift towards autonomous mobile robots poses new challenges, especially in terms of real-time responsiveness and safety requirements. In uncertain and dynamic environments, the ability to detect obstacles accurately and rapidly is crucial for these robots to perceive and navigate their surroundings [4]. Despite limited prior knowledge and potential dynamism, they must effectively detect obstacles while on the move.

Differential mobile robots are affordable and robust, making them an ideal choice for various applications. With efficient locomotion capabilities, they can traverse diverse terrains and maneuver effectively in tight spaces. LiDAR sensors, widely employed in autonomous mobile robots, offer advantages in distance measurement. They provide high scanning precision, a large detection range, and are insensitive to lighting conditions [5]. In this project, our focus is on utilizing 2D LiDAR sensors to obtain reliable 2D point-cloud measurements for detecting material obstacles in the surroundings.

To ensure accurate localization and mapping in unknown indoor environments, we employ SLAM techniques that leverage LiDAR data for real-time position estimation and map updates. This enables safe and efficient navigation, even in environments with limited or no pre-existing maps. Additionally, our project includes a deep learning-based obstacle avoidance algorithm, which uses deep neural networks to train the robot in making real-time decisions when faced with obstacles. This adaptive approach enhances the robot's navigation performance and behavior adaptation in different environments.

Our project's goal is to design a mobile robot system that combines the cost-efficiency and effectiveness of a differential robot platform with the accuracy and reliability of LiDAR sensors for obstacle detection and avoidance. This integrated system will enable autonomous mobile robot navigation in unknown environments with dynamic obstacles, addressing the challenges of real-time responsiveness and safety requirements.

To facilitate seamless communication between different components of the robot system, we integrate it with ROS 2, a popular robot operating system. This integration enhances system interoperability and simplifies the development process. Furthermore, simulation-based evaluation allows us to comprehensively analyze and assess the system's performance under various scenarios and environments, providing insights into its robustness and reliability.

By addressing the research questions related to obstacle detection, SLAM algorithms, deep learning-based obstacle avoidance, integration with ROS 2, and simulation-based evaluation, this project aims to contribute to the development of an efficient, affordable, and capable mobile robot system. In conclusion, this report presents a comprehensive approach to address the challenges of real-time obstacle detection, navigation, and avoidance in unknown environments with dynamic obstacles. By combining the strengths of differential mobile robots, LiDAR sensors, and deep learning techniques, we aim to create an efficient and affordable mobile robot system capable of operating autonomously in diverse and challenging environments.

CHAPTER 1

Problem Description and Literature Review

1.1 Problem Description

Mobile robot navigation in unknown environments with dynamic obstacles presents several challenges that need to be addressed for efficient and safe operation. The ability to detect and avoid obstacles in real-time is critical to ensure the robot's safe navigation and to prevent collisions. Additionally, accurate mapping and localization are essential for effective path planning and autonomous operation in complex environments. The integration of these techniques into the Robot Operating System 2 (ROS2) framework, which offers modularity and scalability, has not been extensively explored.

This project aims to address these challenges and develop an efficient and cost-effective solution for obstacle detection and avoidance in differential mobile robot navigation. By leveraging the strengths of differential robots and LiDAR sensors, the project will focus on developing algorithms and techniques for real-time obstacle detection, accurate localization, efficient path planning, and deep learning-based obstacle avoidance. The proposed project will investigate state-of-the-art techniques in obstacle detection, mapping, and localization for mobile robot navigation. It will explore algorithms that effectively process and analyze data from LiDAR sensors to detect and classify obstacles in real-time. Additionally, advanced SLAM algorithms will be researched to enable accurate mapping and localization in dynamic environments. The implementation of these algorithms, including the deep learning-based obstacle avoidance trained in indoor environments, on the ROS2 framework will provide a robust and scalable platform for testing and evaluation. The deep learning-based obstacle avoidance component will be a crucial part of the developed system. By training the robot in indoor environments using diverse obstacle scenarios, the deep learning model will learn to identify and respond to obstacles in real-time. This approach will enhance the robot's ability to make precise decisions based on its position, orientation, and the presence of obstacles, thereby improving overall navigation safety and efficiency.

The significance of this project lies in the development of an efficient and cost-effective mobile robot system capable of navigating in unknown environments with dynamic obstacles. By leveraging the advantages of differential robot platforms, LiDAR sensors, and deep learning techniques trained in indoor environments, the project aims to address the challenges of real-time obstacle detection while considering cost-efficiency and practicality for real-world applications. The findings of this research will contribute to the field of robotics and autonomous systems by providing insights into effective obstacle detection and avoidance techniques, including the incorporation of deep learning trained in indoor environments, and their integration into the ROS2 framework.

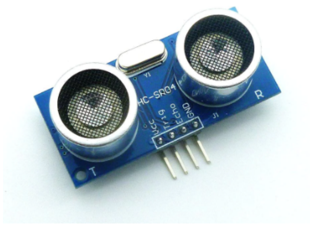
1.2 Overview of mobile robot obstacle avoidance

Mobile robot navigation is a fundamental aspect of autonomous robotics, enabling robots to move and operate in unknown or dynamic environments. It involves the ability to perceive the surroundings, plan optimal paths, and execute motion control to avoid obstacles and reach desired destinations. Effective navigation is crucial for various applications, including industrial automation, logistics, surveillance, search and rescue, and service robotics. Mobile robots can navigate in both indoor and outdoor environments, each presenting unique challenges and requirements.

1. **Indoor Navigation:**
   Indoor navigation typically involves structured environments such as warehouses, hospitals, or factories. Robots operating in indoor environments often rely on predefined paths or maps. They utilize sensors like encoders, gyros, or cameras for localization and navigation. Obstacle avoidance in indoor environments primarily focuses on static objects such as walls, furniture, or machinery.
2. **Outdoor Navigation**:
   Outdoor navigation poses additional challenges due to unstructured and dynamic environments. Robots operating outdoors need to handle uneven terrains, unpredictable obstacles (both static and dynamic), and changing lighting conditions. Outdoor mobile robot navigation requires advanced sensing and perception capabilities to handle these challenges effectively.

In the context of mobile robot navigation, several challenges and requirements arise. Robots must navigate safely and efficiently in environments where obstacles, both static and dynamic, may be present. They should possess the capability to sense and perceive the environment, accurately detect obstacles, and make informed decisions in real-time. Various sensors can be employed for obstacle detection, Figure 1 illustrates the different types of sensors commonly used for obstacle detection.

(a) kinect depth camera


(b) Ultrasonic sensor

(c) RP-LiDAR sensor

Figure 1. Types of sensors for obstacle detection

Subfigure (a) shows vision sensors such as cameras, which provide detailed visual information about the robot's surroundings [6]. Subfigure (b) displays ultrasonic sensors that emit sound waves and measure the time it takes for the waves to bounce back after hitting an object [7]. Finally, subfigure (c) presents LiDAR sensors that use laser beams to generate 3D point clouds of the environment [8, 9].

1. **Vision Sensors:** Vision sensors, such as cameras, provide detailed visual information about the robot's surroundings. In-depth vision sensors, also known as depth cameras or depth sensors, provide not only detailed visual information but also depth information about the robot's surroundings. These sensors capture the 3D structure of the environment by measuring the distance between the sensor and objects in the scene. One popular example of an in-depth vision sensor is the Microsoft Kinect sensor, which combines an RGB camera with an infrared depth sensor.
   Unlike traditional vision sensors that rely solely on 2D images, in-depth vision sensors enable the estimation of object distances and create a depth map of the scene. This additional depth information enhances the understanding of the environment, allowing for more accurate object detection, segmentation, and scene reconstruction. It also provides valuable input for tasks such as 3D mapping, object tracking, and gesture recognition.
   However, in-depth vision sensors have their limitations. They require more processing power compared to regular cameras due to the additional depth data processing. They can also be sensitive to lighting conditions, as variations in lighting can affect the accuracy of depth measurements. Additionally, the complexity of object recognition and classification algorithms increases when incorporating depth information, requiring sophisticated techniques to leverage the full potential of the sensor data.
   Despite these challenges, in-depth vision sensors offer valuable insights into the 3D structure of the environment, making them suitable for applications that require a more detailed understanding of the scene and precise obstacle detection in mobile robot navigation, augmented reality, robotics, and computer vision tasks.[10]
2. **Ultrasonic Sensors:** Ultrasonic sensors emit sound waves and measure the time it takes for the waves to bounce back after hitting an object. They are cost-effective, require less power, and can detect objects within a certain range. However, ultrasonic sensors have limited accuracy and resolution compared to other sensing technologies.
3. **LiDAR Sensors:**
   Lidar, which stands for "Light Detection and Ranging" or "Laser Imaging, Detection, and Ranging," is a device used to measure distances by emitting laser pulses and analyzing the reflected light. It provides precise 3D spatial information about the surrounding environment. A Lidar system typically consists of a laser emitter, scanning mechanism, receiver, and data processing unit. The emitted laser beam

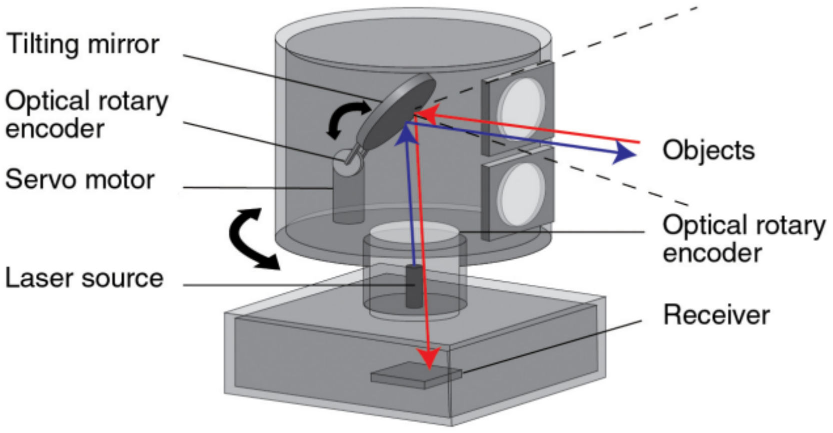


Figure 2. *Principle of mechanical spinning LiDAR [11].*

reflects off objects in its path, and the receiver detects the reflected light. By measuring the time it takes for the laser pulses to return, Lidar can calculate accurate distances to objects. This technology is widely used in various applications, including autonomous vehicles, robotics, mapping, and remote sensing.Figure 2 shows the LIDAR mechanism [6, 8, 12, 7, 9]..

Numerous approaches and techniques have been developed to address the challenges of mobile robot navigation. These include traditional methods such as sensor fusion, path planning algorithms, and control systems based on classical robotics principles. Recent advancements in perception technologies, including computer vision, LiDAR, and depth sensors, have further improved the ability of mobile robots to perceive and understand their surroundings.

Obstacle detection and avoidance play a critical role in mobile robot navigation. Real-time identification and classification of obstacles are essential for safe and efficient movement. Various techniques, such as sensor-based approaches, probabilistic methods, and machine learning algorithms, have been employed for obstacle detection. These techniques rely on data from sensors such as LiDAR, cameras, ultrasonic sensors, or a combination of multiple sensors to accurately perceive and interpret the environment.

The literature on mobile robot navigation provides valuable insights into the challenges, techniques, and advancements in this field. Researchers have explored different algorithms, sensor configurations, and control strategies to improve the navigation capabilities of mobile robots. Additionally, the integration of simultaneous localization and mapping (SLAM) techniques, which enable robots to build maps of their environment while simultaneously estimating their position, has significantly enhanced navigation performance in unknown environments.

Overall, understanding the state-of-the-art in mobile robot navigation is crucial for the successful implementation of the proposed project. It provides a foundation for exploring innovative solutions, developing efficient algorithms for obstacle detection, and integrating SLAM techniques to enable the robot to navigate autonomously and safely in unknown environments. By leveraging the existing knowledge and advancements, this study aims to contribute to the field by addressing the challenges of mobile robot navigation and developing an efficient and cost-effective solution for obstacle detection and avoidance.

1.3 Differential Drive Mobile Robots

Differential mobile robots utilize a differential drive mechanism, where each wheel is independently controlled. This design allows the robot to rotate in place by spinning its wheels in opposite directions. Differential robots are known for their simplicity, robustness, and cost-efficiency. They are well-suited for applications requiring tight maneuverability and can traverse various terrains effectively.

Kinematics: The Differential Drive Wheeled Mobile Robots (DDWMRs) usually have two independently driven wheels and one or more empowered wheels at the rear as a balance. An important issue of the differential driving of mobile robots which needs considering is that their motion controller design is mostly based on kinematic models. The main reason is that dynamic models are more complex than kinematic models and mobile robots usually use only the low speed of the motor to control the loop [13].

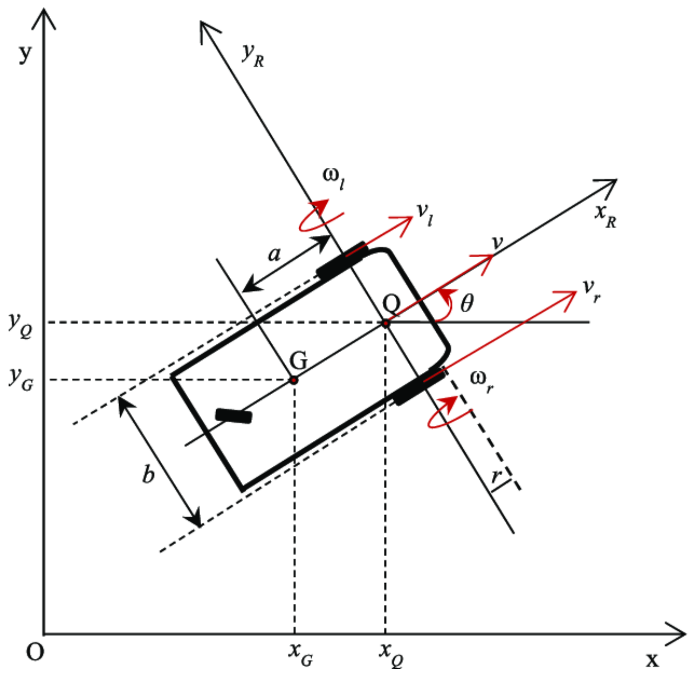


Figure 3. *A geometry of a 3-wheel differential drive mobile robot [14].*

Figure 3 illustrates the geometry of a 3-wheel differential drive mobile robot, where:

- v is the linear velocity of the DDWMR (m/s),
- θ is the orientation of the DDWMR (rad),
- ω_r is the angular velocity of the right wheel (rad/s),
- ω_l is the angular velocity of the left wheel (rad/s),
- v_r is the linear velocity of the right wheel (m/s),
- v_l is the linear velocity of the left wheel (m/s),
- r is the radius of the right and the left wheels (m),
- b is the distance between the right and the left wheels (m),
- Q is the center of the axis between the right and the left wheels,
- G is the center of gravity of the DDWMR,
- a is the distance between Q and G (m).

The kinematic model equations depend on the geometrical structure of the DDWMR. However, most of the 3-wheel DDWMRs have the same kinematic equation which is constructed as follows [15, 16, 17, 18, 14]:

$$x_G = x_Q - a\cos(\theta) \tag{1.1}$$

$$y_G = y_Q - a\sin(\theta) \tag{1.2}$$

We assume that:

- The wheels are rolling without slipping,
- The center of gravity G coincides with the point Q,
- The guidance axis is perpendicular to the robot plane.

Based on Fig.3 we get:

$$v_r = v + \frac{b}{2}\dot{\theta} \tag{1.3}$$
$$v_l = v - \frac{b}{2}\dot{\theta} \tag{1.4}$$

Adding and subtracting Eqs. (1.3) and (1.4), we get:

$$v = \frac{1}{2}(v_r + v_l) \tag{1.5}$$
$$\dot{\theta} = \frac{1}{b}(v_r - v_l) \tag{1.6}$$

Due to the non-slipping assumption, we have $v_r = r\omega_r$ and $v_l = r\omega_l$.

From Fig.3, we get:

$$\dot{x} = \dot{x}_Q = v\cos(\theta) \tag{1.7}$$
$$\dot{y} = \dot{y}_Q = v\sin(\theta) \tag{1.8}$$
$$\dot{\theta} = \omega \tag{1.9}$$

Equations (1.7), (1.8), and (1.9) describe the kinematics of the DDMR, providing essential information for motion control and trajectory planning.

Differential drive mobile robots offer several advantages over other types of mobile robots:

- **Simplicity and Cost-effectiveness:** The differential drive mechanism is relatively simple and straightforward to implement. It requires fewer components and actuators compared to other drive systems, making it cost-effective and easily maintainable.
- **Agility and Maneuverability:** Due to the independent control of each wheel, differential drive robots can exhibit high maneuverability and agility. They can perform tight turns and navigate through narrow spaces effectively. This makes them

suitable for applications that require precise and intricate movements.

- **Energy Efficiency:** Differential drive robots are efficient in terms of power consumption. They require fewer actuators and mechanisms compared to other mobile robot types, resulting in improved energy efficiency. This advantage is particularly important for battery-powered robots or robots operating in resource-constrained environments.
- **Robustness and Stability:** Differential drive robots are inherently stable and robust. They can handle uneven terrains and variations in the ground plane without compromising their performance. This robustness makes them suitable for outdoor applications or environments with challenging surfaces.
- **Versatility and Adaptability:** Differential drive mobile robots can be designed and adapted for various applications. They find applications in exploration, surveillance, industrial automation, and even in hobbyist robotics due to their versatility. Their simple and modular design allows for easy customization and integration of additional sensors or tools.

These advantages make differential drive mobile robots popular in many fields, where their simplicity, agility, energy efficiency, robustness, and versatility are highly valued. They provide an effective solution for tasks that require precise maneuvering, flexibility, and adaptability in different environments.

1.3.1 Odometry

Odometry is a technique used to estimate the position and orientation of a moving object by analyzing its self-generated motion data, such as wheel rotations or inertial sensor readings. It plays a crucial role in navigation and localization systems, providing real-time information about the object's movement in its environment. However, odometry measurements are prone to errors and cumulative drift, requiring integration with other sensing modalities for accurate positioning.

"ros control" Overview

"ros control" is a powerful framework within ROS (Robot Operating System) that provides a standardized interface and architecture for controlling robot hardware. It offers a comprehensive set of libraries and tools designed for developing robust robot controllers and seamlessly integrating them with hardware components [19].

At the core of "ros control" are several essential components:

- **Hardware Interface**: Serving as a vital link between low-level hardware and higher-level controllers, the hardware interface defines the necessary commands and states required to effectively control robot actuators and retrieve sensor data.
- **Controller Manager**: The controller manager handles the lifecycle management of robot controllers. It is responsible for loading, starting, stopping, and seamlessly switching between controllers as needed. By interfacing with the hardware interface, the controller manager can send commands and receive valuable feedback from the robot's sensors and actuators.
- **Controllers**: Controllers play a pivotal role in executing control algorithms and strategies for the robot. They utilize sensor feedback and generate appropriate control commands to achieve desired robot behavior and motion.
- **Joint State Controller**: The joint state controller collects crucial information such as joint position, velocity, and effort from the hardware interface. This data is then published as ROS topics, providing a standardized way to access and utilize joint state information.
- **Robot State Publisher**: By combining joint state information with the robot's kinematic model, the robot state publisher computes and publishes the robot's pose, which includes both position and orientation, as a ROS topic. This pose information is valuable for various higher-level tasks, such as localization and mapping.

Odometry Computation in "ros control".

In "ros control", odometry for a differential drive robot is computed by utilizing the wheel velocities provided by the hardware interface. Through a process called dead reckoning, these wheel velocities are integrated over time to estimate the robot's pose, which consists of its position and orientation.

The mathematical description for integrating the wheel velocities over time to estimate the robot's pose is as follows:

Let v_l represent the linear velocity of the left wheel, v_r denote the linear velocity of the right wheel, L indicate the fixed wheelbase of the robot, and Δt represent the time interval between successive updates of the wheel velocities. We can calculate the incremental displacement Δx, Δy, and change in orientation $\Delta\theta$ over this time interval.

The incremental displacement is given by:

$$\Delta s = \frac{v_l + v_r}{2} \cdot \Delta t \tag{1.10}$$

$$\Delta x = \Delta s \cdot \cos(\theta) \tag{1.11}$$

$$\Delta y = \Delta s \cdot \sin(\theta) \tag{1.12}$$

The change in orientation is determined by:

$$\Delta\theta = \frac{v_r - v_l}{L} \cdot \Delta t \tag{1.13}$$

Starting from an initial pose (x_0, y_0, θ_0), we can obtain the updated pose by integrating these incremental values over time:

$$x_t = x_{t-1} + \Delta x \tag{1.14}$$

Starting from an initial pose (x_0, y_0, θ_0), we can obtain the updated pose by integrating these incremental values over time:

$$x_t = x_{t-1} + \Delta x \tag{1.15}$$

$$y_t = y_{t-1} + \Delta y \tag{1.16}$$

$$\theta_t = \theta_{t-1} + \Delta\theta \tag{1.17}$$

where x_t, y_t, and θ_t represent the updated pose of the robot at time t, while $(x_{t-1}, y_{t-1}, \theta_{t-1})$ represent the previous pose. where x_t, y_t, and θ_t represent the updated pose of the robot at time t, while $(x_{t-1}, y_{t-1}, \theta_{t-1})$ represent the previous pose.

It's important to note that these equations provide a basic estimation of the robot's pose based solely on the wheel velocities. However, errors can accumulate over time due to factors such as wheel slip and uneven terrain. Therefore, additional techniques like sensor fusion with external localization systems can be employed to enhance the accuracy and reliability of pose estimation [19].

1.3.2 Challenges of Odometry Computation for Localization

Odometry computation, which relies on measuring wheel movements and rotations, is commonly used for robot localization. However, it faces several challenges that can lead to inaccuracies and drift over time. Some of the common problems include:

Wheel slippage: When a robot encounters slippery surfaces or uneven terrains, the wheels may slip, causing discrepancies between the expected and actual robot movements. This can result in inaccurate odometry estimates and subsequent localization errors [20].

Wheel calibration: Imperfections in wheel manufacturing or variations in terrain conditions can lead to variations in wheel diameter or slippage characteristics. These discrepancies can introduce biases in odometry computations, affecting localization accuracy.

Cumulative errors: Odometry computations are prone to cumulative errors that accumulate over time. Each small error in measuring wheel movements can lead to significant drift in the estimated robot pose as the robot moves along its trajectory [20].

Simultaneous Localization and Mapping (SLAM) Solution: To overcome the limitations of odometry-based localization, Simultaneous Localization and Mapping (SLAM) algorithms are employed. SLAM combines sensor measurements, such as laser scans or camera images, with odometry information to simultaneously construct a map of the environment and estimate the robot's pose within that map.

SLAM addresses the challenges of odometry computation for localization in the following ways:

Data fusion: By fusing data from multiple sensors, such as LIDAR or cameras, SLAM algorithms can compensate for the limitations of odometry. Sensor data provides additional information about the environment, allowing the algorithm to better estimate the robot's pose and reduce the impact of odometry errors [20].

Loop closure detection: SLAM algorithms incorporate loop closure detection techniques to identify previously visited locations. By detecting loops in the robot's trajectory, SLAM algorithms can correct accumulated odometry errors and align the estimated map with the real-world environment [21, 22].

Map refinement: SLAM algorithms continuously refine the constructed map based on new sensor measurements and odometry updates. By iteratively optimizing the map and the robot's pose estimates, SLAM algorithms can improve localization accuracy and mitigate drift caused by odometry errors [21].

By leveraging sensor data and advanced algorithms, SLAM provides a robust solution to address the challenges of odometry computation for localization. It enables accurate mapping of the environment while simultaneously estimating the robot's pose within that map, resulting in more reliable and precise localization capabilities for autonomous robots [23, 20].

Including SLAM in the navigation system enhances the overall accuracy and reliability of the robot's localization, making it an essential component in autonomous navigation systems operating in complex indoor environments [23, 24].

1.4 SLAM and Path Planning in Robotics

This section provides the theoretical background and mathematical descriptions for both SLAM and path planning problems, as well as discusses the methods that will be used in this project.

1.4.1 Simultaneous Localization and Mapping (SLAM)

Simultaneous Localization and Mapping (SLAM) is a fundamental technique in mobile robot navigation that addresses the challenge of estimating the robot's position (localization) while simultaneously building a map of the environment. SLAM algorithms combine sensor measurements, such as odometry, LiDAR scans, or camera images, with probabilistic models to estimate the robot's pose and create a representation of the environment. SLAM techniques are critical for mobile robots operating in unknown or unexplored environments, as they enable the robot to create maps on-the-fly and localize themselves without relying on pre-existing maps. Advanced SLAM algorithms, such as GraphSLAM or FastSLAM, have been developed to handle dynamic environments, large-scale mapping, and the presence of dynamic obstacles[25] .

Formulation of SLAM Problem

SLAM is a fundamental problem in robotics that was first introduced in 1986 at the IEEE Robotics and Automation Conference in San Francisco by researchers Peter Cheeseman, Jim Crowley, and Hugh Durrant-Whyte [21]. The primary goal of SLAM is to place a mobile robot in an unknown environment and simultaneously build a model map while determining the robot's location within that map using different sensors. A significant challenge in SLAM is dealing with noise in sensor measurements and uncertainties in robot motion.[26]

Mathematical Description of SLAM

Let $X_T = \{x_0, x_1, x_2, \ldots, x_T\}$ represent the robot path, where x_0 is known. The relative motions of the robot are given by $U_T = \{u_0, u_1, u_2, \ldots, u_T\}$, where u_t represents the robot motion between time steps $t-1$ and t. $M_T = \{m_0, m_1, m_2, \ldots, m_{n-1}\}$ represents the true map of the environment, where n is the number of landmarks, and m_i are the vectors representing the positions of the landmarks. $Z_T = \{z_0, z_1, z_2, \ldots, z_T\}$ represents the robot measurements at each time step.

SLAM can be defined as the problem of recovering the map M and robot path X_T from the odometry U_T and observations Z_T.

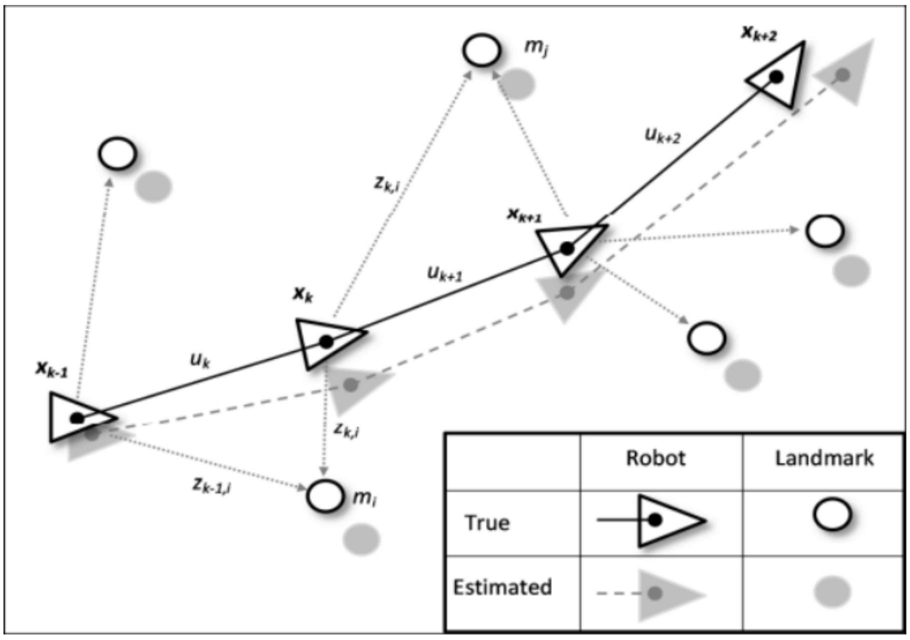


Figure 4. Example of the SLAM problem [21]

The full SLAM problem estimates the posterior probability over the full data X_T and M and is given by:

$$p(X_T, M | Z_T, U_T) \tag{1.18}$$

The online SLAM problem estimates the posterior probability over the current pose x_t and map M given the observations and odometry up to time t:

$$p(x_t, M | Z_t, U_t) \tag{1.19}$$

Solutions to the SLAM Problem

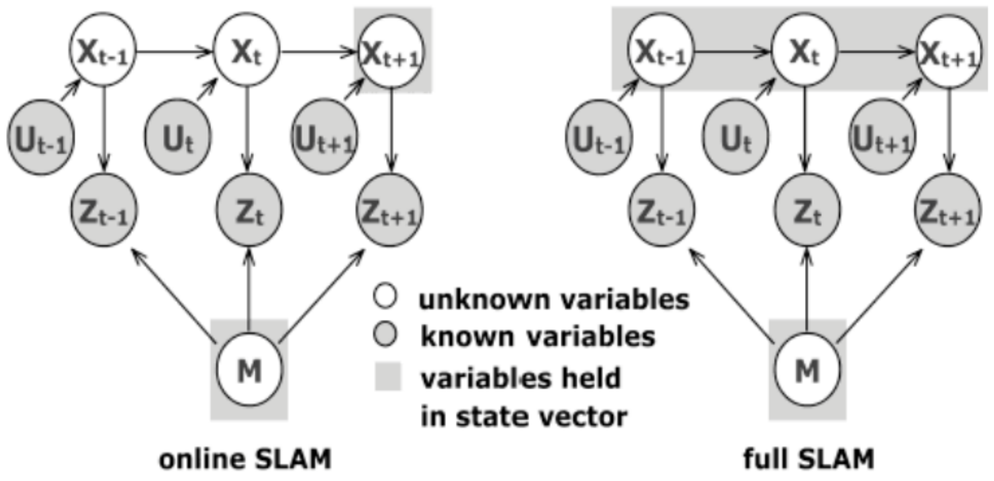


Figure 5. Forms of the SLAM problem [27]

Solving the SLAM problem requires finding a representation for both the observation and motion model that allows for efficient computation of the prior and posterior distributions. Various algorithms have been developed to tackle the SLAM problem.

SLAM algorithms

- **EKFSLAM** (Extended Kalman Filter SLAM) is a well-known and widely used algorithm. It uses the Extended Kalman Filter to estimate the robot's position and map the environment. EKFSLAM has easy implementation and is suitable for small-scale maps. However, it has limitations in handling large maps due to its computational complexity, which grows quadratically with the number of landmarks[28].
- **FastSLAM** is a particle filtering-based algorithm that represents the posterior belief by a set of particles, each of which carries a map and a pose hypothesis. It can handle non-linearities and uncertainties in the robot's motion and measurement models. FastSLAM is more flexible than EKFSLAM and can use negative information, improving map accuracy. It is particularly useful in scenarios where the environment is highly dynamic[29]. However, FastSLAM requires a large number of particles to be robust, making it computationally demanding.
- **GraphSLAM** is a graph-based algorithm that models the environment as a graph, with nodes representing robot poses and landmarks as well as edges representing constraints between them. It solves the SLAM problem by optimizing the graph, estimating the robot's trajectory and the map simultaneously. GraphSLAM offers

excellent flexibility and scalability, making it suitable for large-scale mapping. It can handle loop closures and re-linearize the graph to incorporate new information. GraphSLAM is known for its robustness and ability to scale well with the size of the environment. Additionally, GraphSLAM provides the flexibility to handle both Gaussian and outlier distributions, allowing for more accurate and robust map estimation. It can incorporate negative information and effectively deal with loop closures by re-linearizing the graph. These characteristics make GraphSLAM well-suited for complex and dynamic environments.
Moreover, GraphSLAM exhibits excellent scalability and parallelizability. It can handle large-scale mapping tasks and efficiently utilize parallel computing resources. This scalability and parallelizability are particularly valuable for your project, where you may encounter large datasets or require real-time mapping capabilities.

based on the comparison in Table 1, the strengths of GraphSLAM in terms of flexibility, scalability, and robustness make it the most suitable algorithm for this project. It can effectively handle the challenges of mapping and localization in real-world environments, providing accurate and reliable results. Here is the comparison of the three main categories of SLAM algorithms, where:

Table 1. Comparison of the three main categories of SLAM algorithms [30].

Algorithm	Complexity	Distribution	Characteristics
EKFSLAM	$O(n^2)$	Gaussian	Easy implementation, limited scalability
FastSLAM	$O(k \cdot \log n)$	Any	Can use negative information, scalability
GraphSLAM	$O(e)$	Gaussian+outlier rejection	Re-linearization, flexibility, scalability

n is the number of landmarks.
k is the number of particles.
e is the number of edges.

1.4.2 Path Planning Techniques

Path planning is a fundamental task in robotics that involves finding an optimal path for a robot to navigate from its current position to a desired goal while avoiding obstacles in the environment. Various path planning techniques have been developed to address this problem. Here are some commonly used path planning techniques:

1. **Grid-Based Methods**: Grid-based methods discretize the environment into a grid of cells. They represent obstacles as occupied cells and free space as unoccupied cells. The robot's path is planned by searching through this grid using algorithms such as RRT algorithm to find the shortest path.
2. **Potential Field Methods**: Potential field methods model the environment as a potential field, where attractive forces pull the robot towards the goal and repulsive forces push it away from obstacles. The robot plans its path by following the gradient of the potential field toward the goal while avoiding high-repulsive regions.
3. **Sampling-Based Methods**: Sampling-based methods, such as Rapidly-exploring Random Trees (RRT) and Probabilistic Roadmaps (PRM), build a roadmap or tree structure in the configuration space of the robot. These methods randomly sample the configuration space, connect the samples to form a graph, and then search for a feasible path through this graph.
4. **Visibility Graphs**: Visibility graphs construct a graph by connecting visible points in the environment. The vertices of the graph represent points of interest, and the edges represent straight-line paths between visible points. Path planning is performed by finding a path in this graph using algorithms like Dijkstra's algorithm.
5. **Optimization-Based Methods**: Optimization-based methods formulate path planning as an optimization problem. They define an objective function that balances criteria such as path length, clearance from obstacles, and smoothness. Algorithms like the A* algorithm or genetic algorithms can be used to search for the optimal path that minimizes the objective function.

These are just a few examples of path planning techniques used in robotics. The choice of technique depends on factors such as the environment complexity, computational resources, and specific requirements of the robot's task. Researchers continue to develop new algorithms and improve existing techniques to achieve more efficient and robust path planning for various robotic applications.

Rapidly Exploring Random Trees (RRT)

Rapidly Exploring Random Trees (RRT) is a widely used algorithm in robotics for path planning. Its primary objective is to explore the state space efficiently by incrementally constructing a tree of connected nodes. RRT generates random nodes within the state space and connects them to form a path toward the goal position, ensuring collision-free paths based on map constraints. Although the resulting path may not be optimal, RRT is renowned for its ability to explore large state spaces quickly [31].

The RRT algorithm follows the steps outlined below:

1. Set up a graph with the initial pose q_{start} as the only vertex.

2. For each iteration, randomly sample a point q_{rand} within the obstacle-free space C_{free}.

3. Find the nearest vertex q_{near} in the tree to the new sample q_{new} within a specified distance threshold ϵ.

4. If there are no obstacles between q_{near} and q_{new}, add q_{rand} to the vertex set and include the edge $(q_{\text{near}}, q_{\text{rand}})$ in the edge set.

Figure 6 provides an illustration of the RRT path planning process.

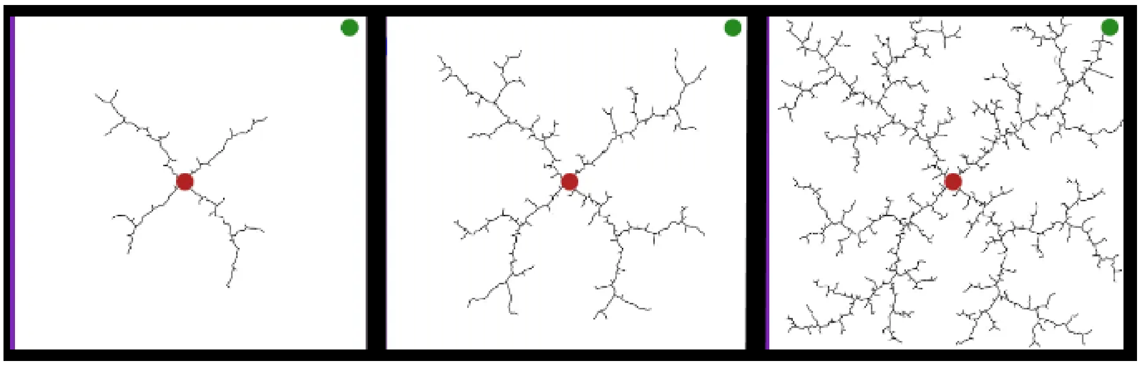


Figure 6. *Illustration of RRT path planning [31].*

By iteratively expanding the tree towards the goal position, RRT gradually constructs a feasible path in complex environments. Although the resulting path might not be the shortest or optimal, RRT's efficiency and ability to handle high-dimensional state spaces make it a valuable algorithm for robotic path planning tasks.

1.5 Deep Learning-Based Obstacle Avoidance

1.5.1 Introduction to Deep Learning

Deep learning is a specialized branch within the field of artificial intelligence that primarily concentrates on developing expansive models of neural networks, enabling them to effectively derive precise decisions based on data. This area of study is especially well-suited for circumstances where the data exhibits intricate characteristics and substantial datasets are readily accessible[32].

In the domain of robotics and autonomous systems, deep learning has shown great promise in various applications, including perception and localization, object recognition, and decision-making[33]. An essential application encompassing both obstacle avoidance and path planning lies in the domain of robotics and autonomous systems, wherein deep learning techniques are employed. The objective of these techniques is to empower robots and autonomous vehicles with the capability to discern obstacles present in their environment and navigate around them in a proficient manner.

1.5.2 Brief Overview of Deep Learning Principles

Deep learning principles are rooted in the concept of artificial neural networks, which consist of interconnected layers of artificial neurons or nodes. These networks are trained using large data-sets, where the weights and biases of the network's connections are adjusted iteratively to minimize the difference between the predicted output and the actual output [34].

The key principles underlying deep learning for collision-free trajectory generation include:

- **Neural Network Architecture**: Deep learning models for obstacle avoidance typically employ convolutional neural networks (CNNs) or recurrent neural networks (RNNs). CNNs are particularly effective in image-based perception tasks, as they can automatically learn relevant features from raw sensor data, such as camera images or LIDAR scans. RNNs, on the other hand, are well-suited for sequential data, such as time-series sensor readings or trajectory information.
- **Training Data Acquisition**: The success of deep learning heavily relies on the availability of large and diverse training datasets. In the context of obstacle avoidance, this entails collecting labeled data that includes both obstacle-free and obstacle-rich scenarios. The data may be acquired through sensors, such as cameras or depth sensors, and annotated with corresponding obstacle labels or bounding boxes.
- **Supervised Learning**: In supervised learning, deep learning models are trained using input-output pairs, where the inputs are sensor data representing the environment, and the outputs are the desired obstacle avoidance actions. By minimizing the discrepancy between predicted and desired outputs, the network learns to associate specific input patterns with appropriate obstacle avoidance responses.
- **Transfer Learning**: Transfer learning is a valuable technique in deep learning, allowing models trained on large data-sets in related domains to be fine-tuned for specific tasks with smaller data-sets. By leveraging pre-trained models on general object recognition tasks, deep learning models for obstacle avoidance can benefit from learned features and reduce the need for extensive training on limited obstacle-specific data-sets.
- **Model Evaluation and Validation**: Deep learning models for obstacle avoidance need to undergo rigorous evaluation and validation processes. This involves testing

the trained models on unseen data or real-world scenarios, assessing their accuracy, robustness, and generalization capabilities. Evaluation metrics such as precision, recall, and F1 score can quantify the performance of the models and guide further improvements.

By employing these deep learning principles, researchers and engineers strive to develop effective obstacle detection and avoidance systems that can reliably navigate complex environments. However, challenges such as data-set limitations, computational power constraints, and model interpretation ability continue to be areas of active research and development. Here's a simple example of a flowchart illustrating the general process of deep learning:

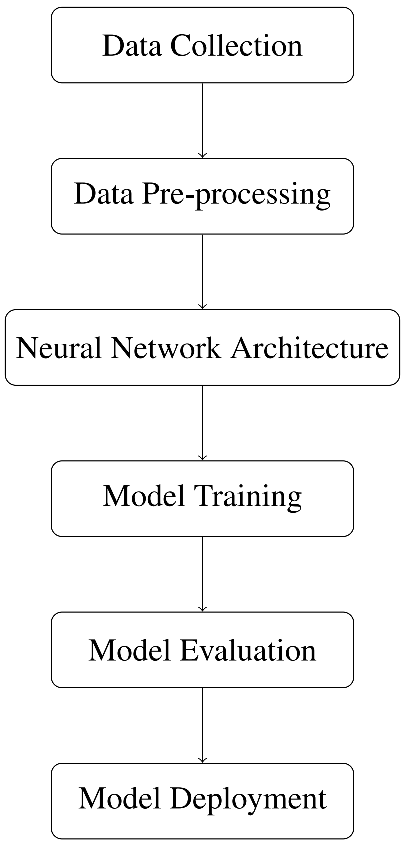


Figure 7. Flowchart of the Deep Learning Process

The flowchart in Figure 7 depicts the typical steps involved in a deep learning process. It begins with raw data, which undergoes pre-processing to transform it into a suitable format for the neural network. The neural network is then trained using the pre-processed data, and the trained model is evaluated using appropriate metrics. Finally, the validated model is ready for deployment in real-world applications.

This flowchart provides a high-level overview of the deep learning process, showcasing the sequential stages involved in training and deploying deep learning models.

1.5.3 Methods of Deep Learning for Obstacle Avoidance and Path Planning

Obstacle avoidance is a critical aspect of various applications, such as autonomous vehicles, robotics, and surveillance systems. Deep learning techniques have shown great promise in tackling this challenge by enabling the extraction of meaningful information from sensor data for accurate obstacle detection and avoidance. In this section, we discuss several widely employed methods in deep learning for obstacle avoidance.

- **Convolutional Neural Networks (CNNs)** have emerged as a powerful method for image-based obstacle detection. Leveraging their ability to capture spatial patterns and features from images, CNNs excel at tasks such as object detection and segmentation. They can effectively process raw input images or transformed representations of LiDAR point cloud data[35].
- **Recurrent Neural Networks (RNNs)** are particularly suitable for processing sequential data, making them valuable for obstacle avoidance tasks involving time-series sensor data. RNNs can model temporal dependencies and capture dynamic patterns, enabling tasks such as trajectory prediction and motion planning[36].
- **Long Short-Term Memory (LSTM)** networks, a type of RNN, excel in handling long-term dependencies in sequential data. Their application to obstacle avoidance scenarios is particularly advantageous when historical sensor data, such as LiDAR point clouds or camera frames, need to be considered for accurate predictions or decisions[37].
- **Generative Adversarial Networks (GANs)** have gained attention for their ability to generate synthetic data that closely resembles real-world obstacles. GANs can be leveraged to augment existing data-sets or generate synthetic training data, thereby enhancing the robustness and diversity of obstacle avoidance models[38].
- **Transfer learning** techniques enable the utilization of pre-trained deep learning models, which have been trained on large-scale data-sets for tasks such as image recognition or object detection. By fine-tuning or adapting these models to obstacle avoidance tasks, transfer learning can leverage their learned representations, potentially reducing training time and data requirements[39].
- **Reinforcement Learning (RL)** is another method for obstacle avoidance that involves an agent interacting with an environment and learning optimal actions through trial and error. RL algorithms, such as Q-learning and Deep Q-Networks (DQN), can learn to navigate complex environments by maximizing long-term rewards. RL-based approaches have been successfully applied to obstacle avoidance tasks, allowing agents to learn effective strategies for avoiding obstacles [40].

- **Feed-forward Neural Networks (FNNs)**, also known as multi-layer perceptrons, have been widely employed for obstacle avoidance tasks. These networks, comprising an input layer, one or more hidden layers, and an output layer, are adept at processing high-dimensional data. FNNs, including feed-forward neural networks, have been successfully utilized for obstacle detection and decision-making based on 2D LiDAR data.

These methods offer a range of techniques for obstacle avoidance using deep learning, providing flexibility and adaptability to various scenarios and data types.

Among these methods, the best methods for 2D LiDAR data in deep learning algorithms for obstacle avoidance include Convolutional Neural Networks (CNNs) and Feed-forward Neural Networks (FNNs). CNNs excel at processing images and can effectively extract features from 2D LiDAR data representations, while FNNs provide a powerful tool for real-time decision-making based on such data.

CHAPTER 2

Lidar-Based Obstacle Avoidance

Lidar-based obstacle detection and avoidance, when combined with Simultaneous Localization and Mapping (SLAM) and deep learning techniques, offers a powerful solution for robust navigation in complex environments. Lidar sensors provide accurate distance measurements and generate detailed 3D point clouds, enabling precise obstacle detection. SLAM algorithms enhance localization accuracy and create environment maps, while deep learning models trained on diverse obstacle scenarios improve real-time obstacle recognition and response. This fusion of Lidar, SLAM, and deep learning enables efficient and reliable obstacle detection and avoidance for autonomous systems.

2.1 State-of-the-art techniques and advancements in lidar-based obstacle avoidance

LiDAR-based obstacle avoidance has witnessed significant advancements in recent years, revolutionizing the field with cutting-edge techniques and technologies. One remarkable development lies in the integration of deep learning algorithms, such as convolutional neural networks (CNNs) and recurrent neural networks (RNNs), for highly accurate object detection and classification within LiDAR point cloud data. These powerful algorithms excel at identifying intricate patterns and features from vast training data-sets, enabling robust obstacle detection even in challenging environments. By harnessing the potential of deep learning, LiDAR-based systems have achieved remarkable precision and minimized false positives, ensuring the safety and reliability of autonomous vehicles and robots.

Moreover, the fusion of LiDAR sensors with simultaneous localization and mapping (SLAM) algorithms has propelled obstacle avoidance capabilities to new heights. SLAM algorithms allow robots and vehicles to simultaneously create a map of their surroundings while localizing themselves within that map. By integrating LiDAR data into SLAM frameworks, obstacles can be accurately identified and localized in real-time. This integration has proven invaluable, enabling autonomous systems to navigate dynamic environments, avoid obstacles, and plan optimal paths effectively.

In addition to advanced deep learning and SLAM techniques, refined LiDAR data processing and feature extraction methodologies have played a crucial role in enhancing obstacle detection. Traditional LiDAR data processing involved computationally intensive segmentation and clustering techniques. However, recent advancements have introduced efficient voxel-based or grid-based representations that facilitate real-time obstacle detection and tracking. By organizing LiDAR point clouds into spatially structured voxel grids or grids, these methods enable efficient processing and quicker response times, empowering autonomous systems to make instantaneous decisions and avoid obstacles in real-world scenarios[41].

Furthermore, the development of advanced filtering and noise reduction techniques has significantly improved the quality of LiDAR data for obstacle avoidance purposes. Signal processing approaches, such as adaptive filtering, outlier removal, and temporal filtering, effectively mitigate noise, outliers, and spurious reflections in the point cloud data. These techniques enhance the accuracy and reliability of LiDAR measurements, leading to more precise obstacle detection and improved overall system performance.

LiDAR sensor technology has also witnessed notable advancements, offering sensors specifically tailored for obstacle avoidance applications. Solid-state LiDARs, for instance, provide enhanced reliability, compact form factors, and higher scanning speeds compared to traditional mechanical spinning LiDARs. These technological strides enable faster data acquisition, increased resolution, and wider field-of-view, empowering LiDAR-based obstacle avoidance systems to perceive their surroundings with remarkable detail and accuracy.

In summary, the rapid progress in LiDAR-based obstacle avoidance stems from a convergence of advanced deep learning algorithms, SLAM techniques, refined data processing methodologies, and state-of-the-art LiDAR sensor technologies. These developments collectively equip autonomous systems with the ability to detect obstacles accurately, localize themselves in dynamic environments, and plan optimal paths. By embracing these advancements, LiDAR-based obstacle avoidance has transformed autonomous vehicles and robots into highly capable and safe entities, capable of navigating complex real-world scenarios with confidence.

2.2 LIDAR based obstacle-detection methods

Detection methods based on 2D LiDAR often incorporate region segmentation or clustering algorithms to divide distance measurements into correlated blocks from which different practical properties can be derived [42]. Then, the shapes of obstacles can be estimated and the stored data amount can be decreased. The segmentation process often compares two subsequent laser scan points from the LiDAR measurements, where the distance threshold is determined as constants or adapted to the scenarios. In a study by Authors et al., the measurements were divided into segments and polygonal obstacles were constructed using line fitting and corner extraction. A detailed comparison and explanation of some segment extraction methods from 2D Lidar laser points are shown in [43]. Premebida discussed some algorithms for segmenting 2D laser scans, as well as different approaches for feature recognition and extraction of three geometric primitives, including lines, circles, and ellipses [44] . Clustering laser point clouds is used to extract ordered obstacle information from cluttered point clouds and then intuitively obtain obstacle information [45] . In a study by Fernández et al., a convolution operation was used to split the laser data into clusters and detected lines in each previously detected cluster. The clustering process helps the robot fully recognize obstacles and perceive the environment[46].In terms of representing the surrounding spatial environment, the obstacle detection process using LiDAR sensors can be mainly classified into two representation approaches: grid-based and vector-based methods.

Grid-based obstacle detection method

Many techniques employ a 2D occupancy grid to represent the environment for obstacle detection and tracking. Occupancy grid maps consist of evenly spaced grid cells, which depict complex geometries when the grid resolution is sufficiently high.

In Figure 8, a sample occupancy grid map is shown, where black cells represent obstacles, gray cells indicate unknown areas and white cells represent empty space. Each cell represents a square area of the environment and stores a value indicating occupancy. Cells can be labeled as "occupied," "unknown," and "free" or can represent occupancy probability [47]. Grid-based methods have the advantage of not requiring prior knowledge of objects in the scene, allowing for the detection of various types of elements [42]. For example, Vu et al [48] updated occupancy grid maps incrementally, enabling the detection of dynamic objects without prior knowledge. Mori et al. [49] utilized grid trajectories formed by ordering laser points on the grid map to detect dynamic obstacles, aiding in segmentation and providing information about the speed and size of dynamic objects. Chen et al [50] proposed a grid-based approach to identify real-time moving obstacles by comparing sequential grid maps.

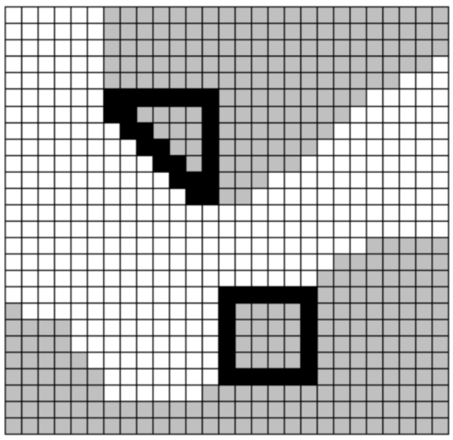


Figure 8. An example of occupancy grid representation of the environment[51].

They achieved the state of the same grid cells over time. In [52], the local map stored in the occupancy grid map was treated as a pixel map and merged into a CNN for road direction determination.

Grid-based methods are suitable for environments where features are challenging to identify and detect, making them well-suited for noisy sensors in outdoor settings. These methods also provide a framework for integrating different sensor types while considering the uncertainty of sensor data [48]. Grid maps help reduce point cloud data since densely packed laser points can be mapped to the same grid cell. With an appropriate resolution, the data can be easily searched while retaining sufficient information about detected obstacles [53]. However, higher grid resolution requires more memory and computational power for processing the divided grids. Additionally, establishing boundaries for the environment can be challenging. For instance, in open spaces like parks, acquiring enough free space becomes difficult. Grid-based methods may compromise accuracy to simplify and reduce processing of bounded data [51].

Vector-based obstacle detection method

The vector-based method is another commonly adopted approach to represent the environment, where obstacles are modeled using predefined geometric features such as lines, circles, boxes, ellipses, and rectangles. This representation enables the use of analytical geometry to calculate distances and similarities between objects [54]. [55] developed a system to handle non-rigid objects like pedestrians or foliage. They identified stable features, reasoned about occlusion geometry, and detected objects accordingly. Petrovskaya and Thrun [56] modeled the geometric features of tracked vehicles, predicting their forms and obtaining more stable vehicle reference points over time. MacLachlan and Mertz [57] fitted rectangular models to laser points, considering the corners of the rectangle as features based on laser scanner characteristics. In [40], the authors introduced a feature extraction approach based on prediction, detecting lines and circles from LiDAR data without prior knowledge of the surrounding environment. Similarly, in [40], they detected the legs of people in an indoor environment by extracting fourteen features from the laser scan and utilizing AdaBoost to create a classifier model.

Obstacle detection methods based on geometric features rely on each obstacle having at least one major feature or defined shape. These methods require prior knowledge of the elements to be detected and search for them in each frame. However, a drawback is that the real shape of obstacles can be lost, limiting the information provided by knowing the similarity between two obstacles [54]. Compared to grid-based methods, the vector-based approach offers a compact representation of the surrounding environment, making it suitable for sparse scenarios with low memory usage [40]. The compact representation allows for efficient processing and storage of the environment information.

In summary, the grid-based obstacle detection method utilizes a 2D occupancy grid to represent the environment, dividing it into grid cells to detect and track obstacles. This method does not rely on prior knowledge of objects and can detect various elements. On the other hand, the vector-based obstacle detection method represents the environment using predefined geometric features, enabling analytical calculations and detection of specific shapes. This method requires prior knowledge of the objects to be detected. Both methods have their advantages and limitations, and the choice depends on the specific requirements of the application and the characteristics of the environment.

2.3 Integration of Lidar and SLAM for Obstacle Avoidance - Lidar Data Processing and Feature Extraction

2.3.1 KartoSLAM

KartoSLAM, developed by Karto Robotics at SRI International, is a GraphSLAM algorithm that has demonstrated excellent performance in real-world scenarios and is robust against noise interference. One of the key strengths of KartoSLAM is its utilization of a highly optimized and non-iterative Cholesky matrix decomposition solver for sparse linear systems [58].It is a widely used Simultaneous Localization and Mapping (SLAM) algorithm provided by the SLAM Toolbox package in ROS (Robot Operating System). It is designed to effectively create 2D maps and estimate the robot's pose in robotic systems. in the ROS version of KartoSLAM, handling scan matching and loop-closure procedures [59]. Notably, KartoSLAM efficiently maintains a pose graph, making it suitable for large-scale environments while minimizing memory. The Karto SLAM algorithm, based on the Rao-Blackwellized Particle Filter (RBPF) framework, incorporates several unique features and improvements for enhanced efficiency and accuracy. Let's examine how Karto SLAM performs each step of building a map:

1. **Initialization:** Karto SLAM starts with an initial set of randomly distributed particles representing the robot's pose and a rough estimate of the map.
2. **Motion Update:** Using odometry data, each particle's pose is updated based on the robot's motion model. This step estimates the robot's displacement and orientation change.
3. **Scan Matching:** Karto SLAM employs scan matching techniques to align the

current laser scan measurements with the map. It finds the best pose estimate that minimizes the discrepancy between the observed scan and the expected scan generated from the particle's pose hypothesis. The scan matching algorithm in Karto SLAM efficiently determines the robot's position by comparing the laser scans with the map and adjusting the particle poses accordingly.

4. **Map Update:** The laser scan measurements are used to update the occupancy grid map representation. Karto SLAM adjusts the map based on the scan matching results and particle weights. This step refines the map by incorporating new information obtained from the laser scans and improves its accuracy over time.
5. **Re-sampling:** Re-sampling is performed based on particle weights to maintain a representative set of particles. Higher-weighted particles have a higher chance of being selected, ensuring an accurate approximation of the posterior probability density. Re-sampling helps in adapting the particle distribution to reflect the most likely robot poses and map hypotheses.
6. **Map Optimization:** Karto SLAM applies map optimization techniques to refine the estimated map. It minimizes the error between observed laser scans and map predictions by adjusting the map's parameters. This optimization step further improves the map's accuracy and alignment with the environment.
7. **Loop Closure Detection:** Karto SLAM incorporates loop closure detection mechanisms to identify previously visited locations and correct pose errors. Adjustments to the map and particle weights improve map consistency and alignment. Loop closure detection helps in detecting revisited areas and reducing the accumulation of pose estimation errors over time.

The Karto SLAM algorithm employs particles to approximate the posterior probability density of the robot's pose and the map. Each particle i consists of a pose hypothesis $x_t^{(i)}$ and a map hypothesis $m^{(i)}$. The overall estimate of the posterior probability density is approximated as a sum of weighted particles.

By leveraging scan matching techniques, Karto SLAM aligns laser scan measurements with the map, iteratively adjusting the particle's pose to minimize discrepancies. Additionally, map optimization techniques refine the estimated map by adjusting its parameters.

Karto SLAM, available in the SLAM Toolbox package of ROS, provides a powerful tool for mapping and localization in robotic systems, enabling effective obstacle avoidance and navigation in dynamic environments.

2.3.2 Rao-Blackwellized Particle Filter (RBPF)

In 2000, Murphy introduced the Rao-Blackwellized particle filter algorithm to integrate LIDAR with SLAM for obstacle avoidance in dynamic environments [60]. The RBPF algorithm combines LIDAR data processing, feature extraction, and SLAM techniques to estimate the robot's position, create an accurate map of the environment, and plan collision-free paths.

The RBPF algorithm consists of the following stages:

1. Prediction stage: Given the previously estimated pose $\mathbf{x}_{t-1}$, the particle filter generates a set of particles $\{\mathbf{x}_t^{[i]}\}_{i=1}^N$ based on the state transition function $p(\mathbf{x}_t|\mathbf{x}_{t-1}, \mathbf{u}_t)$, where $\mathbf{u}_t$ represents the control inputs. Each particle represents a possible pose hypothesis.

2. Correction stage: As new LIDAR observations arrive, the importance weights $w_t^{[i]}$ for each particle are calculated. These weights represent the probability of observing the LIDAR measurements given the corresponding particle pose. The weights are computed using the observation model $p(\mathbf{z}_t|\mathbf{x}_t^{[i]})$, where $\mathbf{z}_t$ denotes the LIDAR measurements.

3. Re-sampling stage: The particles are Re-sampled according to their weights, with higher-weighted particles being more likely to be selected. This step ensures that particles with higher probabilities are retained while discarding particles with lower probabilities, maintaining a diverse set of particles representing the posterior distribution.

4. Map estimation: For each Re-sampled particle $\mathbf{x}_t^{[i]}$, the corresponding map estimate is calculated from the trajectory and LIDAR observations. The map estimate captures the environment's structure and obstacles based on the accumulated information.

By efficiently combining LIDAR data processing and SLAM techniques, the RBPF algorithm accurately estimates the robot's pose and creates an environment map.

2.3.3 Scan Matching

Scan matching, utilizing LIDAR data, is a crucial component of obstacle avoidance systems. It aligns acquired LIDAR scans with the reference map to detect obstacles accurately in real-time. The scan matching process involves the following steps:

1. Feature extraction: The raw LIDAR data is processed to extract meaningful features that

represent the environment's structure, such as lines, corners, and planar surfaces. The extracted features, denoted as $\mathbf{Z}_t = \{\mathbf{z}_t^{[j]}\}_{j=1}^{M}$, capture specific environmental characteristics.

2. Scan matching algorithm: The Iterative Closest Point (ICP) algorithm, along with its variants, is commonly used to align the current LIDAR scan with the reference map. The algorithm iteratively minimizes the distance between corresponding points in the scans, adjusting the robot's pose estimation $\mathbf{x}_t$ to align with the environment. The ICP algorithm aims to find the transformation matrix $\mathbf{T}_{\text{ICP}}$ that minimizes the error function, capturing the discrepancy between the observed and reference scans.

Various matching methods, such as feature-to-feature matching, point-to-feature matching, and point-to-point matching, can be employed to optimize the trade-off between computational efficiency and accuracy.

By performing scan matching on LIDAR data, the system aligns the robot's pose and accurately detects obstacles, facilitating real-time obstacle avoidance.

2.3.4 Graph Optimization

Graph optimization is a widely used technique in SLAM to refine the robot's pose estimates and improve map accuracy. It formulates the SLAM problem as a graph, where the robot's poses are represented as nodes/vertices, and the relationships between poses are represented as edges. The graph optimization process involves the following tasks[61]:

1. Graph construction: The graph is constructed by adding nodes for each estimated pose $\mathbf{x}_t$ and edges that capture the relationships between poses. The edges can be established based on odometry information, visual features, or other sensor measurements. This front-end process accumulates sensor information and establishes the initial graph representation.

2. Graph optimization: The goal of graph optimization is to adjust the robot's pose estimates to satisfy the constraints imposed by the edges as much as possible. This back-end process refines the pose estimates and improves the overall consistency of the map. Various optimization algorithms, such as Karto SLAM and Cartographer, are employed for this purpose. Graph optimization aims to find the optimal set of poses $\{\mathbf{x}_t^*\}$ that minimizes the error function, capturing the deviation between estimated poses and the constraints imposed by the edges.

Graph optimization enables the refinement of pose estimates and enhances the accuracy of the environment map, leading to more robust obstacle avoidance in SLAM systems.

2.3.5 Data Association

Data association is a critical step in SLAM that involves establishing correspondences between sensor measurements obtained at different times and locations, or between map features. In the context of LIDAR data processing and feature extraction, data association refers to associating LIDAR measurements with map features to determine their correspondence and origin in the environment. This process ensures accurate state estimation and map update. The correctness of data association greatly impacts the performance of the SLAM system [20].

The data association problem can be mathematically described as follows: Given a set of LIDAR measurements $\mathbf{Z}_t$ and a map representation $\mathcal{M}$, the goal is to find the correspondence between each measurement and the corresponding map feature. Mathematically, data association aims to determine the correspondence function $\phi(\mathbf{Z}_t, \mathcal{M})$ that establishes the relationships between measurements and map features, ensuring the accurate update of the map and state estimation.

Accurate data association is crucial for robust obstacle avoidance, as it enables the SLAM system to correctly perceive the environment and make informed decisions.

2.4 2D LiDAR-based Deep Learning for Obstacle Avoidance and Path Planning

Deep Learning techniques are emerging as a viable solution for addressing obstacle avoidance challenges in autonomous mobile robots. When faced with stationary and moving obstacles in real-world scenarios, it is imperative for mobile robots to navigate towards a goal while ensuring collision avoidance and safety [62]. The integration of 2D LiDAR sensors with deep learning techniques has revolutionized obstacle avoidance for autonomous systems. Neural networks, particularly feed-forward neural networks (FNNs), play a pivotal role in processing the rich point cloud data obtained from 2D LiDAR sensors. FNNs excel at learning complex patterns and relationships within high-dimensional data, enabling informed decision-making for safe real-time navigation [63].

2.4.1 Neural Networks

Neural networks are statistical models inspired by the structure of the brain. They possess the ability to adapt and learn by estimating the parameters of a population using only a limited number of examples[64].Fundamentally, a neural network comprises essential components, including input and output layers, as well as one or more hidden layers. The input layer serves as the entry point for data, which subsequently undergoes propagation through the hidden layers before reaching the output layer. Within these hidden layers, each individual neuron conducts mathematical computations on the received data and transmits the calculated outcomes to the subsequent neurons within the network [65]. This adaptability and learning capability make neural networks powerful tools in various domains, as they can capture intricate patterns and relationships within data. In the context of obstacle avoidance and path planning, neural networks offer promising solutions for enabling autonomous systems to navigate safely and effectively in complex environments.

2.4.2 feed-forward Neural Networks

feed-forward neural networks (FNNs), also known as multi-layer perceptrons, are a fundamental type of artificial neural network. They consist of an input layer, one or more hidden layers, and an output layer, where information flows only in one direction without loops or feedback connections[66]. FNNs are widely used for tasks such as pattern recognition, classification, and regression.

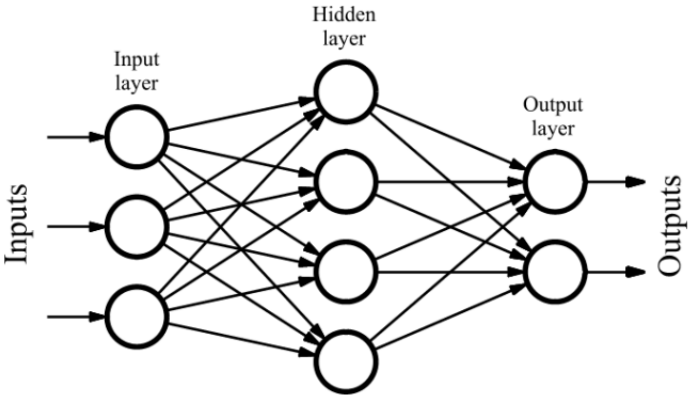


Figure 9. Architecture representation of a 3-layer feed-forward neural network [67].

Several studies have showcased the effectiveness of FNNs in obstacle detection and avoidance. Nejatbakhsh et al. [63] proposed an FNN-based approach for obstacle detection in autonomous driving applications using LiDAR data. The FNN model successfully classified LiDAR scans into obstacle and non-obstacle categories, enabling safe navigation for autonomous vehicles.

2.4.3 Data Pre-processing in Feed Forward Neural Networks

When applying deep neural networks to lidar tasks, it is generally advised to input the data directly into the network without performing dimensionality reduction or feature projection

[34]. This recommendation is based on the belief that such projections may lead to the loss of valuable information that could be essential for the task. Instead, it is preferred to provide the raw data to the network, allowing it to learn and extract relevant features on its own, given a sufficient number of examples. By avoiding premature feature extraction, the network has the potential to capture intricate patterns and relationships within the data, leading to improved performance in obstacle avoidance and other lidar-related tasks.

The first step in the FNN-based processing pipeline is data Pre-processing. This involves transforming the raw LiDAR data into a suitable format for input to the FNN model. Common Pre-processing techniques include down-sampling the point cloud, normalizing the range measurements, and converting the data into a structured representation, such as a grid or an image.

Data processing techniques In the context of 2D LiDAR-based deep learning for obstacle avoidance and path planning, several data processing techniques are commonly employed to prepare the raw LiDAR data for input to the feed-forward neural network (FNN) model. These techniques include normalization, down sampling, and one-hot encoding, each serving a specific purpose in data pre-processing.

- **Normalization** is a technique used to re-scale the range of numerical data to a standard range. It ensures that different features or variables have a similar scale, preventing one feature from dominating the learning process due to its larger magnitude. One common normalization method is max-min scaling, which transforms the data to a range between 0 and 1[68].
  Max-Min Scaling , also known as min-max scaling, is a normalization technique that linearly re-scales the data to a specific range, typically between 0 and 1. The formula for max-min scaling is given by:

  $$X_{\text{scaled}} = \frac{X - X_{\text{min}}}{X_{\text{max}} - X_{\text{min}}} \tag{2.1}$$

  where X is the original data, X_{min} is the minimum value in the data, and X_{max} is the maximum value in the data. This transformation ensures that the data is within a consistent range, which can be beneficial for the learning process of the FNN.
- **Down Sampling** is a technique used to reduce the number of data points in a data-set while preserving important information. In the context of 2D LiDAR data, down sampling can be employed to decrease the density of the point cloud, reducing computational complexity and memory requirements without significantly sacrificing critical feaeural networks are statistical models that draw inspiration from the intri-

cate structure of the human brain. These models possess the remarkable capability to adapt and learn, even when provided with a limited number of examples, enabling them to estimate population parameters [64]. In the domain of obstacle avoidance and path planning, neural networks have emerged as a promising solution for enabling autonomous systems to navigate complex environments with effectiveness and safety.

When employing deep neural networks for lidar-related tasks, it is widely recommended to feed the data directly into the network without undergoing any form of dimensionality reduction or feature projection [34]. This approach aims to prevent the loss of valuable information that may prove crucial for the task at hand. By inputting raw data, the network gains the autonomy to learn and extract pertinent features from an adequate number of examples,tures [69].

Down sampling techniques can include various methods such as voxelization or grid-based sampling, where the point cloud is divided into a regular grid and only one point per grid cell is retained. Another approach is random sampling, where a subset of points is randomly selected from the original point cloud.

- **One-Hot Encoding**One-hot encoding is a technique used to represent categorical variables as binary vectors. In the context of obstacle detection and avoidance, one-hot encoding can be applied to represent different classes or categories of obstacles[70].

  For example, if the LiDAR data needs to classify obstacles into three categories: pedestrian, vehicle, and cyclist, one-hot encoding can represent each category as a binary vector of length 3. The vector [1, 0, 0] would represent pedestrian, [0, 1, 0] would represent vehicle, and [0, 0, 1] would represent cyclist.

  One-hot encoding enables the FNN model to effectively learn and differentiate between different obstacle categories by representing them as distinct binary patterns.

These data processing techniques, including normalization, down sampling, and one-hot encoding, play a crucial role in preparing the 2D LiDAR data for input to the FNN model. They contribute to enhancing the efficiency and effectiveness of the learning process, enabling the FNN to make accurate predictions for obstacle detection and avoidance.

Once the data is pre-processed, it is fed into the FNN model for further processing. The FNN architecture consists of an input layer, one or more hidden layers, and an output layer. Each layer is composed of interconnected nodes or neurons, which perform computations on the input data.

During the forward pass of the FNN algorithm, the pre-processed LiDAR data propagates through the network from the input layer to the output layer. At each layer, the neurons apply a set of learned weights and biases to the input data, perform activation functions, and produce intermediate representations. These intermediate representations capture the complex relationships between the input LiDAR data and the desired output, which is obstacle detection and avoidance in this case.

The output layer of the FNN model produces the final predictions, indicating the presence or absence of obstacles and the corresponding actions to avoid them. The FNN is trained using labeled data, where the ground truth obstacle information is known. The training process involves adjusting the network's parameters (weights and biases) to minimize the difference between the predicted obstacle labels and the ground truth labels. This optimization is typically performed using techniques like stochastic gradient descent (SGD) or more advanced algorithms such as Adam.

By training on a diverse data-set that captures different obstacle scenarios, FNNs can learn to generalize and make accurate predictions in real-time obstacle avoidance scenarios using 2D LiDAR data.

In summary, processing 2D LiDAR data with FNNs involves Pre-processing the raw point cloud, feeding it into the FNN model for forward pass computations, and training the FNN using labeled data to make accurate predictions for obstacle detection and avoidance. The effectiveness of this approach has been demonstrated in various studies, highlighting the potential of FNNs in ensuring safe and efficient navigation of autonomous systems.

CHAPTER 3

Simulation Environment And virtual hardware setup

This section presents an in-depth overview of the simulation environment and the virtual hardware setup utilized in this project. The simulation phase was carried out leveraging the Robot Operating System (ROS) and Gazebo simulator, which provided a robust framework for testing and evaluating the obstacle avoidance capabilities. By emulating real-world conditions, we were able to subject the robot to various challenging scenarios and evaluate its performance in a reliable and repeatable manner. The following subsections delve into the software components and virtual environment construction in detail.

3.1 Software Components

The successful implementation of the simulation environment relied on the utilization of various software components, each serving a specific purpose in the development process. These components are seamlessly integrated within the ROS framework, enabling efficient communication and coordination. his section aims to provide a comprehensive overview of three essential aspects of robotics development: ROS communication mechanisms, the Gazebo simulator for robot simulation, and LiDAR (Light Detection and Ranging) configuration. These concepts are fundamental to understanding the intricacies of robotics engineering.

3.1.1 Robotic Operating System(ROS)

The Robotic Operating System (ROS) is an open-source framework that has garnered extensive adoption within the robotics domain. It serves as a comprehensive platform comprising a multitude of tools, libraries, and standardized conventions[71], all aimed at facilitating modular development and enabling the construction of sophisticated robot systems. The framework encompasses a wide array of functionalities, encompassing hardware abstraction, low-level device control, inter-process communication via message-passing, package management, visualization tools such as RViz2, and more.

The critical components of ROS implementation are nodes, messages, topics, and services[72]. Nodes are independent processes responsible for performing computations or tasks. Each node is designed to handle specific functions within a system. For example, one node may gather sensory data from a LiDAR sensor, process it, and transmit it to another node. The receiving node can then utilize the data for tasks like detection or navigation algorithms, and may subsequently send commands to a third node to provide feedback for robot control. Nodes communicate with each other through topics, which define the format of the messages transmitted. Messages are structured data formats including primitive types, constants, or arrays. There are two main modes of message communication between nodes in ROS: the topic-based asynchronous publisher/subscriber mode and the service-based synchronous server/client pattern. In the publisher/subscriber mode, publisher nodes directly send messages to topics, and any node can retrieve data from it simply by subscribing to those topics via subscribers. Topics enable one-way communication and are particularly useful for the continuous transmission of sensor data. An example of publisher/subscriber communication in ROS is depicted in Figure 10.

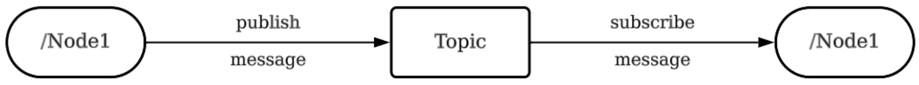


Figure 10. The publisher/subscriber ROS communication

On the other hand, the server/client pattern involves a node advertising a service and waiting for requests from service clients. After receiving a request, the node responds accordingly. Unlike topics, services facilitate one-time message communication. Once the request and response exchange is completed, the connected nodes disconnect [73]. Figure 11 illustrates an example of service-based communication in ROS.

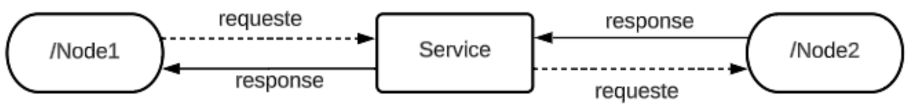


Figure 11. The server/client ROS communication

The decision to utilize the publisher/subscriber communication pattern instead of the service/client pattern for receiving data from the LiDAR sensor in the autonomous navigation map construction employing SLAM in ROS 2 was made based on careful consideration of the application's requirements.By employing publisher/subscriber, a continuous stream of LiDAR sensor data can be efficiently delivered to the SLAM algorithms in real-time, ensuring a constant and up-to-date representation of the environment. This continuous stream facilitates the construction of an accurate and timely map while concurrently estimating the robot's pose.
To visualize the results of the obstacle avoidance robot simulation, ROS provides a powerful and versatile visualization tool called rviz. Rviz is an acronym for ROS visualization, serving as a versatile 3D visualization platform designed for robots, sensors, and algorithms. Similar to other ROS tools, it possesses the capability to adapt to various robotic systems and swiftly tailor its configuration to suit specific applications [74].

One of the standout features of rviz is its ability to render realistic 3D models of the robot and the environment it operates in. This feature allows researchers and developers to import and visualize detailed models of the robot, including its sensors, chassis, and other components. It provides a visually immersive representation of the robot in the simulated environment, making it easier to understand its physical characteristics and interactions with the surroundings.

In addition to visualizing the robot model, Rviz has the capability to display different types of data that flow through a standard ROS system, with a particular focus on the three-dimensional aspects of the data. Within ROS, all types of data are associated with a specific frame of reference [74]. Users can display data from various sensors used in the obstacle avoidance system, such as LiDAR point clouds, camera images, or ultrasonic sensor readings. This visualization capability is invaluable for gaining insights into how the robot perceives its environment and assessing the accuracy and quality of sensor readings.

By utilizing the rich visualization capabilities of rviz, researchers and developers can gain valuable insights into the performance of their obstacle avoidance system. The combination of realistic robot models, visualized sensor data, interactive controls, overlays, and debugging information empowers users to thoroughly analyze and evaluate the robot's behavior, make informed adjustments to the algorithms, and enhance the overall navigation capabilities of the robot.

3.1.2 Gazebo simulator

Gazebo simulator functions as a sophisticated 3D open-source dynamics simulator with the capability to accurately simulate a multitude of robots, sensors, and objects in both indoor and outdoor environments. In a manner akin to game engines, Gazebo simulator offers a higher degree of fidelity in physical simulation, a comprehensive range of sensors, and interfaces that cater to user and program requirements. It also ensures the faithful representation of physical interactions between objects, including precise simulation of rigid body physics [75]. The Gazebo simulator is predominantly employed by academics and developers for indoor simulation. Its emphasis lies in creating a realistic world for robots, relying heavily on physics-based features [76]. The Gazebo simulator offers several significant advantages, such as a robust physics engine, open-source code, and user-friendly graphical interfaces. Users have the option to import pre-existing simulated robots or construct new models using geometrical primitives. By utilizing odometry and sensor data, the simulation model can replace the actual robots while performing robot movement calculations [77].

Figure 12 the general architecture of Gazebo simulator components. A Gazebo simulator world encompasses a collection of robots and objects within the simulated environment, including the robot model, the created environment, and global parameters such as ambient conditions, lighting, and physics properties. In the middle hierarchy, the model comprises a combination of joints, sensors, and bodies. Libraries interact with the Gazebo simulator at the lower level, ensuring that the model remains unaffected by changes in specific tools. Lastly, models employ the shared memory interface to receive commands from clients and return data [78].

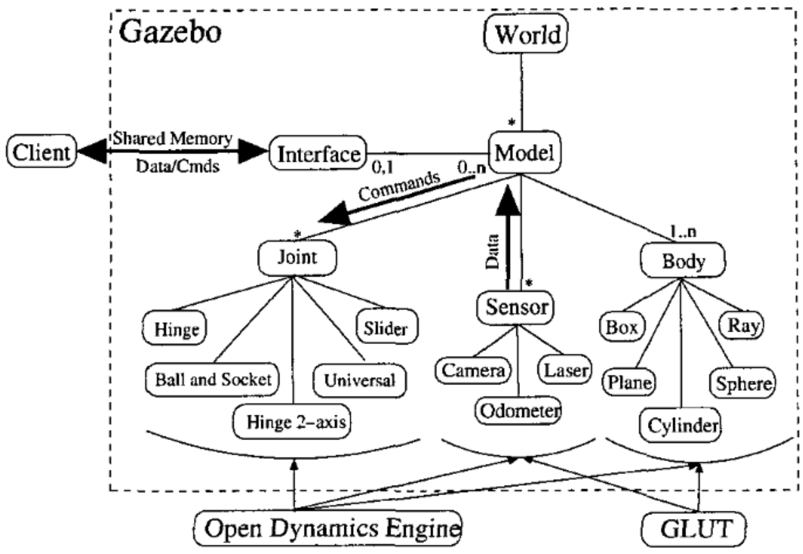


Figure 12. The structure of Gazebo simulator components [78]

When undertaking the design of a mobile robot, our objective is to construct a model that possesses the requisite capabilities, including accuracy and speed, while ensuring the desired levels of reliability and maneuverability to stabilize the mechanical structures. The analysis process heavily relies on the morphology of the robot and can be employed to determine the optimal arrangement of components such as actuators and sensors to fulfill the robot's intended purpose [76].URDF (Unified Robot Description Format) is a file format for specifying the geometry and organization of robots in ROS.

The URDF (Unified Robot Description Format) serves as a file format within the ROS framework, providing a means to define the geometry and structure of robots. The primary building blocks of URDF are the "joint" and "link" elements. The "link" element defines the physical structure of the robot, encompassing properties such as form, mass, friction, bounce factors, and visual characteristics like color, texture, and transparency. On the other hand, the "joint" element establishes connections between the links of the robot's body and imposes motion constraints. Figure 13 visually presents the URDF model of the developed robot, featuring multiple links and joints, with a LiDAR sensor integrated into the main body, and the illustration of the 3D model of the same URDF is presented in Figure 14 and Figure 15.

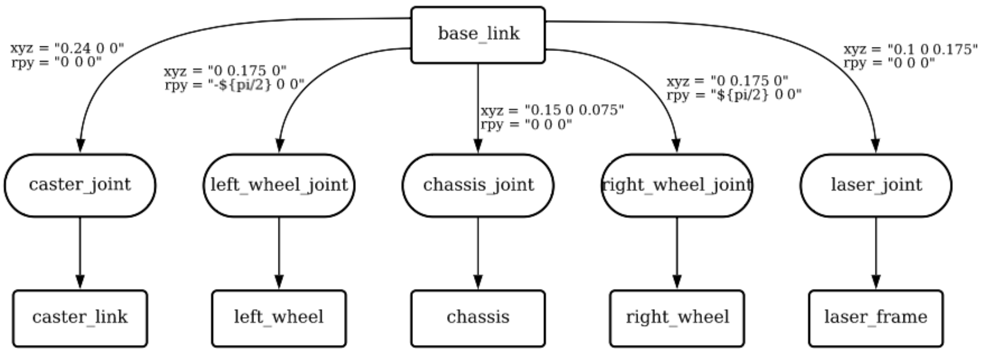


Figure 13. URDF model of the designed robot

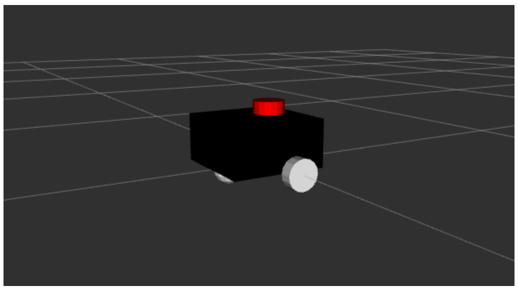


Figure 14. Robot Model Designed in Gazebo

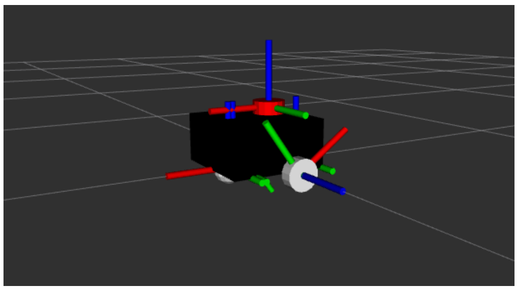


Figure 15. Illustration of Joints in the Robot Model

3.1.3 2D LiDAR sensor

In the preceding chapter, an extensive exploration was undertaken to elucidate the significance of LiDAR sensors in our obstacle avoidance system. LiDAR, an abbreviation for Light Detection and Ranging, encompasses a remote sensing technology that leverages rotating beams to emit pulsed light in a 360-degree direction. By calculating the distance between the LiDAR sensor and surrounding obstacles through analysis of the reflected light, a comprehensive distance map of the environment is generated. Two main categories of LiDAR sensors exist: 2D and 3D. Whereas 2D LiDAR lacks depth information and functions within a flattened two-dimensional space, 3D LiDAR sensors capture height information, yielding voluminous point cloud data, necessitating augmented computational resources and incurring higher costs [79]. For this project, our focus is on a 2D LiDAR sensor.

The selection of the RP-Lidar [80] sensor for our project was predicated on meticulous evaluation of several decisive factors. Recognized for its dependable performance, cost-effectiveness, and seamless integration with the ROS framework, the RP-Lidar emerged as the preferred choice. Its ability to furnish precise distance measurements, expansive field of view, and real-time obstacle detection capability rendered it ideal for our intended application. Moreover, the RP-Lidar strikes an optimal balance between affordability, computational efficiency, and obstacle detection accuracy, aligning aptly with our specific requirements.

Table 2 presents a comprehensive overview of the measurement performance of the RP-Lidar sensor. This table highlights key parameters that are crucial for our project's success.

Table 2. The measurement performance of RP-Lidar

Parameters	**Values**
Sampling Frequency	5000 Hz
Scanning Frequency	10Hz-20Hz
Range	0.1m - 12m
Angular Range	360 deg
Distance Resolution	0.1% of the range
Angular Resolution	1 deg

The Sampling Frequency refers to the rate at which the RP-Lidar sensor samples the environment, with a typical value of approximately 5,000 Hz, determines the rate at which the RP-Lidar sensor samples the environment. A higher sampling frequency ensures a more comprehensive representation of the surroundings, allowing for accurate mapping and timely detection of obstacles.The Scanning Frequency, typically around 20 Hz, represents the number of complete scans performed by the RP-Lidar sensor per second. This parameter directly influences the system's ability to perceive and respond to dynamic changes in the environment. A higher scanning frequency enables more frequent updates of the obstacle map, providing real-time information for safe navigation.The Angular Range of 360 degrees provides a complete field of view, allowing the RP-Lidar sensor to capture data from all directions. This wide coverage is vital for detecting obstacles in the entire environment and ensuring comprehensive situational awareness.The Range parameter, with a maximum measurement distance of approximately 12 meters, determines the sensor's capability to detect objects within a considerable range. This extended range allows for early detection of obstacles, providing the robot with sufficient time to plan appropriate avoidance maneuvers.The Distance Resolution, typically set at 1 centimeter, represents the minimum distance that the RP-Lidar sensor can differentiate. This parameter influences the system's ability to perceive fine details in the environment, enabling accurate detection and avoidance of small objects or obstacles located in close proximity.Lastly, the Angular Resolution, with a value of 1 degree, defines the smallest angular increment between two adjacent measurement points. This parameter ensures a high level of detail in the generated point cloud, enabling precise mapping of the environment and accurate localization of obstacles [80]. Within the Gazebo simulator simulator, this sensor is integrated into the robot and accessed through the Gazebo simulator laser scan plugin interface, which provides point cloud data via the **"libGazebo simulator ros ray sensor.so"** module.

3.1.4 3D Indoor Environment Setup

In the Gazebo simulator, an indoor environment with static obstacles was created to test the functionality of the obstacle avoidance system. The environment was carefully designed, taking into consideration the distribution, shapes, and sizes of the initial obstacles.

One of the advantages of using the Gazebo simulator is the flexibility to add obstacles at any time during the path planning process. This dynamic feature allowed for on-the-fly obstacle insertion, simulating real-world scenarios where obstacles may appear unexpectedly. The ability to modify the environment during the planning process provided a realistic and challenging testing environment for the obstacle avoidance system.

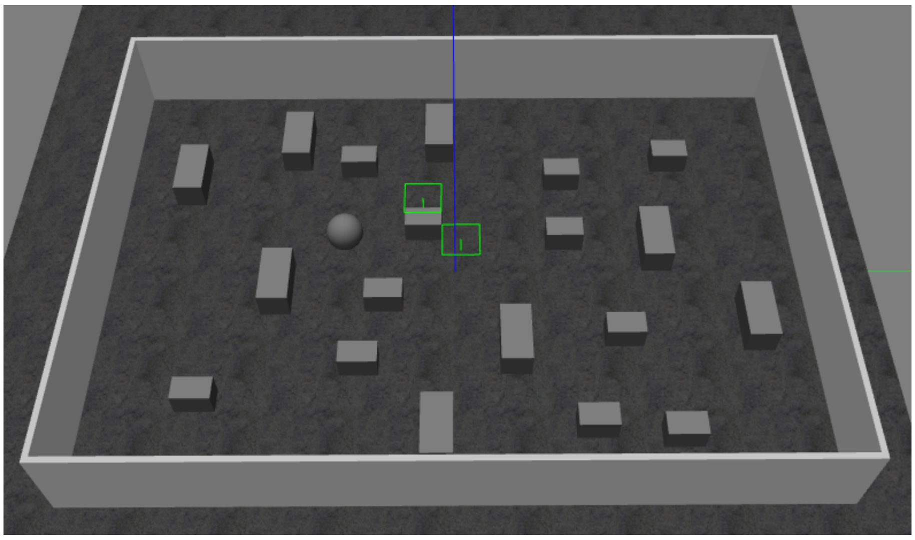


Figure 16. Simulated indoor environment with static obstacles

Figure 16 showcases the initial environment used for testing the obstacle avoidance system. With the simulator's capability to add obstacles dynamically, new obstacles could be introduced at any point in the path planning process. This feature allowed for a comprehensive evaluation of the system's adaptability and responsiveness to changing environments, enhancing the robustness and reliability of the obstacle avoidance algorithm.

The ability to add obstacles during the path planning process simulated real-world scenarios more accurately, where the environment is subject to changes and the robot must continuously adapt to its surroundings. This capability in the Gazebo simulator facilitated thorough testing and fine-tuning of the obstacle avoidance system, ensuring its effectiveness in dynamic environments.

3.2 SLAM Algorithm Selection and Implementation

In the process of selecting a suitable SLAM algorithm for our implementation, various factors such as computational efficiency, accuracy, and ease of integration were carefully considered. After comprehensive evaluation and analysis, we have determined that Karto-SLAM is the optimal choice for our project, primarily due to its compatibility with the SLAM Toolbox within the Robot Operating System (ROS) framework.

One of the primary reasons for choosing Karto slam is its compatibility with ROS. ROS provides a comprehensive framework for robot development, offering a wide range of tools, libraries, and packages that facilitate the implementation process. Karto slam seamlessly integrates with ROS's sensor and actuator interfaces, enabling effortless integration of lidar sensors and robot control. This compatibility simplifies the overall system architecture and ensures smooth data exchange between different components.

Another key advantage of selecting KartoSLAM is its efficient performance, particularly in scenarios where computational resources are limited. A comparative study conducted by[81] evaluated the performance of Gmapping, Cartographer, and KartoSLAM algorithms. The results demonstrated that while KartoSLAM may generate slightly less accurate maps compared to Cartographer, it significantly outperforms in terms of computational cost. This aspect played a crucial role in our decision-making process as we sought an algorithm that could run efficiently on our hardware with limited computing power.

Moreover, the modular architecture of ROS opens up possibilities for advanced functionalities in conjunction with Karto slam. ROS's navigation stack, for instance, offers components like global and local planners, enabling autonomous navigation based on the generated maps. By utilizing these capabilities, the mobile robot can navigate efficiently, avoiding obstacles and reaching target locations effectively.

In summary, the selection of KartoSLAM within the SLAM Toolbox of ROS was motivated by its computational efficiency, compatibility with our limited computing resources, and the wealth of resources available within the ROS community. These factors ensure a cost-effective, well-supported, and reliable SLAM solution for our project.

3.3 Path Planning and Obstacle Avoidance Strategy

The Rapidly-exploring Random Tree algorithm is widely used in robotics for its ability to efficiently explore state spaces and generate feasible paths. By incrementally adding nodes and connecting them, RRT explores the space in a randomized manner, avoiding exhaustive exploration and reducing computational complexity. It is particularly useful in high-dimensional and complex environments, where finding an optimal path is challenging.

The RRT algorithm incrementally builds a tree by randomly sampling nodes and connecting them to the existing tree structure. The resulting tree represents a feasible path from the initial pose to the goal position[31].

The implementation of the RRT algorithm in ROS for autonomous navigation involved the following steps:

Firstly, a dedicated ROS package was created to encapsulate the RRT algorithm implementation and related nodes, ensuring a structured and modular design. Custom ROS messages were defined to establish a standardized communication interface between the

Algorithm 1: RRT Algorithm

```
Graph $T = (V, E)$, initial pose $q_{\text{start}}$ Tree $T = (V, E)$
Function RRT ($T$, $q_{start}$):
    Initialize tree: $V \leftarrow \{q_{\text{start}}\}$, $E \leftarrow \emptyset$;
    for each iteration do
        $q_{\text{rand}} \leftarrow$ SampleRandomNode();
        $q_{\text{near}} \leftarrow$ Nearest($T = (V, E)$, $q_{\text{rand}}$);
        $q_{\text{new}} \leftarrow$ Steer($q_{\text{near}}$, $q_{\text{rand}}$);
        if NoObstacle($q_{near}$, $q_{new}$) then
            $V \leftarrow V \cup \{q_{\text{new}}\}$;
            $E \leftarrow E \cup \{(q_{\text{near}}, q_{\text{new}})\}$;
    return $T = (V, E)$;
```

nodes, facilitating seamless data exchange within the ROS ecosystem.

A ROS node was developed to subscribe to the map topic, which received the map data generated by SLAM algorithm. This same node encompassed the implementation of the RRT algorithm, utilizing the received map data. The RRT algorithm, executed within this node, incrementally constructed the RRT tree while integrating effective obstacle avoidance strategies, resulting in the generation of feasible paths. Subsequently, this node published the generated path on the path topic.

The planned paths generated by the RRT algorithm were seamlessly integrated with the robot's control system. To accomplish this, a separate ROS node was created or modified to establish an efficient interface with the robot's control hardware. By incorporating the RRT-generated paths, the robot achieved accurate navigation within the environment while effectively avoiding obstacles.

Consequently, another ROS node was developed to subscribe to both the pose and path topics. This node received the robot's current pose or position information from the pose topic and acquired the generated path from the path topic, which was published by the RRT algorithm node. Leveraging this information, the node performed calculations to determine the linear and angular velocities required for the robot to navigate towards the goal. These computed velocities served as control commands to govern the robot's motion.

To simplify the initialization process and facilitate testing, a launch file was developed. This launch file orchestrated the startup of all the necessary nodes for the RRT algorithm, including the map subscriber/publisher, path subscriber, and robot control nodes. By executing this launch file, the behavior of the robot in a simulated or real-world environment could be observed and analyzed.

By following these systematic steps, the RRT algorithm was successfully implemented in ROS, empowering autonomous robots with efficient path planning and obstacle avoidance capabilities in complex environments.

3.4 Deep Learning Approach

In this section, we present a detailed methodology for the implementation of Feedforward Neural Networks (FNNs) in the detection and avoidance of obstacles using a data-set of 2D LiDAR scans. Our proposed methodology aims to leverage the inherent capabilities of FNNs to address the critical task of obstacle detection and avoidance specifically within indoor environments.

3.4.1 Data Collection

To ensure the acquisition of high-quality data for our obstacle avoidance and path planning system, we conducted a meticulous data collection process. This process involved a manual approach that prioritized the selection of optimal actions at each step.

The inputs to our model comprise the x-coordinate, and y-coordinate of the displacement vector between the position of the robot and its goal, along with the robot orientation around the z axis obtained from pose estimation data. Additionally, we incorporate comprehensive lidar scan data. The lidar scan captures 360 distance measurements in a 180-degree range, resulting in a 363-element input vector. This rich sensory information enables our system to perceive the surrounding environment, including obstacles and their distances from the robot. By leveraging this detailed input data, our system effectively maps the environment, plans optimal paths, and successfully avoids obstacles during real-time navigation.

In our approach, we devised an action space consisting of 11 discrete actions, each denoted by a specific index m ranging from 0 to 10. These indexes correspond to the output generated by our model, allowing for a seamless mapping between the model's output and the corresponding angular velocities. All actions in this space share a consistent linear velocity (v), while their angular velocities (w_m) vary based on the equation:

$$w(m) = -0.8 + 0.16 \cdot m \tag{3.1}$$

.

This formulation facilitates a comprehensive exploration of angular movement while maintaining a constant linear velocity across the action set.

In order to ensure the collection of a high-quality and well-organized dataset, our data collection process involved the coordination of three essential nodes, which communicated with each other using the publisher/subscriber communication paradigm, as previously described. This approach facilitated efficient and reliable data transfer among the interconnected nodes, ensuring seamless coordination and data synchronization throughout the data collection process.

1. **Motion Control Node**
   The motion control node serves as the primary component for controlling the robot's motion. It issues velocity commands and signals the selection of an action through the publication of a boolean variable triggering the other nodes. This node ensures precise control over the robot's movement, accurately executing the chosen actions during the data collection process.
2. **Lidar Data Capture Node**
   The lidar data capture node plays a crucial role in perceiving the robot's surroundings. It subscribes to two topics: the "scan" topic, which provides lidar scan data, and the "action selection" topic, where the motion control node publishes the selected actions. By subscribing to the "action selection" topic, the lidar node acquires the relevant lidar data for each selected action.
3. **Pose Estimation Node**
   The pose estimation node collects data for determining the robot's position and orientation. It subscribes to the "odom" topic, which contains the robot's pose calculated using wheel odometry, as well as the "action selection" topic. This allows the node to retrieve odometry data whenever an action is selected. Upon initialization, the pose estimation node requests information about the goal coordinates, computing the distance between the robot and the goal, then publishes the resulting x-coordinate, y-coordinate along side the robot's orientation around the z-axis.

The manual approach, coupled with the coordinated functioning of the nodes, enables us to effectively capture the necessary information and lay a solid foundation for subsequent stages of system development.

In Figure 17, we present the communication flow among the three nodes engaged in the data-set collection process. The diagram showcases the publisher/subscriber ROS communication mechanism employed to establish effective coordination and data exchange. This communication framework ensures the synchronization of data acquisition and plays a crucial role in facilitating the collection of essential information required for the subsequent stages of our obstacle avoidance and path-planning system.

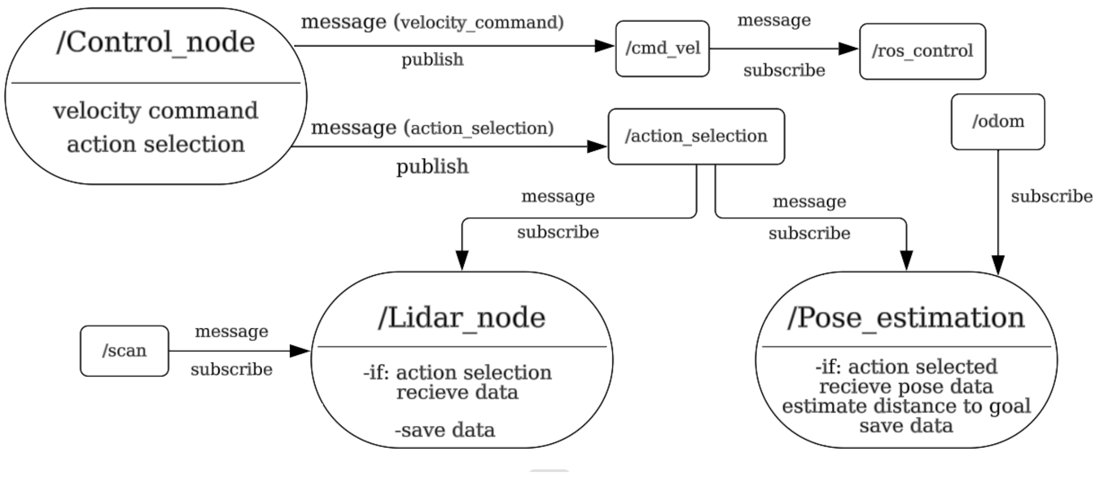


Figure 17. the communication flow among the three nodes.

The experimental environment utilized in this study was a custom-built simulation environment specifically designed within the Gazebo simulator as shown in Figure 16.

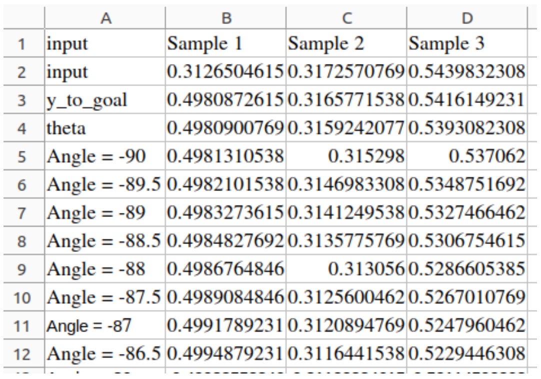
3.4.2 Data Pre-processing

In the data pre-processing phase, a range of techniques were applied to ensure the optimal preparation of input and output data for our obstacle avoidance and path planning system. The following methods were employed:

1. **Normalization of Lidar Data**: The lidar scan data was subjected to normalization using the max-min scaling approach. This technique standardized the data range, enabling more effective training and modeling processes. It enhanced the convergence of lidar data by bringing it within a consistent and manageable scale.
2. **Normalization of Coordinates**: The x and y coordinates underwent max-min scaling as well. This normalization process harmonized the scales of the coordinates, promoting better convergence during training and enabling the model to generalize effectively across various scenarios.
3. **Normalization of Robot Angle**: To account for the robot's orientation, the robot angle was also normalized using max-min scaling. By normalizing the angle values, we achieved a uniform distribution, effectively capturing the complete range of

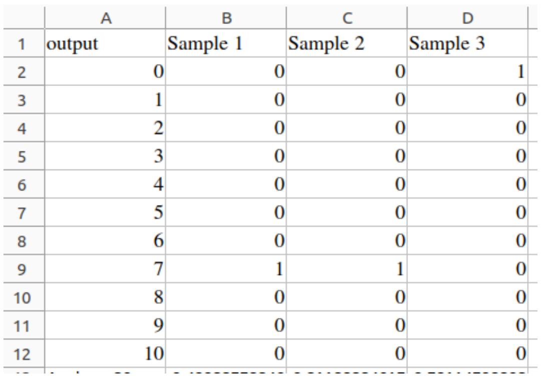
possible orientations.

4. **Combining Data in a Single Data-Frame**: The normalized lidar data, coordinates, and robot angle were combined into a single, cohesive Data-Frame. This integration of data facilitated streamlined data handling and simplified subsequent stages of the pipeline, ensuring efficient processing and utilization of the combined information.
5. **One-Hot Encoding for Output**: Since the desired output involved an action index, which functions as a categorical variable, OHE was employed. This encoding technique transformed the output labels into a binary vector format. By doing so, we avoided any potential disturbances in the output and ensured that the model returned an integer value representing the intended action.



	A	B	C	D
1	input	Sample 1	Sample 2	Sample 3
2	input	0.3126504615	0.3172570769	0.5439832308
3	y_to_goal	0.4980872615	0.3165771538	0.5416149231
4	theta	0.4980900769	0.3159242077	0.5393082308
5	Angle = -90	0.4981310538	0.315298	0.537062
6	Angle = -89.5	0.4982101538	0.3146983308	0.5348751692
7	Angle = -89	0.4983273615	0.3141249538	0.5327466462
8	Angle = -88.5	0.4984827692	0.3135775769	0.5306754615
9	Angle = -88	0.4986764846	0.313056	0.5286605385
10	Angle = -87.5	0.4989084846	0.3125600462	0.5267010769
11	Angle = -87	0.4991789231	0.3120894769	0.5247960462
12	Angle = -86.5	0.4994879231	0.3116441538	0.5229446308

(a) Input dataset



	A	B	C	D
1	output	Sample 1	Sample 2	Sample 3
2	0	0	0	1
3	1	0	0	0
4	2	0	0	0
5	3	0	0	0
6	4	0	0	0
7	5	0	0	0
8	6	0	0	0
9	7	1	1	0
10	8	0	0	0
11	9	0	0	0
12	10	0	0	0

(b) Output/Targets dataset

Figure 18. Example of a sample of the collected data.

By diligently applying these data pre-processing techniques, we have attained optimal scaling, integration, and encoding of the input and output data. Figure 18 shows an example of the processed data, demonstrating the input features (LIDAR scans and pose information) and the corresponding output action.

3.4.3 Model Architecture

The model architecture encompasses a feed-forward neural network (FFNN) implemented using the Keras framework, adhering to best practices in deep learning. The FFNN comprises an input layer, two hidden layers, and an output layer.

The input layer is a dense layer with 64 neurons, matching the dimensionality of the input data, specifically (363). It serves as the entry point for the neural network, receiving the input data and propagating it forward through the subsequent layers. Two hidden layers follow the input layer, each consisting of 128 neurons. The inclusion of two hidden layers enhances the model's ability to learn intricate relationships within the data. To introduce non-linearity and facilitate comprehensive feature extraction, the rectified linear unit (ReLU) activation function is employed in both hidden layers. ReLU is defined mathematically as :

$$f(x) = max(0, x) \tag{3.2}$$

where x represents the input to the activation function. ReLU maps all negative input values to zero while leaving positive input values unchanged. This activation function introduces non-linearity into the model and helps address the vanishing gradient problem commonly encountered in deep neural networks.

n the domain of multi-class classification, the output layer of the neural network comprises a dense layer consisting of 12 neurons, which corresponds to the number of classes involved in the classification problem. The softmax activation function is employed in this layer to acquire class probability distributions. By design, the softmax function ensures that the output values remain non-negative and sum up to unity, thereby enabling the model to furnish probabilistic predictions for multi-class classification tasks.

Mathematically, the softmax activation of an input vector x, possessing a dimensionality of n, is defined as follows:

$$\text{softmax}(x[i]) = \frac{e^{x[i]}}{\sum_{j=1}^{n} e^{x[j]}}, \quad \text{for } i = 1 \text{ to } n, \tag{3.3}$$

Here, $e^{x[i]}$ represents the exponential of the i-th element of the input vector, and $\sum()$ denotes the summation of the exponential values across all elements within the vector. The softmax function transforms the input values into a probability distribution over the classes, wherein each output value represents the probability of the corresponding class being the correct class. This attribute empowers the model to make confident predictions by assigning higher probabilities to the most probable classes.

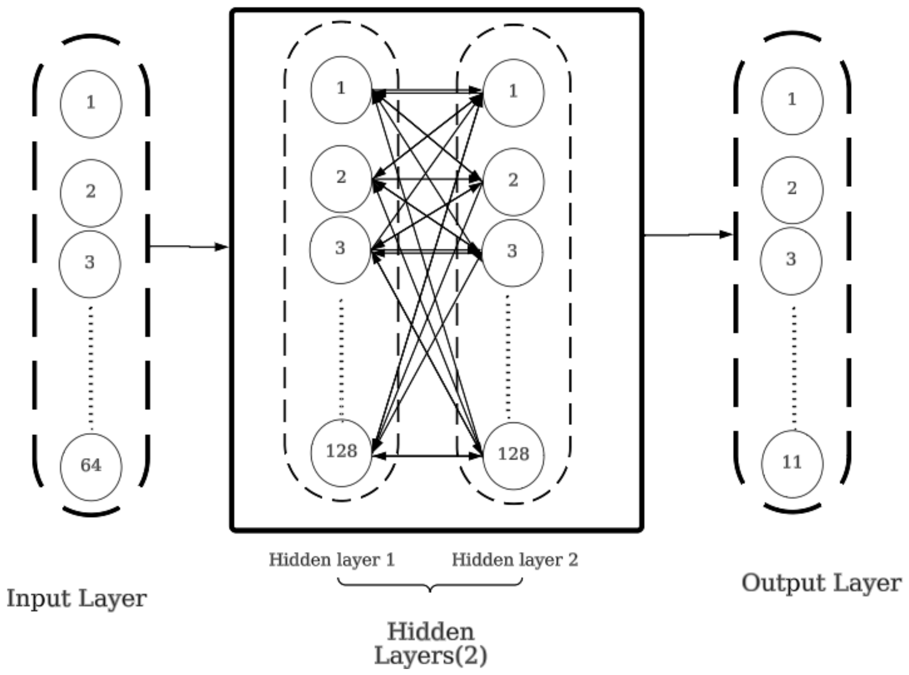


Figure 19. Model architecture.

Figure 19 presents a schematic representation of the model's architecture, illustrating its sequential configuration. This diagram elucidates the sequential flow of information, commencing from the input layer, traversing through the hidden layers, and culminating at the output layer.

3.4.4 Training Process

The model training process was conducted with utmost precision and meticulousness to optimize performance and foster effective learning. The dataset was diligently partitioned into two distinct subsets: a training subset comprising approximately 75% of the data and an independent testing subset containing the remaining 25%. This careful division facilitated a comprehensive assessment of the model's generalization capabilities, providing valuable insights into its efficacy and dependability when confronted with previously unseen data. Essential hyper-parameters, such as the learning rate and batch size, were carefully determined to balance convergence speed and resource efficiency.

During the training phase, the Adam optimizer algorithm was employed. Adam combines the advantages of adaptive learning rates from AdaGrad and efficient gradient-based optimization from RMSProp. By leveraging adaptive learning rates and efficient parameter updates, the Adam optimizer promoted the model's optimization performance [82]. This choice was particularly advantageous for our model, as it required capturing intricate relationships and exhibiting robust generalization capabilities.

The choice of loss function is a critical aspect of the model training process. For this study, the categorical cross-entropy loss function was employed, which is widely used in multi-class classification tasks. This loss function quantifies the disparity between the predicted class probabilities and the true class labels. Mathematically, the categorical cross-entropy loss can be expressed as:

$$\text{Categorical Cross-Entropy Loss} = -\sum_{i=1}^{N}\sum_{j=1}^{C} y_{ij}\log(p_{ij}) \tag{3.4}$$

Here, N represents the number of samples, C is the number of classes, y_{ij} denotes the true probability of class i for sample j, and p_{ij} is the predicted probability of class i for sample j. The loss is calculated as the sum of the logarithmic differences between the predicted and true class probabilities. By utilizing the categorical cross-entropy loss function, the model training process aims to minimize the discrepancy between predicted probabilities, which in our approach are described by the OHE representation of the 11 integer outputs.

By methodically incorporating rigorous deliberations into the segregation of data, the selection of crucial hyper-parameters, and the utilization of the widely acclaimed Adam optimizer algorithm, the process of training the model was meticulously devised with the aim of augmenting learning, optimizing performance, and fostering dependable predictions.

CHAPTER 4

Results And Discussion

This chapter presents the outcomes of map construction using the Simultaneous Localization and Mapping (SLAM) technique and obstacle detection using the Rapidly-Exploring Random Trees (RRT) algorithm in the simulated environment of the Gazebo simulator within ROS2. The Karto SLAM algorithm was employed to generate a precise and comprehensive map of the environment, allowing for an accurate representation of the unknown indoor space. Additionally, the RRT algorithm was utilized to effectively detect and avoid obstacles during autonomous navigation.

The results obtained from the map construction process using SLAM will be showcased and extensively discussed, highlighting the algorithm's performance and efficiency in gradually updating and accurately representing the environment. Furthermore, the obstacle detection and avoidance capabilities enabled by the RRT algorithm will be presented, demonstrating its effectiveness in ensuring safe navigation.

In addition, this chapter includes a comparison of pose computation results, where the advantages of SLAM-based pose estimation over odometry-based estimation are discussed. The findings emphasize the suitability of SLAM for accurate localization and mapping in real-world scenarios.

Moreover, the chapter presents the outcomes of training a deep learning model for obstacle avoidance using a collected data-set. The limitations in terms of accuracy attributed to odometry pose computation errors and challenges in data-set collection due to computational constraints are discussed. Future work is proposed to address these limitations by incorporating SLAM pose data, which is expected to enhance the obstacle avoidance model's performance and the agent's ability to navigate complex environments.

The comprehensive presentation and discussion of the obtained results in map construction, obstacle detection, pose computation comparison, and deep learning model training provide valuable insights into the performance, limitations, and potential improvements of the implemented algorithms. These findings contribute to the development of autonomous robotic systems and pave the way for future enhancements and real-world applications.

4.1 SLAM-Based Map Construction Results

The Karto SLAM algorithm was employed to construct a map of the unknown indoor environment in Gazebo. This algorithm effectively generated a detailed and accurate map by combining sensor data from the mobile robot with odometry information. The resulting map provided a comprehensive representation of the environment, enabling further analysis and navigation planning.

Mapping Initialization:

The robot is initially placed at the coordinate (0.0) within the unknown indoor environment, and it starts generating the map within a range of 360 degrees and 12 meters.

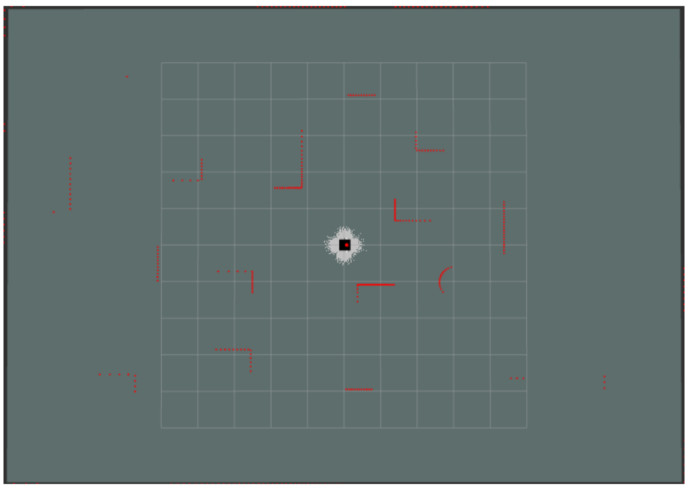
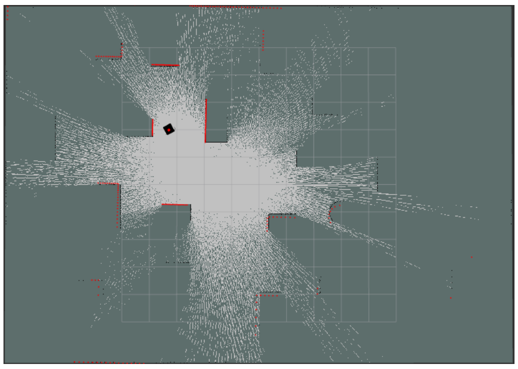
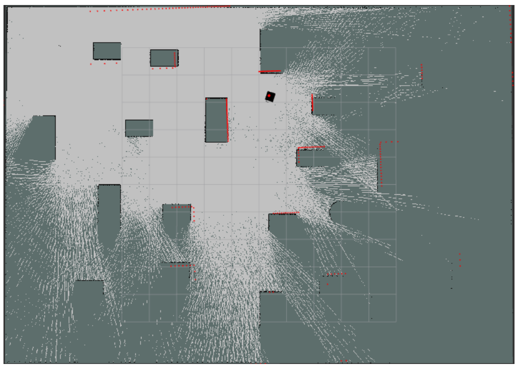


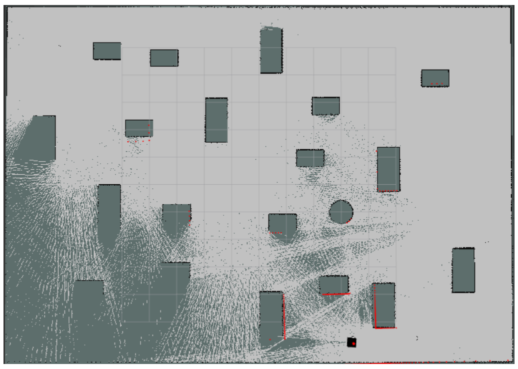
Figure 20. Mapping initialization.

Map construction and final mapping result:
As the robot moves in the environment, it matches the previous LIDAR scans with the new ones and updates the map. This process continues until the final mapping of the entire environment is achieved, differentiating between obstacles and free spaces.

The SLAM-based map construction process described has successfully generated a highly accurate and detailed map of the unknown indoor environment. The algorithm begins with mapping initialization, gradually updating the map by matching previous and new LIDAR scans. The progressive stages of map construction demonstrate the algorithm's effectiveness, culminating in a comprehensive representation of the entire environment. The SLAM approach proves reliable for mapping unknown indoor spaces with high accuracy.


(a) Intermediate map construction. The map is gradually updated as the robot explores the environment.


(b) Intermediate map construction. The robot continues to map the environment, distinguishing between obstacles and free space.


(c) Intermediate map construction. The mapping process provides a detailed representation of the environment.

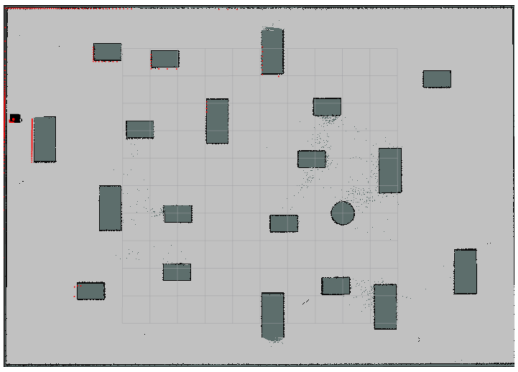

(d) Final map construction. The robot completes the mapping process, creating a comprehensive map of the environment.

Figure 21. Progressive stages of map construction. The figures demonstrate the gradual updates in the map as the robot explores the environment.

Figure 21 illustrates the various stages of map construction, showing the progressive updates in the map as the robot explores the environment. Figure 21a depicts an intermediate stage, capturing the evolving map with increasing detail. Finally, Figure 21d represents the final map, which provides a comprehensive representation of the entire environment.

4.2 Comparison of Pose Estimation Results

The comparison of pose estimation results provides insights into the accuracy and reliability of different methods in determining the robot's pose. In this evaluation, we compare the pose obtained from odometry, SLAM, and the ground truth (real pose) resulting from a complete path generation and obstacle avoidance command .

4.2.1 Odometry-Based Pose Estimation

The odometry-based pose estimation relies on wheel encoders and motion models to estimate the robot's position and orientation. However, due to inherent errors in the odometry measurements and accumulated uncertainties over time, the estimated pose may deviate from the actual robot's pose. Figure 22 shows the comparison between the odometry-based pose and the real pose. It can be observed that the odometry-based pose exhibits noticeable deviation (3.25 meters) from the ground truth, resulting in less accurate localization.

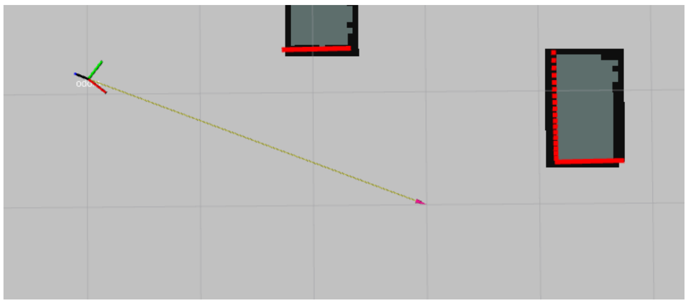


Figure 22. Comparison of odometry-based pose estimation with the real pose.

4.2.2 SLAM-Based Pose Estimation

The SLAM-based pose estimation leverages simultaneous mapping and localization techniques to estimate the robot's pose while constructing a map of the environment. By integrating sensor measurements such as laser scans and odometry, SLAM algorithms provide more accurate and robust pose estimates. Figure 23 illustrates the comparison

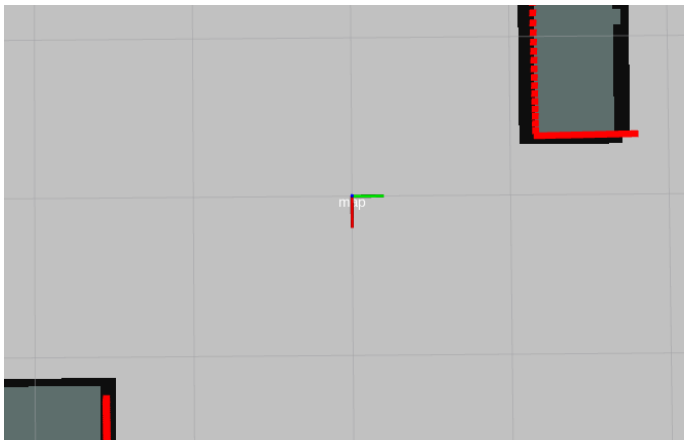


Figure 23. Comparison of SLAM-based pose estimation with the real pose.

between the SLAM-based pose and the real pose. It can be observed that the SLAM-based pose closely aligns with the ground truth, indicating a higher level of accuracy compared to odometry-based estimation.

4.2.3 Comparison and Discussion

Comparing the pose estimation results, it becomes evident that the SLAM-based pose estimation outperforms odometry-based estimation in terms of accuracy and closeness to the real pose. The SLAM algorithm's ability to incorporate sensor measurements and update the map in real-time enables it to compensate for odometry errors and environmental changes, resulting in more reliable pose estimates.

The superior accuracy of SLAM-based pose estimation has significant implications for various robotic applications, such as autonomous navigation, mapping, and object manipulation. By providing a more precise estimation of the robot's pose, SLAM enables robots to navigate complex environments with greater efficiency and reliability.

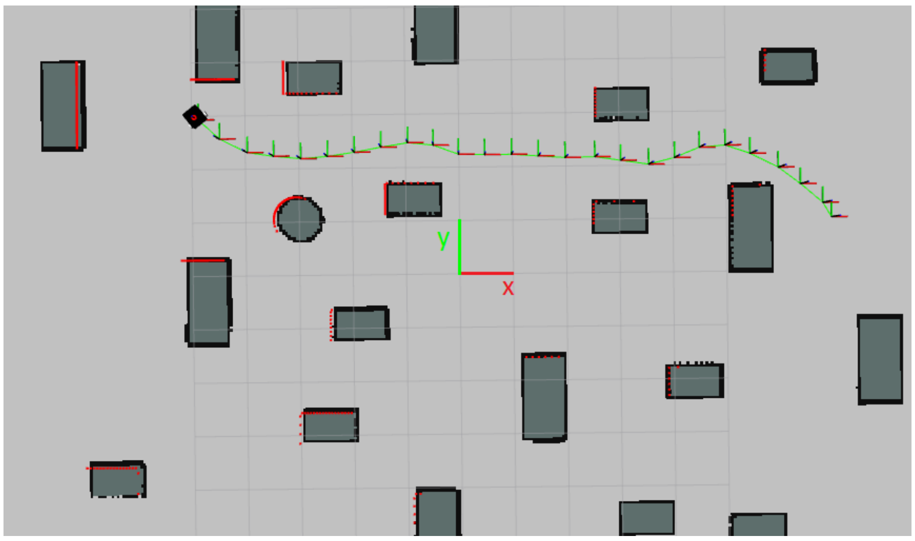
Overall, the comparison results highlight the advantages of SLAM-based pose estimation over odometry-based estimation, demonstrating its suitability for tasks that require accurate localization and mapping in real-world scenarios.

4.3 RRT-Based Obstacle Avoidance Results

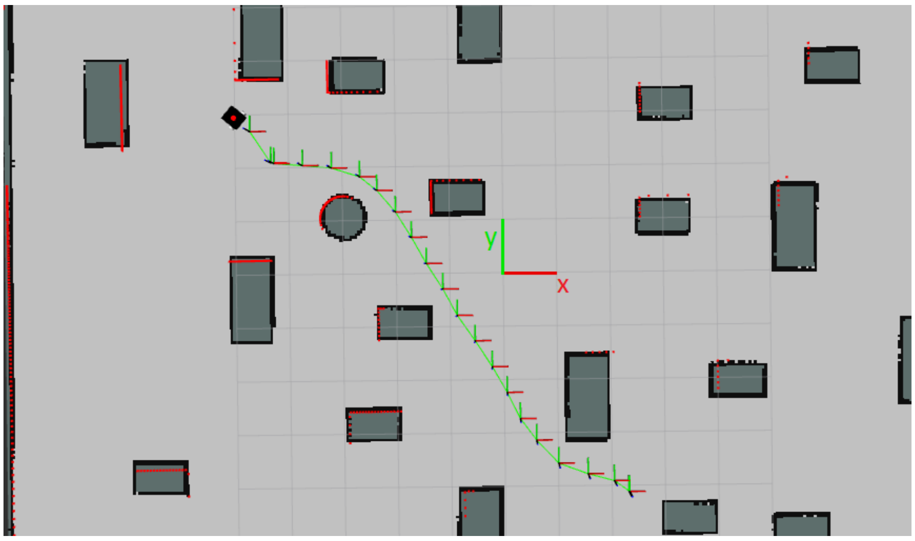
The RRT-based obstacle avoidance approach demonstrates effective navigation in the simulated indoor environment. The algorithm successfully generates feasible paths that avoid obstacles, allowing the robot to reach its goal while maintaining a safe distance from the detected obstacles.

First scenario: The robot's goal coordinates were set to (-5.0 ; 4.0), and the robot was initially posed at different places: (7.0 ; 1.0) in Figure 24a , and (2.3 ; -4.2) in Figure 24b.

Figure 24 illustrates the results of the RRT-based obstacle avoidance approach in the first scenario:



(a) Robot initially posed at (7.0, 1.0).

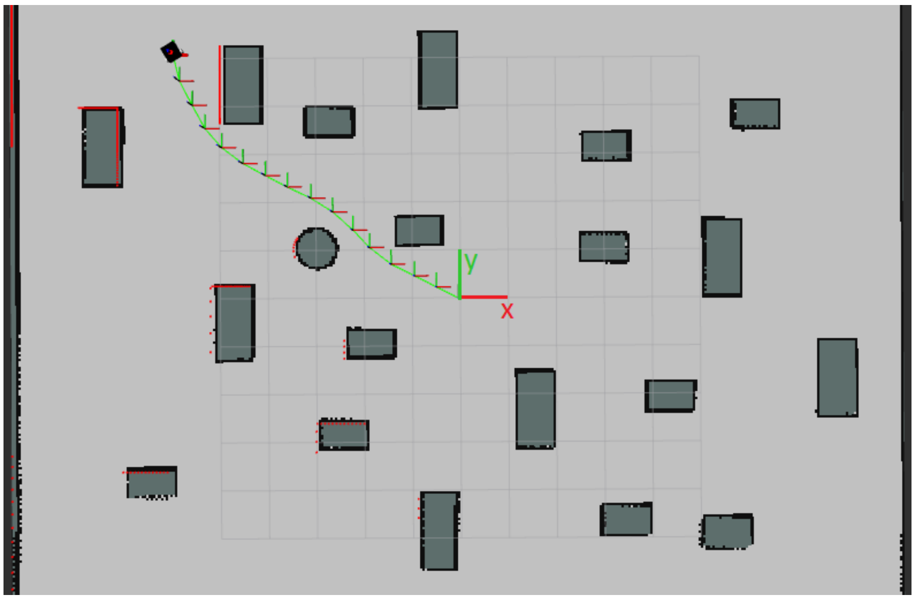


(b) Robot initially posed at (2.3 ; -4.2).

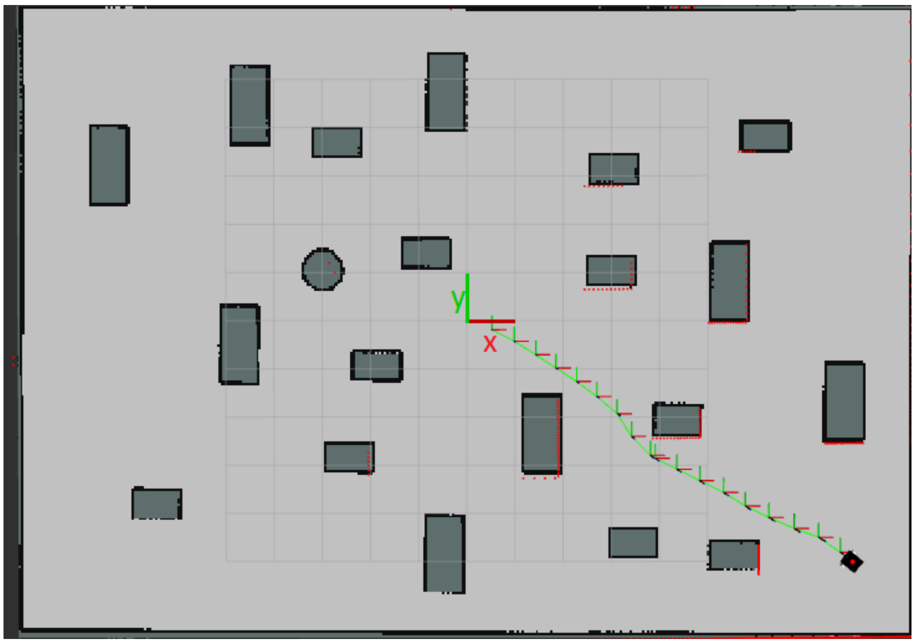
Figure 24. Different initial robot poses with the same goal coordinates.

Second scenario:

The initial coordinates were set to (0.0 ; 0.0), and the robot's goal coordinates were set to different places: (-6.0 ; 5.0) in Figure 25a and (8.0 ; -5.0) in Figure 25b.
Figure 25 illustrates the results of the RRT-based obstacle avoidance approach in the second scenario:



(a) Goal set at (-6.0 ; 5.0).



(b) Goal set at (8.0 ; -5.0).

Figure 25. Different goal coordinates with the same initial robot pose.

These results cover all possible scenarios for the mobile robot's autonomous navigation. In all scenarios, the robot successfully reached its goal while avoiding obstacles, demonstrating the effectiveness of the RRT-based obstacle avoidance approach. The RRT algorithm proves to be a highly accurate path planning algorithm, providing reliable paths that enable the robot to navigate autonomously.

The RRT algorithm efficiently explores the state space and generates feasible paths by dynamically adapting to the obstacle configuration. The algorithm maintains a safe distance from obstacles by considering the robot's kinematic constraints and the environment's obstacles throughout the navigation process. This capability enhances the robot's autonomy and enables it to navigate through complex environments effectively.

4.4 Deep Learning Approach Results

This section presents the outcomes of the deep learning approach for training the obstacle avoidance model. The model underwent 50 epochs of training using a small data-set comprising 700 samples. The evaluation of the model's performance was based on essential metrics, including loss and accuracy.

To illustrate the effectiveness of the trained obstacle avoidance model, Figure 26 provides a graphical representation of the performance metrics. The graph presents the training and validation accuracy as well as the corresponding loss over the course of the 50 training epochs.

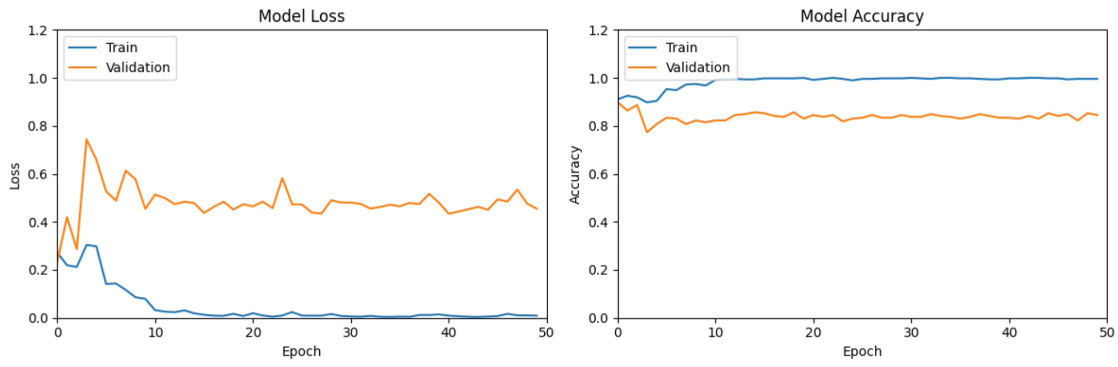


Figure 26. Performance metrics of the trained obstacle avoidance model.

Initially, both the training and validation accuracy showed a positive trend, indicating the model's ability to learn from the training data. However, after a certain point, a noticeable gap emerged between the training and validation accuracy, suggesting the onset of overfitting.

In terms of the loss metric, we observed variations throughout the training process, reflecting the model's dynamic adjustments. The initial loss was recorded at 0.2145, with an accuracy of 0.9435 in the training set. In the validation set, the corresponding values were

0.2833 for loss and 0.8914 for accuracy.

As the training progressed, the loss and accuracy values exhibited fluctuations. While the loss metric showed variations, indicating shifts in the model's performance, the accuracy metric consistently improved, reaching a peak value of 1.0000 at the conclusion of the training.

Overall, the deep learning approach exhibits promising results in training the obstacle avoidance model, showcasing the potential for further optimization in future iterations.

4.4.1 Results Analysis

The training of our deep learning model on the collected data (700 samples) after yielded results: as can be seen in figure 26 the model shows a decreasing trend in both training and validation loss, which is desirable. It indicates that the model is learning and improving over time. Similarly, the training and validation accuracy shows an increasing trend, which suggests that the model is able to classify the data correctly. However, the accuracy achieved was not as high as desired. Additionally, there is a slight gap between the training and validation accuracy. This indicates the possibility of over-fitting, where the model may have memorized the training data too well and is not generalizing well to unseen data. This can be attributed to two main factors.

First, the uncertainty in the pose estimation based on odometry as seen in figure 22. Inaccurate pose information can adversely affect the model's ability to make precise decisions based on the agent's position and orientation.

Another factor contributing to the limited accuracy is the computational power limitations that constrained the collection of a large amount of data. Gathering a comprehensive data-set for training requires significant time and computational resources. Given the constraints, it was challenging to collect a sufficiently diverse and representative data-set, which could have contributed to the sub-optimal performance of the trained model.

4.4.2 Conclusion

In conclusion, the deep learning approach employed for training the obstacle avoidance model has yielded promising results. The accuracy of the model consistently improved throughout the training process, culminating in a peak value at the conclusion of the training. However, it is worth noting that a discernible disparity emerged between the training and validation accuracies, suggesting the presence of potential overfitting concerns.

Several factors contributed to the observed outcomes. The inherent uncertainty associated with pose estimation based on odometry exerted an influence on the model's decision-making capabilities. Moreover, the limited availability of computational resources posed constraints on the acquisition of a more extensive and diverse dataset.

4.5 Discussion of Results

This chapter presents the outcomes of map construction using SLAM and obstacle detection using the RRT algorithm in the simulated environment of the Gazebo simulator within ROS2. The Karto SLAM algorithm was employed to generate a precise and comprehensive map of the environment. Additionally, the RRT algorithm was utilized for effective obstacle detection and avoidance during autonomous navigation.

The results obtained provide compelling evidence of the performance and efficiency of the implemented algorithms in the simulated environment. The SLAM-based map construction process exhibited remarkable accuracy and detail in generating maps of unknown indoor environments. The algorithm's ability to incrementally update the map as the robot explored the environment enabled it to differentiate between obstacles and free spaces effectively. These findings confirm the reliability and accuracy of the SLAM approach for mapping unknown indoor spaces.

Likewise, the RRT-based obstacle avoidance approach proved to be highly successful in navigating the simulated indoor environment. The algorithm efficiently generated feasible paths that avoided obstacles, allowing the robot to reach its destination while maintaining a safe distance from detected obstacles. The RRT algorithm's adaptability to various obstacle configurations and its ability to explore the state space contributed to its effective and efficient navigation capabilities.

Furthermore, the comparison of pose estimation results revealed the superiority of SLAM-based pose estimation over odometry-based estimation in terms of accuracy and closeness to the real pose. By incorporating sensor measurements and dynamically updating the map in real-time, the SLAM algorithm compensated for odometry errors and environmental changes, resulting in more reliable pose estimates.

However, it is important to acknowledge the limitations observed during the deep learning model training for obstacle avoidance. Due to the limited accuracy and quantity of the collected data, the model experienced over-fitting, impacting its performance. To address this limitation, future work should focus on collecting and training the model with a more extensive data-set, potentially incorporating SLAM pose data to enhance accuracy and reliability. By leveraging SLAM, it is anticipated that the obstacle avoidance model will achieve improved performance and enhance the agent's ability to navigate through complex environments.

4.6 Limitations and Future Work

The development and evaluation of the obstacle avoidance system highlighted several limitations and areas for future work. Some of the key limitations include:

- Computational power constraints hindered the collection of a large and diverse data-set, affecting the model's performance.
- The reliance on odometry poses computation introduced errors that impacted the accuracy of the trained model.
- The current training algorithm may not generalize well to unseen scenarios and obstacle configurations.

To overcome these limitations and further enhance the obstacle avoidance system, the following future work is proposed:

- Collection and training the obstacle avoidance model using SLAM pose data for improved accuracy and reliability.
- Investigation techniques to improve the generalization capabilities of the trained model, enabling it to adapt to unseen scenarios and obstacles.
- Conduction of extensive real-world testing and evaluation to validate the system's performance in different environments and conditions.

By addressing these limitations and pursuing the proposed future work, it is anticipated that the obstacle avoidance system can be further refined and optimized, making it more effective and reliable in real-world applications.

4.7 Conclusion

In conclusion, the implemented SLAM and RRT algorithms demonstrated their effectiveness in map construction and obstacle avoidance, respectively, in the simulated environment. The Karto SLAM algorithm successfully generated a precise and comprehensive map of the unknown indoor environment, while the RRT algorithm enabled the robot to navigate autonomously while avoiding obstacles.

The results obtained provide valuable insights into the performance and efficiency of these algorithms in the simulated environment. The SLAM-based map construction process showcased the algorithm's ability to gradually update the map and accurately represent the environment. The RRT-based obstacle avoidance approach demonstrated its capability to generate feasible paths that ensured safe navigation.
However, the deep learning algorithm for obstacle avoidance exhibited limitations in terms of accuracy due to errors in odometry pose computation and constraints in data-set collection.
Future work should focus on addressing these limitations by incorporating SLAM pose data, which offers higher accuracy and reliability. By leveraging SLAM, it is expected that the obstacle avoidance model will achieve improved performance and enhance the agent's ability to navigate through complex environments. The comparison of pose estimation results highlighted the advantages of SLAM-based pose estimation over odometry-based estimation, indicating its suitability for accurate localization and mapping in real-world scenarios.

Overall, the results and discussion presented in this chapter provide valuable insights into the performance and limitations of the implemented algorithms. These findings contribute to the development of autonomous robotic systems, paving the way for further enhancements and real-world applications. Future research directions may involve testing the algorithms in physical environments, considering additional sensor inputs, and optimizing their performance for specific tasks and constraints, thereby addressing the limitations highlighted in this chapter.

General Conclusion

In conclusion, this project aimed to design a cost-efficient and robust obstacle avoidance system by integrating Simultaneous Localization and Mapping (SLAM) with Rapidly-exploring Random Trees (RRT) and exploring the use of deep learning for obstacle detection and avoidance. The SLAM and RRT methodologies consistently demonstrated high accuracy in mapping the environment and generating collision-free paths.

However, the deep learning-based obstacle avoidance component faced challenges and exhibited limitations in terms of accuracy. Despite these challenges, a thorough evaluation was conducted to analyze the problem and identify potential solutions.

To address these limitations, future work should focus on improving the deep learning component. This can be achieved by collecting a more diverse and accurate data-set , refining the model architecture to capture more intricate obstacle features, and optimizing the training process to enhance the model's performance.

Although the deep learning-based obstacle avoidance system did not meet the desired accuracy level within the scope of this project, it serves as a valuable starting point for further research. By addressing the identified challenges and incorporating advancements in deep learning techniques, it is anticipated that the system's accuracy can be significantly improved.

In summary, this project successfully demonstrated the effectiveness of integrating SLAM and RRT for accurate mapping and path planning in obstacle avoidance systems. While the deep learning-based method faced challenges, it highlighted areas for improvement and identified potential solutions for future work. By addressing the limitations and leveraging advancements in deep learning, it is expected that a more accurate and reliable obstacle avoidance system can be realized, further enhancing the cost-efficiency and robustness of the overall system.

References

[1] Mary B Alatise and Gerhard P Hancke. “A review on challenges of autonomous mobile robot and sensor fusion methods”. In: *IEEE Access* 8 (2020), pp. 39830–39846.

[2] Jesús Hamilton Ortiz. “Industry 4.0: Current status and future trends”. In: (2020).

[3] Shiyong Wang et al. “Towards smart factory for industry 4.0: a self-organized multi-agent system with big data based feedback and coordination”. In: *Computer networks* 101 (2016), pp. 158–168.

[4] Maria Isabel Ribeiro. “Obstacle avoidance”. In: *Instituto de Sistemas e Robótica, Instituto Superio Técnico* 1 (2005).

[5] Angelo Nikko Catapang and Manuel Ramos. “Obstacle detection using a 2D LIDAR system for an Autonomous Vehicle”. In: *2016 6th IEEE International Conference on Control System, Computing and Engineering (ICCSCE)*. IEEE. 2016, pp. 441–445.

[6] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. “ImageNet classification with deep convolutional neural networks”. In: *Advances in neural information processing systems*. 2012, pp. 1097–1105.

[7] Lucas Rausch and Cyrill Stachniss. “Robust and efficient obstacle detection for autonomous driving in urban environments”. In: *IEEE Robotics and Automation Letters* 6.2 (2021), pp. 1812–1819.

[8] Jaesik Lim et al. “Real-time semantic segmentation for autonomous driving based on LIDAR data”. In: *2020 IEEE Intelligent Transportation Systems Conference (ITSC)*. IEEE. 2020, pp. 3773–3778.

[9] Yusheng Zhang, Keqiang Xu, and Jun Wang. “Review of obstacle detection and tracking for intelligent vehicles”. In: *IEEE Transactions on Intelligent Transportation Systems* 20.3 (2019), pp. 1103–1117.

[10] Radu Bogdan Rusu, Nico Blodow, and Michael Beetz. “Fast point feature histograms (FPFH) for 3D registration”. In: *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2009, pp. 3212–3217.

[11] Konrad Wenzl, Heinrich Ruser, and Christian Kargel. “Decentralized multi-target-tracking using a lidar sensor network”. In: *2012 IEEE International Instrumentation and Measurement Technology Conference Proceedings*. IEEE. 2012, pp. 2492–2497.

[12] Morgan Quigley et al. “ROS: an open-source Robot Operating System”. In: *ICRA workshop on open source software*. Vol. 3. 3.2. 2009, p. 5.

[13] Robins Mathew and Somashekhar S Hiremath. “Development of waypoint tracking controller for differential drive mobile robot”. In: *2019 6th International Conference on Control, Decision and Information Technologies (CoDIT)*. IEEE. 2019, pp. 1121–1126.

[14] Alexandr Stefek et al. “Energy comparison of controllers used for a differential drive wheeled mobile robot”. In: *IEEE Access* 8 (2020), pp. 170915–170927.

[15] Mahesh Yallala and SJ Mija. “Path tracking of differential drive mobile robot using two step feedback linearization based on backstepping”. In: *2017 International Conference on Innovations in Control, Communication and Information Systems (ICICCI)*. IEEE. 2017, pp. 1–6.

[16] Diana Diaz and Rafael Kelly. “On modeling and position tracking control of the generalized differential driven wheeled mobile robot”. In: *2016 IEEE International Conference on Automatica (ICA-ACCA)*. IEEE. 2016, pp. 1–6.

[17] Alexandr Štefek et al. “Differential drive robot: Spline-based design of circular path”. In: *Dynamical Systems: Theoretical and Experimental Analysis: Łódź, Poland, December 7-10, 2015*. Springer. 2016, pp. 331–342.

[18] Spyros G Tzafestas. *Introduction to mobile robot control*. Elsevier, 2013.

[19] *ROS Wiki: RobotSetup/Odom*. `http://wiki.ros.org/navigation/Tutorials/RobotSetup/Odom`. Accessed: <may 2023>.

[20] Sebastian Thrun. “Probabilistic algorithms in robotics”. In: *Ai Magazine* 21.4 (2000), pp. 93–93.

[21] Hugh Durrant-Whyte and Tim Bailey. “Simultaneous localization and mapping: part I”. In: *IEEE robotics & automation magazine* 13.2 (2006), pp. 99–110.

[22] Raul Mur-Artal and Juan D Tardós. “Orb-slam2: An open-source slam system for monocular, stereo, and rgb-d cameras”. In: *IEEE transactions on robotics* 33.5 (2017), pp. 1255–1262.

[23] Cesar Cadena et al. “Past, present, and future of simultaneous localization and mapping: Toward the robust-perception age”. In: *IEEE Transactions on robotics* 32.6 (2016), pp. 1309–1332.

[24] Michael Montemerlo et al. “FastSLAM: A factored solution to the simultaneous localization and mapping problem”. In: *Aaai/iaai* 593598 (2002).

[25] AbdelKarim ZIANI-KERARTI. “Simultaneous Localization And Mapping”. PhD thesis. Directeur; M. MOKHTARI Rida/CO-Directeur; M. ARICHI Fayssal, 2021.

[26] Abu Bakar Sayuti HM Saman and Ahmed Hesham Lotfy. “An implementation of SLAM with extended Kalman filter”. In: *2016 6th International Conference on Intelligent and Advanced Systems (ICIAS)*. IEEE. 2016, pp. 1–4.

[27] Luciano Buonocore, Cairo Lúcio Nascimento Júnior, and Areolino de Almeida Neto. “Solving the Indoor SLAM Problem for a Low-Cost Robot using Sensor Data Fusion and Autonomous Feature-based Exploration.” In: *ICINCO (2)*. 2012, pp. 407–414.

[28] Javier Civera et al. “1-point RANSAC for EKF-based structure from motion”. In: *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE. 2009, pp. 3498–3504.

[29] Michael Montemerlo. “FastSLAM: A factored solution to the simultaneous localization and mapping problem with unknown data association”. In: *PhD thesis, Carnegie Mellon University* (2003).

[30] Michael Montemerlo and Sebastian Thrun. *FastSLAM: A scalable method for the simultaneous localization and mapping problem in robotics*. Vol. 27. Springer, 2007.

[31] Steven M LaValle et al. “Rapidly-exploring random trees: A new tool for path planning”. In: (1998).

[32] John D Kelleher. *Deep learning*. MIT press, 2019.

[33] Sorin Grigorescu et al. “A survey of deep learning techniques for autonomous driving”. In: *Journal of Field Robotics* 37.3 (2020), pp. 362–386.

[34] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. “Deep learning”. In: *nature* 521.7553 (2015), pp. 436–444.

[35] Leon O Chua and Tamas Roska. “The CNN paradigm”. In: *IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications* 40.3 (1993), pp. 147–156.

[36] Larry R Medsker and LC Jain. “Recurrent neural networks”. In: *Design and Applications* 5 (2001), pp. 64–67.

[37] Alex Graves and Alex Graves. “Long short-term memory”. In: *Supervised sequence labelling with recurrent neural networks* (2012), pp. 37–45.

[38] Kunfeng Wang et al. “Generative adversarial networks: introduction and outlook”. In: *IEEE/CAA Journal of Automatica Sinica* 4.4 (2017), pp. 588–598.

[39] Chuanqi Tan et al. “A survey on deep transfer learning”. In: *Artificial Neural Networks and Machine Learning–ICANN 2018: 27th International Conference on Artificial Neural Networks, Rhodes, Greece, October 4-7, 2018, Proceedings, Part III 27*. Springer. 2018, pp. 270–279.

[40] Yilu Zhao and Xiong Chen. “Prediction-based geometric feature extraction for 2D laser scanner”. In: *Robotics and Autonomous Systems* 59.6 (2011), pp. 402–409.

[41] Maurice Quach et al. “Survey on deep learning-based point cloud compression”. In: *Frontiers in Signal Processing* (2022).

[42] Victor Vaquero, Ely Repiso, and Alberto Sanfeliu. “Robust and real-time detection and tracking of moving objects with minimum 2D LIDAR information to advance autonomous cargo handling in ports”. In: *Sensors* 19.1 (2018), p. 107.

[43] Viet Nguyen et al. “A comparison of line extraction algorithms using 2D laser rangefinder for indoor mobile robotics”. In: *2005 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE. 2005, pp. 1929–1934.

[44] Cristiano Premebida and Urbano Nunes. “Segmentation and geometric primitives extraction from 2d laser range data for mobile robot applications”. In: *Robotica* 2005 (2005), pp. 17–25.

[45] Lanxiang Zheng et al. “The obstacle detection method of uav based on 2D lidar”. In: *IEEE Access* 7 (2019), pp. 163437–163448.

[46] Carlos Fernández et al. “Clustering and line detection in laser range measurements”. In: *Robotics and Autonomous Systems* 58.5 (2010), pp. 720–726.

[47] Alberto J Álvares et al. “A navigation and path planning system for the Nomad XR4000 mobile robot with remote web monitoring”. In: *ABCM Symposium Series in Mechatronics*. Vol. 1. 2004, pp. 18–24.

[48] Trung-Dung Vu, Olivier Aycard, and Nils Appenrodt. “Online localization and mapping with moving object tracking in dynamic outdoor environments”. In: *2007 IEEE Intelligent Vehicles Symposium*. IEEE. 2007, pp. 190–195.

[49] Taketoshi Mori et al. “Moving objects detection and classification based on trajectories of LRF scan data on a grid map”. In: *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE. 2010, pp. 2606–2611.

[50] Baifan Chen et al. “Real-time detection of dynamic obstacle using laser radar”. In: *2008 The 9th International Conference for Young Computer Scientists*. IEEE. 2008, pp. 1728–1732.

[51] Rasoul Mojtahedzadeh. “Robot obstacle avoidance using the Kinect”. In: *Master of Science Thesis Stockholm, Sweden* (2011).

[52] Ran Sun. “Performance evaluation and improvement of LiDAR-based obstacle detection algorithm”. In: ().

[53] Eric N Willcox III. “Forward perception using a 2d lidar on the highway for intelligent transportation”. PhD thesis. Thesis, Worcester Polytechnic Institute, 2016. 4, 2016.

[54] Martin Dekan et al. “Moving obstacles detection based on laser range finder measurements”. In: *International Journal of Advanced Robotic Systems* 15.1 (2018), p. 1729881417748132.

[55] Christoph Mertz et al. “Moving object detection with laser scanners”. In: *Journal of Field Robotics* 30.1 (2013), pp. 17–43.

[56] Anna Petrovskaya and Sebastian Thrun. “Model based vehicle detection and tracking for autonomous urban driving”. In: *Autonomous Robots* 26.2-3 (2009), pp. 123–139.

[57] Rob MacLachlan and Christoph Mertz. “Tracking of moving objects from a moving vehicle using a scanning laser rangefinder”. In: *2006 IEEE Intelligent Transportation Systems Conference*. IEEE. 2006, pp. 301–306.

[58] Regis Vincent, Benson Limketkai, and Michael Eriksen. “Comparison of indoor robot localization techniques in the absence of GPS”. In: *Detection and sensing of mines, explosive objects, and obscured targets XV*. Vol. 7664. SPIE. 2010, pp. 606–610.

[59] Kurt Konolige et al. “Efficient sparse pose adjustment for 2D mapping”. In: *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE. 2010, pp. 22–29.

[60] Kevin Murphy and Stuart Russell. “Rao-Blackwellised particle filtering for dynamic Bayesian networks”. In: *Sequential Monte Carlo methods in practice* (2001), pp. 499–515.

[61] Giorgio Grisetti, Cyrill Stachniss, and Wolfram Burgard. “Improved techniques for grid mapping with rao-blackwellized particle filters”. In: *IEEE transactions on Robotics* 23.1 (2007), pp. 34–46.

[62] Pham Xuan Hien and Gon-Woo Kim. “Goal-oriented navigation with avoiding obstacle based on deep reinforcement learning in continuous action space”. In: *2021 21st International Conference on Control, Automation and Systems (ICCAS)*. IEEE. 2021, pp. 8–11.

[63] Amin Nejatbakhsh. *Scalable Tools for Information Extraction and Causal Modeling of Neural Data*. Columbia University, 2022.

[64] Hervé Abdi, Dominique Valentin, and Betty Edelman. *Neural networks*. 124. Sage, 1999.

[65] Nissen Steffen. “Neural Networks made simple”. In: *Fast neural network library (Fann)* (2005), pp. 14–15.

[66] Murat H Sazli. “A brief review of feed-forward neural networks”. In: *Communications Faculty of Sciences University of Ankara Series A2-A3 Physical Sciences and Engineering* 50.01 (2006).

[67] Ramon Quiza and JP Davim. “Computational modeling of machining systems”. In: *Intelligent machining: modeling and optimization of the machining processes and systems. London: ISTE* (2009), pp. 173–213.

[68] J Han. *M. Kamber, and J. Pei,“Data Mining Concepts and Techniques”*. 2011.

[69] Arun A Ross and Rohin Govindarajan. “Feature level fusion of hand and face biometrics”. In: *Biometric technology for human identification II*. Vol. 5779. SPIE. 2005, pp. 196–204.

[70] Francois Chollet. “The limitations of deep learning”. In: *Deep learning with Python* (2017).

[71] Lea Steffen et al. “Reactive neural path planning with dynamic obstacle avoidance in a condensed configuration space”. In: *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2022, pp. 8101–8108.

[72] Carol Fairchild and Thomas L Harman. *ROS robotics by example*. Packt Publishing Ltd, 2016.

[73] Mirco De Marchi et al. “Efficient ROS-Compliant CPU-iGPU communication on embedded platforms”. In: *Journal of Low Power Electronics and Applications* 11.2 (2021), p. 24.

[74] Morgan Quigley, Brian Gerkey, and William D Smart. *Programming Robots with ROS: a practical introduction to the Robot Operating System*. " O’Reilly Media, Inc.", 2015.

[75] Wei Qian et al. “Manipulation task simulation using ROS and Gazebo”. In: *2014 IEEE International Conference on Robotics and Biomimetics (ROBIO 2014)*. IEEE. 2014, pp. 2594–2598.

[76] Zandra B Rivera, Marco C De Simone, and Domenico Guida. “Unmanned ground vehicle modelling in Gazebo/ROS-based environments”. In: *Machines* 7.2 (2019), p. 42.

[77] Maxim Sokolov et al. “3D modelling and simulation of a crawler robot in ROS/Gazebo”. In: *Proceedings of the 4th International Conference on Control, Mechatronics and Automation*. 2016, pp. 61–65.

[78] Nathan Koenig and Andrew Howard. “Design and use paradigms for gazebo, an open-source multi-robot simulator”. In: *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)(IEEE Cat. No. 04CH37566)*. Vol. 3. IEEE. 2004, pp. 2149–2154.

[79] Deepali Ghorpade, Anuradha D Thakare, and Sunil Doiphode. “Obstacle detection and avoidance algorithm for autonomous mobile robot using 2D LiDAR”. In: *2017 International Conference on Computing, Communication, Control and Automation (ICCUBEA)*. IEEE. 2017, pp. 1–6.

[80] Ran Sun. “Performance evaluation and improvement of LiDAR-based obstacle detection algorithm”. In: ().

[81] X Fan, Y Wang, and Z Zhang. “An evaluation of Lidar-based 2D SLAM techniques with an exploration mode”. In: *Journal of Physics: Conference Series*. Vol. 1905. 1. IOP Publishing. 2021, p. 012021.

[82] Zijun Zhang. “Improved adam optimizer for deep neural networks”. In: *2018 IEEE/ACM 26th international symposium on quality of service (IWQoS)*. Ieee. 2018, pp. 1–2.