PEOPLE'S DEMOCRATIC REPUBLIC OF ALGERIA

Ministry of Higher Education and Scientific Research

University of Amar Telidji - Laghouat

Faculty of Technology

Department of Electronics

MASTER THESIS

DOMAINE: Science & Technology

OPTION: Automation & Industrial Computing

BENAIDJA Manal & BENKOUIDER Assia

Theme

AI-Driven Mobile Robot Navigation: Precision in Road Following and Objective Pursuit

Jury members:

BELKHIRI Mohammed	Pr	President
ABOUCHABANA Nabil	MCB	Examiner
OUBBATI Brahim Khalil	MCB	Supervisor

2023 / 2024

Abstract

In recent years, mobile robots have become essential in various fields due to their ability to perform tasks intelligently and efficiently. This project aims to control a two-wheeled mobile robot and make it follow a path using artificial intelligence. Sensors like cameras and LiDAR are key components of this project. We used specific algorithms to control the robot with a keyboard and joystick and enabled autonomous navigation with a camera and deep learning techniques. Our goal is to develop a mobile robot that can accurately follow a set path, map its surroundings, and move smoothly without issues using a PID controller. The experimental results showed good performance in road following and autonomous navigation.

Keywords: Mobile Robot ,ROS Robot operating system ,AI Artificial intelligence, LiDar, PID controller .

Résumé

Ces dernières années, les robots mobiles sont devenus essentiels dans divers domaines en raison de leur capacité à effectuer des tâches de manière intelligente et efficace. Ce projet vise à contrôler un robot mobile à deux roues et à lui faire suivre un chemin en utilisant l'intelligence artificielle. Les capteurs tels que les caméras et le LiDAR sont des éléments clés de ce projet. Nous avons utilisé des algorithmes spécifiques pour contrôler le robot à l'aide d'un clavier et d'un joystick et avons permis une navigation autonome à l'aide d'une caméra et de techniques d'apprentissage profond. Notre objectif est de développer un robot mobile capable de suivre avec précision un chemin défini, de cartographier son environnement et de se déplacer en douceur sans problème à l'aide d'un contrôleur PID. Les résultats expérimentaux ont montré de bonnes performances

en matière de suivi de route et de navigation autonome.

Keywords: Robot mobile ,ROS Système d'exploitation du robot ,AI Intelligence artificielle, LiDar, Régulateur PID .

ملخص

لعبت الروبوتات المتنقلة في السنوات الأخيرة دورًا بارزًا في مجالات متعددة، حيث تم تطويرها للقيام بمهام مختلفة بذكاء وكفاءة. تهدف هذه المذكرة إلى التحكم في روبوت متحرك ذو عجلتين وتتبع المسار باستخدام الذكاء الاصطناعي. بالإضافة إلى ذلك، يعتمد المشروع على استخدام أجهزة الاستشعار التي لعبت دورًا رئيسيًا في مشروعنا، مثل الكاميرا والليدار. ومن خلال تطبيق خوارزميات معينة، تمكنا من التحكم في الروبوت عبر لوحة المفاتيح وعصا التحكم ، بالإضافة إلى إجراء التنقل المستقل باستخدام الكاميرا وتقنيات التعلم العميق. الهدف النهائي هو تطوير روبوت متنقل قادر على اتباع مسار محدد بدقة من خلال استخدام وحدة التحكم تناسبي تكاملي تفاضلي ، ورسم خرائط المناطق المحيطة به والتحرك بسلاسة دون التعرض لمشاكل.

الكلمات المفتاحية: ، نظام التشغيل للروبوت روس ، ، الروبوت المتنقل ،متحكم تناسبي تكاملي تفاضلي ، التنقل .

Acknowledgements

We would like to express our sincere gratitude and deep thanks to our esteemed supervisors, for their guidance, encouragement and continuous support which were crucial in the successful completion of my Master's thesis. Their valuable guidance has contributed greatly to the development of research and improvement of its quality. We would also like to express our deep thanks to Professor Dr. Brahim Khalil Oubbati for this valuable contributions and thoughtful suggestions throughout the process of preparing the thesis. These contributions have contributed significantly to enhancing the quality of research and its development. In conclusion, we would like to express our deep gratitude to our dear families. Their constant love and unwavering support have been our pillar of strength throughout this academic pursuit. We are grateful for the sacrifices they made to ensure our success and realize that their love for us never ends.

We thank you all for your continued support and encouragement, and we will do our best to be worthy of this support and achieve more successes in the future.

BENAIDJA Manal & **BENKOUIDER Assia**

Laghouat University

July 2024

Acronyms

ROS Robot Operating System

AI Artificial intelligence

ADS Autonous Driving Cars

SAE Society Of Automotive Engineers

IMU Inertial Meosurement Units

GNSS Global Navigation Satellite System

LTS Long-Term Support

TCP Transmission Control Protocol

IP Internet Protocol

ResNet 18 Residual Network-18

ICR Instantaneous Center Of Rotation

PID Proportional-Integral-Derivative

MPC Model Predictive Control

LQR Linear Quadratic Regulator

LQG Linear Quadratic Gaussian

LIDAR Light Detection and Ranging

GPU Graphics Processing Unit

MSE Mean Square Error

DSP Digital Signal Processor

SLAM Simultaneous Localisation and Mapping

Rvis Ros Visualization

Contents

List of Figures

General Introduction

Mobile robots have been extensively studied with the aim of employing them in various application domains like industry, military operations, mining, planetary exploration, and more [1].These robotic systems are regarded as vehicles whose kinematic models emulate the manoeuvrability of an automobile. Path following stands out as a fundamental motion task for robots, entailing the robot's endeavour to navigate and adhere to a geometric path defined by a series of oriented waypoints, starting from an initial configuration, all accomplished without human intervention. The robot's configuration encompasses its main body's position and orientation, along with the steering wheels' angle. Traditionally, achieving this task involves depicting stationary obstacles and the mobile robot's configuration on a two-dimensional map. Waypoints are then generated by a planner that must consider the robot's geometric structure and kinematics. The rationale behind employing Artificial Intelligence (AI) for path tracking lies in its capacity to manage intricate systems, drawing inspiration from human expertise to do so effectively.
This master thesis is divided into four chapters besides the introduction and the conclusion:

- **Chapter 1** we will present the like-car levels and his Architecture of Vehicle System, highlighting their real-life applications and the pivotal role of Artificial Intelligence (AI) in shaping their capabilities. Additionally, we'll discuss the Robot Operating System (ROS).

- **Chapter 2** This chapter introduces the mathematical model for a two-wheeled mobile robot, designs a classical controller for precise path tracking, and incorporates artificial intelligence, specifically ResNet-18, to generate optimal paths

and improve the robot's adaptability.

- **Chapter 3** Path following using classical controller (PID) This chapter focuses on the path following of car-like vehicles using classical control techniques, specifically the Proportional-Integral-Derivative (PID) controller. are commonly used in various applications, such as autonomous navigation and robotics. Also in this chapter we will do two applications to control the robot using the keyboard and joystick.Achieving accurate and robust path following is crucial for the successful operation of these vehicles. where This chapter explores the principles, design, and implementation of the PID controller for path following in car-like vehicles.

- **Chapter 4** Experiments and evaluation This chapter focuses on the experimental setup and evaluation of path-following methods for car-like vehicles. and The robot's ability to navigate autonomously using lidar and IMU ...Without any problem , and also The performance and effectiveness of the implemented algorithms are evaluated through a series of experiments. The results obtained from these experiments provide insight into the capabilities and limitations of path-following technologies, allowing a comprehensive evaluation of their performance.

Chapter 1

Autonomous Driving Cars

1.1 Introduction

Robots have been widely used in recent years, particularly in production lines where they can significantly reduce the time required to manufacture products and lessen the reliance on human labour in industries. Additionally, robots can also free humans from hazardous or risky tasks such as military operations, firefighting, underwater exploration, and space exploration. As robots increasingly integrate into daily life, such as office automation and surgical procedures, they may also assist individuals with disabilities in their daily activities. Looking ahead, autonomous robots are poised to have a significant impact on human life in the future.

1.2 Backgrouand

The term "Autonomous mobile robot" refers to a system capable of independent navigation, executing actions or tasks without reliance or external human intervention [2]. These robots exhibit diverse locomotion methods, including flying, swimming, crawling, walking or rolling, making them versatile across industries like factories, hospitals and transportation [3]. This master thesis delves into the crucial application of wheeled mobile robots (autonomous driving cars (ADCs)).

Autonomous cars, also known as self-driving cars or driver-less vehicles, represent a sophisticated system that offers numerous societal benefits, including a decrease in

road accidents, enhanced safety and convenient transportation for passengers. they can significantly mitigate the impact of driver errors as a leading cause of car collisions. Moreover, they offer specialized mobility solutions for individuals unable to drive due to visual disabilities or physical limitations [4].

1.2.1 Autonomous cars Levels:

As per the Society of Automotive Engineers (SAE), autonomous cars are categorized into five levels of autonomy, outlined hereafter :

- Level zero (no automation): at this level, the driver is solely responsible for the driving tasks.
- Level one (driver assistance): It encompasses a single automated function, such as adaptive cruise control
- Level two (partial automation):It contains several automated functions such as longitudinal or lateral control in addition to contingency braking.
- Level three (conditional automation):The vehicle can perform many automated functions under definite conditions, but the driver is necessary to perform control if the vehicle leaves the automated functions
- Level four (high automation): The vehicle autonomously performs all driving functions under certain conditions, rendering the driver unnecessary at this level.
- Level five (full automation): At this level, the car autonomously handles all driving functions under various conditions without requiring a driver.

1.3 Architecture of an Autonomous Vehicle System

The architecture of an autonomous driving cars system is divided into two parts: Hardware architecture and software architecture as shown in Figure 1.1

The hardware structure can be divided into three main components: sensor devices, microcontrollers, and actuation systems. Conversely, the software framework can be

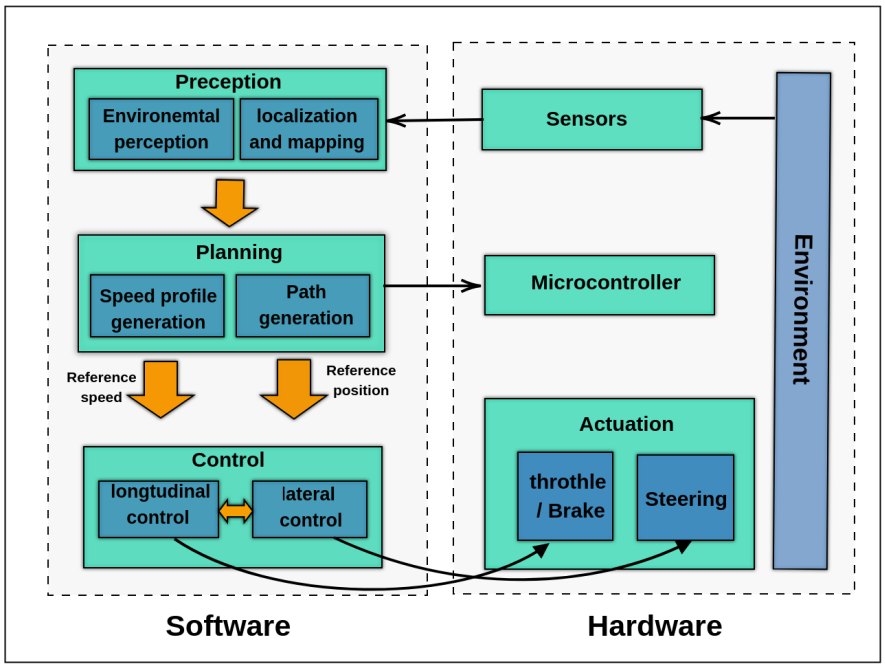

Figure 1.1: Architecture of an Autonomous Vehicle System.

categorized into three key aspects: perception, planning, and control. Python software is employed for programming microcontrollers [5]

Planning: The planning functions in autonomous driving are structured into three layers: route planning, behavioural planning, and motion planning. In route planning, the Autonomous Driving Controller (ADC) performs calculations to identify the optimal path from the present location to the destination, leveraging map data . Once a route plan is established, the ADC adheres to road regulations and driving norms, primarily managed by the behavioral layer [6]. Motion planning is then utilized to determine the specific trajectory and speed required by the control system.

Perception: The perception part receives the information from the sensor devices; including the environmental perception, localization, and mapping [7]. Environmental perception involves gathering data about the autonomous vehicle's surroundings, including object detection [8]. Conversely, localization enables an Autonomous Driving Controller (ADC) to determine its position within a map of the environment, including its orientation. Localization heavily relies on sensor technologies like Inertial Measurement Units (IMUs), Global Navigation Satellite System (GNSS), odometry, cameras, or lidar .

The mapping component of the ADC is crucial for constructing an accurate representation of its surroundings, which can be based on either a global or local map. This map must be highly precise to ensure the safe operation of the ADC [9]

Car or Robot Mobile Control The control system of an Autonomous Driving Controller (ADC) is divided into longitudinal and lateral control. Longitudinal control manages the throttle and brake actuators, while lateral control adjusts the steering angle of the vehicle. The primary role of the control system is to implement decisions made by the planning module. It achieves this by transmitting commands to the actuators, ensuring the vehicle navigates safely and effectively in its environment [10]

This thesis concentrates on investigating the control aspect of Autonomous Driving Controllers (ADCs) to achieve precise tracking of road speed and trajectory within the environment. Accurate control is crucial for ensuring safe and efficient driving operations. This precision is achieved by determining the optimal gains for the controllers. Traditional adjustment methods, relying on mathematical calculations and trial and error, do not guarantee the best gains. Consequently, optimization techniques are utilized to identify and implement the most effective gains

1.4 AI techniques

The different techniques that used for autonomous driving control are classified as follow :

- Computer Vision techniques: used to analyze images and video to distinguish objects, track them, determine their condition, and analyze the environment surrounding the vehicle.

- Deep learning :Deep Learning is a branch of AI that uses Deep Neural Networks to extract valuable information and learn complex representations from data. It enables pattern recognition, decision-making, and prediction without explicit programming, relying on extensive training data and intensive training processes.

- Reinforcement Learning:Learning through interaction with the environment, maximizing rewards by taking correct actions and avoiding wrong ones, using the

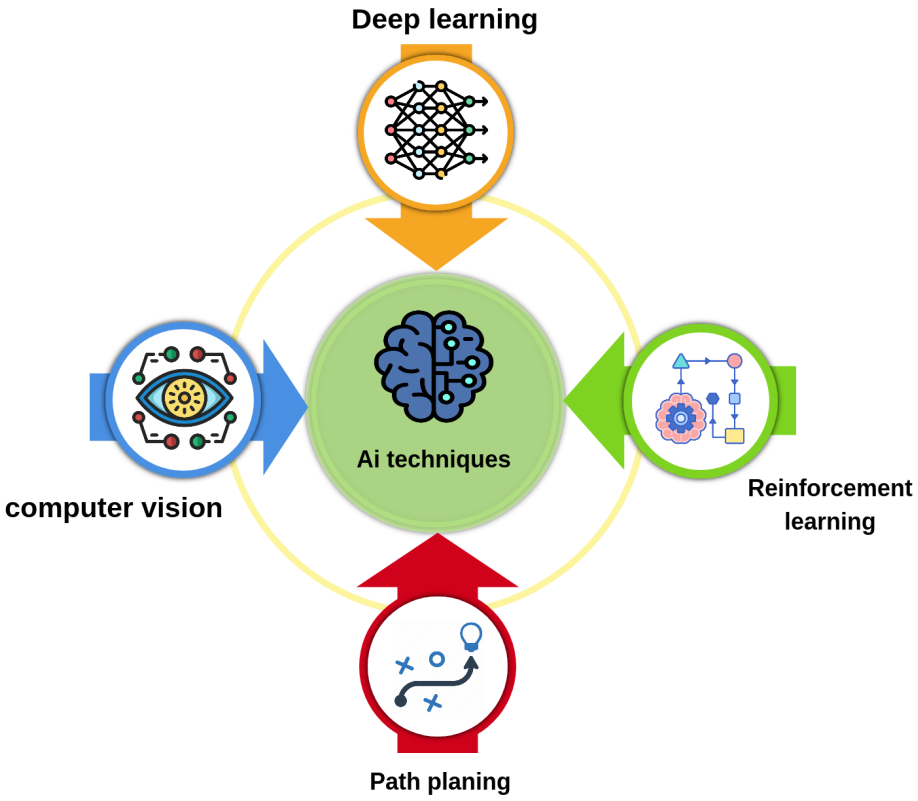

Figure 1.2: AI techniques for autonomous cars.

reward and punishment mechanism.

- Path planning: refers to the process of determining the optimal path for a robot or vehicle to achieve a specific goal. It involves converting data received from sensor and environmental information systems into a specific movement plan. It uses intelligent planning algorithms to balance efficiency, safety and obstacle avoidance.

1.5 Framework and Different Environment used for this thesis

Automated mobile navigation based on artificial intelligence is revolutionizing the field of robotics by integrating artificial intelligence (AI) algorithms and advanced sensing technologies to achieve accuracy in following the road and striving towards the goal. Among the most important components used to maintain the obtained result is “following the road accurately” which is: Ros and sensors "camera".

1.5.1 Robotic Operating System

ROS is a comprehensive software framework that facilitates communication and control of robotic hardware across various platforms, offering libraries, tools, and conventions for simplifying the development of complex robot behaviors. It handles hardware abstraction, device drivers, process communication, and package management, allowing developers to concentrate on higher-level tasks like navigation, manipulation, perception, and planning. There are many versions of ROS that presented hereafter:

1.5.2 Versions of ROS

ROS has undergone several major iterations, each introducing distinct features and compatibility enhancements. Among the noteworthy versions of ROS are:

- **ROS Noetic**

ROS Noetic, the latest LTS " Long-Term Support" version released in 2020, offers compatibility with Ubuntu 20.04 and newer editions, introducing notable improvements, bug fixes, and new features, making it the preferred choice for advancing robot development projects.
Importance of ROS Noetic in mobile robotics: robust integration, flexible programming, powerful libraries, advanced applications, improved performance, increased productivity.

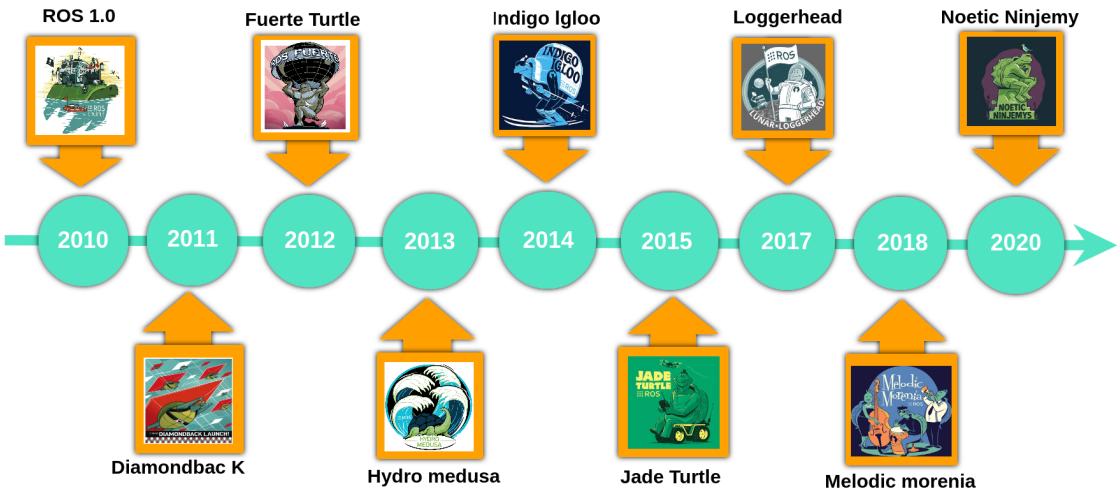

Figure 1.3: ROS versions.

1.5.3 ROS Architecture

From figure 1.4 can see that, the ROS architecture begins with the ROS Master, enabling node discovery and communication without manual IP addresses. Nodes register with the ROS Master, which facilitates message routing via a lookup table. Nodes communicate by publishing to and subscribing to Topics over a TCP/IP-like protocol. For specific data requests, ROS uses Services where nodes register services with the ROS Master. This allows nodes to request and provide data on demand.

Where the definition of nodes have been presented hereafter :

ROS Master: manages communication between distributed components in ROS.

ROS Nodes: are independent software modules that perform specific tasks and communicate via topics and services.

ROS Topics: facilitate efficient indirect data exchange between nodes using a publisher-subscriber model.

Messages: define the communication data format for consistent information exchange between nodes.

Launch Files: organize and run groups of nodes and settings for specific ROS system

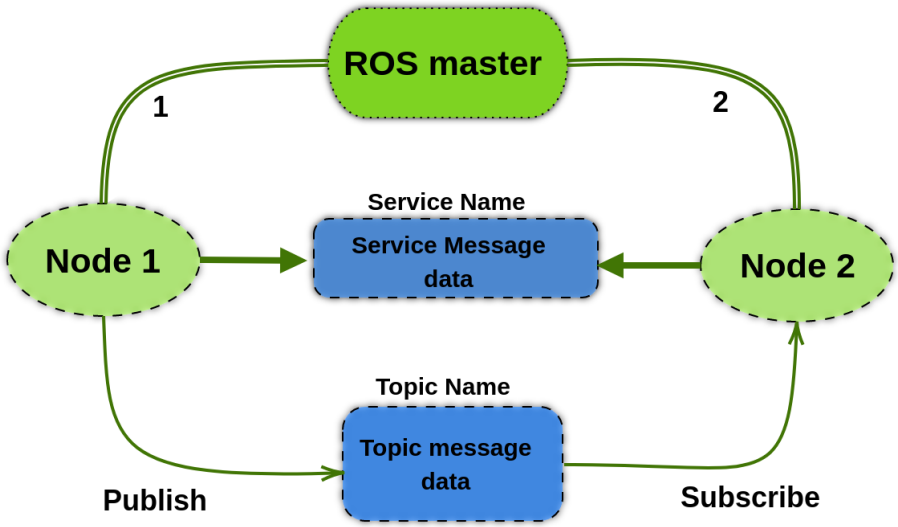

Figure 1.4: Example under ROS environment

configurations.

Publisher and Subscriber: mechanisms allow nodes to publish and receive information on topics.

Services: Services enable two-way communication and data exchange between nodes using a request-response model.

1.5.4 Camera sensor

A robot with a camera figure 1.5 combines visual input and mobility to perform tasks like object recognition, surveillance, and navigation . Controlled by a microcontroller, it captures real-time video, enabling functions such as obstacle avoidance and remote monitoring. This makes it valuable in fields like security, search and rescue, inspection, and research, enhancing capabilities for interacting with environments.

Figure 1.5: Sensors (camera)

1.6 Conclusion

In this chapter, we have thoroughly examined autonomous vehicles and their significance in daily life, providing an overview in the "Background" section. We analyzed the various levels of vehicle autonomy, from assistive driving systems to fully autonomous vehicles, detailed in the "Levels of Self-Driving Vehicles" section. The "Autonomous Vehicle System Architecture" section offered an in-depth look at the system architecture, highlighting key components and their interactions. Additionally, we explored the essential role of artificial intelligence technologies, the ROS operating system, and camera sensors. The ROS operating system facilitates seamless communication and integration among components, while the camera sensors enhance the vehicle's ability to analyze images and recognize the surrounding environment.

Chapter 2

Mathematical modeling

2.1 Introduction

The differential drive wheeled mobile robot is very common in the robotics market. Almost every beginner in robotics has built this type of platform because it is relatively easy to construct. Despite its popularity, meaning its control behavior is non-linear. Many control methods have been developed to address this issue, including linearization around an operating point, dynamic feedback linearization, and non-linear algorithms that ensure stability using Lyapunov functions.

In this chapter, we will present the mathematical molel for mobile robot based on two wheeled design. Also design a classical controller to ensure the mobile robot tracks its reference value, taking into account settling time, overshoot and ripples. Moreover, artificial intelligence has been integrated to generate the references value (path).Specifically, residual Network (ResNte-18) plays a key role in generating the optimal path for robot, enhancing its intelligence and adaptability to the surrounding environment.

2.2 Mathematical model for mobile robot

A mathematical model of a mobile robot is a mathematical representation of its motion, dynamics, and control algorithms used to control it. We can use this model to simulate the robot's behavior under certain conditions, improve its performance, and understand

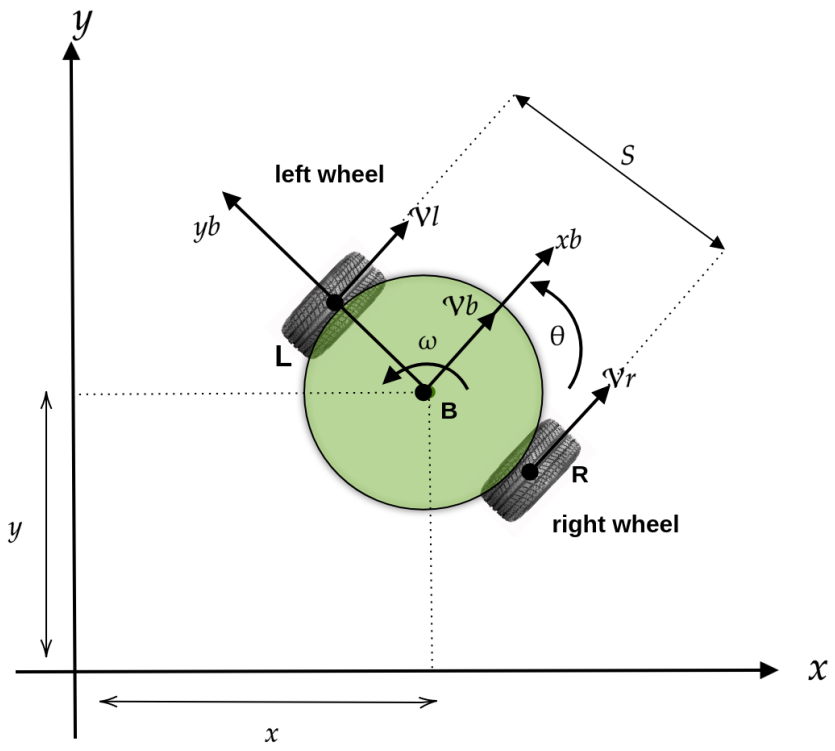

Figure 2.1: Chassis scheme of mobile robot with two wheeled

its response. The mathematical model contributes to the design and improvement of control systems and provides tools for planning and forecasting of the mobile robot.

The fig. 2.1 illustrates a robot and its reference frames. The X and Y axes define the global reference frame, also known as the inertial frame. In contrast, the x and y axes define the local reference frame, or the robot frame. The coordinates x and y represent the robot's position in the global reference frame at point B, while θ denotes the angular difference between the global and local reference frames. The distance between the two driven wheels is denoted by L. Consequently, the robot's pose can be represented as the vector $q = [x, y, \theta]^T$, comprising these three components.

mathematical model :

The velocity of the right (Vr) and left (Vl)wheels can be given as follow :

$$Vl = Vb - \frac{wL}{2} \tag{2.1}$$

$$Vr = Vb - \frac{wL}{2} \tag{2.2}$$

From Eqs 2.1 and 2.2, the linear and angular velocities (Vb and w) can be obtained by :

$$Vb = \frac{Vr + Vl}{2} \tag{2.3}$$

$$w = \frac{Vr - Vl}{2} \tag{2.4}$$

Robots designed with this architecture are subject to nonholonomic constraints, which arise from the machine's inability to move laterally relative to the wheel direction. These constraints can be described as follows:

$$A(q)\dot{q} = 0 \tag{2.5}$$

where A(q) is a full rank matrix linked with kinematic constraints :

$$A(q)\dot{q} = \begin{bmatrix} \sin\theta & -\cos\theta & 0 \end{bmatrix} \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = 0 \tag{2.6}$$

A schematic diagram of the mobile root is presented in Fig. 2.1. The kinematic equations of the two wheeled mobile robot are derived as follows:

$$\begin{cases} \dot{x} &= V_B \; cos \; (\theta) \\ \dot{y} &= V_B \; sin \; (\theta) \\ \dot{\theta} &= \omega \end{cases} \tag{2.7}$$

Can be written compactly :

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} cos(\theta) & 0 \\ sin(\theta) & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} V_b \\ \omega \end{bmatrix} \tag{2.8}$$

Where:

x and y : are the translation coordinates

θ : is the angle of rotation of the robot

ω : is the instaneous angular velocity of the robot body

V_B : is the velocity of the point B

2.3 Modeling based on Instant Center Rotation

To ensure a single Instantaneous Center of Rotation (ICR) [11], the axes of the robot's wheels must intersect at a point (see fig. 2.2). In car-type robots, this means the wheels of the front axle must be oriented differently. Additionally, due to varying radii of curvature in their trajectories, the speeds of the wheels differ. This differential steering system is called Ackerman.

ICR model :

form figures 2.2 and 2.3, we have the intensity of the velocity V_B and ω is :

$$V_b = \omega\, l \implies V_b = \frac{V_R + V_L}{2} \tag{2.9}$$

$$\omega = \frac{V_R - V_L}{2}$$

Through the equations ω and V_B ,we obtain the following equations:

$$\begin{bmatrix} V_b \\ \omega \end{bmatrix} = \begin{bmatrix} \frac{1}{2} & \frac{1}{2} \\ -\frac{1}{s} & \frac{1}{s} \end{bmatrix} \begin{bmatrix} V_L \\ V_R \end{bmatrix} \tag{2.10}$$

By substituting the equation 2.10 into the equation , we obtain :

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} cos(\theta) & 0 \\ sin(\theta) & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \frac{1}{2} & \frac{1}{2} \\ -\frac{1}{s} & \frac{1}{s} \end{bmatrix} \begin{bmatrix} V_L \\ V_R \end{bmatrix} \implies \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} \frac{cos(\theta)}{2} & \frac{cos(\theta)}{2} \\ \frac{sin(\theta)}{2} & \frac{sin(\theta)}{2} \\ -\frac{1}{s} & \frac{1}{s} \end{bmatrix} \begin{bmatrix} V_L \\ V_R \end{bmatrix} \tag{2.11}$$

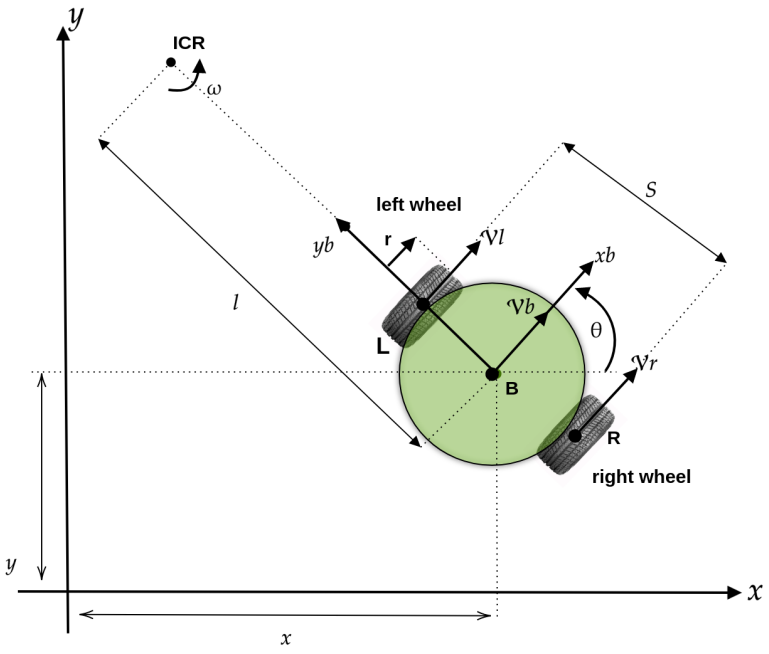

Figure 2.2: Representing the axes and setting the center.

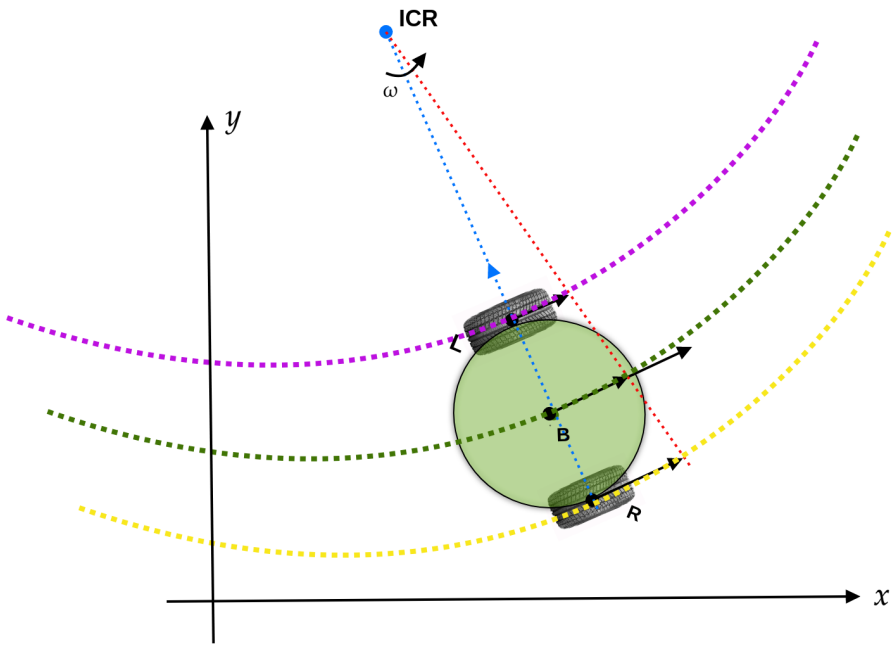

Figure 2.3: Angular speeds.

The system of equations (2.5) relates the controlled wheel velocity projections of the centre B of the robot and the angular velocity of the robot .

The last systeme equations can be expandes as follows:

$$\begin{cases} \dot{x} = \frac{cos(\theta)}{2} V_L + \frac{cos(\theta)}{2} V_R \\ \dot{y} = \frac{sin(\theta)}{2} V_L + \frac{sin\ (\theta)}{2} V_R \\ \dot{\theta} = -\frac{1}{s} V_L + \frac{1}{s} V_R \end{cases} \tag{2.12}$$

We know that the wheel velocities are actually functions of the wheel angular velocities $\dot{\phi}_L$ and $\dot{\phi}_R$:

$$\begin{cases} V_L &= r\ \dot{\phi_L} \\ V_R &= r\ \dot{\phi_R} \end{cases} \tag{2.13}$$

The two equations can be compactly written in the vector-matrix from :

$$\begin{bmatrix} V_L \\ V_R \end{bmatrix} = \begin{bmatrix} r & 0 \\ 0 & r \end{bmatrix} \begin{bmatrix} \dot{\phi_L} \\ \dot{\phi_R} \end{bmatrix} \tag{2.14}$$

Finally, we get the final statement:

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} \frac{r\ cos\ (\theta)}{2} & \frac{r\ cos\ (\theta)}{2} \\ \frac{r\ sin\ (\theta)}{2} & \frac{r\ sin\ (\theta)}{2} \\ -\frac{r}{s} & \frac{r}{s} \end{bmatrix} \begin{bmatrix} \dot{\phi_L} \\ \dot{\phi_R} \end{bmatrix} \Longrightarrow \begin{cases} \dot{x} &= \frac{r\dot{\phi_L}}{2}\ cos(\theta)\ +\ \frac{r\dot{\phi_R}}{2}\ cos(\theta) \\ \dot{y} &= \frac{r\dot{\phi_L}}{2}\ sin\ (\theta)\ +\ \frac{r\dot{\phi_R}}{2}\ sin(\theta) \\ \dot{\theta} &= -\ \frac{r}{s}\ \dot{\phi_L}\ +\ \frac{r}{s}\ \dot{\phi_R} \end{cases} \tag{2.15}$$

The equation (2.15) is the final equation derived it relates the angular velocities of the wheels with the velocity projections of the center of the robot and the robot's angular velocity.

2.4 Path Following

There are several techniques used in path tracking for mobile robots. Here are some basic techniques:

PID control: is a classic technique for path following that adjusts the robot's steering angle based on the error between the desired path and the robot's position, utilizing proportional, integral, and derivative terms[12] for smooth and accurate control.

Model Predictive Control (MPC): is an optimization-based technique that predicts and optimizes the robot's control inputs to minimize a cost function , allowing it to follow a desired path while considering dynamics, constraints, and dynamic obsta-

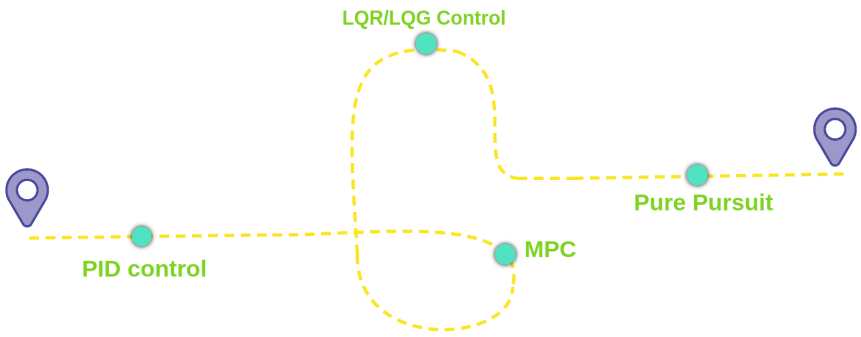

Figure 2.4: Different technique used for Path following application.

cles [13].

LQR/LQG Control: is an optimal control approach that minimizes a quadratic cost function (LQR) or incorporates system uncertainties (LQG) through a Kalman filter,[14] enabling effective path following by appropriately formulating the cost function and system dynamics.

2.4.1 Challenges faced by robots:

There are several significant challenges that robots encounter when it comes to path following. Some of the key challenges include:

- Environment changes: They can affect the robot's ability to follow the path accurately, as it may face challenges such as the presence of unexpected obstacles or changes in lighting that affect its sensing and determination of its path.

- Inaccuracy in sensing: It can affect the robot's performance in following the path, as errors in measurements or unwanted noise interference may occur.

- Obstacle handling: involves designing strategies to safely avoid or interact with obstacles for the robot to follow the path.

2.5 Classical controller For tracking task

The PID (Proportional-Integral-Derivative) controller is a feedback control mechanism commonly used in mobile robots to regulate and maintain desired behavior. It continuously calculates an error signal based on the difference between the desired and actual states, and adjusts control outputs accordingly to minimize the error and achieve stable and accurate robot motion.

The equation of a PID controller is typically represented as:

$$U(t) = K_p\, e(t) + K_i \int e(t)\, dt + K_d \frac{de(t)}{dt} \tag{2.16}$$

We considering the following system dynamic :

$$\begin{cases} \dot{x} &= \frac{r\dot{\phi_L}}{2}\, cos(\theta) + \frac{r\dot{\phi_R}}{2}\, cos(\theta) \\ \dot{y} &= \frac{r\dot{\phi_L}}{2}\, sin\,(\theta) + \frac{r\dot{\phi_R}}{2}\, sin(\theta) \\ \dot{\theta} &= -\frac{r}{s}\,\dot{\phi_L} + \frac{r}{s}\,\dot{\phi_R} \end{cases} \tag{2.17}$$

The main aim is to design a PID controller to push the mobile robot to track a desired trajectory.

We take the errors as the following :

$$\begin{cases} e_x = x_d - x \\ e_y = y_d - y \\ e_\theta = \theta_d - \theta \end{cases} \tag{2.18}$$

Where X_d, y_d and θ_d are the desired position and orientation.

After all, there are two task need to controlled which are the position and orietation.

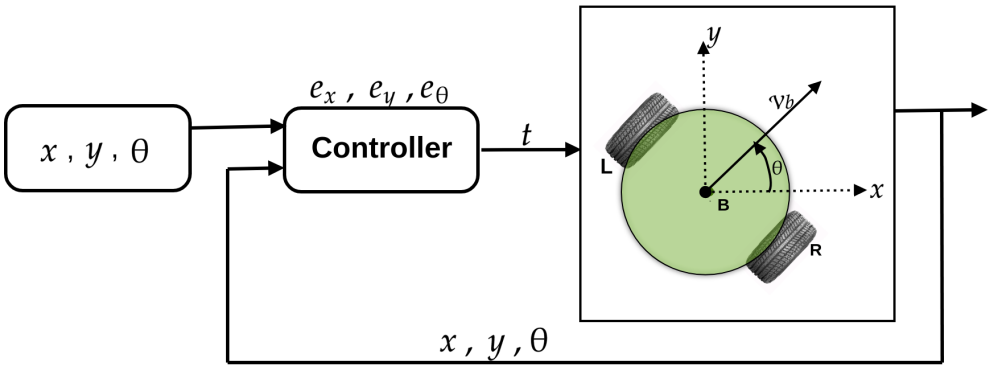

Figure 2.5: Closed-loop wheeled mobile robot system.

For Position control we can write :

$$\begin{cases} V = K_p^x e_x + K_i^x \int e_x dt + K_d^x \frac{de_x}{dt} \\ \omega = K_p^y e_y + K_i^y \int e_y dt + K_d^y \frac{de_y}{dt} \end{cases} \tag{2.19}$$

For orientation control we get:

$$\omega = k_p^\sigma e_\sigma + K_i^\sigma \int e dt + k_d^J \frac{de_\sigma}{dt} \tag{2.20}$$

from eqs 2.18, 2.19 and 2.20 we can write :

$$\begin{aligned} v &= \frac{r}{2}\left(\dot{\phi}_L + \dot{\phi}_R\right) \\ \omega &= \frac{r}{s}\left(\dot{\phi}_R - \dot{\phi}_L\right) \end{aligned} \tag{2.21}$$

Where $\dot{\phi}_L$ and $\dot{\phi}_R$ are given by :

$$\begin{aligned} \dot{\phi}_L &= \frac{2V - s\omega}{2r} \\ \dot{\phi}_R &= \frac{2V + s\omega}{2r} \end{aligned} \tag{2.22}$$

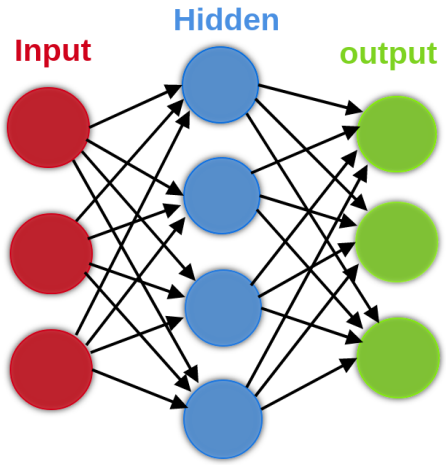

Figure 2.6: General scheme of neural

2.6 Neural prediction controllers

Neural prediction controllers utilize artificial neural networks to optimize control actions for car-like vehicles, ensuring precise path following and improved navigation accuracy. Various techniques have been proposed for such applications in the literature[15]. We propose employing the Residual Network (ResNet) algorithm for this application, given its popularity and outstanding performance in the field of image processing.

2.6.1 Resnet18 :

Definition

ResNet-18 (Residual Network) is a deep convolutional neural network architecture that consists of 18 layers, including convolutional, pooling, and fully connected layers. It is known for its innovative use of residual blocks and skip connections, which enable effective training of deep networks and alleviate the vanishing gradient problem. ResNet-18 has achieved remarkable performance in image classification tasks, demonstrating its effectiveness in extracting meaningful features and achieving high accuracy.

The reason for using it

We use ResNet-18 in our project due to its ability to handle complex visual perception

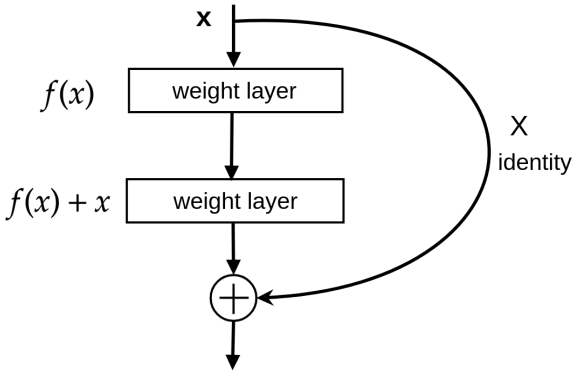

Figure 2.7: skip connection

tasks efficiently and accurately, enabling the robot to understand and navigate its environment effectively. This makes it a suitable choice for mobile robots with limited computational resources.

Architecture

ResNet-18 addresses two key challenges: skip connections and the use of max pooling:

- **The vanishing gradient :** the solution is Skip connections, also known as shortcut connections, are a fundamental component of ResNet-18. They allow the network to learn residual functions, capturing the difference between the desired output and the current output. By incorporating skip connections, ResNet-18 mitigates the problem of vanishing gradients, enabling effective training of deep networks[16]. These connections provide alternative paths for gradient flow, helping to preserve and propagate important information throughout the network.

- **Max Pooling:** Max pooling is a common down sampling technique used in convolutional neural networks. However, it can lead to a loss of spatial information and potentially compromise the network's ability to precisely localize objects. ResNet-18 addresses this issue by strategically placing skip connections that bypass the pooling layers. By preserving the spatial information through skip connections, ResNet-18 retains finer-grained details and improves the network's ability to localize objects accurately, enhancing its overall performance in tasks such as object recognition and classification.

2.7 Reinforcement learning :

Reinforcement learning is a branch of artificial intelligence concerned with developing methods and models to enable automated systems to make intelligent decisions through their interaction with their environment. Reinforcement learning is based on the concept of reward and punishment, where the system seeks to maximize reward and avoid punishment by experiencing and evaluating a set of behaviors. The goal of the reinforcement learning process is to identify the behavior that leads to the best long-term results[17]. Reinforcement learning techniques are used in a wide range of applications including robotics, video games, financial trading, production industries, and others, where they can achieve intelligent adaptation and informed decision-making in changing and unpredictable environments.

2.8 Conclusion

In this chapter, we did the mathematical model of mobile robot featuring a differential drive and two-wheel configuration.We then used the Classical controller PID for the path-following task.To expand our horizons further, we set out to explore various technologies that pave the way for achieving highly accurate and efficient path tracing such as MPC, LQR and LQG, which offer tantalizing possibilities for achieving precise path tracking. Finally, we used on neural prediction controllers, with a particular focus on the ResNet18 (residual network) model. This model is the latest technology for image recognition and computer vision tasks. It is extracted prominently It extracts salient features from visual data, enabling the robot to make informed decisions and interact with its surrounding environment.

Chapter 3

Driving the mobile robot

3.1 Introduction

In this chapter, we delve into the construction of our mobile robot and delve into the intricacies of data collection and tracking algorithms, which are paramount for achieving accurate path tracing. A major emphasis is placed on the sensor system, which plays a pivotal role in gathering data and monitoring the robot's trajectory. Through seamless integration of the tracking algorithm, we unlock the potential for precise path tracing, allowing our mobile robot to navigate with utmost precision and efficiency.

3.2 Robot Devices

The robot is composed of a variety of meticulously chosen components. These components include:

- **JetBot ROS :** is an open-source robotics platform that integrates NVIDIA JetBot hardware with the Robot Operating System (ROS) for advanced AI-powered robotics applications.

- **IMU** : The IMU (Inertial Measurement Unit) plays a vital role in a mobile robot by providing real-time measurements of orientation, acceleration, and angular velocity, enabling tasks such as navigation, control, and motion estimation.

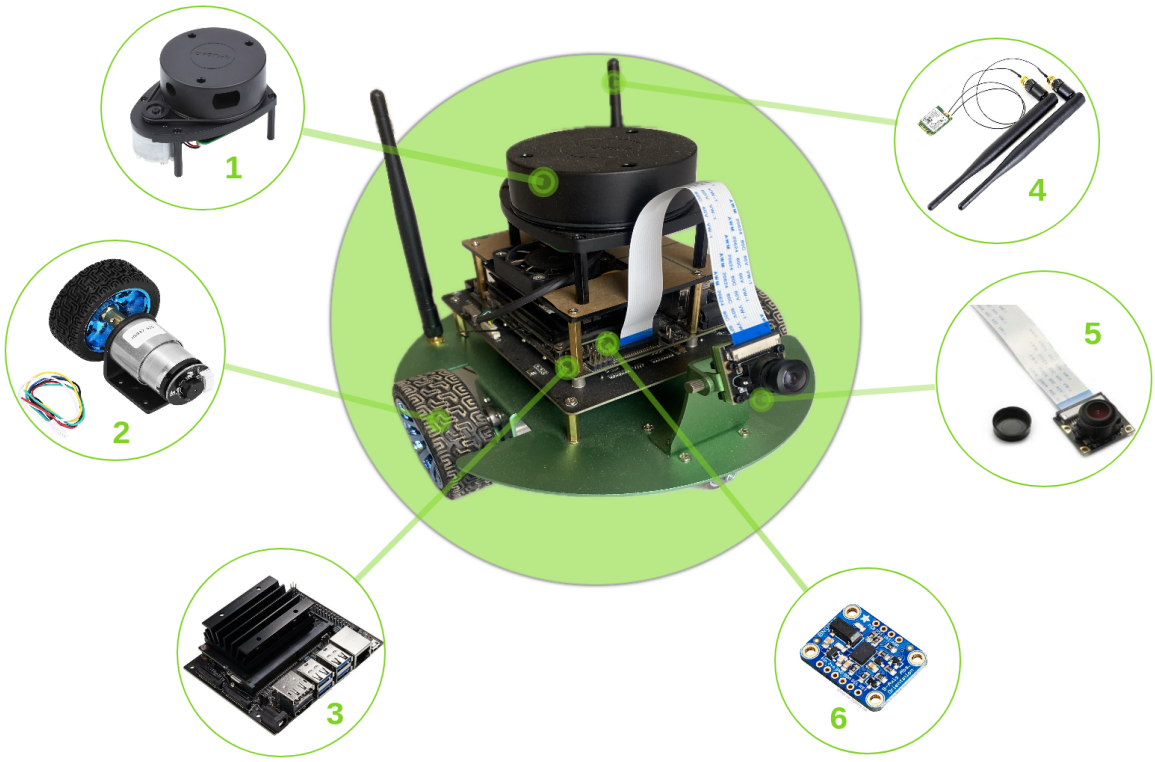

Figure 3.1: Devices of robot.

- **RPLIDAR A1 :**is an affordable 2D laser scanner that offers 360-degree environmental scanning with reliable distance measurement for robotics, mapping, and navigation applications.

- **High power encoder motor:** is a robust motor equipped with an encoder for precise control of speed and position, ideal for applications requiring high torque and accurate motion feedback in robotics and industrial automation.

- **The Wireless AC8265 with antennas:** is a high-performance Wi-Fi adapter that supports dual-band 802.11ac, offering fast and reliable wireless connectivity with enhanced range and stability for various devices and applications.

- **He IMX219-160 camera:** is a high-quality, wide-angle 8-megapixel camera module designed for use with devices like the Raspberry Pi, offering a 160-degree field of view for capturing detailed images and videos.

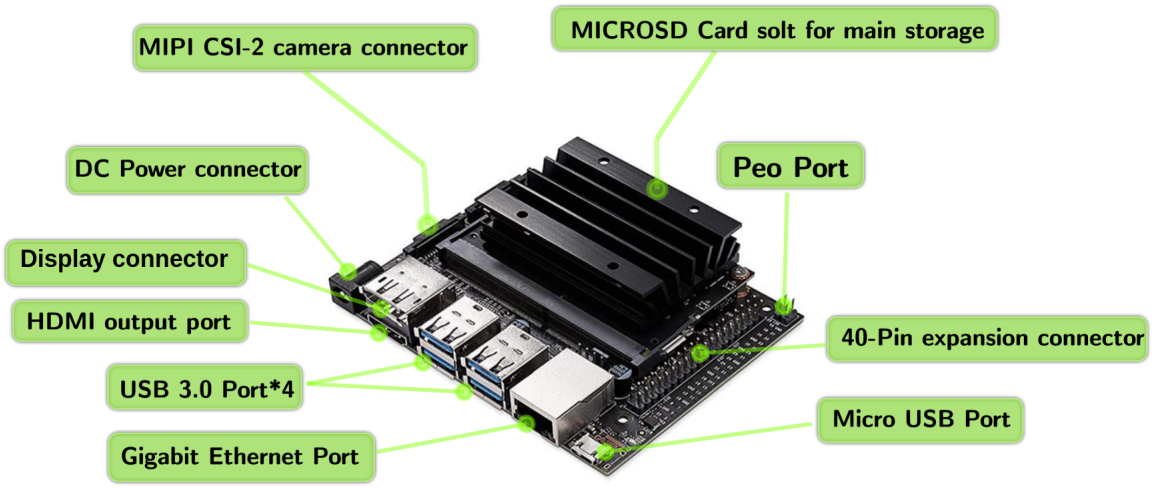

Figure 3.2: jetson nano

3.3 The NVIDIA Jetson Nano module:

The NVIDIA Jetson Nano module plays a critical role in our project by providing high-performance computing power for AI and computer vision applications, enabling tasks such as object detection, localization, and autonomous navigation. Its small size, GPU-accelerated processing, and support for various sensors and peripherals make it an essential component for developing intelligent and capable mobile robotic systems[18].

The key features of the NVIDIA Jetson Nano Developer Kit are present in figure 3.2.

3.4 First application :

3.4.1 Controlling Robot Motion Using Keyboard Input

The initial code for our project enables us to control the robot using a keypad, creating a direct and responsive communication channel for efficient operation. This keypad-controlled mechanism allows for fluid and precise robot management. To begin, we need to import the essential Robot class from the robot package, which offers convenient motor control capabilities.

Import the Robot class from the jetbot module:

```
from jetbot import Robot
```

Upon successfully importing the Robot class, we may then proceed to initialize an instance of this class utilizing the syntax provided below.:

```
robot = Robot()
```

3.4.2 Robot Manipulation through Command Inputs

After creating our robot instance, we can now use this instance to control the robot. To achieve a rotation of 40 of the robot's maximum speed :

```
robot.left(speed=0.4)
```

In order to stop the robot, we use this instruction:

```
robot.stop()
```

3.4.3 Link motors to traitlets

Generally this part of the code is used the ipywidgets library to create two vertical sliders, namely "left" and "right," with a range of -1.0 to 1.0 and a step size of 0.01,These sliders are then placed within a horizontal box container.The container is displayed in the output of the current cell, providing a user-friendly means of controlling and adjusting the respective "left" and "right" values.

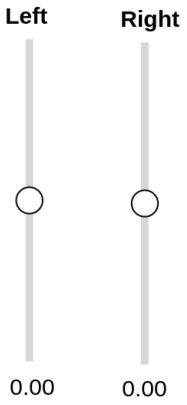

Figure 3.3: The sliders

```
import ipywidgets.widgets as widgets.
from IPython.display import display.
import ipywidgets.widgets as widgets
from IPython.display import display
#create two sliders with range [-1.0, 1.0]
left slider = widgets.FloatSlider(description='left', min=-1.0,
max=1.0,step=0.01,orientation='vertical')
right slider = widgets.FloatSlider(description='right', min=-1.0,
max=1.0, step=0.01, orientation='vertical')
#Create a horizontal box container to place the sliders next to each other
slider container = widgets.HBox([left slider, right slider])
#display the container in this cell's output
display(slider container)
```

Until we establish the connection to the drivers, these sliders will remain unusable. So we'll do the link function from the theme package.

```
import traitlets
left link = traitlets.link((left slider, 'value'), (robot.left motor,'value'))
right link = traitlets.link((right slider, 'value'), (robot.right motor,'value'))
```

By establishing the bidirectional connection through our implemented linking function, any modifications to the motor values from an external source will be reflected in real-time updates of the sliders. Executing the provided code block will allow us to witness this dynamic behavior in action.

```
robot.forward(0.3)
time.sleep(1.0)
robot.stop()
left link.unlink()
right link.unlink()
```

3.4.4 Attach functions to events

In this code, a set of buttons are created to enable control of the robot's movement. The buttons include Stop, Forward, Back, Left, and Right, each with its own design. These buttons are then displayed in a visually organized manner, allowing us to interact with them intuitively and control the robot's actions.

```
# create buttons
button layout = widgets.Layout(width='100px', height='80px',
align self='center')
stop button = widgets.Button(description='stop', button style='danger',
layout=button layout)
forward button = widgets.Button(description='forward',
layout=button layout)
backward button = widgets.Button(description='backward',
layout=button layout)
left button = widgets.Button(description='left', layout=button layout)
```

```
right button = widgets.Button(description='right',
layout=button layout)
# display buttons
middle box = widgets.HBox([left button, stop button,
right button],layout=widgets.Layout(align self='center'))
controls box = widgets.VBox([forward button, middle box,
backward button])
display(controls box)
```

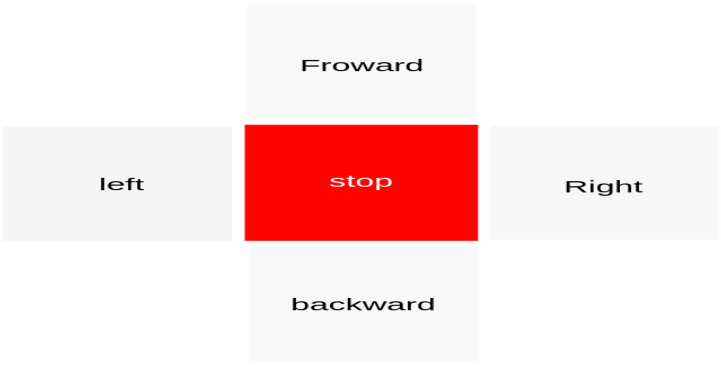

Figure 3.4: The keypad buttons

This code defines functions that allow us to precisely control the robot's movement, enabling it to stop, move forward or backward, and turn left or right for a specified period before stopping.

```
def stop(change):
robot.stop()
def step
forward(change):
robot.forward(0.4)
time.sleep(0.5)
robot.stop()
```

```
def step backward(change):
robot.backward(0.4)
time.sleep(0.5)
robot.stop()
def step left(change):
robot.left(0.3)
time.sleep(0.5)
robot.stop()
def step right(change):
robot.right(0.3)
time.sleep(0.5)
robot.stop()
```

Now we assign specific functions to click events for each button, and create the necessary event handlers for their click actions.

```
stop button.on click(stop)
forward button.on click(step forward)
backward button.on click(step backward)
left button.on click(step left)
right button.on click(step right)
```

3.4.5 Driving the mobile robot via joystick

In this section, we consider using a joystick to generate reference values for the mobile robot. The primary reason for choosing the joystick is to test and evaluate the performance of the PID controller. Unlike a keyboard, which generates rapidly changing reference values, the joystick provides a more nuanced control input, allowing for a more accurate assessment of the PID controller's capabilities.

Experimental Results and Discussion

From Figure 3.6, we can observe that the robot has two reference values for the left and right motors. The primary objective of the PID controller is to ensure that the robot tracks these reference values for the left set ($lset$) and the right set ($rset$). Additionally, Figure 3.6 highlights the following main paths:

- Strength Path [$334.5s < t < 335.4s$ and $336.2s < t < 336.7s$]: In this scenario, both motors have identical set values, maintaining a speed of 29 cm/s. The PID controller's role here is to ensure that both motors adhere to these set values. Experimental results demonstrate that the PID controller performs well in terms of settling time, overshoot, and response stability.

- Right Drift Path [$335.4s < t < 336.2s$]: In this case, the robot receives an input to take a right drift path. Consequently, the set values for the two motors differ, with the right motor increasing its speed to 38 cm/s and the left motor reducing its speed to 19 cm/s. The PID controller effectively manages the tracking control

Figure 3.5: driving mobile robot using joy stick.

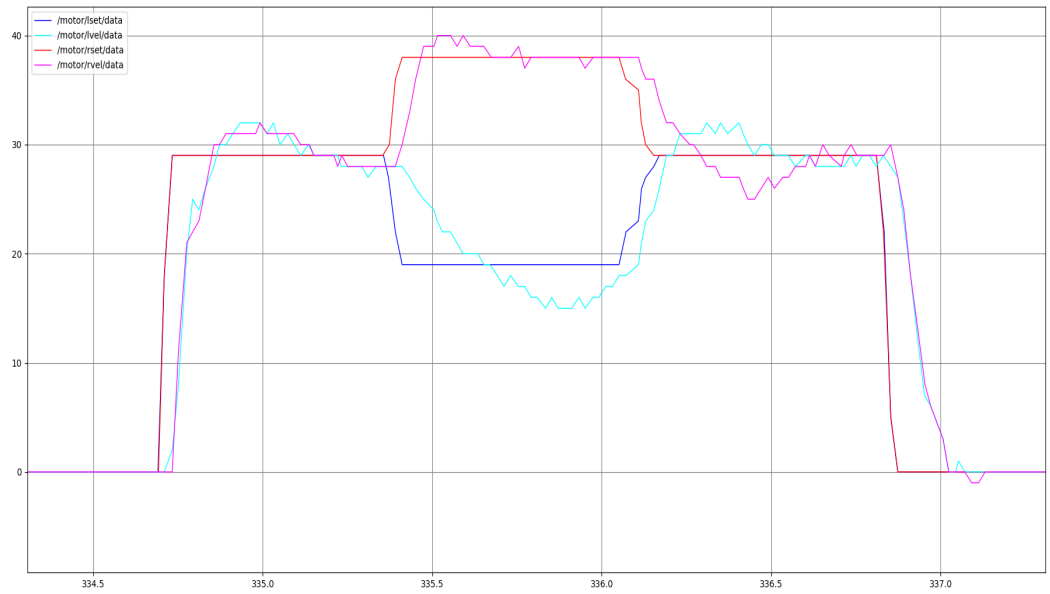

Figure 3.6: Experimental responses for different paths tracked by the robot using PID controller.

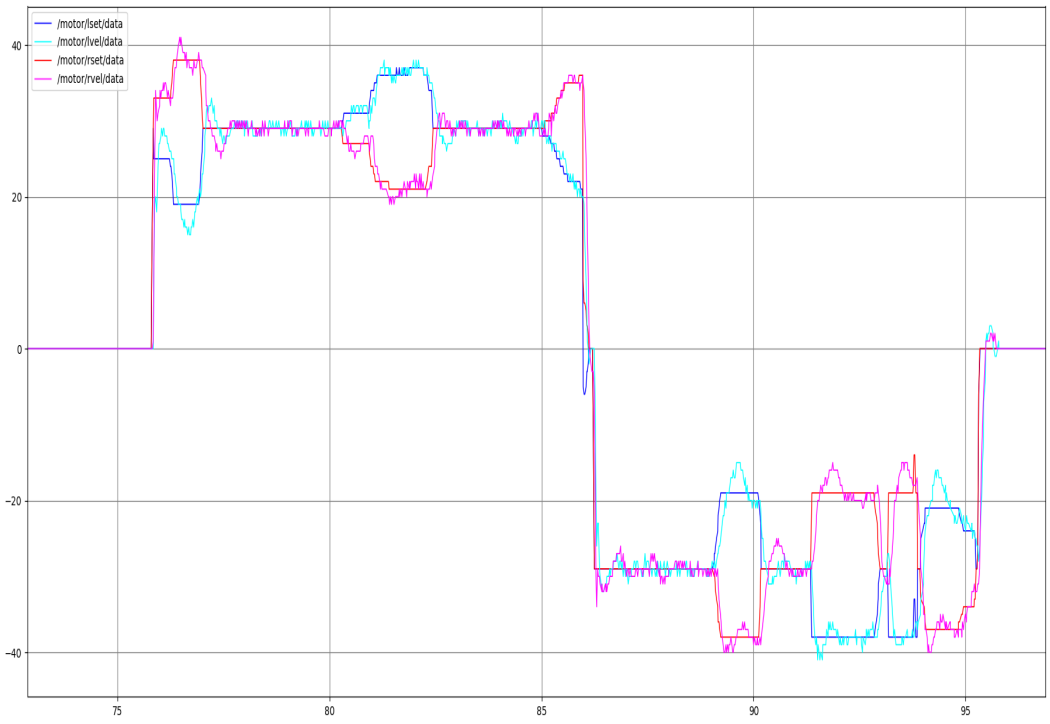

Figure 3.7: Experimental responses for different paths tracked by the robot using PID controller.

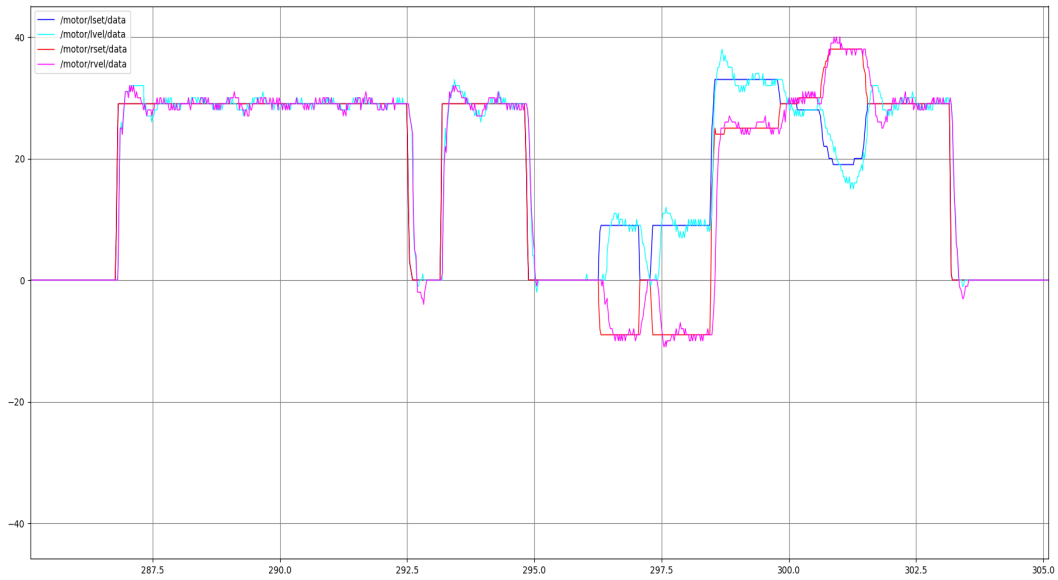

Figure 3.8: Experimental responses for different paths tracked by the robot using PID controller.

for this path, maintaining good performance in terms of settling time, overshoot, and response stability.

For other scenarios, such as backward and left paths, the obtained results are represented in the following figures:

Figure 3.7 ($86s < t < 89s$): In this timeframe, the robot follows its reference value for the backward path. Both motors operate at the same negative speed value (-30 cm/s), indicating a backward movement.

Figures 3.7 and 3.8: These figures illustrate the robot's performance in various directions. The PID controller effectively ensures that the mobile robot tracks the set paths with commendable robustness, particularly in terms of settling time, overshoot, and ripples.

3.5 Road Following using AI technique

After the previous application, which allowed us to understand the main behavior of our system and design a robust classical controller (PID) to track the reference value, we move on to the road-following application. Here, we use a camera to generate optimal

paths for the mobile robot. To achieve this challenge , we employ image processing based on neural network algorithm. This application is based on three key steps, which are presented hereafter:

3.5.1 Data collection

Our goal is to enable the robot to navigate autonomously and follow a pre-defined path. Using a single camera sensor and deep learning techniques and by leveraging the power of the NVIDIA Jetson Nano platform, the robot can efficiently process visual data and use a neural network to accurately recognize and interpret the road ahead. This approach ensures smooth path recognition without any disturbances or inconsistencies, which ultimately facilitates direct and efficient implementation. The way we will do this is as follows:

First we will imports necessary libraries and modules for creating an interactive widget-based interface to display a camera feed and capture snapshots using the JetBot robot platform.

```
import ipywidgets
import traitlets
import ipywidgets.widgets as widgets
from IPython.display import display
from jetbot import Robot, Camera, bgr8_to_jpeg
from uuid import uuid1
import os
import json
import glob
import datetime
import numpy as np
import cv2
import time
from jupyter_clickable_image_widget import ClickableImageWidget
```

Secondly we set the directory path using DATASET DIR= dataset- XY in which the data set will be stored or retrieved.

```
DATASET DIR = 'dataset xy'
```

Next, we proceed with the setup of essential elements for capturing and storing snapshots from the camera feed. This involves tasks such as directory creation, camera initialization, widget creation for image display, counting existing image files, and defining a function to save snapshots based on click events and coordinates. All these steps are comprehensively explained within this section of the code. try:

```
os.makedirs(DATASET DIR)
except FileExistsError:
print('Directories not created because they already exist')
camera = Camera()
camera widget = clickableimagewidget(width=camera.width,height=camera.height)
snapshot widget= ipywidgets.image(width=camera.width,height=camera.height)
traitlets.dlink((camera, 'value'), (camera widget, 'value'), transform=bgr8 to
jpeg)
count widget= ipywidgets.IntText(description='count')
count widget.value = len(glob.glob(os.path.join(DATASET DIR, '*.jpg')))
```

```
def save snapshot( , content, msg):
if content['event'] == 'click':
data = content['eventData']
x = data['offsetX']
y = data['offsetY']
uuid = 'xy image path = os.path.join(DATASET DIR, uuid + '.jpg')
with open(image path, 'wb') as f:
f.write(camera widget.value)
```

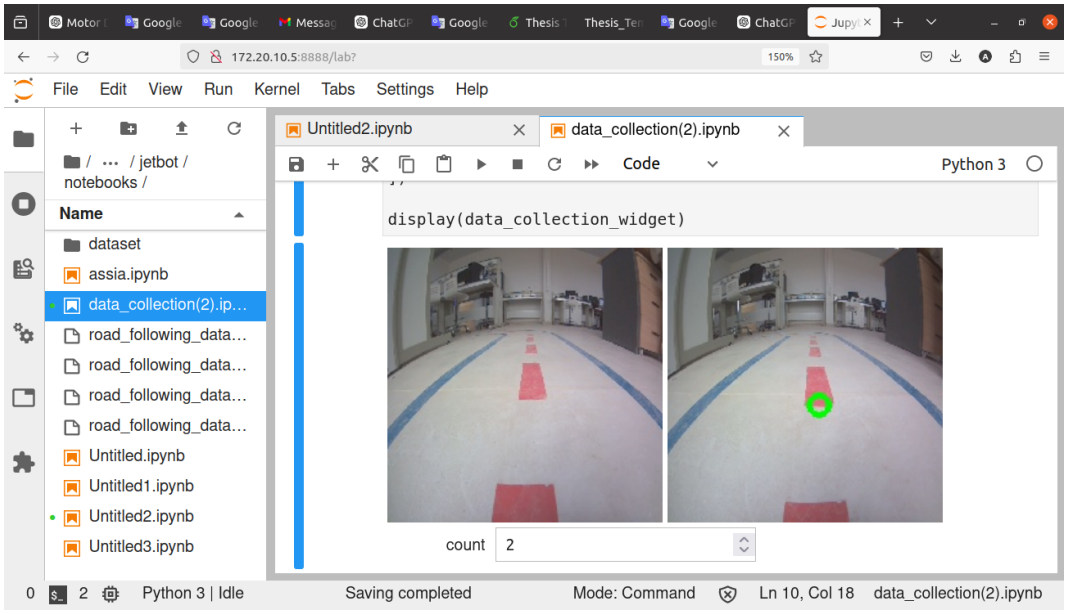

Figure 3.9: Samples from the dataset folder

Finally,This code displays the saved snapshot by updating the snapshot widget with a highlighted circle at the clicked coordinates, while also updating the count widget to reflect the updated number of saved images, providing a visual representation of the data collection process. The code also establishes the necessary event handling and widget organization for a user-friendly data collection interface.

```
# display saved snapshot
snapshot = camera.value.copy()
snapshot = cv2.circle(snapshot, (x, y), 8, (0, 255, 0), 3)
snapshot widget.value = bgr8 to jpeg(snapshot)
count widget.value = len(glob.glob(os.path.join(DATASET DIR, '*.jpg')))
camera widget.on msg(save snapshot)
data collection widget = ipywidgets.VBox([ ipywidgets.HBox([camera widget,
snapshot widget]), count widget ])
display(data collection widget).
```

3.5.2 Training Model (ResNet18)

In this section, we will train our image classifier. Where the best model will be saved Throughout the training process. To achieve this, we will use ResNet18, a widely known convolutional neural network architecture known for its effectiveness in image classification tasks. In addition, we will use the popular deep learning library PyTorch. The following code snippet illustrates our approach:

```
import torch
import torch.optim as optim
import torch.nn.functional as F
import torchvision
import torchvision.datasets as datasets
import torchvision.models as models
import torchvision.transforms as transforms
import glob
import PIL.Image
import os
import numpy as np
```

The code snippet begins by importing key modules from PyTorch and torchvision, facilitating core operations, optimization, neural network functions, dataset management, image transformations, file handling, and array manipulation. These imports enable seamless construction and training of neural networks, utilization of pre-trained models, and manipulation of images within the deep learning framework. This concise and comprehensive import statement optimizes code readability and maintainability, ensuring efficient development and deployment of computer vision applications.

```
def get x(path, width):
"Gets the x value from the image filename"
return (float(int(path.split()[1])) -width/2) / (width/2)
return (float(int(path.split()[1])) - width/2) / (width/2)
def get y(path, height):
"Gets the y value from the image filename"
return (float(int(path.split()[2])) - height/2) / (height/2)


,colframe=green!75!black] class XYDataset(torch.utils.data.Dataset):
def init (self, directory, random hflips=False):
self.directory = directory
self.random hflips = random hflips
self.image paths = glob.glob(os.path.join(self.directory, '.jpg'))
self.color jitter = transforms.ColorJitter(0.3, 0.3, 0.3, 0.3)
def len (self):
return len(self.image paths)
def getitem (self, idx):
image path = self.image paths[idx]
def getitem (self, idx):
image path = self.image paths[idx]
image = PIL.Image.open(image path)
width, height = image.size x =float(get x(os.path.basename(image path),
width))
y = float(get y(os.path.basename(image path), height))
if float(np.random.rand(1)) > 0.5:
image = transforms.functional.hflip(image)
x = -x
image = self.color jitter(image)
image = transforms.functional.resize(image, (224, 224))
image = transforms.functional.to tensor(image)
image = image.numpy()[::-1].copy()
```

```
image = torch.from numpy(image)
image = transforms.functional.normalize(image, [0.485, 0.456,
0.406], [0.229, 0.224, 0.225]) return image, torch.tensor([x, y]).float()
dataset = XYDataset('dataset xy', random hflips=False)
```

We then define utility functions get x and get y to extract coordinates from image file names, followed by a custom dataset class XYDataset to process the image data. It is configured with a directory path and an optional random horizontal flap. The class implements methods to retrieve the length of a dataset and its individual elements. Images are loaded, transformed, and transformed into tensors, while the corresponding coordinates are calculated and returned as a float-tensor. Finally, an XYDataset instance is created with a specified directory path and optional flip settings.

```
test percent = 0.1
num test = int(test percent * len(dataset))
train_dataset,test_dataset=torch.utils.data.random
split(dataset,[len(dataset)num test,num test])
train_loader=torch.utils.data.DataLoader(train_dataset,batch
size=8,shuffle=True,nm workers=0)
test_loader=torch.utils.data.DataLoader(test_dataset,batch
size=8,shuffle=True,numworkers=0)
```

These code segments define data loaders for both the training and testing datasets, setting parameters such as batch size, shuffle, and the number of workers for data loading. They enable efficient iteration over the datasets in batches during model training and evaluation, crucial for managing large-scale datasets in deep learning workflows.

```
model = models.resnet18(pretrained=True)
```

This line instantiates a ResNet-18 model architecture, initialized with pre-trained weights obtained from training on the ImageNet dataset. ResNet-18 is a widely-used convolutional neural network renowned for its effectiveness in various computer vision tasks. By utilizing pre-trained weights, the model benefits from learned features and parameters, enhancing its performance and enabling efficient transfer learning for downstream tasks.

```
model.fc = torch.nn.Linear(512, 2)
device = torch.device('cuda')
model = model.to(device)
```

In this section, the final fully connected layer of the model is replaced with a new linear layer having an output size of 2, which corresponds to the number of output classes or dimensions in the task. Next, the code specifies the device for computation as CUDA "Compute Unified Device Architecture", indicating the usage of a GPU for accelerated processing if available. Finally, the model is transferred to the selected device, ensuring that subsequent computations are performed efficiently on the GPU, if accessible, thereby enhancing performance during training and inference.

```
NUM EPOCHS = 70
BEST MODEL PATH = 'best steering model xy.pth'
best loss = 1e9
optimizer = optim.Adam(model.parameters())
for epoch in range(NUM EPOCHS):
model.train()
train loss = 0.0
for images, labels in iter(train loader):
images = images.to(device)
```

```
labels = labels.to(device)
optimizer.zero grad()
outputs = model(images)
loss = F.mse loss(outputs, labels)
train loss += float(loss)
loss.backward()
optimizer.step()
train loss /= len(train loader)
model.eval()
```

```
test loss = 0.0
for images, labels in iter(test loader):
images = images.to(device)
labels = labels.to(device)
outputs = model(images)
loss = F.mse loss(outputs, labels)
test loss += float(loss)
test loss /= len(test loader)
print('if test loss < best loss:
torch.save(model.state dict(), BEST MODEL PATH)
best loss = test loss
```

In this part, we coordinate the training loop of the model over a specified number of epochs, making use of the Adam optimizer for parameter optimization. During each epoch, the model is set to training mode, and the training data set is iterated to calculate and minimize the mean square error (MSE) loss between the model predictions and the ground truth labels. Likewise, the model is switched to evaluation mode to evaluate its performance on the test dataset, and calculate the test loss. The progress of the training and test losses is printed for each period, and if the test loss improves, the model state dictionary is saved. This iterative process aims to identify and retain

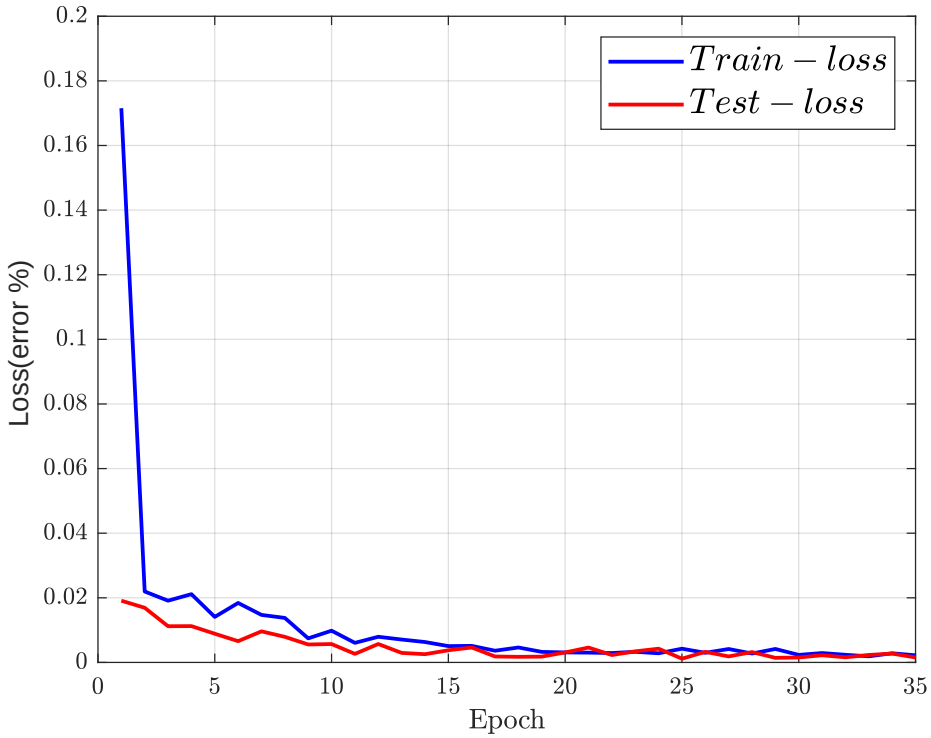

Figure 3.10: The training curve of Resnet18

the best performing model based on the test loss.

results:

Analyze ResNet-18 model curves by plotting training and validation losses against epochs to observe trends in convergence or divergence, indicating potential overfitting. Track training and validation accuracies to ensure model performance aligns with expectations. Use MATLAB's visualization tools to create clear, informative plots with epochs on the x-axis and metrics (loss or accuracy) on the y-axis. Monitor for signs of overfitting, such as increasing validation loss or stagnating validation accuracy compared to training metrics.

3.5.3 Live implementation

In this part, we will use the best model that we trained to move the robot smoothly on road

Load trained Model

First of all, we need to assume we have already downloaded the file "best_steering _model _xy.pth" to our workstation as instructed in the "train_model.ipynb" directory. Now, we need to upload this model file to the same directory on our robot . Once the upload is complete, we should see a file named "best_steering_model_xy.pth" in the directory.

For this initial part we need to execute the code below to initialize the PyTorch model.

```
import torchvision
import torch
model=torchvision.models.resnet18(pretrained=False)
model.fc=torch.nn.Linear(512, 2)
```

Next, load the trained weights from the best_steering_model_xy.pth file that we uploaded before.

```
model.load_state_dict(torch.load('best_steering_model_xy.pth'))
```

The model weights are currently stored in the CPU memory. Thereby, we need to execute the code below to transfer them to the GPU device.

```
device = torch.device('cuda')
model = model.to(device)
model = model.eval().half()
```

Pre-Processing function

We have now loaded our model, but there's a slight issue: the format we used to train our model doesn't match the camera's format. To fix this, we need to do some pre-processing, which involves the following steps:

- Convert from HWC layout to CHW layout.

- Normalize the data using the same parameters we used during training (our camera provides values in the [0, 255] range, and training loaded images in the [0, 1] range, so we need to scale by 255.0).
- Transfer the data from CPU memory to GPU memory.
- Add a batch dimension.

```
import torchvision.transforms as transforms
import torch.nn.functional as F
import cv2
import PIL.Image
import numpy as np
mean = torch.Tensor([0.485, 0.456, 0.406]).cuda().half()
std = torch.Tensor([0.229, 0.224, 0.225]).cuda().half()
def preprocess(image):
image = PIL.Image.fromarray(image)
image = transforms.functional.to_tensor(image).to(device).half()
image.sub_(mean[:, None, None]).div_(std[:, None, None])
return image[None, ...]
```

For the camera activation, execute the following code :

```
from IPython.display import display
import ipywidgets
import traitlets
from jetbot import Camera, bgr8_to_jpeg
camera = Camera()
image_widget = ipywidgets.Image() traitlets.dlink((camera, 'value'), (image_widget, 'value'), transform=bgr8_to_jpeg)
display(image_widget)
```

also here we need to import robot library to drive the motors.

```
from jetbot import Robot
robot= Robot()
```

Next, we'll create a function that triggers whenever the camera detects a change. This function will :

- Pre-process the camera image.
- Run the neural network.
- calculate the steering value.
- control the motors using PID controller.

```
angle = 0.0
angle_last = 0.0
def execute(change):
global angle, angle_last
image = change['new']
xy = model(preprocess(image)).detach().float().cpu().numpy().flatten()
x = xy[0]
y = (0.5 - xy[1]) / 2.0
xslider.value = x
y_slider.value = y
speed_slider.value = speed_gain_slider.value
angle = np.arctan2(x, y)
pid = angle * steering_gain_slider.value + (angle-angle) * steer-
ing_dgain_slider.value
angle_last = angle
steering_slider.value = pid + steering_bias_slider.value
execute('new': camera.value)
camera.observe(execute, names='value')
```

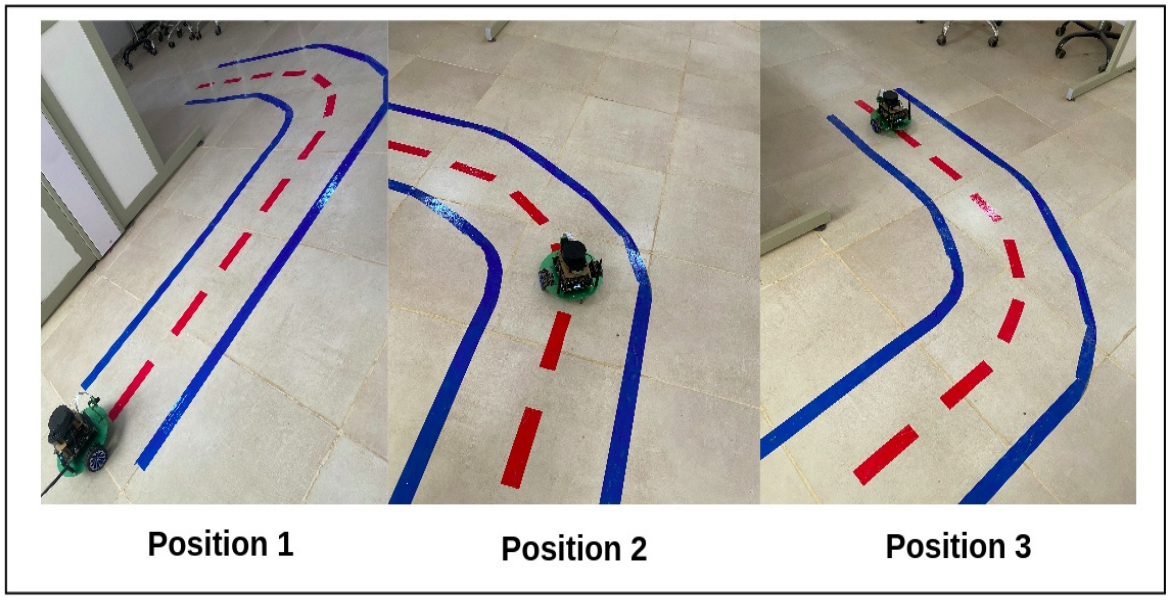

Figure 3.11: Experimental validation : the robot track the road

3.5.4 Experimental Results and discussion

After training our robot to follow a set path, we tested it at three different speeds: low, medium, and high. This was done to evaluate the performance of the PID controller. The differences observed in these tests are shown in the following figures.

From the figure 3.11 ,we can see that when we place our robot at the starting point (Position1), the neural network generates a reference value by comparing the real environment captured by the camera to the desired path. After this, the PID controller takes over, adjusting the two motors to follow these reference values ($rset$ and $lset$) see Figures 3.12 ,3.13 and 3.14. To test the robustness of the PID controller combined with AI-generated reference values, we considered three different speeds. At low speed (25% of the maximum speed of the two motors), the PID controller showed good robustness and performance in tracking the reference values with respect good settling time and small overshoot (see figure 3.12)

Additionally, to evaluate the ability of the PID controller, we also tested a very curved path (see Figure 3.11, position 2) and increased the speed to 60%. The PID controller maintained good performance in terms of response time, overshoot, and ripples, allowing the robot to smoothly follow the path.

At speeds up to 80% (see figure 3.14), the PID controller still delivered good results

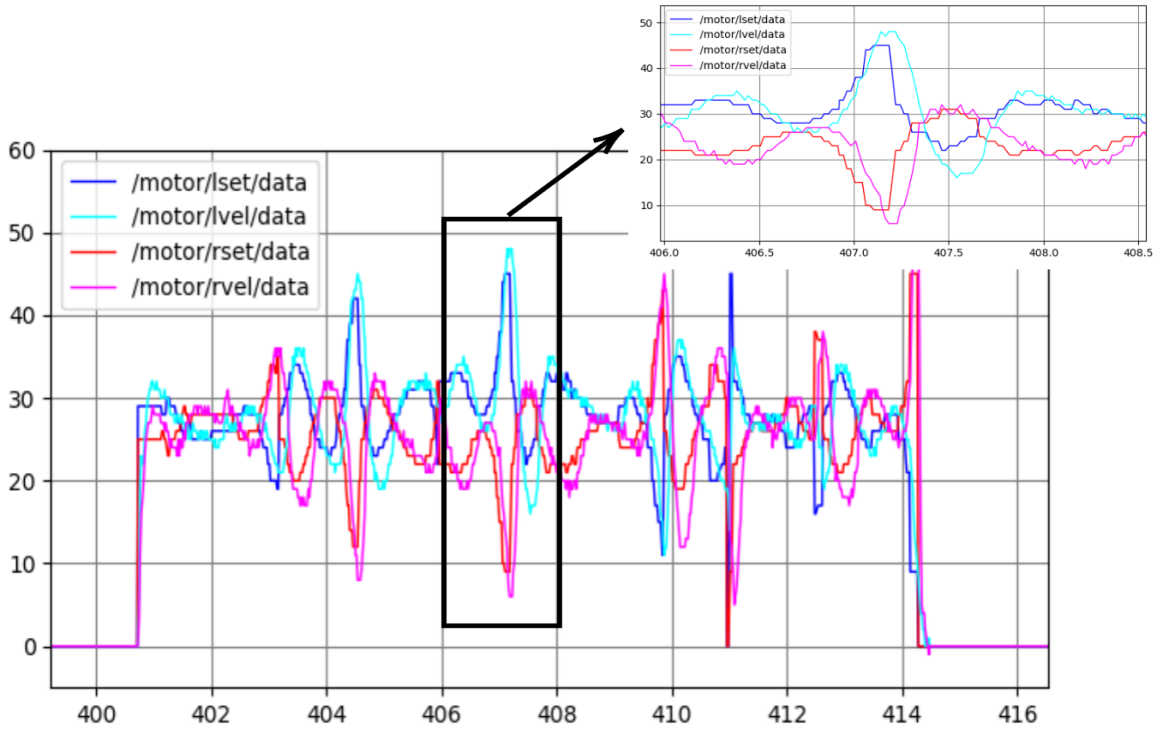

Figure 3.12: Experimental response of the robot following the road at 25% of speed

and performance, though with a noticeable steady-state error depending on the position (1, 2, or 3). This variation is due to the reference input and the nonlinear limitations of the system's behavior.

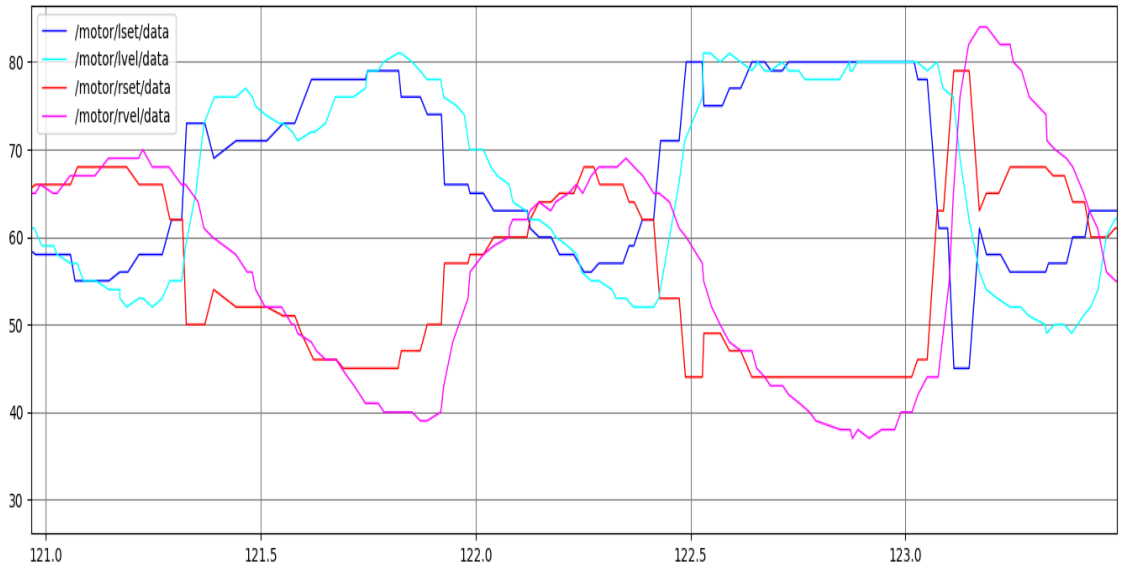

Figure 3.13: Experimental response of the robot following the road at 60% of speed

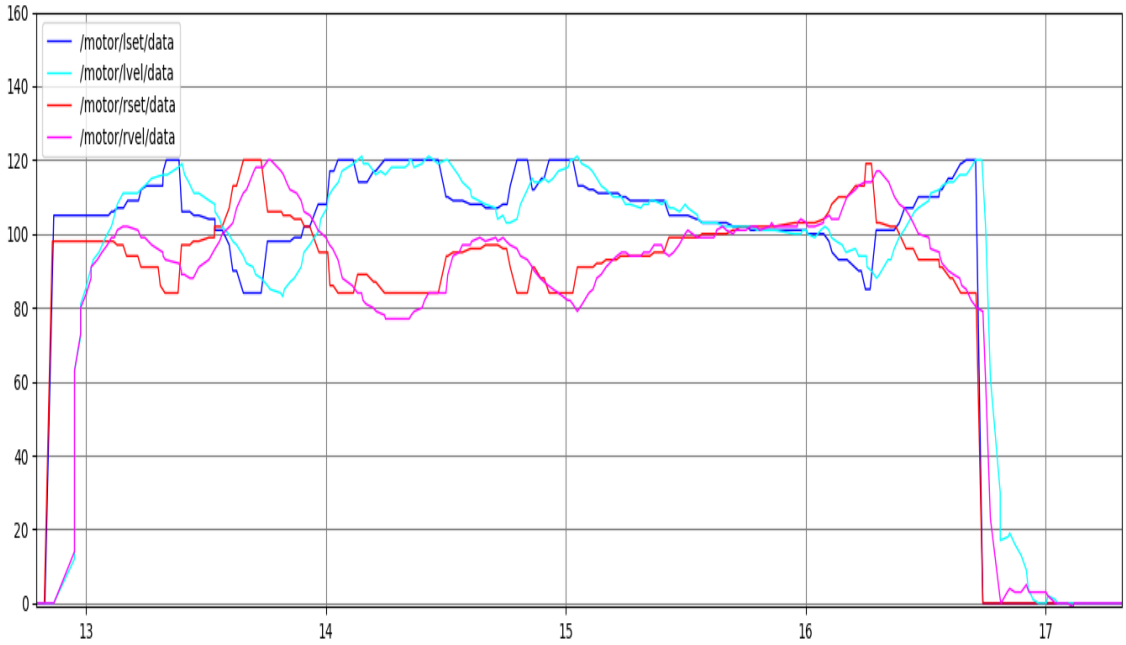

Figure 3.14: Experimental response of the robot following the road at 80% of speed

3.6 Conclusion

This chapter serves as a comprehensive guide to building a mobile robot, exploring its various components, and showcasing a range of applications focused on achieving precise and smooth control. Through initial use of the keyboard and joystick interfaces and analysis of the PID results, it becomes clear that the joystick achieves superior results compared to the keyboard. Furthermore, optimal robot training contributes to achieving accurate navigation capabilities. Furthermore, the integration of PID control proved to be indispensable in maintaining stability and achieving positive results throughout the robot's navigation journey, underscoring the importance of accurate pose estimation for reliable operation.

Chapter 4

Path following with Navigation

4.1 Introduction

In this application, we aim to enable our robot to navigate automatically in an unknown environment and follow the given instructions to complete its mission. In fact, to do this challenge task, To tackle this challenging task, we collaborate with colleagues working under the same supervisor on a project titled "Investigating and Implementing ROS-based Autonomous Navigation for Mobile Robots using SLAM Algorithms." Our colleagues, Nour Otmani and Makhloufi Maroua, have already completed the part of the project involving the creation of a 2-D map. After building the 2-D map, the PID controller, designed in a previous section, is used to guide the robot to navigate accurately and efficiently to the target position.

4.2 Additional equipment used for this application

4.2.1 RPLIDAR A1:

The RPLIDAR figure 4.1 is a 2D laser scanner that can perform a 360° scan within a 12 m range. Lidar plays a crucial role in mobile robot navigation by enabling environment mapping, obstacle detection and avoidance, localization, and path [19]. It allows robots to perceive their surroundings, create accurate maps, navigate safely, and adapt their trajectories in real-time to ensure efficient and reliable movement in

Figure 4.1: RPLIDAR A1

complex environments. LiDAR systems consist of the following main components:

- Laser source: emits laser beams to illuminate the environment.
- Scanner: Directs laser beams in different directions to cover a wide field of view.
- Photodetector: Receives and measures the intensity of laser rays reflected from objects.
- Timing electronics: precisely measures the time it takes laser beams to travel and return, allowing distance calculations.
- Optics: Focuses laser beams and collects reflected light to make precise distance measurements.
- Control and processing unit: Manages the Lidar system, synchronizes laser emission, processes signals from the photodetector, and performs data processing and analysis.

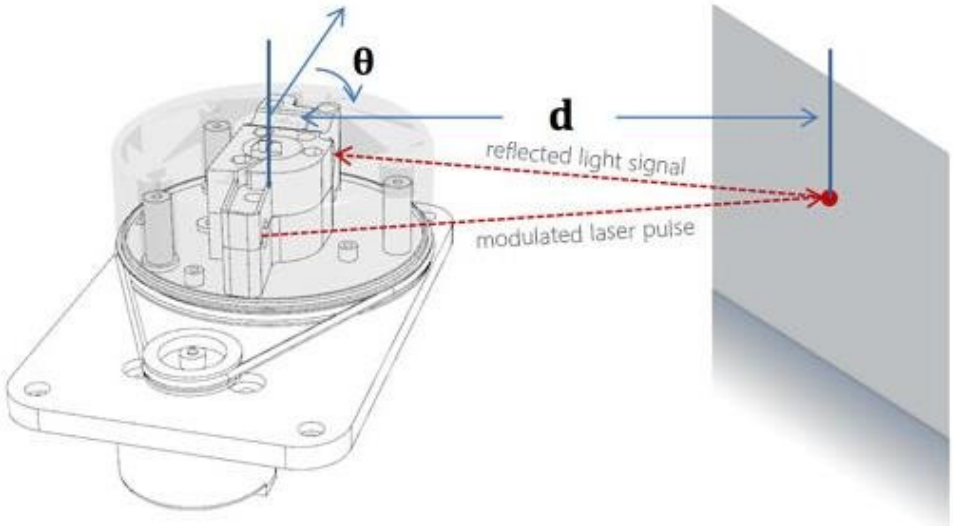

Figure 4.2: RPLIDAR ranging mechanism

4.2.2 Mechanism:

The RP LiDAR mechanism functions by emitting a modulated laser pulse towards an object. The pulse reflects off the object and returns to the LiDAR sensor. The sensor measures the time taken for the pulse to travel to the object and back, known as the time of flight. Using the speed of light, the distance (d) to the object is calculated. The LiDAR unit rotates, allowing it to scan the environment at various angles (θ). By combining the distance measurements with the corresponding angles, a 2D map of the surroundings is constructed, providing spatial information for applications like navigation and object detection.

4.3 Odometry :

Odometry in a mobile robot refers to the technique of estimating the robot's position and orientation by tracking its movement using sensors, typically wheel encoders. It calculates the distance and direction traveled by the robot based on the rotation of its wheels[20]. Odometry provides a local estimate of the robot's position, which can be used for navigation, mapping, and path planning purposes.

4.3.1 Odometry: Estimating Robot Movement and Position

Odometry: Estimating Robot Movement and Position

- Wheel Odometry: Measure wheel rotations using encoders to calculate distance traveled. Difference in distances between wheels determines change in orientation.
- Motion Odometry: Use accelerometers or gyroscopes to measure linear acceleration and angular velocity. Integration over time estimates robot's displacement and orientation.
- Dead Reckoning: Continuously update position and orientation using wheel or motion data. Integrate estimated distances and orientation changes for accurate pose estimation.
- Error Accumulation: Odometry is prone to cumulative errors and drift. Factors like wheel slippage, uneven terrain, or sensor noise introduce inaccuracies over time, affecting position and orientation estimation.

4.4 IMU:

IMU stands for Inertial Measurement Unit [21]. It is a sensor module that combines multiple sensors to measure various aspects of a system's motion, orientation, and environmental conditions. An IMU typically consists of three primary types of sensors:

- Accelerometer: An accelerometer measures linear acceleration along three axes (typically X, Y, and Z). It detects changes in velocity and can determine the direction and magnitude of acceleration.
- Gyroscope: A gyroscope measures angular velocity or rotational rate around each of the three axes. It provides information about the rate of rotation and changes in orientation.
- Magnetometer: A magnetometer measures the strength and direction of the magnetic field. It can be used to determine the orientation relative to the Earth's magnetic field.

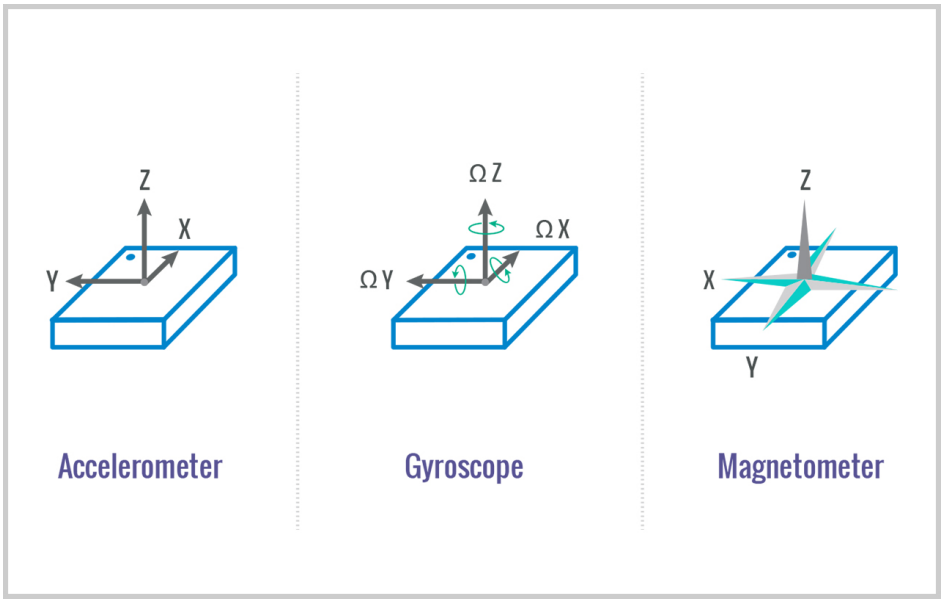

Figure 4.3: IMU Sensor Components

4.5 Path following using polynomials:

In mobile robotics, polynomial functions serve as essential tools for defining and navigating paths of varying complexity. These functions, including linear ($P(x) = a_1x+a_0$), quadratic ($P(x) = a_2x^2 + a_1x + a_0$), and cubic ($P(x) = a_3x^3 + a_2x^2 + a_1x + a_0$) polynomials, offer versatile representations of trajectories such as straight lines, parabolic arcs, and more intricate curves with inflection points. Each polynomial degree determines the path's smoothness and complexity, crucial for applications requiring precise trajectory following in diverse terrains and environments. For instance, a mobile robot equipped with sensors might navigate a quadratic path $P(x) = x^2$ while tracking a designated line, adjusting its movements dynamically based on sensor feedback to maintain alignment and follow the curvature of the path accurately. This approach not only enhances navigation efficiency but also ensures the robot can adapt swiftly to changes in its environment, utilizing mathematical precision to optimize its path-following capabilities in real-world scenarios.

Figure 4.4: Example of a 2-D map created using SLAM based on LiDAR

4.6 Bulid a 2-D Map using SLAM technique

The primary objective of building a 2D map using Simultaneous Localization and Mapping (SLAM) for a mobile robot is to enable autonomous navigation in unknown environments. SLAM techniques allow the robot to simultaneously localize itself within the environment and create a detailed map of its surroundings using onboard sensors like LIDAR . This capability is crucial for tasks such as exploration, surveillance, and logistics, where the robot needs to understand its environment in real-time without relying on pre-existing maps or external infrastructure. By generating accurate maps and continuously updating its position relative to these maps, SLAM empowers robots to navigate dynamically and adapt to changes in their surroundings, thereby enhancing their autonomy and operational capabilities in various applications.[19].

4.7 Practical Implementations :

The previously designed controller strategy is now practically applied to the mobile robot to follow the required paths. In this application, the closed-loop controller for

the two motors receives input from a 2-D map generated by the LiDAR-based SLAM technique (see figure 4.5).

Figure 4.5 shows the LTSS Laboratory environment as encountered by the robot and the map created during its movement along a direct path. This direct path helps the robot avoid collisions and conflicts, as seen in Figure 4.6. From Figure 4.6, we can observe the iterative nature of the mapping process with each movement of the robot.

To test the robustness of the control strategy, we conducted experiments with two paths: one with obstacles and one without (see Figures 4.7-4.10). The results indicate that the responses of both motors closely follow their reference values when using the PID controller. Additionally, we achieved excellent accuracy and clarity in the generated map. By following the direct path, the robot improves its ability to make informed decisions and interact effectively with its surroundings.

The sequence of figures effectively illustrates the progress of the mapping process. In Figure 4.6, we see the map displayed in RVIS, a visualization tool commonly used for mapping and navigation. This demonstrates that the mapping algorithm successfully generated an initial map based on the robot's initial movements.

Figure 4.5: LTSS laboratory with its 2-D map

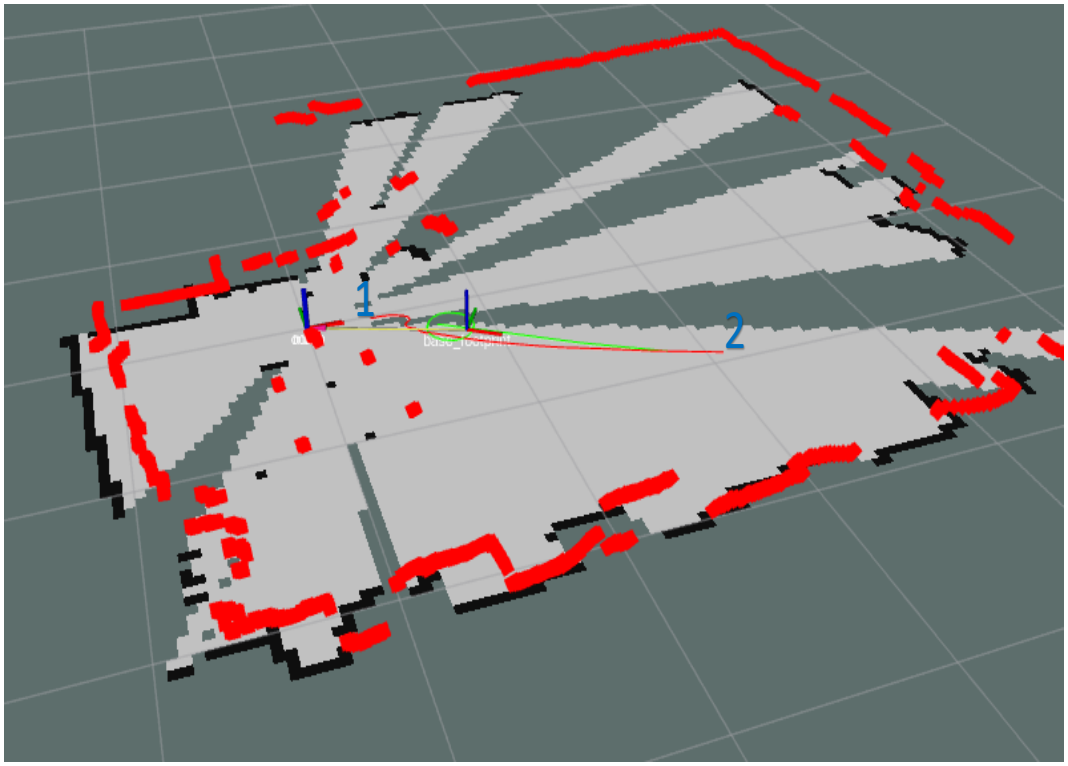

Figure 4.6: Start building a 2-D map and navigation.

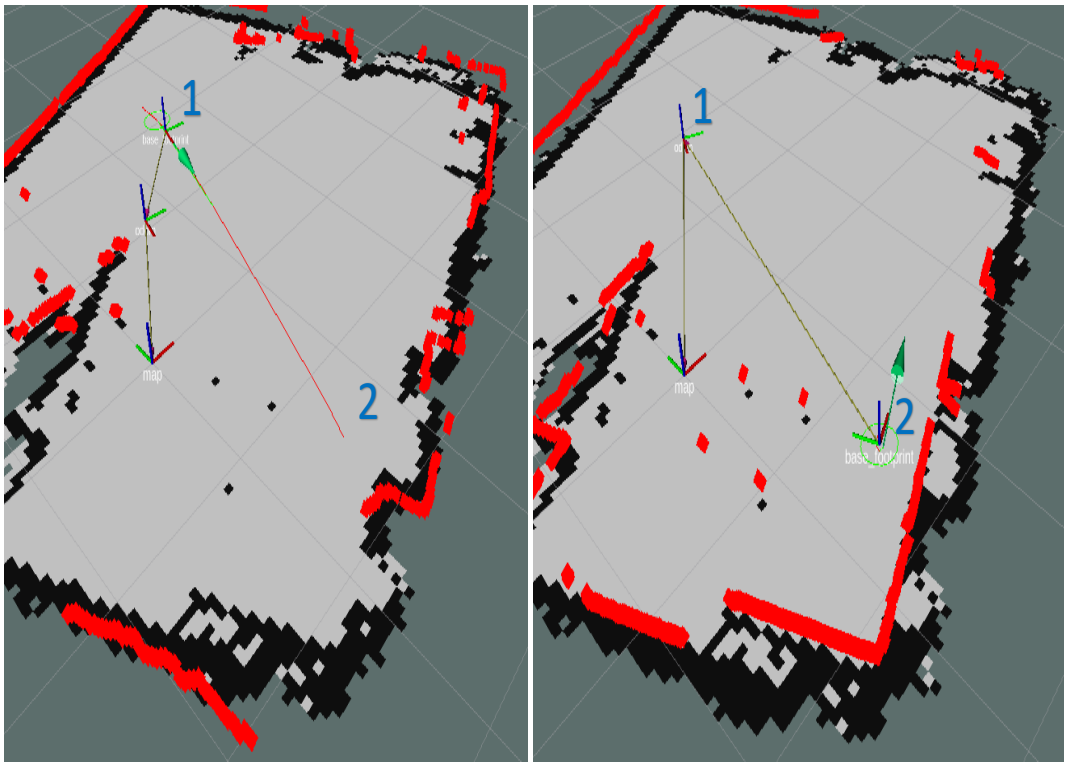

Figure 4.7: First path case without dynamic obstacle.

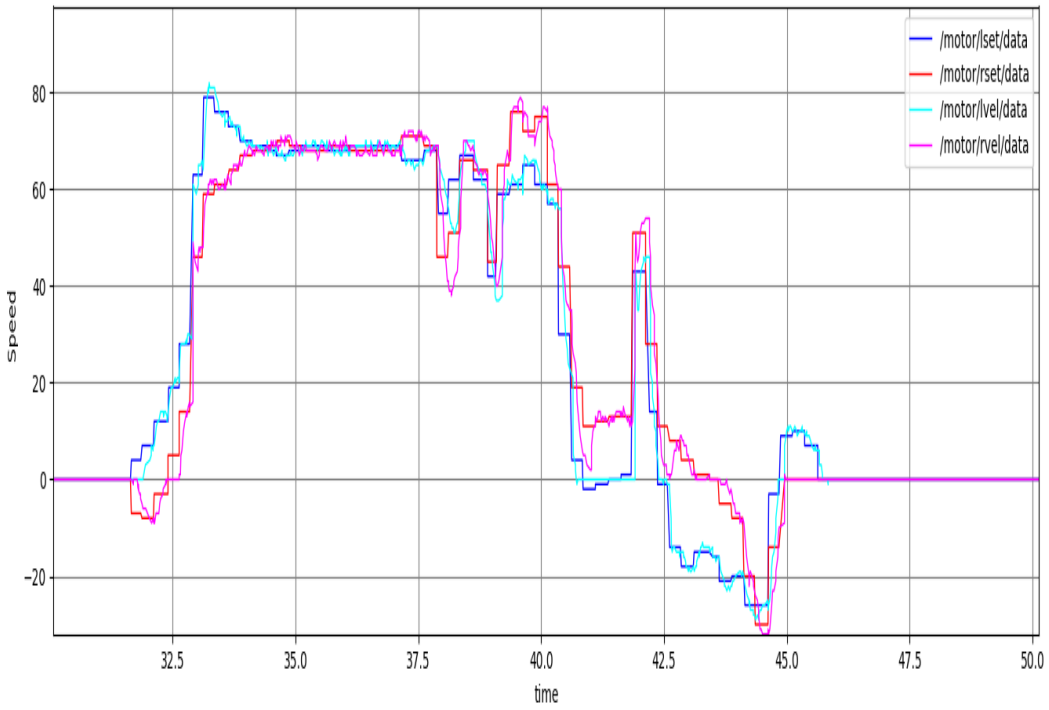

Figure 4.8: Different experimental responses of two motors using PID controller

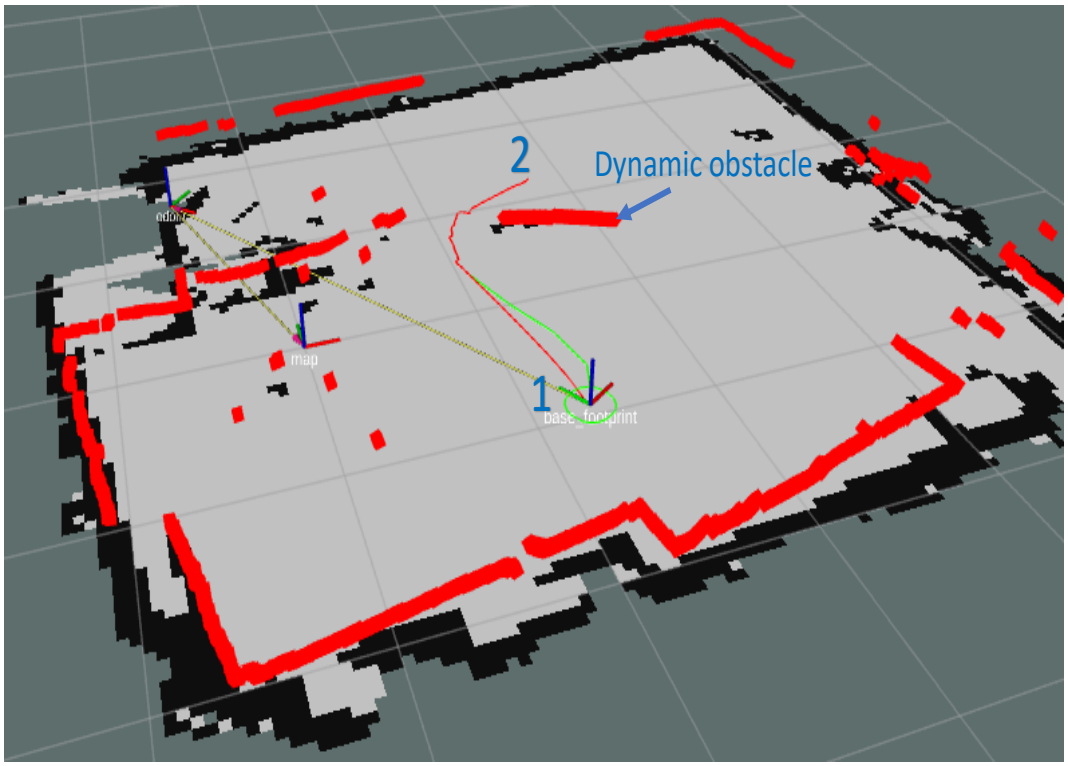

Figure 4.9: Second path case with dynamic obstacle

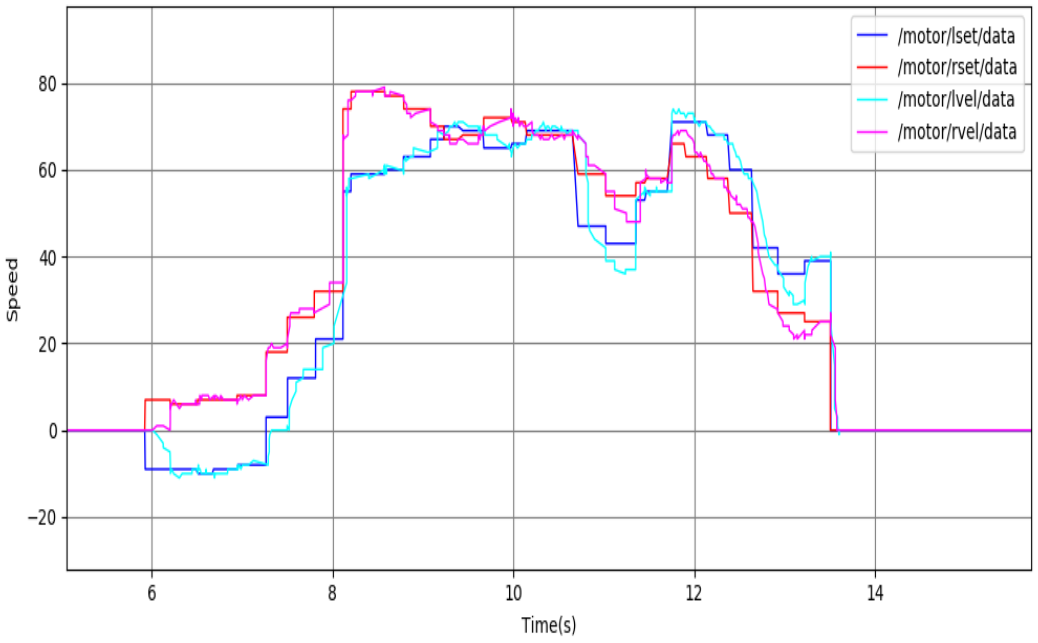

Figure 4.10: Different experimental responses of two motors using PID controller

4.8 Conclusion

In conclusion, this chapter highlighted the significance of path following with navigation using LiDAR and SLAM technology. By leveraging LiDAR sensors, the robot achieved accurate environmental perception, enabling it to detect obstacles and plan optimal paths. The integration of SLAM algorithms facilitated simultaneous mapping and localization, ensuring precise navigation without encountering issues. The use of LiDAR and SLAM proved instrumental in drawing detailed maps of the environment and tracking the robot's path accurately. This combination of technologies allowed the robot to adapt to dynamic surroundings, avoid collisions, and ensure efficient and safe navigation.

General Conclusion

The work presented in this master's thesis is dedicated to mobile robots for path tracking and navigation using a range of techniques.

In the first chapter, we discussed some generalizations about mobile robots, their role, and their applications in various fields. We learned about the levels of self-driving cars and the Architecteur of their systems. We discussed a range of AI technologies and touched on the ROS platform. We discussed its basic concepts that provide tools that help in building and developing robot programs and applications.

Then we showed the mathematical model of the two-wheeled mobile robot. We then used the PID controller, which allows the robot to move smoothly, and learned about neural prediction controllers, especially Resnet 18, which played an essential role in analyzing images to enable the robot to recognize the environment and make decisions. After that, we presented the most important components of the robot that allow it to move, such as: IMU, JetBot ROS, RPLIDAR, Wireless.... Then we implemented two applications for the keyboard and joystick based on specific algorithms. Then we took the data collection and training model to get the best model for the robot to follow the path.

In the last section, experimental results demonstrate the successful use of LIDAR and SLAM to enable the robot to perform simulations effectively. These technologies played a crucial role in mapping the environment, determining the robot's location, and ensuring accurate navigation.

In future work, we aim to explore additional techniques to enhance the mobile robot's capabilities, including using 3D maps to create a detailed representation of the environment, improving robot localization, obstacle detection, and path planning capabilities. Additionally, incorporating multiple cameras into the robot's sensor array will pro-

vide improved visual coverage and depth perception, enabling object recognition and tracking and better understanding of the scene. These developments will ultimately enhance the robot's capabilities to navigate and interact, leading to more efficient and intelligent mobile robot systems.

Bibliography

[1] Klaus Schilling and Christoph Jungius. Mobile robots for planetary exploration. Control Engineering Practice, 4(4):513–524, 1996.

[2] Gerald Cook and Feitian Zhang. Mobile robots: Navigation, control and sensing, surface robots and AUVs. John Wiley & Sons, 2020.

[3] David Filliat. Robotique mobile. PhD thesis, EDX, 2011.

[4] Nicu Bizon, Lucian Dascalescu, and Naser Mahdavi Tabatabaei. Autonomous vehicles: intelligent transport systems and smart technologies. (No Title), 2014.

[5] Mehmet Serdar Güzel, Mehmet Kara, and Mehmet Sıtkı Beyazkılıç. An adaptive framework for mobile robot navigation. Adaptive Behavior, 25(1):30–39, 2017.

[6] Jingke Wang, Yue Wang, Dongkun Zhang, Yezhou Yang, and Rong Xiong. Learning hierarchical behavior and motion planning for autonomous driving. In 2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pages 2235–2242. IEEE, 2020.

[7] Yufeng Yue and Danwei Wang. Collaborative Perception, Localization and Mapping for Autonomous Systems, volume 2. Springer Nature, 2020.

[8] Qiping Chen, Yinfei Xie, Shifeng Guo, Jie Bai, and Qiang Shu. Sensing system of environmental perception technologies for driverless vehicle: A review of state of the art and challenges. Sensors and Actuators A: Physical, 319:112566, 2021.

[9] Guillaume Bresson, Zayed Alsayed, Li Yu, and Sébastien Glaser. Simultaneous localization and mapping: A survey of current trends in autonomous driving. IEEE Transactions on Intelligent Vehicles, 2(3):194–220, 2017.

[10] ZHAO Jin and Abdelkader El Kamel. Integrated longitudinal and lateral control system design for autonomous vehicles. IFAC Proceedings Volumes, 42(19):496–501, 2009.

[11] Lionel Clavien, Michel Lauria, and François Michaud. Instantaneous centre of rotation based motion control for omnidirectional mobile robots with sidewards off-centred wheels. Robotics and Autonomous Systems, 106:58–68, 2018.

[12] George Gillard. An introduction to pid controllers, 2017.

[13] Carlos E Garcia, David M Prett, and Manfred Morari. Model predictive control: Theory and practice—a survey. Automatica, 25(3):335–348, 1989.

[14] Rupak Majumdar and Sadegh Soudjani. The computability of lqr and lqg control. In Proceedings of the 24th International Conference on Hybrid Systems: Computation and Control, pages 1–7, 2021.

[15] Nathan A Spielberg, Matthew Brown, Nitin R Kapania, John C Kegelman, and J Christian Gerdes. Neural network vehicle models for high-performance automated driving. Science robotics, 4(28):eaaw1975, 2019.

[16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In Proceedings of the IEEE conference on computer vision and pattern recognition, pages 770–778, 2016.

[17] Julian Ibarz, Jie Tan, Chelsea Finn, Mrinal Kalakrishnan, Peter Pastor, and Sergey Levine. How to train your robot with deep reinforcement learning: lessons we have learned. The International Journal of Robotics Research, 40(4-5):698–721, 2021.

[18] Gordeev Alexey, Vladimir Klyachin, Kurbanov Eldar, and Aleksandr Driaba. Autonomous mobile robot with ai based on jetson nano. In Proceedings of the Future Technologies Conference (FTC) 2020, Volume 1, pages 190–204. Springer, 2021.

[19] Shubham Nagla. 2d hector slam of indoor mobile robot using 2d lidar. In 2020 international conference on power, energy, control and transmission systems (ICPECTS), pages 1–4. IEEE, 2020.

[20] Kok Seng Chong and Lindsay Kleeman. Accurate odometry and error modelling for a mobile robot. In Proceedings of International Conference on Robotics and Automation, volume 4, pages 2783–2788. IEEE, 1997.

[21] Irem Toroslu and Mustafa Doğan. Effective sensor fusion of a mobile robot for slam implementation. In 2018 4th International Conference on Control, Automation and Robotics (ICCAR), pages 76–81. IEEE, 2018.