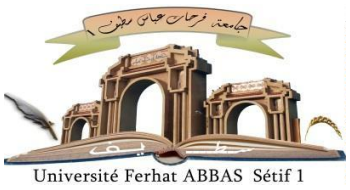
الجمهورية الجزائرية الديمقراطية الشعبية

People's Democratic Republic of algeria

Ministry of Higher Education and scientific Research



UNIVERSITY FERHAT ABBAS - SETIF 1

FACULTY OF TECHNOLOGY

THESIS

Submitted to the Department of Electronics

In Fulfillment of the Requirements for the degree of

DOCTORATE

Domain: Science and Technology

Field : Electronics Specialty : Instrumentation

By : **Mr. Medjaldi Amar**

THEME

Enhancing Vision-Based Navigation of Autonomous Guided Vehicles Through Deep Learning and Cloud Integration

Defended on **08 / 01 / 2026** in front of Jury :

Amardjia Nourredine	Professor	Univ. Ferhat Abbas Sétif 1	President
Nora Karkar	M.C.A.	Univ. Ferhat Abbas Sétif 1	Thesis director
Yacine Slimani	M.C.A.	Univ. Ferhat Abbas Sétif 1	Co-director
Foudil Abdessemed	Professor	Univ. Batna 2	Examiner
Bekkouche Toufik	Professor	Univ. M El Bachir El Ibrahimi	Examiner
Zerroug Nadjet	M.C.A.	Univ. Ferhat Abbas Sétif 1	Examiner
Khier Benmahammed	Professor	Univ. Ferhat Abbas Sétif 1	Guest

الجمهورية الجزائرية الديمقراطية الشعبية

République Algérienne Démocratique et Populaire

Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

UNIVERSITÉ FERHAT ABBAS - SÉTIF 1

FACULTÉ DE TECHNOLOGIE

THÈSE

Présentée au Département d'Électronique

En vue de l'obtention du diplôme de

DOCTORAT

Domaine : Sciences et Technologies

Filière : Électronique Spécialité : Instrumentation

Par : **M. Medjaldi Amar**

THÈME

Amélioration de la navigation basée sur la vision des véhicules autonomes guidés grâce à l'apprentissage profond et à l'intégration du cloud

Soutenue le **08 / 01 / 2026** devant le jury :

Amardjia Noureddine	Professeur	Univ. Ferhat Abbas Sétif 1	Président
Nora Karkar	M.C.A.	Univ. Ferhat Abbas Sétif 1	Directrice de thèse
Yacine Slimani	M.C.A.	Univ. Ferhat Abbas Sétif 1	Co-directeur
Foudil Abdessemed	Professeur	Univ. Batna 2	Examinateur
Bekkouche Toufik	Professeur	Univ. M El Bachir El Ibrahimi	Examinateur
Zerroug Nadjet	M.C.A.	Univ. Ferhat Abbas Sétif 1	Examinatrice
Khier Benmahammed	Professeur	Univ. Ferhat Abbas Sétif 1	invité

Dedication

This thesis is dedicated to my beloved ***family*** *and* ***friends****,*
whose love, patience, and unwavering support have accompanied me throughout this journey.

To the soul of my dear ***mother****,*
may Allah (Subḥānahu wa Taᶜālā) grant her mercy and the highest ranks of Jannah.
Her love and prayers remain my greatest source of strength.

To my ***father****,*
for his sacrifices, encouragement, and constant belief in me.

To my ***brother Oussama*** *and my* ***sisters****,*
for their presence, motivation, and sincere support.

To my beloved ***fiancée****,*
for her love, patience, and understanding, which brought balance and strength to my life.

To my close friends ***Fares****,* ***Djallel Eddine****,* ***Yacine****,* ***Hakim****,*
Isslam*,* ***Issam****,****Imad***
and all those who stood by me, thank you for your friendship, encouragement, and positive energy.

Acknowledgments

First and foremost, I would like to express my sincere gratitude to my thesis director, **Dr. Nora KERKAR**, from the Department of Electronics at Ferhat Abbas University of Sétif 1, for her invaluable supervision, scientific guidance, and continuous encouragement throughout this research. Her expertise, availability, and patience have played a key role in the completion of this thesis.

I am equally grateful to my co-director, **Dr. Yacine SLIMANI**, for his constructive advice, support, and technical insights that have enriched the quality of this work at every stage.

I would also like to thank **Professor AMARDJIA Nourredine**, President of the jury, for his time, critical reading, and thoughtful feedback.

My sincere appreciation goes to the esteemed members of the defense jury for accepting to examine this work and for their valuable comments:

- **Pr. Foudil ABDESSEMED** – University of Batna 2
- **Pr. BEKKOUCHE TOUFIK** – University of Mohamed El Bachir El Ibrahimi
- **Dr. Zerroug Nadjet** – Ferhat Abbas University of Sétif 1
- **Pr. Khier BENMAHAMMED** – Ferhat Abbas University of Sétif 1

I am grateful to the members of the **(LSI)** at Ferhat Abbas University of Sétif 1 for their support and technical assistance during my research.

Many thanks also go to my fellow PhD colleagues and researchers who contributed directly or indirectly to my progress.

List of publications

Journal Articles:

1. A. Medjaldi, Y. Slimani, N. Karkar, *"Object Detection and Distance Estimation Using Google Cloud Vision API,"* *Engineering, Technology & Applied Science Research (ETASR)*, vol. 14, no. 1, Dec. 2023.
   DOI: 10.48084/etasr.10135

Conference Papers:

1. A. Medjaldi, Y. Slimani, N. Karkar, *"Object Detection and Distance Estimation Using Google Cloud Vision API,"* *Oral presentation at the BRD International Conference on Scientific and Academic Research (ICSAR)*, Konya, Turkey, Dec. 25–26, 2023.
   DOI: 10.59287/as-proceedings.607

2. A. Medjaldi, Y. Slimani, N. Karkar, *"Object Localization Using RGB-D Camera and YOLOv3,"* *Oral presentation at the 2nd International Conference on Frontiers in Academic Research (ICFAR)*, Konya, Turkey, Dec. 2023.
   DOI: 10.59287/as-proceedings.423

others:

1. A. Medjaldi,N. Karkar, Y. Slimani, *"Vision-based Automatic guidance using Yolo and KinectV1 RGB-D Image"* *Participation in the UFAS1 Doctorials Day JD'23*

2. A. Medjaldi, Y. Slimani, N. Karkar, *"Object Detection and Distance estimation using cloud Computing Vision Service"* *Participation in the UFAS1 1st Scientific Day on Artificial Intelligence for IoT (AI4IOT'23)*

List of Abbreviations

AGV	Autonomous Guided Vehicle
AP	Average Precision
BCE	Binary Cross Entropy
BTSD	Belgium Traffic Sign Dataset
CE	Cross Entropy
CIoU	Complete Intersection over Union
CNN	Convolutional Neural Network
DFL	Distributed Focal Loss
DIoU	Distance Intersection over Union
Fast R-CNN	Fast Region-based Convolutional Neural Network
Faster R-CNN	Faster Region-based Convolutional Neural Network
GIoU	Generalized Intersection over Union
GTSRB	German Traffic Sign Recognition Benchmark
HOG	Histogram of Oriented Gradients
IoT	Internet of Things
IoU	Intersection over Union
IR	Infrared
LiDAR	Light Detection and Ranging
Mask R-CNN	Mask Region-based Convolutional Neural Network
mAP	Mean Average Precision
MIT	Massachusetts Institute of Technology
MLP	Multi-Layer Perceptron
MSE	Mean Squared Error
RBF	Radial Basis Function
RBFN	Radial Basis Function Network
RADAR	Radio Detection and Ranging
R-CNN	Region-based Convolutional Neural Network
RGB-D	Red Green Blue–Depth
RNN	Recurrent Neural Network
SIFT	Scale-Invariant Feature Transform
SOM	Self-Organizing Map

SPP-Net	Spatial Pyramid Pooling Network
SSD	Single Shot MultiBox Detector
SURF	Speeded-Up Robust Features
SVM	Support Vector Machine
ToF	Time-of-Flight
TSRD	Traffic Sign Recognition Database
YOLO	You Only Look Once

Abstract

ملخص

يهدف هذا العمل إلى تعزيز تنقل المركبات الموجهة ذاتيًا (AGVs) عبر دمج التعلم العميق مع أنظمة الرؤية الحاسوبية المعتمدة على السحابة. يستند إلى تقنيات الكشف المتقدمة لتمييز الإشارات المرورية والعوائق في الوقت الحقيقي باستخدام الشبكات العصبية. تم تطوير منصة محاكاة بالاعتماد على روبوت 3-DX Pioneer للقيام برسم الخرائط ثلاثية الأبعاد والتنقل في بيئات منظمة، مع اعتماد الحوسبة السحابية لتفويض المعالجة وتحسين الأداء. تسهم هذه البنية في رفع ذكاء ومرونة AGVs في السياقات الصناعية والحضرية.

الكلمات المفتاحية: المركبات الموجهة ذاتيًا، رسم الخرائط ثلاثية الأبعاد، التنقل، التعلم العميق، الرؤية الحاسوبية

Abstract

This work integrates cloud-based computer vision and deep learning in order to enhance the navigation of autonomous guided vehicles (AGVs). It allows the ability to recognize obstacles and traffic signs in real time by using deep neural networks. For 3D mapping and configured environment navigation, a simulation platform using the Pioneer 3-DX robot was developed. Cloud computing improves system scalability and offloads processing. Intelligent and versatile AGVs for urban and industrial applications are guaranteed by the suggested structure.

Keywords: AGV, 3D mapping, navigation, deep learning, computer vision

Résumé

Ce travail intègre la vision par ordinateur basée sur le cloud et l'apprentissage profond afin d'améliorer la navigation des véhicules guidés autonomes (AGV). Il permet de reconnaître en temps réel les obstacles et les panneaux de signalisation en utilisant des réseaux neuronaux profonds. Afin de réaliser la cartographie en 3D et la navigation dans un environnement configuré, une plateforme de simulation utilisant le robot Pioneer 3-DX a été développée. Le cloud computing améliore la scalabilité du système et réduit le traitement. La structure proposée garantit des AGV intelligents et polyvalents pour des applications urbaines et industrielles.

Mots-clés : AGV, cartographie 3D, navigation, apprentissage profond, vision par ordinateur

Contents

List of Figures

List of Tables

List of Algorithms

General Introduction

Autonomous mobile robotics has become a major area of study and development in both academic and industrial sectors in recent years. A historical categorization of industrial robots has been proposed in terms of "generations" [1], spanning from the first generation (1950–1967) to the current fourth generation (2000–present) of intelligent and networked systems. This growing interest is fueled by rapid advances in information and communication technologies, which have expanded the possibilities for intelligent, connected, and adaptive systems. Alongside the rise of artificial intelligence, big data analytics, and the Internet of Things (IoT), these developments are reshaping traditional industries and catalyzing the evolution of smart environments, particularly in applications such as autonomous vehicles, smart factories, and logistics.

The core of mobile robotics lies in the ability to navigate environments independently, without human intervention. This involves gathering and analyzing sensory information to aid in decision-making, mapping, and localization. The operating environment of the robot and mission-specific requirements typically influence the design of the perception system. Robots benefit from incorporating complementary or redundant sensors to improve perception accuracy and reliability, much as humans do. These sensors are generally categorized into two groups: *exteroceptive sensors*, which provide information about the external environment, and *proprioceptive sensors*, which provide information about internal states. Data fusion techniques are used to combine inputs from multiple sources, such as LiDAR, RADAR, stereo cameras, and RGB-D cameras, thus improving operational integrity and robustness, especially in degraded environments.

Historically, mobile robots relied on pre-mapped 2D or 3D environments to guide their movement. Navigation strategies were typically categorized into two types: global navigation, which often uses heuristic methods for pathfinding, and local navigation, also known as conventional navigation. However, these approaches suffer from limitations such as inflexibility, high cost, and a lack of adaptability to dynamic environments. Modern autonomous systems must be capable of adaptive decision-making to effectively navigate unfamiliar or changing terrain, avoid obstacles, and dynamically update planned routes.

Advancements in deep learning have significantly transformed the field of computer vision. Neural network architectures, particularly Convolutional Neural Networks (CNNs)[2], have enabled robots to interpret complex visual data with high accuracy. Early detection methods used classifiers like Support Vector Machines (SVMs)[3] along with manually crafted features such as SIFT[4], HOG[5],, and SURF[6]. While effective to some extent, these traditional methods were hindered by scalability issues, limited generalization, and reduced robustness. Deep learning has overcome these challenges by replacing handcrafted features with automatically learned representations. CNNs have emerged as the dominant paradigm for object detection, broadly categorized into two types: *two-stage models* R-CNN[2], SPP-Net[7] , Fast R-CNN [8], Faster R-CNN [9], and Mask R-CNN [10], which first generate region proposals and then perform classification, and *one-stage models* YOLO algorithm [11], Single Shot MultiBox Detector (SSD)[12] , CenterNet[13], and EfficientDet[14]., which perform detection and classification simultaneously, offering faster inference times at competitive accuracy.

Among these, YOLO (You Only Look Once) has significantly improved the ability of robots to recognize, classify, and track objects in real time, facilitating more efficient scene understanding. YOLO eliminates the need for complex preprocessing and enables rapid inference. Introduced by J. Redmon et al., the YOLO architecture has undergone several iterations focused on improving accuracy and speed, incorporating advanced features such as pooling layers and streamlined convolutional designs.

When combined with RGB-D sensors, these models allow robots to acquire rich three-dimensional representations of their surroundings, enhancing environmental mapping and obstacle avoidance. This integration also benefits traffic sign detection, a critical component in mobile robotic applications.

Furthermore, cloud computing technologies are increasingly being integrated into mobile robotics. These technologies enable offloading of computationally intensive tasks and data storage to remote servers, thus reducing on-board hardware requirements. This shift enhances scalability, reduces cost, and improves system flexibility. Cloud services offer access to advanced algorithms and shared datasets for applications such as object recognition, facial recognition, and speech processing.

These converging technologies are particularly relevant for Autonomous Guided Vehicles (AGVs), where performance, safety, and adaptability are essential. Incorporating cloud-based solutions into AGV systems opens new opportunities for real-time data sharing, collaborative learning, and intelligent fleet coordination. As a result, AGVs can become more efficient, responsive, and adaptable to the dynamic demands of modern industrial and urban settings.

Research Objectives

The main objectives of this research are summarized as follows:

- Enhance real-time robot navigation by integrating state-of-the-art deep learning algorithms for traffic sign detection, specifically using YOLOv8 in combination with RGB-D camera data.
- Develop a simulation platform to support and advance research in the field of autonomous mobile robotics.
- Integrate cloud services into 3D mapping systems to improve the robot's understanding of its environment and contribute to the evolution of intelligent navigation systems.

Thesis Organization

This thesis is organized as follows:

- **Chapter 1: Overview of Robotics** Introduces Autonomous Mobile Robots (AMRs), types of locomotion, wheeled mobile robots, and modeling methods. It details kinematic and dynamic modeling of Differential Drive Mobile Robots (DDMRs), including the Lagrangian approach and control strategies for the Pioneer 3-DX robot. The chapter also covers perception systems, including proprioceptive and exteroceptive sensors, and calibration of Kinect RGB-D sensors.
- **Chapter 2: Deep Learning Architectures for Visual Perception in Autonomous Systems** Covers the fundamentals of artificial neural networks (ANNs), activation functions, training techniques, and different network types (feedforward, recurrent, convolutional, etc.). This chapter also introduces state-of-the-art object detection techniques with a focus on one-stage models such as YOLO and SSD, and their integration with RGB-D sensors.
- **Chapter 3: Deep Learning for Object Detection and Recognition** Focuses on YOLO-based object detection pipelines, traffic sign recognition datasets, evaluation metrics, and YOLOv8 implementation. Real-time obstacle detection and avoidance using RGB-D sensors for AGVs is presented, with system architecture and experimental results.

- **Chapter 4: Cloud Vision-Based AGV Navigation** Explores 3D object representation using RGB-D point clouds, filtering and clustering (DBSCAN), grayscale intensity methods, and a cloud-enabled system architecture to enhance real-time perception and navigation.
- **Finally : Conclusion** Summarizes key findings, contributions, limitations, and proposes directions for future research in intelligent and cloud-based mobile robotics.

Chapter 1

An overview of robotics

Chapter Overview

This section introduces the scope and motivation of the chapter, outlining the fundamental concepts required for understanding autonomous mobile robots and their role in modern robotic systems.
Chapter Organization:

Contents

1.1 Introduction

Mobile robots, defined by their ability to traverse their environment, form a distinct category compared to their stationary counterparts - manipulator robots. Although the term often refers to wheeled robots, there are various locomotion methods, which lead to multiple types such as walkers, submarines, and aerial vehicles. UNIMATE, the first industrial robot to be used in a General Motors factory, marked the start of the adventure in 1961. This signaled the start of a new age, which was followed by other developments:

- 1962: The Rancho Arm demonstrated the possibility of robotic limbs with human-like dexterity and was created for people with disabilities.
- 1965: Artificial intelligence applications in a variety of fields were made possible by DENDRAL, a groundbreaking expert system.
- 1968: The Tentacle Arm showed off its inventive movement capabilities, drawing inspiration from the octopus.
- 1969 saw the development of automation with the Stanford Arm, the first electrically powered and computer-controlled robot arm.
- 1970: Shakey, the first AI-controlled mobile robot, was a major advancement in autonomous mobility.
- 1974: With its touch and pressure sensors, the Silver Arm showed off its sophisticated talents in challenging assembly jobs.
- 1979: The Stanford Cart demonstrated the promise of mobile robots in dynamic contexts by navigating a room on its while outfitted with a camera and image processing.

However, the evolution of robots transcended industrial applications. While the Robot Institute of America's (RIA) definition focuses on reprogrammable manipulators for material handling, it fails to capture the full spectrum of robotic functionalities. The broader Webster definition aptly describes robots as automatic devices performing tasks traditionally ascribed to humans:

There are two main types of mobile robots:

- Teleoperated robots: These robots are controlled by a human operator from a distance.
- Autonomous robots: These robots make their own decisions and do not need human intervention.

Mobile robots are used in a variety of applications, including:

- Manufacturing: Mobile robots are used to assemble products, move materials, and inspect parts.
- Healthcare: Mobile robots are used to deliver medication, clean hospital rooms, and perform surgery.
- Logistics: Mobile robots are used to move goods in warehouses and distribution centers.

Figure 1.1: Shakey the Robot – 1966

• Military: Mobile robots are used to perform reconnaissance, disarm bombs, and provide support to soldiers.

Wheeled mobile robots dominate the category, with studies beginning in the mid-1960s after manipulator robots Figure 1.1 . Their apparent simplicity (planar mechanisms and linear actuators) made them ideal initial subjects for studying autonomous systems. Despite their simplicity, these systems have faced many challenging problems, many of which are still unsolved even today .

This contrasts with manipulator robots, commonplace in industry. Industrial adoption of mobile robots is, however, limited due to various complexities. Unlike manipulator robots designed for repetitive tasks in controlled environments, mobile robots are meant to operate autonomously in unstructured or poorly structured environments, presenting significant challenges.

Despite these challenges, mobile robotics has significantly advanced our understanding of localization and navigation for autonomous systems. The diverse problems even a simple wheeled mobile robot presents make it a worthy subject of study and a valuable foundation for exploring more complex mobile systems.

For a mobile robot, movement often relies on components prone to slippage like wheels or tracks. This disconnects the potential readings from proprioceptive sensors from the robot's actual position. Therefore, localization, the ability to determine its position relative to the environment, becomes essential for a mobile robot, achieved through appropriate sensors. These sensors serve a dual purpose: perception, allowing the robot to sense its surroundings at varying distances, and localization, enabling it to determine its position within that environment. Perception is crucial for safety, environmental modeling, and self-localization within the

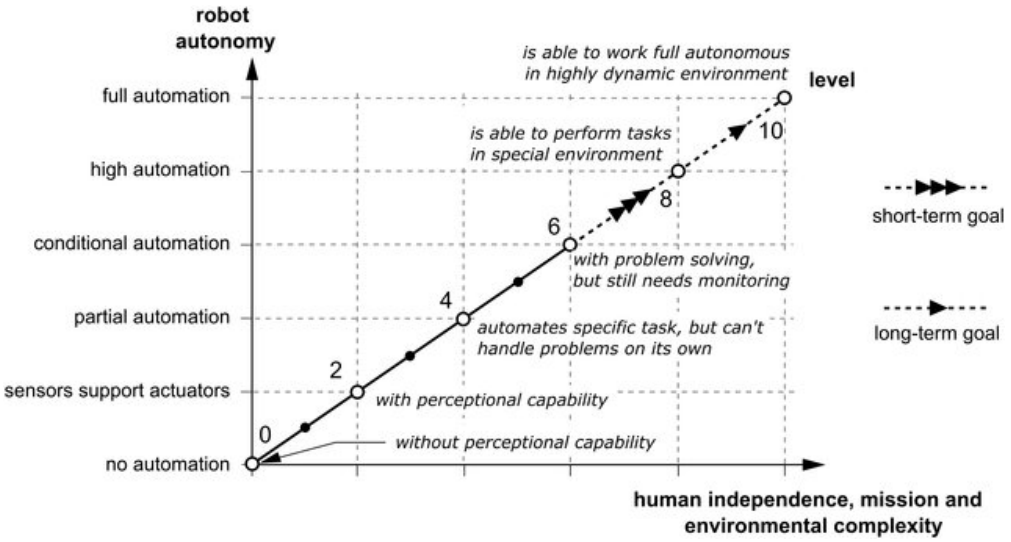


Figure 1.2: Robot autonomy based on ALFUS, the autonomy levels for unmanned systems.

environment. Localization allows the robot to position itself relative to a starting point or a pre-mapped environment.

1.2 Autonomous Mobile Robots

Automated guided Mobile robots are now a vital part of intralogistics, the movement of goods within a facility. Their advanced technology and proven effectiveness have led to their widespread adoption across various industries and production lines. Autonomy, the ability to operate independently in real-world environments for extended periods, is a significant factor influencing the complexity of mobile robot systems. First, achieving full autonomy, mimicking living systems, was the perceived goal. However, recent discourse in robotics emphasizes semiautonomy as the most efficient approach, allowing robots and humans to leverage their strengths through collaborative task sharing. Nonetheless, the degree of autonomy remains a crucial performance metric for mobile robots Figure 1.2[15].

Automated Guided Mobile robots find their primary application in intralogistics, encompasses the organization, control, optimization, and execution of internal goods and material flows within:

- Industry: Streamlining production lines and material handling.
- Trade: Optimizing logistics, information channels, and goods transfer in commercial settings.
- Public Institutions: Enhancing efficiency in material movement and logistics

management.

1.3 Type of locomotion

The core of any mobile robot's design is its locomotion system. This choice is driven by two key factors: the environment the robot operates in (land, water, air, etc.) and technical requirements like maneuverability, controllability, and terrain compatibility. These factors influence how a robot moves, leading to various gaits like walking, rolling, jumping, or swimming.

Based on their locomotion systems, mobile robots fall into several categories:

1. **Stationary:** These robots, like robotic arms, stay fixed in one place.
2. **Land-based:**
    - **Wheeled:** These robots move on wheels, offering good speed and efficiency on flat surfaces.
    - **Legged:** Inspired by animals, these robots can navigate uneven terrain but might be slower.
    - **Tracked:** These robots use continuous tracks for better traction and stability on difficult terrain.
    - **Hybrid:** Combining different locomotion methods, these robots offer versatility for complex environments.
3. **Air-based:** These robots fly, using propellers, wings, or other mechanisms.
4. **Water-based:** Submarines or other robots navigate underwater environments.
5. **Other:** This category encompasses robots with unique locomotion methods, like those that climb or burrow.

Each category has its own pros and cons, making the choice of locomotion system crucial for the robot's effectiveness in its intended application[16].

1.4 Wheeled Mobile Robots

Wheeled robots dominate mobile robot development due to their ease of control, stability, energy efficiency, and speed. They can fulfill various functions with different configurations like driving wheels, steering wheels, or a combination of both. However, unlike legged robots, their application is limited to flat, hard

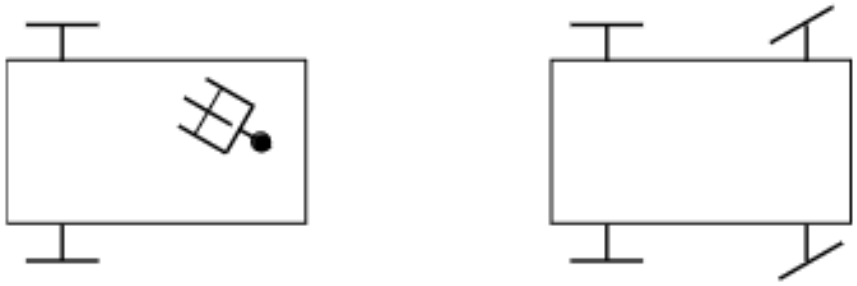


Figure 1.3: Unicycle-type and car-like mobile robots.

surfaces. Uneven terrain demands larger wheels, and less stable surfaces can cause slipping or getting stuck, creating control challenges. Despite these limitations, wheeled robots come in various forms based on wheel positioning and number, primarily categorized as unicycles, tricycles, car-like robots, and omnidirectional robots[17]. The wheel reigns supreme as the most popular locomotion mechanism in both mobile robotics and human-made vehicles (. Its key strengths lie in:

- High Efficiency: Wheeled robots achieve excellent efficiency, due to their relatively simple design.
- Simplified Balance: Unlike legged robots, wheeled robots prioritize designs that ensure all wheels maintain contact with the ground, eliminating balance as a major research concern. Three wheels are sufficient for guaranteed stability, with two-wheeled designs requiring special mechanisms for balance.

The two common types of wheeled mobile robots Figure 1.3 [18]:

- A unicycle-type robot has two independently powered wheels on a fixed axle, along with one or more passive caster wheels for stability.
- A car-like robot (with rear-wheel drive) has a powered rear axle and one or two steerable front wheels.

The number of wheels plays a crucial role in a mobile robot's design, influencing its stability, maneuverability, and terrain suitability. Below is a breakdown of common robot categories based on their wheel count:

1. **Single-wheeled (Unicycle):** These robots are inherently unstable and require complex controls for balance.
2. **Two-wheeled:** These robots offer better stability than single-wheeled ones. Examples include the Roomba vacuum cleaner and Segway.
3. **Three-wheeled:** There are two main types:
    - Differently steered (two powered wheels + one free-turning wheel).

- Single-actuator driven with a steered third wheel.

4. **Four-wheeled:** These robots are more stable due to a wider center of gravity. They can be:
   - Differentially steered (like rovers),
   - Two-by-two powered wheels (like tanks),
   - Car-like steering (widely used in self-driving cars and logistics).
5. **Five-wheeled:** Designed for rough terrains, these robots offer enhanced contact and stability. They can be used for obstacle crossing and climbing slopes.
6. **Six-wheeled:** These robots, like the Mars rovers, offer superior traction and stability on rugged terrain. They often utilize complex suspension systems for better control and obstacle navigation. Examples include Sojourner, Spirit, Opportunity, Curiosity, and Shrimp.

More than six wheels: Robots like the Octopus with unique wheel configurations aim to tackle challenging terrains with enhanced maneuverability and obstacle handling capabilities.

1.5 Modelling Wheeled Mobile Robots

The modeling of wheeled mobile robots—particularly differential-drive configurations—requires a layered approach that encompasses three fundamental stages: *kinematic modeling*, *dynamic modeling*, and *actuator modeling* [19]. Each of these layers serves a distinct role in characterizing the robot's behavior, from its geometric motion to the physical forces acting upon it, and ultimately to the control inputs driving its actuators.

Kinematic modeling establishes the motion equations based on the geometric and velocity relationships, abstracting away the influence of forces and masses. It focuses solely on how the robot's configuration evolves over time due to wheel rotations. In contrast, **dynamic modeling** incorporates Newtonian mechanics, accounting for forces, torques, and energy to describe the system's behavior more realistically. Finally, **actuator modeling** bridges the gap between control signals and physical movement by describing how motor inputs produce wheel torques and rotations [20, 21].

Consider a differential-drive mobile robot (DDMR) equipped with two identical wheels, each with a radius R, separated by a distance $2L$ (i.e., L from the centerline). As shown in Figure 1.4, the robot's local coordinate system is fixed at

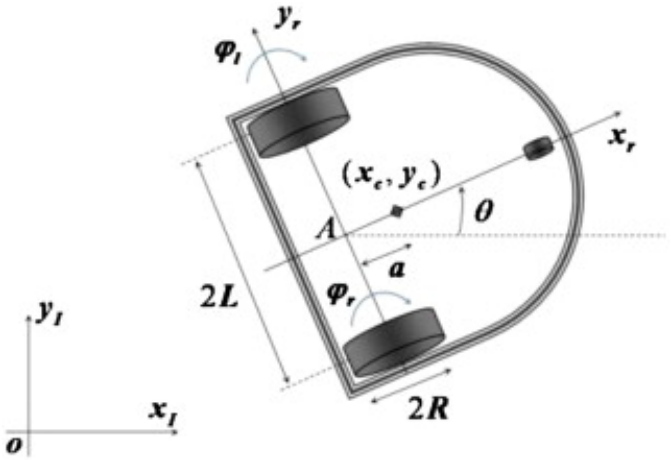


Figure 1.4: Differential-drive mobile robot model and reference frames

point A, the midpoint between the two wheels. The center of mass C is positioned along the robot's longitudinal axis at a distance d from point A [21].

Reference Frames

Two coordinate frames are considered for motion modeling:

- **Inertial Frame** $\{X_I, Y_I\}$: A fixed global reference frame attached to the environment.
- **Robot Frame** $\{X_r, Y_r\}$: A body-fixed frame attached to the robot, centered at point A and aligned with its forward direction.

The pose of the robot with respect to the inertial frame is defined as

$$\mathbf{q}^I = \begin{bmatrix} x_a \\ y_a \\ \theta \end{bmatrix}, \tag{1.1}$$

where (x_a, y_a) denote the position of point A in the inertial frame and θ is the robot orientation with respect to the X_I-axis.

Let $\mathbf{p}^r$ and $\mathbf{p}^I$ denote the coordinates of an arbitrary point expressed in the robot and inertial frames, respectively:

$$\mathbf{p}^r = \begin{bmatrix} x^r \\ y^r \end{bmatrix}, \quad \mathbf{p}^I = \begin{bmatrix} x^I \\ y^I \end{bmatrix}. \tag{1.2}$$

The relationship between these coordinates is given by the rigid-body transformation

$$\mathbf{p}^I = \mathbf{R}(\theta)\,\mathbf{p}^r + \mathbf{t} \tag{1.3}$$

where $\mathbf{t} = \begin{bmatrix} x_a \\ y_a \end{bmatrix}$ is the translation vector, and $\mathbf{R}(\theta)$ is the planar rotation matrix defined as

$$\mathbf{R}(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \tag{1.4}$$

Differentiating (1.3) with respect to time yields the velocity relationship.

$$\dot{\mathbf{p}}^I = \mathbf{R}(\theta)\,\dot{\mathbf{p}}^r + \dot{\mathbf{R}}(\theta)\,\mathbf{p}^r + \dot{\mathbf{t}}, \tag{1.5}$$

where the time derivative of the rotation matrix is given by

$$\dot{\mathbf{R}}(\theta) = \dot{\theta}\begin{bmatrix} -\sin\theta & -\cos\theta \\ \cos\theta & -\sin\theta \end{bmatrix}. \tag{1.6}$$

This expression highlights that the rotation matrix is time-varying and that its derivative explicitly depends on the angular velocity $\dot{\theta}$. Such a formulation is fundamental in mobile robot kinematics, navigation, and control.

1.5.1 Kinematic Constraints of the differential-Drive robot

1. No Lateral Slip Motion: Because of this limitation, the robot can only move in a curved motion—either forward or backward—and not sideways. This condition indicates that the center-point A in the robot frame has zero velocity along the lateral axis:

$$\dot{y}_a^r = 0 \tag{1.7}$$

As shown in (1.7), the lateral velocity in the robot frame is zero. Using the orthogonal rotation matrix $\mathbf{R}(\theta)$, the velocity in the inertial frame yields:

$$-\dot{x}_a \sin\theta + \dot{y}_a \cos\theta = 0 \tag{1.8}$$

Equation (1.8) represents the nonholonomic constraint of the robot.

2. Pure Rolling Constraint: Every wheel retains a single point of contact (P) with the ground according to the pure rolling constraint. Both the longitudinal axis (X_r) and the lateral axis (Y_r) of the wheel are free of slippage. Refer to Figure 1.5[22].

The velocities at the contact points in the robot frame are related to the angular wheel velocities as follows:

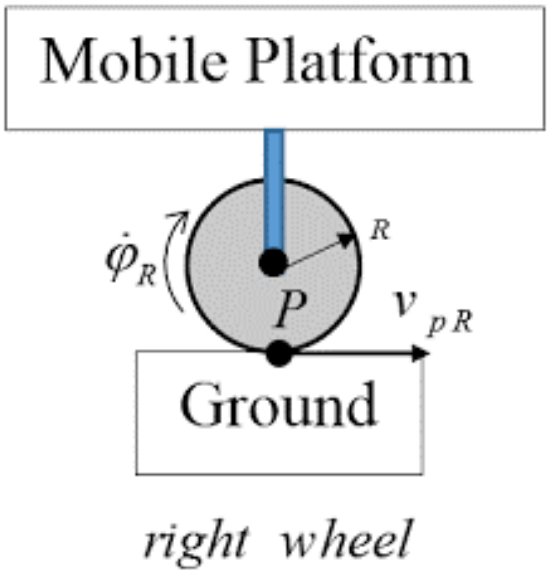


Figure 1.5: Rolling motion constraints

$$\begin{cases} v_{pR} = R\,\dot{\phi}_R \\ v_{pL} = R\,\dot{\phi}_L \end{cases} \tag{1.9}$$

where: - R is the radius of the wheel, - $\dot{\phi}_R$ and $\dot{\phi}_L$ are the angular velocities of the right and left wheels, respectively, - v_{pR} and v_{pL} are the linear velocities at the contact points of the right and left wheels.

In the inertial frame, the velocities of the contact points can be expressed as a function of the velocity of the robot's center point A:

$$\dot{x}_{pR} = \dot{x}_a + L\dot{\theta}\cos\theta \tag{1.10}$$
$$\dot{y}_{pR} = \dot{y}_a + L\dot{\theta}\sin\theta \tag{1.11}$$

$$\dot{x}_{pL} = \dot{x}_a - L\dot{\theta}\cos\theta \tag{1.12}$$
$$\dot{y}_{pL} = \dot{y}_a - L\dot{\theta}\sin\theta \tag{1.13}$$

Using the rotation matrix $\mathbf{R}(\theta)$, the rolling constraint equations can be written as:

$$\dot{x}_{pR}\cos\theta + \dot{y}_{pR}\sin\theta = R\dot{\phi}_R \tag{1.14}$$
$$\dot{x}_{pL}\cos\theta + \dot{y}_{pL}\sin\theta = R\dot{\phi}_L \tag{1.15}$$

Substituting the contact point velocities into the rolling constraints, the three constraint equations can be written in matrix form:

$$\Lambda(q)\,\dot{q} = 0 \tag{1.16}$$

where:

$$\Lambda(q) = \begin{bmatrix} -\sin\theta & \cos\theta & 0 & 0 & 0 \\ \cos\theta & \sin\theta & L & -R & 0 \\ \cos\theta & \sin\theta & -L & 0 & -R \end{bmatrix} \tag{1.17}$$

and the generalized velocity vector is:

$$\dot{q} = \begin{bmatrix} \dot{x}_a \\ \dot{y}_a \\ \dot{\theta} \\ \dot{\phi}_R \\ \dot{\phi}_L \end{bmatrix} \tag{1.18}$$

Additionally, the wheel velocities are:

$$\begin{cases} v_R = R\dot{\phi}_R \\ v_L = R\dot{\phi}_L \end{cases} \tag{1.19}$$

The above constraint matrix $\Lambda(q)$ will be used in the next section for the dynamic modeling of the Differential-Drive Mobile Robot (DDMR).

1.5.2 Kinematic Modeling of a DDMR

Kinematic modeling provides a mathematical framework for describing the motion of robotic systems without considering the forces or torques responsible for that motion. In the context of a Differential Drive Mobile Robot (DDMR), this modeling is vital for understanding how wheel rotations translate into overall linear and angular displacements of the robot's body.

A DDMR navigates using two independently actuated wheels mounted on either side of its chassis. Let R denote the radius of each wheel, and L the half-distance between the wheels (i.e., the distance from the center of the robot to either wheel). The rotational speeds of the right and left wheels, $\dot{\phi}_R$ and $\dot{\phi}_L$ respectively, directly influence the robot's translational and rotational behavior.

The linear velocity v of the robot in its body-fixed coordinate frame is obtained as the mean of the two wheel velocities:

$$v = \frac{v_R + v_L}{2} = \frac{R}{2}(\dot{\phi}_R + \dot{\phi}_L) \tag{1.20}$$

Similarly, the robot's angular velocity ω, which defines the rate of rotation about its vertical axis, is derived from the difference in wheel speeds:

$$\omega = \frac{v_R - v_L}{2L} = \frac{R}{2L}(\dot{\phi}_R - \dot{\phi}_L) \tag{1.21}$$

Combining these relationships, the velocity vector of the robot at its geometric center A can be expressed in its **local coordinate frame** as:

$$\begin{bmatrix} \dot{x}_a^r \\ \dot{y}_a^r \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} \frac{R}{2} & \frac{R}{2} \\ 0 & 0 \\ \frac{R}{2L} & -\frac{R}{2L} \end{bmatrix} \begin{bmatrix} \dot{\phi}_R \\ \dot{\phi}_L \end{bmatrix} \tag{1.22}$$

To express the robot's movement in the **global (inertial) coordinate frame**, we apply a rotation matrix that accounts for the robot's orientation θ:

$$\dot{q}^I = \begin{bmatrix} \dot{x}_a^I \\ \dot{y}_a^I \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} \frac{R}{2}\cos\theta & \frac{R}{2}\cos\theta \\ \frac{R}{2}\sin\theta & \frac{R}{2}\sin\theta \\ \frac{R}{2L} & -\frac{R}{2L} \end{bmatrix} \begin{bmatrix} \dot{\phi}_R \\ \dot{\phi}_L \end{bmatrix} \tag{1.23}$$

Equation (1.23) encapsulates the **forward kinematics** of the DDMR, mapping wheel velocities to global pose derivatives. Alternatively, the same transformation can be represented using the robot's computed linear and angular velocities:

$$\dot{q}^I = \begin{bmatrix} \dot{x}_a^I \\ \dot{y}_a^I \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} \cos\theta & 0 \\ \sin\theta & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} v \\ \omega \end{bmatrix} \tag{1.24}$$

This final formulation highlights how the robot's local motion components are projected into the inertial frame based on its orientation. It is especially useful in trajectory tracking and navigation algorithms, where real-world positioning is critical.

1.5.3 Dynamic Modeling of the DDMR

Unlike kinematics, which disregards the different forces at play, dynamics examines how a mechanical system moves while accounting for them. Research on DDMR motion simulation and the development of different motion control methods both require the DDMR dynamic model [23, 24].

A non-holonomic DDMR with n generalized coordinates $(q_1, q_2, \ldots, q_n)$, subject to m constraints, can be described by the following equations of motion:

$$M(q)\ddot{q} + V(q,\dot{q})\dot{q} + F(\dot{q}) + G(q) + \tau_d = B(q)\tau - \Lambda^T(q)\lambda \tag{1.25}$$

where:

- $M(q)$: $n \times n$ symmetric positive definite inertia matrix
- $V(q, \dot{q})$: Centripetal and Coriolis matrix
- $F(\dot{q})$: Surface friction matrix
- $G(q)$: Gravitational vector
- τ_d: Vector of bounded unknown disturbances, including unmodeled dynamics
- $B(q)$: Input matrix
- τ: Input vector
- $\Lambda^T(q)$: Matrix associated with kinematic constraints
- λ: Vector of Lagrange multipliers

1.5.4 Lagrange Dynamic Approach

The Lagrangian method is a powerful approach for deriving the dynamic equations of a mechanical system by considering kinetic and potential energies. Since the DDMR operates on a flat surface, the potential energy is zero [25].

The general Lagrange equation is:

$$\frac{d}{dt}\left(\frac{\partial L}{\partial \dot{q}_i}\right) - \frac{\partial L}{\partial q_i} = F - \Lambda^T(q)\lambda \tag{1.26}$$

where $L = T - V$ is the Lagrangian, $q = [x_a, y_a, \theta, \phi_R, \phi_L]^T$ is the vector of generalized coordinates, and $\Lambda(q)\lambda$ represents the constraint forces.

The total kinetic energy T is the sum of:

$$T_c = \frac{1}{2}m_c v_c^2 + \frac{1}{2}I_c\dot{\theta}^2 \tag{1.27}$$

$$T_{wR} = \frac{1}{2}m_w v_{wR}^2 + \frac{1}{2}I_m\dot{\theta}^2 + \frac{1}{2}I_w\dot{\phi}_R^2 \tag{1.28}$$

$$T_{wL} = \frac{1}{2}m_w v_{wL}^2 + \frac{1}{2}I_m\dot{\theta}^2 + \frac{1}{2}I_w\dot{\phi}_L^2 \tag{1.29}$$

Using geometric relationships:

$$v_c = \sqrt{\dot{x}_a^2 + \dot{y}_a^2 + d^2\dot{\theta}^2 + 2d(\dot{y}_a\cos\theta - \dot{x}_a\sin\theta)\dot{\theta}} \tag{1.30}$$

$$v_{wR} = \sqrt{\dot{x}_a^2 + \dot{y}_a^2 + L^2\dot{\theta}^2 + 2L(\dot{y}_a\cos\theta - \dot{x}_a\sin\theta)\dot{\theta}} \tag{1.31}$$

$$v_{wL} = \sqrt{\dot{x}_a^2 + \dot{y}_a^2 + L^2\dot{\theta}^2 - 2L(\dot{y}_a\cos\theta - \dot{x}_a\sin\theta)\dot{\theta}} \tag{1.32}$$

The total kinetic energy becomes:

$$T = \frac{1}{2}m(\dot{x}_a^2 + \dot{y}_a^2) - m_c d\dot{\theta}(\dot{y}_a \cos\theta - \dot{x}_a \sin\theta) + \frac{1}{2}I_w(\dot{\phi}_R^2 + \dot{\phi}_L^2) + \frac{1}{2}I\dot{\theta}^2 \tag{1.33}$$

where:

$$m = m_c + 2m_w, \quad I = I_c + m_c d^2 + 2m_w L^2 + 2I_m$$

Substituting into the Lagrange equation yields:

$$m\ddot{x}_a - md\ddot{\theta}\sin\theta - md\dot{\theta}^2\cos\theta = C_1 \tag{1.34}$$

$$m\ddot{y}_a - md\ddot{\theta}\cos\theta - md\dot{\theta}^2\sin\theta = C_2 \tag{1.35}$$

$$I\ddot{\theta} - md\ddot{x}_a\sin\theta + md\ddot{y}_a\cos\theta = C_3 \tag{1.36}$$

$$I_w\ddot{\phi}_R = \tau_R + C_4 \tag{1.37}$$

$$I_w\ddot{\phi}_L = \tau_L + C_5 \tag{1.38}$$

In matrix form:

$$M(q)\ddot{q} + V(q,\dot{q})\dot{q} = B(q)\tau - \Lambda^T(q)\lambda \tag{1.39}$$

To remove λ, apply:

$$\dot{q} = S(q)\eta, \tag{1.40}$$

$$\ddot{q} = \dot{S}(q)\eta + S(q)\dot{\eta} \tag{1.41}$$

Premultiplying by $S^T(q)$:

$$\bar{M}(q)\dot{\eta} + \bar{V}(q,\dot{q})\eta = \bar{B}(q)\tau \tag{1.42}$$

with:

$$\bar{M}(q) = \begin{bmatrix} I_w + \frac{R^2}{4L^2}(mL^2 + I) & \frac{R^2}{4L^2}(mL^2 - I) \\ \frac{R^2}{4L^2}(mL^2 - I) & I_w + \frac{R^2}{4L^2}(mL^2 + I) \end{bmatrix} \tag{1.43}$$

$$\bar{V}(q,\dot{q}) = \begin{bmatrix} 0 & \frac{R^2}{2L}m_c d\dot{\theta} \\ -\frac{R^2}{2L}m_c d\dot{\theta} & 0 \end{bmatrix}, \tag{1.44}$$

$$\bar{B}(q) = I_{2\times 2} \tag{1.45}$$

The system can also be represented in terms of linear v and angular ω velocities:

$$\begin{bmatrix} m_0 & 0 \\ 0 & I_0 \end{bmatrix}\begin{bmatrix} \dot{v} \\ \dot{\omega} \end{bmatrix} + \begin{bmatrix} 0 & -m_c d\omega^2 \\ m_c d\omega & 0 \end{bmatrix}\begin{bmatrix} v \\ \omega \end{bmatrix} = \begin{bmatrix} \frac{1}{R} & 0 \\ 0 & \frac{L}{R} \end{bmatrix}\begin{bmatrix} \tau_R + \tau_L \\ \tau_R - \tau_L \end{bmatrix} \tag{1.46}$$

with:

$$m_0 = m + \frac{2I_w}{R^2}, \quad I_0 = I + \frac{2L^2 I_w}{R^2}$$

This reduced model is well-suited for control and simulation of the DDMR.

1.6 Control Model of the Pioneer 3-DX Robot

The Pioneer 3-DX is a differential drive robot with two diametrically opposed wheels of radius r and axle length l. The robot's posture with respect to an inertial frame $\mathcal{F}_R(O, x, y)$ is given by the state vector:

$$q = \begin{bmatrix} x & y & \theta & \phi_R & \phi_L \end{bmatrix}^T \tag{1.47}$$

Where: - x, y are the Cartesian coordinates of the center of mass, - θ is the robot's heading angle relative to the x-axis, - ϕ_R, ϕ_L are the angular displacements of the right and left wheels.

The kinematic constraints of the robot are:

$$\dot{y}\cos\theta - \dot{x}\sin\theta = 0 \tag{1.48}$$

$$\dot{x}\cos\theta + \dot{y}\sin\theta + \frac{l}{2}\dot{\theta} - r\dot{\phi}_R = 0 \tag{1.49}$$

$$\dot{x}\cos\theta + \dot{y}\sin\theta - \frac{l}{2}\dot{\theta} - r\dot{\phi}_L = 0 \tag{1.50}$$

Equation (1.48) enforces the non-slipping constraint: motion occurs only along the x_b axis. Equations (1.49) and (1.50) model pure rolling for each wheel.

Subtracting (1.50) from (1.49) gives a holonomic relation:

$$l\dot{\theta} = r(\dot{\phi}_L - \dot{\phi}_R) \quad \Rightarrow \quad \theta(t) = \frac{r}{l}(\phi_L - \phi_R) + C \tag{1.51}$$

The Pioneer 3-DX model is equivalent to the unicycle model:

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} \cos\theta & 0 \\ \sin\theta & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} v \\ \omega \end{bmatrix} \tag{1.52}$$

Where: - v is the linear velocity (longitudinal), - ω is the angular velocity.

The relationship to the wheel velocities is:

$$\begin{bmatrix} v_R \\ v_L \end{bmatrix} = \begin{bmatrix} 1 & \frac{l}{2} \\ 1 & -\frac{l}{2} \end{bmatrix} \begin{bmatrix} v \\ \omega \end{bmatrix} \quad \text{and} \quad v_R = r\dot{\phi}_R, \quad v_L = r\dot{\phi}_L \tag{1.53}$$

Table 1.1: Robot motion based on left and right wheel velocities

v_L	v_R	**Angular Velocity** ω	**Robot Motion**
> 0	> 0 and $v_L = v_R$	0	Move forward (straight)
< 0	< 0 and $v_L = v_R$	0	Move backward (straight)
> 0	> 0 and $v_L < v_R$	> 0	Turn left (forward arc)
> 0	> 0 and $v_L > v_R$	< 0	Turn right (forward arc)
< 0	< 0 and $v_L < v_R$	> 0	Turn left (backward arc)
< 0	< 0 and $v_L > v_R$	< 0	Turn right (backward arc)
> 0	< 0	$\neq 0$	Rotate in place (counterclockwise)
< 0	> 0	$\neq 0$	Rotate in place (clockwise)
$= 0$	$= 0$	0	Stationary

1.6.1 Control Strategy

To control the robot, you command the linear and angular velocities v and ω based on a path-following algorithm . These are then converted into individual wheel angular velocities $\dot{\phi}_R$ and $\dot{\phi}_L$ using:

$$\dot{\phi}_R = \frac{v + \frac{l}{2}\omega}{r}, \quad \dot{\phi}_L = \frac{v - \frac{l}{2}\omega}{r} \tag{1.54}$$

These values are fed to the motor controllers of the robot wheels.

Algorithm 1 Pioneer 3-DX Wheel Velocity Control

Require: Desired linear velocity v, angular velocity ω
Require: Wheel radius r, axle length l
Ensure: Right and left wheel angular velocities $\dot{\phi}_R$, $\dot{\phi}_L$

1: **Input:** v, ω, r, l
2: Compute right wheel linear velocity:
3: $\quad v_R \leftarrow v + \frac{l}{2} \cdot \omega$
4: Compute left wheel linear velocity:
5: $\quad v_L \leftarrow v - \frac{l}{2} \cdot \omega$
6: Convert to angular wheel speeds:
7: $\quad \dot{\phi}_R \leftarrow \frac{v_R}{r}$
8: $\quad \dot{\phi}_L \leftarrow \frac{v_L}{r}$
9: **Output:** $\dot{\phi}_R$, $\dot{\phi}_L$

1.7 Perception Systems in Mobile Robotics

The perception function is the ability to acquire and understand sensory information about the environment. It is a critical first step for a mobile robot, as it provides the information needed for subsequent localization and mapping tasks.

First, the choice of a perception system is often dependent on the robot's operating environment and the functionalities implemented on the robot to fulfill its mission.

Second, a perception system consisting of a single sensor will rarely be sufficient to accurately perceive the environment. This is like humans, who rely on multiple sensory organs to understand their surroundings. A mobile robot's perception system will typically integrate multiple sensors of complementary types to enrich sensory information or of redundant types to address the issue of degraded mode operation. In this context, data fusion methods are generally used to process this sensory information.

Third, the cost of integrating sensors on autonomous vehicles is a significant factor. The desired precision and a high acquisition frequency will increase the cost of a sensor. This is a constraint that will inevitably influence the choice of a perception system.

Sensors are classified into two families:

- **Proprioceptive sensors** provide information about the robot's internal behavior, i.e., its state at a given time.
- **Exteroceptive sensors** provide information about the world outside the robot.

1.7.1 Proprioceptive sensors

Proprioceptive sensors are responsible for monitoring the internal state of a robot. They play a crucial role in regulating locomotion and maintaining balance by providing real-time feedback about the robot's own motion and orientation. These sensors are generally categorized into two main groups:

- **Motion sensors** measure the robot's movement. This includes sensors such as odometers, accelerometers, Doppler radars, and optical sensors.
- **Attitude sensors** measure the robot's orientation. This includes sensors such as gyroscopes, inertial sensors, inclinometers, and magnetometers.

Global Navigation Satellite Systems (GNSS), such as GPS and Galileo, are among the most widely used positioning technologies. Inertial Measurement Units (IMUs)

Figure 1.6: Wheel odometry with an optical encoder [Pololu Corporation 2016]
frames

provide complementary motion information. GNSS receivers are capable of measuring the position, velocity, and, in some cases, the heading of an ego vehicle. The actual positioning techniques and the adjustments made have a significant impact on accuracy. In addition, the IMU records the ego vehicle's accelerations and angular rotation rate; the three readings taken together can be used to infer the vehicle's three-dimensional orientation. Lastly, we have sensors for wheel odometry. This sensor measures the wheel rotation rates and uses them to calculate the ego car's estimated speed and heading rate of change. This is the same sensor that monitors your car's mileage. [26].

Odometers

By measuring the rotation of its wheels, odometers are utilized to measure the curvilinear movements of a robot. The wheels' elementary rotations are integrated to calculate the robot's relative position. Odometers consist of incremental encoders that enable measurement of rotation angles with a precision that is dependent on the encoder's resolution. The displacement information will require knowledge of the diameter of the wheels, the wheelbase, the mechanical and kinematic structure of the vehicle. This sensor is widely used in mobile robotics since it has the advantage of being simple to implement and inexpensive[27, 28][29].

Accelerometers

Accelerometers are a common tool for measuring physical activity. They work by detecting changes in speed (acceleration) over time. Acceleration is typically measured in meters per second squared (m/s^2). An accelerometer is a sensor that measures linear acceleration at a given point. In practice, acceleration is measured using a test mass M, of mass m, connected to a sensor housing. The principle of this sensor is to measure the non-gravitational mass force that must be applied to M to keep it in place in the housing when an acceleration is applied to the housing [30], [31]. The calculation of the robot's elementary displacement is obtained by double integration of this information. This double integration leads to significant error accumulation. This sensor is more expensive than odometers [28].

Gyroscopes

Gyroscopes are heading sensors that keep their course with respect to a stationary reference frame. As a result, they provide a precise estimate of a mobile system's direction. Gyroscopes are used to measure angular velocity, or an object's rate of rotation.

There are two types of gyroscopes: mechanical gyroscopes and optical varieties. mechanical gyroscopes. A fast-spinning rotor's inertial characteristics are essential to the idea of a mechanical gyroscope. The gyroscopic precession is the name of the relevant characteristic. In place of moving mechanical components, optical gyroscopes are angular speed sensors that employ two monochromatic light beams, or lasers, emitted from the same source. Gyroscopes can be used to correct odometry errors by providing information about the robot's orientation. By combining the data from a gyroscope and an odometer, it is possible to calculate the robot's position more accurately [32].

Inertial Measurement Unit (IMU)

An Inertial Measurement Unit (IMU) is a self-contained sensing device widely employed for estimating the motion state of mobile platforms, including orientation, angular velocity, and linear acceleration. IMUs do not directly measure position or velocity; instead, these quantities are obtained through temporal integration of inertial measurements. Due to their autonomy and high update rates, IMUs play a critical role in navigation systems, particularly in environments where external positioning signals are unavailable or unreliable.

Conventional IMUs are composed of two primary types of sensors: accelerometers and gyroscopes. Accelerometers measure specific force, which includes inertial acceleration and gravitational components, while gyroscopes measure angular velocity about the sensor axes. To enable full three-dimensional motion sensing, mod-

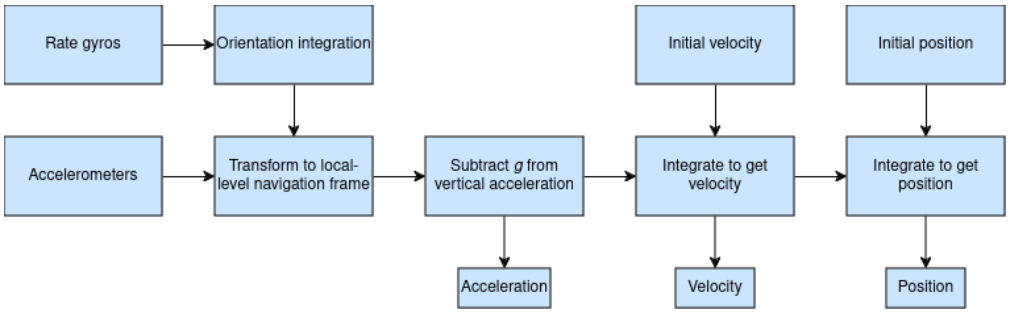


Figure 1.7: Block diagram of an inertial measurement unit (IMU) and associated reference frames

ern IMUs typically incorporate tri-axial accelerometers and tri-axial gyroscopes, providing six degrees of freedom (6-DoF) inertial measurements.

Advances in micro-electro-mechanical systems (MEMS) technology have led to the development of compact, low-cost IMUs with improved robustness and reduced power consumption. In more advanced configurations, IMUs may also integrate additional sensors such as magnetometers, enabling enhanced attitude estimation through sensor fusion techniques [33].

A typical IMU architecture and its associated reference frames are illustrated in Fig. 1.7, highlighting the relationship between the sensor measurements and the body-fixed coordinate system.

1.7.2 Exteroceptive Sensors

Exteroceptive sensors allow robots to interact with their surroundings. They function like external sense organs, enabling the acquisition of information from the environment. This information can take various forms. Therefore, measurements from exteroceptive sensors are interpreted by robots to produce meaningful environmental features[34].

Ultrasonic Sensor (Time-of-Flight, Sound) (Sonars)

Ultrasonic sensors use sound waves to detect range, hence their original term for sound navigation and ranging. Sonar sensors are low-cost, short-range monitoring devices. A number important factors are taken into consideration while choosing sonar, including their cost, detecting field of view, and maximum range that they can measure. A number important factors are taken into consideration when choosing sonar, including its cost, detecting field of view, and maximum range.[35].

The distance d to an object responsible for the reflected acoustic signal can be estimated using the *time-of-flight* (ToF) principle. Assuming propagation in a

homogeneous medium, the distance is given by

$$d = \frac{c\,t}{2}, \tag{1.55}$$

where t denotes the measured round-trip travel time of the acoustic wave and c is the speed of sound in air. The factor $\frac{1}{2}$ accounts for the forward and return propagation of the wave between the sensor and the reflecting object.

The speed of sound in air is a function of the thermodynamic properties of the medium and can be expressed as

$$c = \sqrt{\gamma R T}, \tag{1.56}$$

where γ is the ratio of specific heats of air, R is the specific gas constant, and T denotes the absolute temperature in Kelvin.

This formulation emphasizes the sensitivity of time-of-flight-based distance estimation to environmental conditions, particularly temperature, and highlights the necessity of atmospheric compensation to ensure accurate ranging measurements.

Laser Rangefinder (Time-of-Flight, Electromagnetic)

Light Detection and Ranging (LiDAR), commonly referred to as a laser rangefinder, is an active electromagnetic sensing technology that offers significant advantages over acoustic rangefinders by exploiting the propagation of light rather than sound. In LiDAR systems, narrow laser pulses are emitted toward the surrounding environment, and the distance to objects is determined by measuring the time-of-flight (ToF) of the reflected signals.

By accurately estimating the round-trip travel time of the emitted laser pulse and knowing the speed of light, LiDAR sensors are capable of providing high-resolution range measurements with centimeter-level accuracy. In addition to distance estimation, the intensity of the returned signal can be used to infer surface reflectivity properties, which may aid in object segmentation and scene interpretation.

Most modern LiDAR systems employ a mechanically or electronically actuated scanning mechanism, often consisting of a rotating assembly with multiple vertically stacked laser emitters and receivers. This configuration enables the acquisition of dense three-dimensional point clouds that capture detailed geometric information about the surrounding environment. An example of such a point cloud representation is shown in Fig. 1.8.

As an active sensor, LiDAR operates independently of ambient illumination conditions and maintains reliable performance under low-light or dynamically varying lighting environments. Consequently, LiDAR does not suffer from many of the limitations inherent to passive vision-based sensors, making it particularly

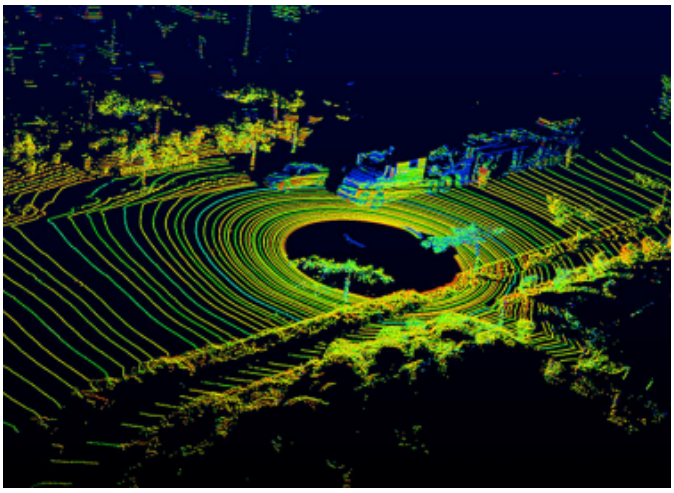


Figure 1.8: Three-dimensional scene geometry reconstructed from a LiDAR point cloud

well suited for perception tasks in autonomous navigation and robotic mapping applications [36].

Time-of-Flight Camera

Similar to a LIDAR, a Time-of-Flight (TOF) camera has the advantage of recording the complete 3D scene simultaneously and without any moving elements. A modulated light pulse is emitted by a 3D TOF camera, which then analyzes the reflected light to produce 3D images. The duration of light's journey to and from an item is measured by the camera.

Since light travels at a known speed, this time difference (phase shift) reveals the object's distance. By calculating this for each pixel, the camera constructs a depth map, which is a 3D representation of the scene[37, 38].

Doppler Effect Sensing (RADAR or Sound)

Longer in use than LIDAR systems, RADAR (Radio Detection and Ranging) sensors are renowned for their reliable detection of large objects under a range of environmental circumstances. RADAR's dependability in inclement weather, including rain or fog, where optical sensors could malfunction, is one of its many noteworthy benefits.

RGB-Depth Camera

From its early conceptual underpinnings to its current application in robotics, gaming, and 3D vision, RGB-D sensing has seen substantial development. Despite their limited hardware capabilities, experimental systems such as MIT's "Put-That-There" offered multimodal interfaces in the late 1970s, laying the foundation for spatial interaction.[39]

Although it needed actual contact, 3D hand tracking was introduced to consumer markets by 1989 with the Nintendo Power Glove, which used magnetic tracking and ultrasonic sensors. Ten years later, 2D contactless gesture tracking using webcams was revolutionized by Sony's EyeToy.[40]

Nintendo's Wii Remote adopted a hybrid strategy in 2006, using inertial sensors and infrared light to track motion more accurately. However, RGB-D sensing did not become well known until the 2010 debut of Microsoft Kinect. Initially created for gaming, the Kinect sensor made depth sensing accessible to a wider audience by utilizing infrared structured light technology. In its subsequent iteration, Time-of-Flight was added. Its low cost and simplicity of use in computer vision and robotics research fundamentally altered the area.[41]

Unlike passive sensors, which depend on an external radiation source, these sensors are usually active, which means they have their own emission source. Two different ideas underlie the operation of active RGB-D sensors: time-of-flight (ToF) measurement, which calculates depth by measuring the time it takes for emitted light to return to the sensor, and active triangulation, which uses structured light projection.

1.8 Kinect RGB-D sensor

The Microsoft Kinect sensor's structured or coded light technology is briefly described in this section. This technique is used in sensors that continuously emit an infrared (IR) light pattern in the direction of the item being examined. A known pattern is created by filtering and structuring this light before it is projected onto the scene. The projected pattern is picked up by an infrared camera, which then examines how it changes when it comes into touch with the surface of the object. The sensor calculates each point's distance from itself by examining these distortions. The Kinect provides three types of data:

- Infrared (IR) Image: A grayscale image representing the intensity of the returned infrared beam from the observed object.
- Depth Map (Depth Image): Derived from the same IR camera, the depth map is obtained by processing the infrared beams, enabling distance measurement between the sensor and objects. It is a 16-bit grayscale image, where each pixel

Figure 1.9: RGB along with depth Image.

directly represents the depth of a corresponding point in the scene.

- Color Image: A standard RGB image of the scene.

The development of computer vision algorithms has been greatly aided by the release of the Kinect V1 RGB-D sensor. Its drawbacks include a limited range and the difficulty to determine depth properly on materials that are extremely reflecting, transparent, or absorbent. Furthermore, the sensor typically provides somewhat noisy depth data. The Kinect V1 sensor features an RGB camera, an infrared laser emitter, and an infrared camera. A laser beam is diffracted into many speckles that are projected onto the scene as part of a triangulation procedure to determine depth. This pattern is recorded by the infrared camera, which then compares it to a reference pattern that has been saved and acquired from a predefined distance. Variations in depth are indicated by displacements in the speckle places, which enables the sensor to calculate a disparity image.The depth of each pixel is then determined based on its disparity. By establishing a depth coordinate system, the perspective center of the infrared camera serves as the origin of the system, representing the 3D coordinates of object points. A right-handed coordinate system is defined where:

- The **Z-axis** extends perpendicularly from the picture plane towards the object,
- The **X-axis** runs along the baseline b between the laser projector and the infrared camera,
- The **Y-axis** is orthogonal to both the X- and Z-axes.

If an object is placed on a reference plane at a given distance Z_0, the speckle projection of the object is visible on the picture plane of the infrared camera. The **disparity** d in the image space represents the amount the speckle changes along

the X-axis as the object moves closer or farther from the sensor. Using triangle similarity, the depth Z_k of a point k in object space is derived from the equations:

$$\frac{D}{b} = \frac{Z_0 - Z_k}{Z_0}$$

Substituting D into the equation and solving for Z_k, we get:

$$Z_k = \frac{Z_0}{1 + \frac{Z_0}{fb} d}$$

This equation serves as the fundamental mathematical model for computing depth from disparity, provided that the constant parameters Z_0, f, and b are determined through calibration.

Once Z_k is known, the planimetric coordinates of each point can be computed using its image coordinates and scaling factor:

$$X_k = -\frac{Z_k}{f} (x_k - x_0 + \delta x)$$

$$Y_k = -\frac{Z_k}{f} (y_k - y_0 + \delta y)$$

where X_k and Y_k are the image coordinates of the point, x_0 and y_0 represent the principal point coordinates, and δx and δy are lens distortion corrections.It is assumed that the image coordinate system is aligned with the baseline, ensuring parallelism with the depth coordinate system.[42] To fully utilize the sensor for mapping applications, analyzing systematic and random errors is essential. Correcting systematic errors is crucial for aligning depth and color data, requiring an accurate depth measurement model and proper calibration parameters.

1.8.1 Calibration

Accurate calibration is crucial for deriving reliable 3D information from the Kinect sensor. However, direct calibration using full-resolution sensor data is impractical due to the inherently lower resolution of the disparity images when compared to the infrared (IR) frames. To ensure pixel-wise correspondence with the disparity data, downsampled infrared images are employed instead for the calibration process.

Before applying the calibration results to compute depth values, a systematic correction must be made to address a discovered offset—a 4-pixel horizontal shift (in the x-direction) between the IR and disparity images. Once this shift is corrected, the calibration parameters obtained from the infrared images can be transferred directly to the disparity data.

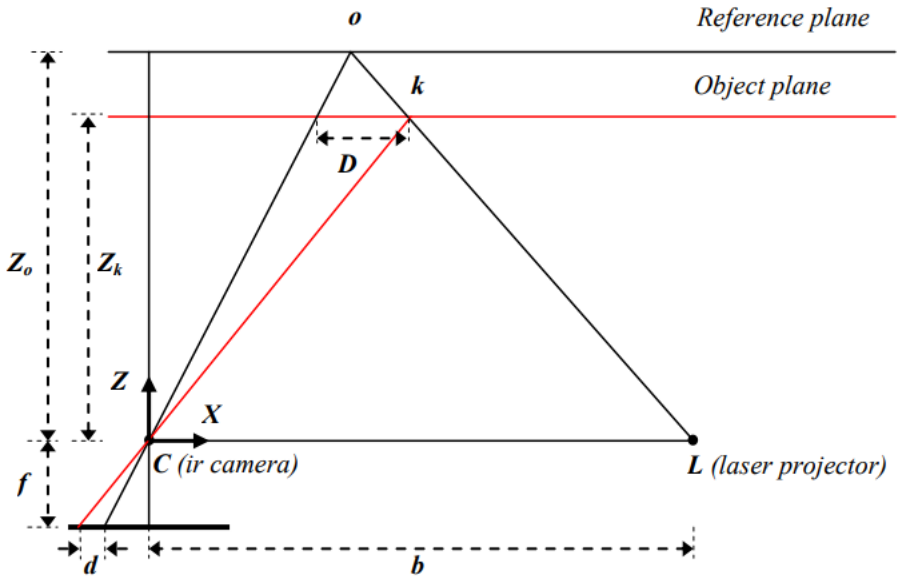


Figure 1.10: Relation between relative depth and measured disparity.

The complete calibration process yields a set of parameters that define the mapping from image measurements to 3D world coordinates. These parameters include the focal length f, the coordinates of the principal point (x_0, y_0), distortion coefficients $(\delta x, \delta y)$, the baseline distance b between the IR projector and camera, and the reference pattern distance Z_0. When the depth reference frame is aligned with the image coordinate frame, any angular misalignment between the baseline and the x-axis can be neglected.

The intrinsic parameters f, x_0, y_0, and distortion coefficients are typically obtained via standard infrared camera calibration procedures. However, estimating the baseline b and the reference distance Z_0 is more complex. Due to bandwidth constraints, the Kinect transmits disparity values as 11-bit integers normalized between 0 and 2047 rather than actual disparities. Hence, the true disparity d must be modeled as a linear transformation of the normalized disparity d' via parameters m and n:

$$d = md' + n$$

This leads to the following depth inversion formula:

$$\left(\frac{m}{fb}\right) d' + \left(\frac{Z_0^{-1} + n}{fb}\right) = Z_k^{-1}$$

This equation establishes a linear relationship between the inverse depth Z_k^{-1}

and the normalized disparity d'. To estimate the coefficients m and n, a least-squares regression can be applied using known distances of planar or object surfaces and their corresponding disparity measurements. Nevertheless, due to the incorporation of normalization constants, b and Z_0 cannot be uniquely resolved without external reference information [42].

Once the full set of calibration parameters is determined, the transformation from image space (x, y, d') to 3D space (X, Y, Z) is fully specified, enabling the construction of dense point clouds from the disparity images.

To integrate color with depth, the spatial relationship between the RGB camera and the depth sensor must also be calibrated. This stereo calibration process includes estimating rotation, translation, and the intrinsic parameters of the RGB camera (e.g., focal length and lens distortion). Since RGB frames are transmitted at a lower resolution, calibration is performed on scaled-down RGB images. With the derived parameters, each 3D point is projected onto the RGB image plane, and its color is determined through interpolation. This facilitates accurate colorization of the point cloud by mapping each 3D point to its corresponding pixel in the RGB frame.

1.9 Conclusion

This chapter has presented a comprehensive overview of autonomous mobile robotics, establishing the theoretical and technological foundations required for the design and deployment of intelligent navigation systems. The discussion began with a general introduction to robotics and autonomous mobile robots, followed by a detailed examination of locomotion mechanisms, with particular emphasis on wheeled mobile robots due to their mechanical simplicity, efficiency, and widespread industrial adoption.

The kinematic and dynamic modeling of differential-drive mobile robots was thoroughly addressed, including nonholonomic constraints, velocity relationships, and Lagrangian-based dynamic formulations. These models form the cornerstone for motion analysis, control design, and trajectory planning, and were subsequently applied to the Pioneer 3-DX robot as a representative platform. The control strategies presented provide the necessary link between theoretical modeling and practical implementation.

Furthermore, the chapter highlighted the critical role of perception systems in mobile robotics, distinguishing between proprioceptive and exteroceptive sensors and analyzing their complementary contributions to state estimation and environmental awareness. This sensor-based perception framework is fundamental to achieving robust autonomy in structured and semi-structured environments.

Special attention was devoted to RGB-D sensing, with a particular focus on

the Microsoft Kinect sensor. Its integrated depth and color sensing capabilities, calibration requirements, and suitability for three-dimensional environment perception were discussed, positioning it as a cost-effective and versatile solution for mapping and navigation tasks.

Overall, this chapter lays the groundwork for the subsequent chapters by providing the essential modeling, control, and perception concepts upon which advanced navigation, sensor fusion, and deep learning-based approaches are developed.

Chapter 2

Deep Learning Architectures for Visual Perception in Autonomous Systems

Chapter Overview

This chapter introduces the fundamental deep learning architectures used for visual perception in autonomous systems. We discuss the components of artificial neural networks, activation functions, training strategies, and various types of neural networks, including CNNs and RNNs, highlighting their applications in robotic perception.

Chapter Organization:

Contents

2.1 Introduction

Artificial Neural Networks (ANNs), which mimic the neurons found in real brains, are made up of interconnected artificial neurons. A simplified representation of a biological neuron is shown in Fig. 2.1. The cell body, dendrites, and axons are the three main parts of a neuron. The cell body, which functions similarly to a thresholding unit, controls all neuronal activity. Dendrites protrude from the cell body to help receive input from other neurons. The axon is a long fiber that carries messages from the cell body to neighboring neurons [43].

In terms of its functionality, a neuron operates in the following manner: dendrites serve as pathways for receiving inputs, connecting to sensory organs such as the eye or other neurons through synapses. The soma integrates these inputs, and once a certain threshold is exceeded, initiates a series of spikes along the axon. Consequently, as the signal spreads, the nucleus returns to its resting state, ceasing the firing process. The transfer of signals to other neurons is facilitated by boutons, with synaptic connections acting as channels for signal transmission [44].

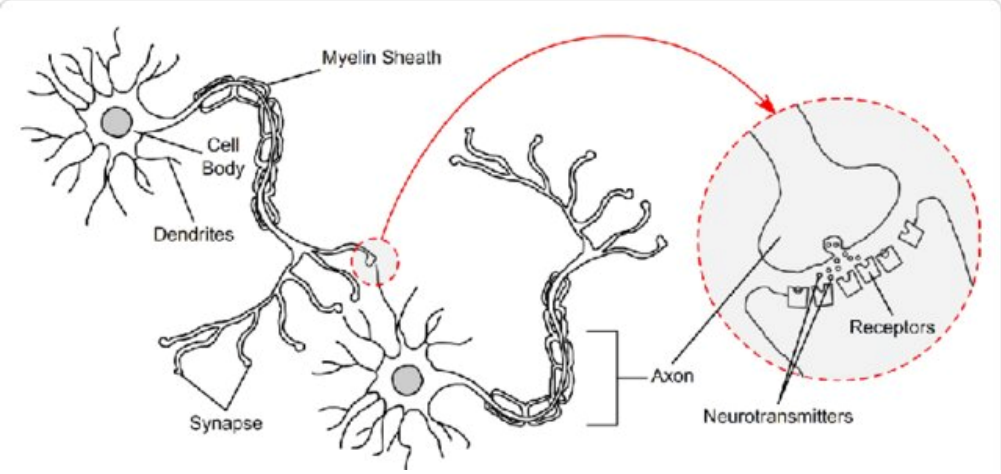


Figure 2.1: Simplified image of a biological neuron

The effectiveness of the nucleus is influenced by the strengths of synapses and the signals received at axon terminals. Furthermore, signal propagation strictly follows a unidirectional pathway—from axon terminals to dendrites—making it impossible for signals to transmit from dendrites to axon terminals. It is important to note that a single neuron can establish connections with thousands of others.

The behavior of a biological neuron can be mathematically represented as:

$$f(x) = G(wx^T + b) \tag{2.1}$$

In this expression, $w \in \mathbb{R}^d$ is the weight vector, $x \in \mathbb{R}^d$ is the input vector, and $b \in \mathbb{R}$ is the bias or intercept term. An artificial neuron performs a weighted

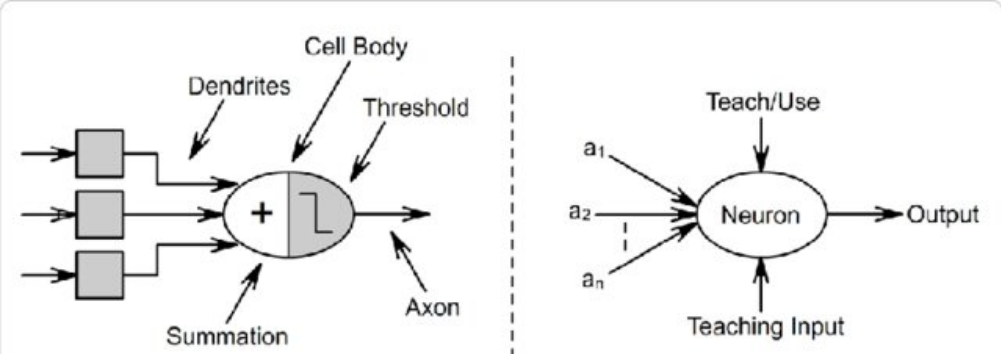


Figure 2.2: Biological neuron mapped to an artificial neuron

summation of its inputs, analogous to the function of the soma in a biological neuron. The weights represent the synaptic strengths, while the inputs represent signals from other neurons or sensors. The function $G(x) : \mathbb{R} \to \mathbb{R}$ is a nonlinear activation function that applies a transformation to its input. This activation function simulates the threshold mechanism in biological neurons, producing an output based on the neuron's potential, $wx^T + b$.This abstraction of biological processing into an artificial neuron is illustrated in Fig. 2.2.

2.2 Components of Artificial Neural Networks

2.2.1 Input Layer

Receives the input data intended for the neural network. Without performing any computations on the input values, it directly forwards them to the subsequent layer.

2.2.2 Hidden Layers

Hidden layers reside within the neural network. Each node in the input layer is linked to nodes in the hidden layers, with every hidden layer incorporating an activation function.

2.2.3 Output Layer

This represents the final layer of the neural network, tasked with making predictions. Each node in the hidden layers connects to the output layer, transmitting the outputs from the hidden layers for evaluation. The output layer also employs an activation function to determine the probability of various outcomes.

2.2.4 Weights

Weights are values that the model learns during training. As inputs are fed into the neural network, the model assigns a value to each based on their significance, which represents a weight at a high level.

2.2.5 Biases

Biases are constants typically fixed at a value of 1 and function as extra inputs to the next layer. Bias units create outgoing connections with their own weights but are not impacted by the layer above (which lacks incoming connections). These bias units ensure that neuron activation can occur even when all other inputs are zero.

2.3 Activation Functions

In artificial neural networks, activation functions are essential because they transform input signals into output signals that are sent to the network's later layers. Each layer's output, which becomes the input for the layer after it, is obtained by applying an activation function to the weighted sum of the inputs and their corresponding weights in a neural network[45]. The number of layers and, crucially, the activation function selection determine a neural network's prediction accuracy. A common rule of thumb indicates that at least two layers are advantageous, while there are no hard rules dictating the ideal number of layers or the kind of activation function[46],by adding bias and calculating a weighted sum of inputs, activation functions decide whether to activate a neuron. As a result, the neuron's output becomes non-linear and is limited to a range, such [0, 1] or [-1, 1]. Activation functions can be used between two neural networks and are also known as Transfer Functions. To achieve accurate outputs in ANNs, it is essential to update the weights and biases of neurons based on the output error, a process known as Back Propagation (BP).This process is made easier by activation functions, which offer gradients in addition to the error, allowing weights and biases to be adjusted. The most popular optimization technique for ANNs is gradient descent optimization[47]. Prediction mistakes can be decreased by using one or more hidden layers in a neural network,. Prediction accuracy is mostly determined by activation functions, of which non-linear functions are most frequently employed. Non-linear functions allow the network to capture non-linear correlations found in real-world data, in contrast to linear activation functions, which generate outputs equal to the inputs or inputs that have been linearly converted. Due to the non-linear nature of real-world errors, neural networks equipped with non-linear activation functions have superior adaptability and performance compared to those

with linear activation functions. As a result, non-linear activation functions are preferred in neural network architectures to enhance their learning capabilities and predictive accuracy. Nonlinear activation functions are crucial in enabling a neural network to learn a wide array of nonlinear functions, especially when it has sufficient neurons and layers. Activation within a neural network facilitates non-linear mappings by performing mathematical operations on the inputs. Without activation functions, the output signal from a neural network would simply be a linear function, a polynomial of the first degree. While linear equations are easy to understand and solve, their complexity is limited. They lack the capability to learn and recognize intricate relationships within data. A neural network without activation functions behaves similarly to a linear regression model, resulting in restricted performance and power in most cases. The goal is for neural networks to go beyond learning and computing linear functions. We want them to tackle more complex tasks, such as modeling intricate data types like images, videos, audio, speech, and text. To achieve this, activation functions are crucial.

2.4 TYPES OF ACTIVATION FUNCTIONS

Neural networks rely on net inputs, which are processed to produce an output called the activation of the unit. An activation function, sometimes known as a threshold function or transfer function, is applied to enable a scalar-to-scalar translation in order to achieve this transformation. By keeping neuron outputs within a limited range, squashing functions are essential for restricting their amplitude. These routines guarantee that the output signal stays inside a specified boundary by compressing its amplitude. The Binary Step Function, Linear, Sigmoid, Tanh, ReLU, Leaky ReLU, Parametrized ReLU, Exponential Linear Unit, Swish, and SoftMax are examples of frequently used activation functions. Neural networks' performance and behavior are influenced by these functions in a variety of ways.

2.4.1 Sigmoid (Logistic) Activation Function

The following equations provide the sigmoid activation function[48]:

$$f_{\text{Sigmoid}}(x) = \frac{1}{1 + e^{-x}} \tag{2.2}$$

When plotted, the function appears as shown in Figure 2.3.

The sigmoid function is one of the most prevalent activation functions used in neural networks due to its non-linear nature. It transforms input values into a bounded range between 0 and 1, a characteristic that has enabled its extensive

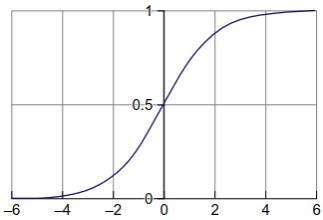


Figure 2.3: Sigmoid activation function.

use in neural networks for many years. Furthermore, it is biologically inspired, mimicking the firing behavior of biological neurons.

Despite its advantages, the sigmoid function suffers from issues such as saturation and the vanishing gradient problem. Its derivative is given by:

$$f'(x) = f_{\text{Sigmoid}}(x) \cdot (1 - f_{\text{Sigmoid}}(x)) \tag{2.3}$$

While this derivative is smooth and differentiable, a notable drawback is that the sigmoid function is not symmetric around zero. This asymmetry can cause all neuron outputs to have the same sign, leading to inefficient training. To address this, scaling or alternative activation functions may be used.

2.4.2 TANH FUNCTION

Another often used activation function in neural networks is the Hyperbolic Tangent (tanh) function, which is essentially a scaled-up sigimoid. Its definition in mathematics is:

$$f_{\tanh}(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} = 2f_{\text{sigmoid}}(2x) - 1 \tag{2.4}$$

The derivative of the tanh function is:

$$f'_{\tanh}(x) = 1 - f^2_{\tanh}(x) \tag{2.5}$$

With output values limited to the range of -1 to 1, the Tanh function preserves continuity and differentiability. Tanh provides steeper gradients than the sigmoid function, which makes it easier to memorize intricate patterns. Tanh is also preferred over the sigmoid function because gradients are not restricted to a single direction due to its zero-centered character.

2.4.3 Rectified Linear Unit FUNCTION

One common non-linear activation function that is often used in neural networks is the Rectified Linear Unit, or ReLU. Its capacity to selectively activate neurons—

ensuring that a neuron deactivates only when the output of the linear transformation is zero—is its primary advantage. In terms of mathematics, it can be stated as :

$$f_{\text{ReLU}}(x) = \max(0, x) \tag{2.6}$$

Its derivative is:

$$f'_{\text{ReLU}}(x) = \begin{cases} 0 & \text{if } x < 0 \\ 1 & \text{if } x \geq 0 \end{cases} \tag{2.7}$$

Since ReLU only stimulates a portion of neurons at a time rather than all of them at once, its efficiency is one of its most prominent advantages. It is crucial to remember that ReLU could run into situations when the gradient value is zero, which would prevent weights and biases from being updated during backpropagation in neural network training.

2.4.4 LEAKY RELU FUNCTION

Leaky ReLU[49],represents an enhanced rendition of the ReLU function. The Learky ReLU defined as follows :

$$f_{\text{lrelu}}(x) = \begin{cases} \alpha x & \text{if } x < 0 \\ x & \text{if } x \geq 0 \end{cases} \tag{2.8}$$

Its derivative is:

$$f'_{\text{lrelu}}(x) = \begin{cases} \alpha & \text{if } x < 0 \\ 1 & \text{if } x \geq 0 \end{cases} \tag{2.9}$$

Leaky ReLU returns the constant value α, in contrast to ReLU, which delivers a function value of zero for negative values. The value of the hyperparameter α typically falls between $[0, 1]$. Setting α to 0.001 is a typical value. By ensuring that Leaky ReLU remains activated for negative inputs, this improvement enhances performance and reduces problems related to whole neuron deactivation.

2.4.5 PARAMETRIZED RELU FUNCTION

A modest change and improved performance are features of the Parametrized ReLU (PReLU), a development of the Rectified Linear Unit (ReLU). In order to learn from data, the neural network treats the difference α as a parameter. Mathematically expressed as:

$$f_{\text{PReLU}}(x) = \begin{cases} x & \text{if } x \geq 0 \\ \alpha x & \text{if } x < 0 \end{cases} \tag{2.10}$$

It effectively addresses the issue of the ReLU function's gradient becoming zero for negative x values by introducing a new parameter, known as the slope, for the negative part of the function.

2.4.6 EXPONENTIAL LINEAR UNIT

Another variation of the Rectified Linear Unit (ReLU) is the Exponential Linear Unit (ELU). By altering the original ReLU $\max(0, x)$ to $\max(-1, x)$, it can be viewed as a smoothed version of the shifted ReLU activation function. The activation function passes a negative value close to the origin via this shift. Mathematically expressed as:

$$f_{\text{ELU}}(x) = \begin{cases} \alpha(e^x - 1) & \text{if } x < 0 \\ x & \text{if } x \geq 0 \end{cases} \tag{2.11}$$

2.5 Training of an Artificial neural network

A neural network becomes trainable once it has been tailored to a specific task or application. Initially, its parameters—commonly referred to as weights—are set using random values. During training, these weights are iteratively updated to reduce the difference between the network's predictions and the actual targets in the training data. As data propagates through the layers, each layer extracts increasingly abstract features, allowing the network to identify the most relevant patterns for improved performance. This learning process is guided by a loss function, which quantifies the error between predicted and true outputs.

For classification problems, **Cross-Entropy Loss (CE)** is a widely adopted objective function. It is mathematically expressed as:

$$CE = -\sum_{x} p(x) \log q(x) \tag{2.12}$$

where x represents a training instance, $p(x)$ is the true probability distribution, and $q(x)$ is the predicted distribution.

In contrast, for regression tasks, the **Mean Squared Error (MSE)**—also known as quadratic loss—is commonly employed:

$$MSE = \frac{1}{|X|} \sum_{x} (y(x) - \hat{y}(x))^2 \tag{2.13}$$

Here, $|X|$ denotes the number of training examples, $y(x)$ is the actual value, and $\hat{y}(x)$ is the predicted output.

Once the loss function is defined, training proceeds through optimization techniques such as **gradient descent** and **backpropagation**. Backpropagation enables the computation of gradients by propagating the prediction error backward through the network, layer by layer. Gradient descent then uses these gradients to update the weights in the direction that minimizes the loss function. Popular optimization algorithms based on this principle include **Stochastic Gradient Descent (SGD)** [50], **Adam** [51], and **RMSprop** [52], each with adaptations for various scenarios in practice.

2.5.1 Regularization

By using a variety of strategies to progressively lower parameter magnitudes over time, regularization helps to lessen the influence of unbounded parameters. By reducing certain weights, the regularization coefficients L1 and L2 prevent overfitting. Simplified hypotheses are easier to generalize when weights are reduced. Unregularized weights have a tendency to overfit the training set, particularly when they are linked to higher-order polynomials in the feature set.

2.5.2 Momentum

In the search space, momentum helps the learning algorithm avoid stagnation areas where it could otherwise get stuck. It helps the updater locate the valleys in the error landscape that lead to the minima.

2.6 Different types of Neural Networks

Neuronal connections distinguish the two primary network architectures: ”feed-forward neural networks” and ”recurrent neural networks.” A feed-forward neural network lacks feedback between inputs and neuron outputs. By means of synaptic connections between outputs and inputs—which may be self-referential or to inputs from other neurons—a recurrent neural network, on the other hand, integrates this type of feedback. Neural networks are often organized in layers. Feed-forward neural networks can be further categorized as ”single-layer” or ”multi-layer” based on the number of layers they contain. [53]

2.6.1 Single layer feedforward neural network

In layered feedforward networks, neurons are arranged in distinct layers. Single-layer networks, which consist of an input layer composed of source nodes connected to an output layer (computation nodes), are the simplest kind of repetition. It’s

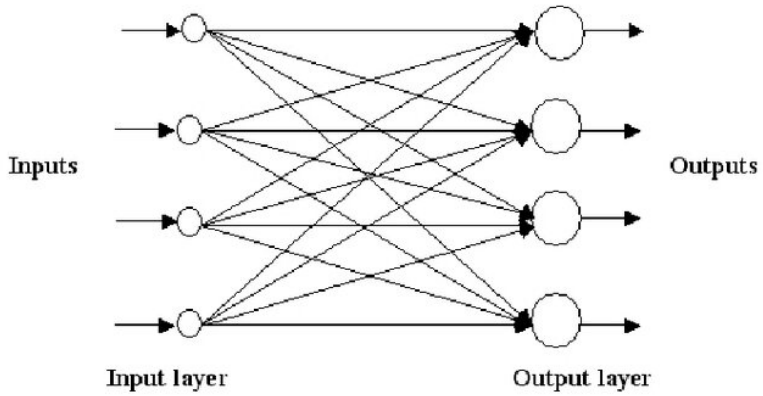


Figure 2.4: Single layer feedforward neural network

interesting to note that the network is feedforward or acyclic since neurons in the input layer do not connect to neurons in the output layer. When the input layer is disregarded, the network is classified as a single-layer network since no computations are performed there.[54]

2.6.2 Multi-Layer Feedforward Neural Networks

Multi-Layer Perceptrons (MLPs) are a core type of architecture in the broader class of Artificial Neural Networks (ANNs), particularly valued for their versatility in handling a wide array of supervised learning problems. These networks are structured as a sequence of layers, beginning with an *input layer* that receives the raw feature vector, followed by one or more *hidden layers* composed of interconnected computational nodes (neurons), and culminating in an *output layer* that produces the final prediction. Each layer in an MLP is densely (fully) connected to the next, creating a deep feedforward architecture that processes data unidirectionally—from input to output—without forming cycles or recurrent paths.

Unlike shallow networks, MLPs are capable of learning intricate mappings from input to output by leveraging **hierarchical feature representations**. Each hidden layer transforms its input space through a set of weighted connections and non-linear activation functions, enabling the network to model complex, high-dimensional decision boundaries.

The learning mechanism of an MLP relies on the *backpropagation algorithm*, which iteratively minimizes a predefined loss function (e.g., mean squared error or cross-entropy) via stochastic or batch gradient descent. The process begins with a **forward propagation** step, where inputs traverse the network to produce

an output. This is followed by a **backward propagation** step, where gradients of the loss function with respect to each weight are computed using the chain rule of calculus. These gradients inform the weight updates that reduce error in subsequent training iterations.

A pivotal factor in the effectiveness of MLPs is the choice of **activation functions**. If only linear activation functions are employed, the network collapses into a single-layer linear model, regardless of depth, due to the compositional properties of linear transformations. To mitigate this, non-linear activation functions are introduced to allow the network to capture non-linear relationships in the data. One of the earliest and most influential non-linear activations is the *sigmoid function*, particularly its single-pole variant [55], which maps input values to a bounded output range and introduces differentiability, a key requirement for gradient-based optimization.

In modern practice, other activation functions such as *ReLU* (Rectified Linear Unit), *tanh*, and *Leaky ReLU* are frequently used due to their advantages in training deep architectures, including mitigation of the vanishing gradient problem. However, the sigmoid function remains foundational in understanding early developments in deep learning and continues to be used in specific contexts such as binary classification tasks and gating mechanisms within more complex neural structures like LSTMs.

Overall, the MLP serves as a fundamental building block in deep learning, providing a conceptual and operational basis for more advanced architectures. Its feedforward nature, universal approximation capability, and compatibility with gradient-based learning frameworks have cemented its role in the evolution of neural computation.

2.6.3 Recurrent Neural Network (RNN)

Recurrent neural networks (RNNs) are distinguished by the presence of at least one feedback connection, which permits activations to repeat in a loop. This feature allows the network to learn sequences, perform temporal processing, recognize or replicate sequences, and form temporal correlations or predictions[56]. Recurrent neural network topologies come in several varieties. One popular kind incorporates extra loops into a traditional Multi-Layer Perceptron (MLP). These architectures incorporate some kind of memory while utilizing the powerful non-linear mapping capabilities of the MLP. Other topologies might use stochastic activation functions and more homogeneous structures that might connect each neuron[57, 58].

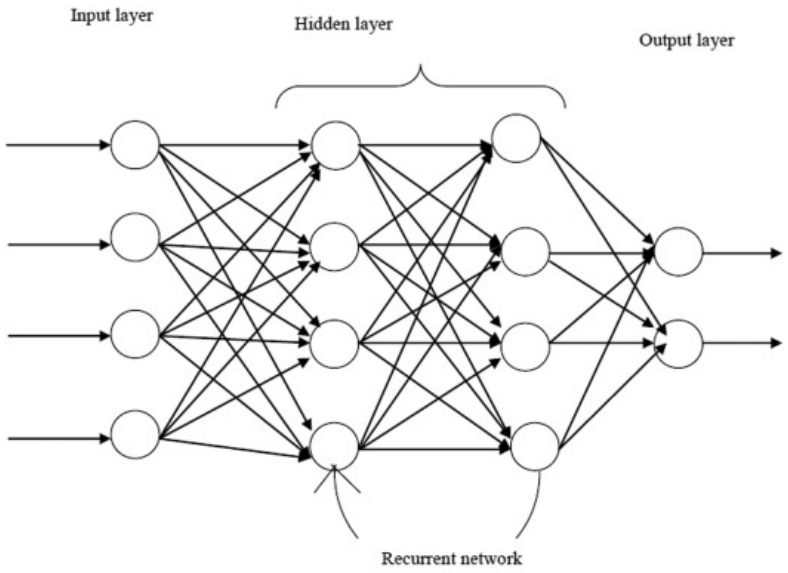


Figure 2.5: Architecture of RNN, recurrent neural networks.

2.6.4 Radial basis function Neural Network

Function approximation theory serves as the foundation for the idea of Radial Basis Function Networks (RBFNs). It entails figuring out the Euclidean distance between each neuron's center and the point being evaluated. To ascertain the weight or effect for every neuron, a radial basis function (RBF), sometimes referred to as a kernel function, is then applied to this distance. The phrase "radial basis" indicates that the function's argument is distance. RBFs function essentially as local receptors, and the proximity of the input to a particular stored vector determines the output.[59] The radial-basis-function (RBF) network consists of three layers.The input space is transformed nonlinearly without weights by the hidden layer. The center and breadth parameters determine the hidden layer's nodes, which are known as radial basis functions.[60]

2.6.5 Kohonen Self Organizing Neural Network

Competitive learning is used to create the Kohonen Self-Organizing Feature Map (SOM), a neural network that self-organizes by adapting to input patterns. A competition is held before to every learning cycle in competitive learning, and the winner is chosen by a set of criteria, usually minimizing the Euclidean distance between the input and weight vectors. After the winning processing element has been chosen, its weight vector is modified in accordance with the learning law that was used. By altering not just the winning processing element but also its

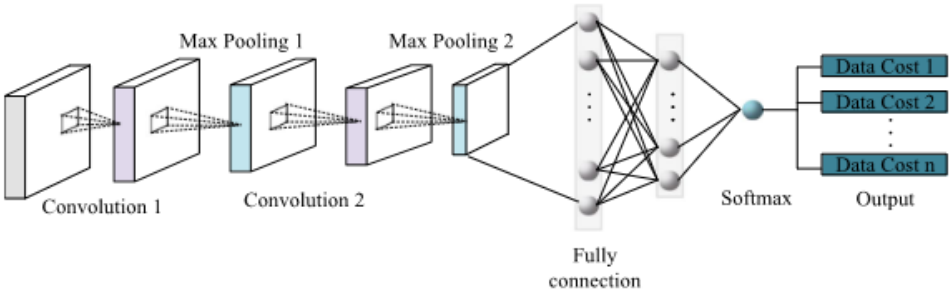


Figure 2.6: A Convolutional Neural Network Structure.

surrounding processing elements, SOM sets itself apart from simple competitive learning. SOM's self-organizing feature depends on taking use of the winning processing element's neighborhood.[61]

2.6.6 Convolutional Neural Network

A specialized architecture within the domain of neural networks, the **Convolutional Neural Network (CNN)** is particularly adept at handling data with grid-like structures—most notably, images. CNNs demonstrate strong performance in both *supervised* and *unsupervised* learning contexts. In supervised settings, they are trained to map input data to known output labels, while in unsupervised scenarios—where label information is absent—they aim to discover hidden patterns or model the data's underlying distribution.

The foundation of any CNN lies in its **convolutional layers**, which serve as the primary mechanism for feature extraction. These layers employ a series of learnable filters, or *kernels*, which are applied across the input space via the convolution operation. This process generates *feature maps* that emphasize local spatial patterns and structures within the data [62], enabling the network to detect edges, textures, and increasingly complex features as depth increases.

Convolution layer

The convolutional layer, which is in charge of obtaining various features from discrete local areas in the input data, is made up of a collection of convolution kernels, each of which is a neuron. The three dimensions of each convolution kernel are depth (D), width (W), and length (L). The convolution kernels in a CNN's convolutional layer have artificially specified dimensions; the kernel size is represented by the product of length and width, L $\times$ W. Commonly utilized sizes 3 $\times$ 3, 5 $\times$ 5, and so forth. The number of channels, also referred to as depth or the quantity of feature maps, represents the number of feature maps

produced by each layer within the CNN. Additionally, the depth of the convolution kernel aligns with the number of channels. The CNN's computational cost and feature extraction capabilities are significantly influenced by the feature map's depth, which is indicated by the number of channels. Increasing the number of channels can improve the CNN's ability to extract features, but it also increases the computational cost. The following is a mathematical expression for the convolution operation applied to a continuous function:

$$s(t) = \int x(a)w(t-a)\,da \tag{2.14}$$

This procedure is also represented by the asterisk (*) as follows:

$$s(t) = (x * w)(t) \tag{2.15}$$

symbolizes how a convolution kernel (or filter) moves across an input signal or picture. The output feature map is created by multiplying the kernel values element-wise by the matching input values at each point, then adding the results. Convolution can be conceptualized as the computation of a weighted average of the function $x(a)$ using a kernel function $w(a)$. When one function is shifted over the other, it acts as a gauge of how much overlap there is between them. In the context of Convolutional Neural Networks (CNNs), the function x typically represents the input, and w denotes the kernel (or filter). The discrete convolution operation is defined as:

$$s(t) = (x * w)(t) = \sum_{a=-\infty}^{\infty} x(a)\,w(t-a) \tag{2.16}$$

In machine learning applications, both the input and the kernel are generally represented as multi-dimensional arrays. These arrays are referred to as *tensors*. The input tensor stores the data, while the kernel tensor contains the learnable parameters of the model.

Although the theoretical definition involves an infinite summation, in practical implementations, the summation is performed over a finite number of array elements. It is typically assumed that the tensor values outside the defined range are zero, effectively limiting the computation to a finite domain. Convolutions can also be applied to multiple axes simultaneously. For instance, when a two-dimensional kernel K is applied to a two-dimensional image I as input, the two-dimensional convolution can be defined as:

$$S(i,j) = (I * K)(i,j) = \sum_{m}\sum_{n} I(i-m, j-n)K(m,n) \tag{2.17}$$

This equation defines how the two-dimensional input I and kernel K are convolved to produce the output $S(i, j)$, where the summation is performed over both axes (i.e., the m and n indices).

Pooling Layer

Pooling Layer

In the architectural hierarchy of a **Convolutional Neural Network (CNN)**, the *pooling layer* plays a vital role in the transformation pipeline that follows the convolutional and activation stages. A typical convolutional block processes the input data through a sequence of operations: feature extraction via convolution, non-linear transformation through an activation function (commonly $ReLU$), and finally spatial dimensionality reduction through pooling.

The **pooling operation** is a form of non-parametric downsampling that systematically reduces the resolution of feature maps while retaining the most informative characteristics. Among various strategies, *max pooling* is the most widely adopted. It partitions the input feature map into fixed-size windows (e.g., 2×2) and selects the maximum value within each window to construct the output. This mechanism enables the network to retain only the most prominent activations in localized regions, effectively highlighting dominant features such as edges, corners, or textures.

From a computational perspective, pooling contributes to reduced spatial dimensionality, which leads to fewer parameters in subsequent layers and thus lowers memory and computational demands. More importantly, it introduces a level of **translational invariance**, allowing the model to recognize patterns regardless of slight shifts or distortions in the input. This property is crucial in real-world applications, such as image recognition, where the same object might appear at different locations or orientations.

Beyond max pooling, alternative strategies such as *average pooling*—which computes the mean value within a local neighborhood—can be employed. While average pooling preserves more contextual information, it may lack the discriminatory power of max pooling, particularly in early layers where salient features need to be emphasized.

Recent advances in CNN design sometimes replace or augment traditional pooling with *strided convolutions* or *global pooling*, depending on task-specific requirements. These variations aim to retain spatial context while achieving similar benefits in computational efficiency and invariance.

Figure 2.7 illustrates the functionality of the max pooling operation. Each colored patch denotes a receptive field in the input feature map, with the largest value from each region selected to form a lower-dimensional representation in the

output tensor.

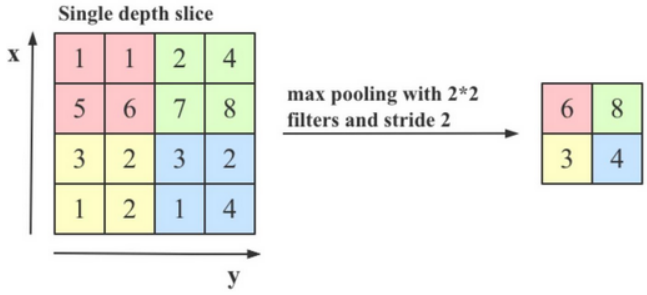


Figure 2.7: Illustration of the max pooling process: the highest activation in each local region is selected to form the downsampled output.

2.7 Conclusion

This chapter has presented a comprehensive and structured analysis of deep learning architectures that underpin modern visual perception systems in autonomous platforms. Beginning with the biological inspiration of artificial neurons, the discussion progressively formalized the mathematical abstraction of neural computation, establishing the conceptual foundation of Artificial Neural Networks (ANNs). The core components of neural architectures—including layers, weights, biases, and activation functions—were examined to highlight their respective roles in enabling non-linear modeling and hierarchical feature learning.

A detailed exploration of activation functions demonstrated their critical influence on network expressiveness, convergence behavior, and training stability. Classical functions such as sigmoid and hyperbolic tangent were discussed alongside modern alternatives including ReLU, Leaky ReLU, PReLU, and ELU, emphasizing their impact on gradient propagation and deep network optimization. The training mechanisms of neural networks were then analyzed through loss functions, backpropagation, gradient-based optimization, and regularization strategies, which collectively ensure generalization and robustness in real-world applications.

The chapter further reviewed major neural network paradigms, ranging from feedforward and recurrent networks to radial basis and self-organizing models, situating each architecture within its functional and computational context. Particular emphasis was placed on Convolutional Neural Networks (CNNs), which represent the cornerstone of visual perception in autonomous systems. Through convolutional and pooling operations, CNNs enable efficient extraction of spatially invariant and semantically rich representations from visual data, making them particularly well-suited for tasks such as object detection, recognition, and scene understanding.

By bridging theoretical principles with architectural design, this chapter establishes the necessary groundwork for understanding advanced perception models deployed in autonomous navigation systems. The concepts and architectures discussed herein serve as the foundation for state-of-the-art object detection frameworks, including YOLO-based models, which will be examined in subsequent chapters. Building upon this theoretical framework, the next chapter focuses on cloud-assisted visual perception and three-dimensional scene representation for Autonomous Guided Vehicle (AGV) navigation, where deep learning models are integrated with RGB-D sensing and spatial clustering techniques to enable robust, real-time decision-making.

Chapter 3

Deep learning for object detection and recognation

Chapter Overview

This chapter focuses on modern deep learning techniques for object detection and recognition in autonomous systems. We explore state-of-the-art detectors, YOLO-based models, evaluation metrics, and practical implementations for real-time obstacle detection using RGB-D sensors. Applications to traffic sign recognition and AGV navigation are also discussed.
Chapter Organization:

Contents

3.1 State-of-the-art

The field of computer vision is broad and extends beyond simple image processing. Its primary goal is to enable computers to comprehend and interpret visual data. Object detection stands as a cornerstone task in computer vision, serving as a foundation for addressing more intricate challenges, including image segmentation, object tracking, and behavior recognition. This process typically involves two main steps: first, accurately localizing potential objects within an image, and second, categorizing these identified objects into distinct classes. Object detection focuses on localized regions of an image and specific object categories. Although Convolutional Neural Networks (CNNs) have been utilized in object detection since the 1990s, early progress was limited by the availability of training data and hardware resources, such as computational power and storage capacity. The breakthrough of CNNs in the 2012 ImageNet challenge reignited interest in deep CNN-based object detection, leading to substantial improvements in detection and recognition rates. Consequently, object detection has found extensive applications across real-world scenarios, including autonomous driving.

Before the advent of deep learning, object detection algorithms primarily relied on manual design and the traditional sliding window approach. Feature extraction techniques, such as Speeded-Up Robust Features (SURF) [5], Histogram of Oriented Gradients (HOG) [4], and Scale-Invariant Feature Transform (SIFT) [3], were commonly employed. Classifiers, including Support Vector Machines (SVMs) [63], were used to train shallow classifiers for individual object classes. However, these traditional approaches exhibited several limitations, including low robustness, poor generalization, and reduced detection accuracy in challenging environments. The main drawbacks stem from two factors: reliance on prior knowledge for feature extraction, which constrains parameter selection and complicates manual tuning, and the shallow nature of classifiers, which necessitates exponentially more parameters and training data for complex detection tasks.

In response to these challenges, Convolutional Neural Networks (CNNs) [2] have gained widespread adoption across various domains for object detection, gradually replacing traditional detection techniques. Deep learning-based models can be broadly categorized into two types: two-stage models and one-stage models. The two-stage model involves object detection in two phases: first, generating region proposals, and then classifying these proposals using a classifier while refining their positions. Noteworthy algorithms in this category include R-CNN [7], SPP-Net [8], Fast R-CNN [9], Faster R-CNN [10], and Mask R-CNN [11]. Conversely, one-stage models directly predict object locations and generate bounding boxes in a single step. Examples of such models include the YOLO algorithm [12], Single Shot MultiBox Detector (SSD) [13], CenterNet [14], and EfficientDet [64].

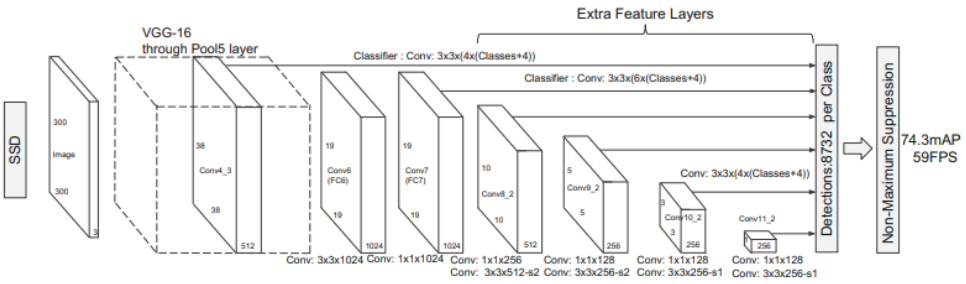


Figure 3.1: Architecture of the SSD model illustrating multi-scale feature prediction.

3.1.1 Single-Stage Object Detectors

Single-stage object detectors have undergone significant evolution, with each successive model introducing architectural innovations and performance optimizations tailored for real-time object detection. Unlike two-stage detectors, which separate the proposal and classification steps, single-stage detectors predict object locations and class labels in a single unified forward pass. This design makes them particularly suitable for real-time applications, such as autonomous driving and robotics. Below is a summary of several notable single-stage architectures, highlighting their key components and contributions to the field.

SSD (Single Shot MultiBox Detector)

The Single Shot MultiBox Detector (SSD) is a pioneering single-stage model that achieves a balance between high detection accuracy and low inference latency. SSD employs a convolutional neural network (CNN) backbone to generate feature maps at multiple scales, enabling robust detection of objects of varying sizes. Each spatial location in these feature maps is assigned a set of anchor boxes, also referred to as default boxes, with different scales and aspect ratios.

For each anchor box, the network simultaneously predicts both the class probabilities and the bounding box offsets. Post-processing is performed using non-maximum suppression (NMS) to remove redundant detections. The SSD detection head, constructed on top of a base CNN (e.g., VGG16), uses a series of convolutional layers to produce dense predictions across all feature levels [65].

The overall SSD architecture, including its multi-scale feature maps and detection head, is illustrated in Figure 3.1.

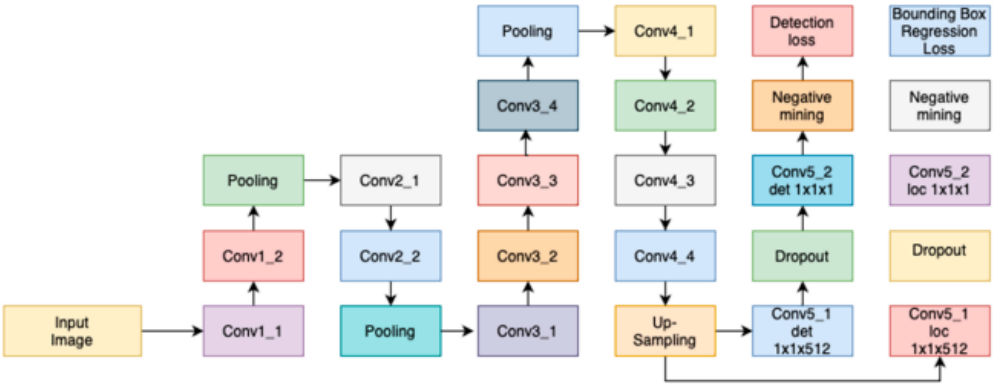


Figure 3.2: Detailed overview of the DenseBox architecture for dense prediction.

DenseBox

Proposed by Huang et al. [66], DenseBox adopts a fully convolutional architecture that merges classification and localization into a single end-to-end pipeline. Unlike SSD or YOLO that rely on pre-defined anchor boxes, DenseBox performs dense predictions at every location across the image grid. This allows for fine-grained object detection, particularly for small or closely clustered objects. After feature extraction using a deep CNN, the network refines bounding box predictions through iterative scoring, followed by non-maximum suppression to finalize results. DenseBox is particularly advantageous in tasks involving face or pedestrian detection due to its ability to densely cover the spatial domain.

RetinaNet

Developed by Lin et al. [67], RetinaNet addresses a key challenge in single-stage detection: the extreme class imbalance between foreground and background samples. To mitigate this, it introduces the **focal loss** function, which down-weights the contribution of easy negatives and focuses the model's learning on hard, informative examples. RetinaNet also employs a **Feature Pyramid Network (FPN)** as its neck, combining semantic-rich features from different levels of the backbone to enhance multi-scale detection. This architectural synergy of FPN and focal loss makes RetinaNet a strong performer in terms of both accuracy and speed.

RFB Net (Receptive Field Block Network)

RFB Net, introduced by Niu and Zhang [68], extends the ideas of SSD by incorporating Receptive Field Blocks (RFBs) to mimic the human visual system's ability to perceive at different scales. The network applies multiple convolutional

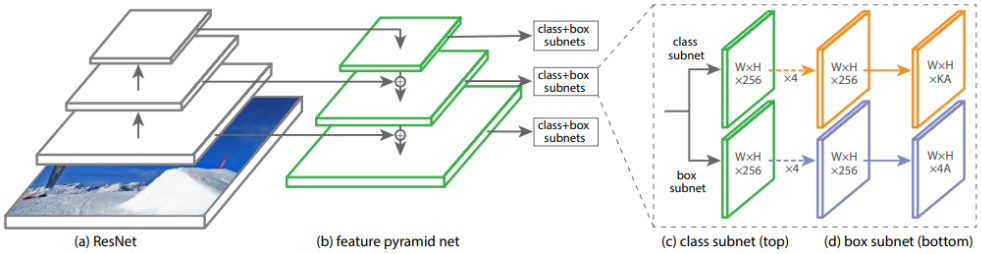


Figure 3.3: RetinaNet framework showcasing the integration of FPN and focal loss.

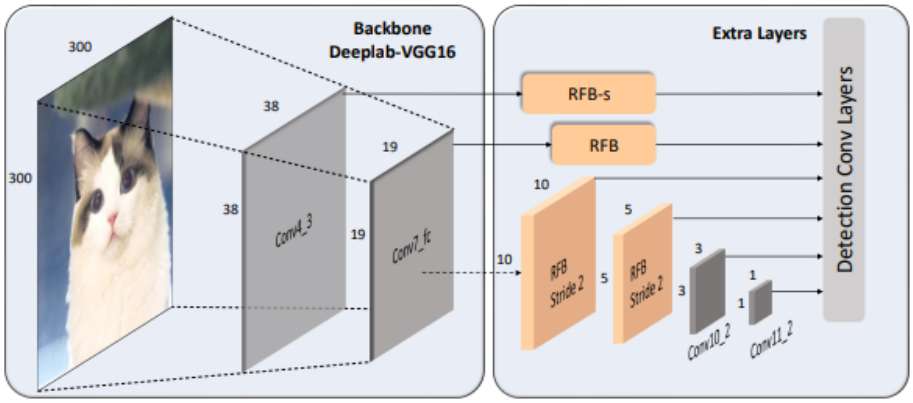


Figure 3.4: Architecture of the RFB Net with Receptive Field Blocks.

filters with varied receptive fields to better capture contextual information. It also integrates anchor boxes of diverse shapes and scales, improving performance on small and irregularly shaped objects. Using the VGG network as its backbone, RFB Net progressively refines both localization and classification outputs through a multi-step process, leading to more precise bounding box predictions.

EfficientDet

Proposed by Tan and Le [69], EfficientDet represents a modern and highly scalable single-stage detector that prioritizes both accuracy and computational efficiency. It leverages a compound scaling method that uniformly adjusts the model's depth, width, and input resolution. At its core is the **EfficientNet** backbone, known for its parameter efficiency, paired with a **Bidirectional Feature Pyramid Network (BiFPN)** that facilitates efficient multi-scale feature fusion. EfficientDet achieves a superior balance between speed and accuracy, making it highly suitable for deployment in edge devices and mobile applications.

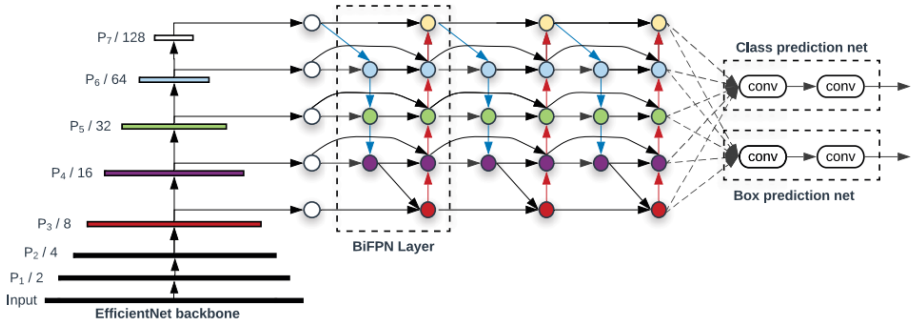


Figure 3.5: EfficientDet model architecture using BiFPN for feature fusion.

YOLO (You Only Look Once) Algorithm

YOLO (You Only Look Once) [70] represents a cutting-edge framework in object detection, distinguished by its unique approach and considerable advantages over traditional detection methods. Unlike classical models that rely on complex multi-stage pipelines, YOLO achieves remarkable inference speed by framing object detection as a single regression problem. Instead of processing multiple region proposals or sliding windows, the network analyzes the entire image in one forward pass, simultaneously predicting bounding boxes and class probabilities. This end-to-end design drastically reduces computational overhead and accelerates detection without sacrificing accuracy.

The YOLO architecture is modular, composed of three fundamental building blocks—*backbone*, *neck*, and *head*—each playing a vital role and evolving across YOLO versions to boost overall performance:

- **Backbone:** Responsible for extracting rich, hierarchical feature representations from input images, the backbone typically utilizes powerful convolutional neural networks pre-trained on expansive datasets such as ImageNet. Popular backbone choices throughout YOLO iterations include architectures like ResNet50, ResNet101, and CSPDarkNet53 [71], which provide a robust foundation for downstream tasks.

- **Neck:** Acting as a bridge between backbone and head, the neck refines and fuses features from different scales and layers. Techniques such as Feature Pyramid Networks (FPN) and Spatial Attention Modules (SAM) are commonly integrated to enhance multi-scale feature aggregation and focus the network's attention on salient regions [72]. This step improves the model's ability to detect objects of varying sizes and complexities.

- **Head:** The final component synthesizes processed feature maps to simultaneously predict bounding box coordinates, objectness scores, and class probabilities. Utilizing multi-scale anchor boxes, the head adapts detection to different object shapes and sizes, ensuring high recall and precision [73].

Since its initial release, YOLO has undergone extensive refinements, from YOLOv1 through to the latest YOLOv11, each version introducing substantial improvements in detection accuracy, processing speed, and computational efficiency. Key advancements include the incorporation of sophisticated loss functions tailored for better bounding box regression and classification, adaptive anchor box mechanisms to handle diverse object scales, and the adoption of lightweight yet powerful backbone networks for rapid feature extraction. Furthermore, improvements in training strategies, such as data augmentation and better optimization algorithms, have enhanced the model's robustness and generalization capability.

YOLO's evolution exemplifies a continuous pursuit to optimize real-time object detection, striking an effective balance between latency and accuracy. Its enduring popularity and state-of-the-art performance make it a benchmark framework in applications demanding high-speed, precise detection, ranging from autonomous vehicles and robotics to surveillance and industrial automation [74].

3.2 YOLO-based Object Detection

YOLO processes the entire image simultaneously, adopting a holistic strategy for both training and inference. Unlike traditional sliding-window or region-proposal techniques, YOLO enhances both accuracy and speed by integrating contextual information about object classes along with their visual features. The approach divides the input image into a grid, and concurrently predicts bounding box coordinates and class probabilities for each cell.

Leveraging a deep Convolutional Neural Network (CNN), YOLO extracts salient features and outputs bounding boxes coupled with class likelihoods. The use of anchor boxes at multiple scales allows the model to effectively detect objects of various sizes.

Each iteration of YOLO introduces architectural and methodological improvements to boost performance. For example, YOLOv4 and YOLOv5 integrate more sophisticated backbone networks and employ advanced loss functions such as Complete Intersection over Union (CIoU) to enhance localization precision. The fundamental YOLO loss function can be expressed as follows:

$$
\begin{aligned}
&\lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 \right] \\
&+\lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[(\sqrt{w_i} - \sqrt{\hat{w}_i})^2 + (\sqrt{h_i} - \sqrt{\hat{h}_i})^2 \right] \\
&+\sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} (C_i - \hat{C}_i)^2 \\
&+\lambda_{\text{noobj}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{noobj}} (C_i - \hat{C}_i)^2 \\
&+\sum_{i=0}^{S^2} \mathbb{1}_{i}^{\text{obj}} \sum_{c \in \text{classes}} (p_i(c) - \hat{p}_i(c))^2
\end{aligned}
\tag{3.1}
$$

Here, $\mathbb{1}_{ij}^{\text{obj}}$ denotes an indicator function that is 1 if the j-th bounding box predictor in cell i is responsible for detecting an object, and 0 otherwise. Similarly, $\mathbb{1}_{i}^{\text{obj}}$ signals whether cell i contains an object.

From its inception with YOLOv1 through to the latest YOLOv11, the model has undergone substantial enhancements aimed at increasing accuracy, inference speed, and computational efficiency. Noteworthy improvements include the introduction of anchor boxes to handle multi-scale detection, refined loss functions for more accurate bounding box regression, and more efficient backbone architectures tailored for faster feature extraction. Each YOLO variant presents unique innovations that continually push the balance between real-time processing and detection precision.

Model	**Year**	**Tasks**	**Contributions**	**Framework**
YOLOv1	2015	Object Detection	Single-stage detector	Darknet
YOLOv2	2016	Object Detection	Multi-scale training	Darknet
YOLOv3	2018	Object Detection	SPP, Darknet-53 backbone	Darknet
YOLOv4	2020	Object Detection	Mish activation, CSPDarknet-53	Darknet
YOLOv5	2020	Detection, Segmentation	Anchor-free, SWISH, PANet	PyTorch
YOLOv6	2022	Detection, Segmentation	Self-attention, anchor-free	PyTorch
YOLOv7	2022	Detection, Tracking	Transformers, E-ELAN	PyTorch
YOLOv8	2023	Detection, Segmentation	GANs, anchor-free	PyTorch

Table 3.1: Overview of YOLO Models

With improved features such as object detection, image classification, and image segmentation, YOLOv8 brought forth even more advancements over its predecessors. One notable enhancement is its anchor-free detection system, which does away with the need for pre-set bounding boxes. YOLOv8 finds the exact center of every object, cutting down on computational overhead and speeding up the process. Five variants of YOLOv8 are available for various tasks: YOLOv8n, YOLOv8s, YOLOv8m, YOLOv8l, and YOLOv8x. While the "n" version is faster and smaller, making it perfect for applications with little resources, the "x" version offers the best accuracy but is slower. We employed the YOLOV8s for this study in order to attain both speed and accuracy in detecting.

The following metrics are used: DFL (Distributed Focal Loss), BCE (Binary Cross Entropy), CE (Cross Entropy), GIoU (Generalized Intersection over Union), DIoU (Distance Intersection over Union), CIoU (Complete Intersection over Union), and MSE (Mean Squared Error).

Model	Bounding Box Regression	Classification	Strengths	Weaknesses
YOLOv1	MSE	BCE	Fast, real-time	Lower accuracy
YOLOv2	SSE	BCE	Improved with anchor boxes	Accuracy issues
YOLOv3	GIoU/DIoU	CE	Improved accuracy	Increased complexity
YOLOv4	CIoU	BCE/Focal Loss	Flexible	Less user-friendly
YOLOv5	CIoU	Focal Loss	Faster, more accurate	Less adaptable
YOLOv6	CIoU/DFL	VariFocal Loss	Efficient	Less research
YOLOv7	CIoU	BCE	Improved accuracy	Higher complexity
YOLOv8	CIoU/DFL	CE	Faster	Case-specific

Table 3.2: Advanced Features of YOLO Models

3.2.1 Traffic Sign Recognition Datasets

Data collection is a critical step in training and testing models, Several datasets are commonly used to evaluate the performance of traffic sign recognition algorithms. These datasets provide a diverse and comprehensive representation of real-world traffic signs, enabling researchers to train and test their models in realistic environments.

German Traffic Sign Recognition Benchmark (GTSRB)

One of the most used datasets in traffic sign recognition research is the German Traffic Sign Recognition Benchmark (GTSRB)[75]. It includes 51,839 high-resolution photos of 43 different types of traffic signs. A vital resource for assessing recognition algorithms, GTSRB is well-known for its sizable dataset, excellent annotations, and realistic scenario modeling. The dataset's preponderance of German

traffic signs, however, would restrict how broadly models trained on it can be applied. Consequently, when similar models are used on traffic signs from different areas, performance might suffer. Despite this drawback, GTSRB's broad coverage and trustworthy annotations make it a benchmark dataset.

Belgium Traffic Sign Dataset (BTSD)

Another well-known dataset in the subject is the Belgium Traffic Sign Dataset (BTSD)[76], which comprises 7,095 high-resolution photos of 62 distinct traffic sign classes from the Netherlands and Belgium. The photographs were gathered from a variety of real-world situations, documenting changes in occlusions, weather, and lighting. BTSD is a useful resource, especially for evaluating models trained on larger datasets, despite being smaller than datasets like GTSRB. It is a valuable dataset for academics working on traffic sign identification algorithms for Belgium and the Netherlands because of its high-quality photos and variety of climatic situations.

Chinese Traffic Sign Database (TSRD)

The National Natural Science Foundation of China (NSFC) provided assistance in the development of the extensive dataset known as the Chinese Traffic Sign Database (TSRD). It has 6,164 photos divided into 58 groups of traffic signs. In order to ensure a varied portrayal of weather, lighting fluctuations, and environmental settings, the photographs were collected from both real-world camera setups and BAIDU Street View.

Table 3.3: Comparison of Traffic Sign Recognition Datasets

Dataset	Number of Classes	Total Images
GTSRB	43	51,839
BTSD	62	7,095
TSRD	58	6,164

3.2.2 Evaluation metrics

The model's effectiveness is assessed using common object detection evaluation metrics such as mAP@50, mAP@95, and the F1 score.

Precision and **Recall** are mathematically expressed as:

$$\text{Precision} = \frac{TP}{TP + FP}, \quad \text{Recall} = \frac{TP}{TP + FN} \tag{3.2}$$

where:

- TP denotes true positives, i.e., correctly identified positive instances,
- FP indicates false positives, instances wrongly predicted as positive,
- FN corresponds to false negatives, actual positives that were missed.

The **F1 score** for the k-th class is calculated by combining precision and recall as:

$$F1_k = \frac{2 \times \text{Recall}_k \times \text{Precision}_k}{\text{Recall}_k + \text{Precision}_k} \tag{3.3}$$

and the overall F1 score across all n classes is obtained by averaging:

$$F1 = \frac{1}{n} \sum_{k=1}^{n} F1_k \tag{3.4}$$

Average Precision (AP) is computed as:

$$AP = \sum_{i=1}^{m} (R_i - R_{i-1}) \cdot P_{\text{interp}}(R_i) \tag{3.5}$$

where R_i represents the recall value at the i-th threshold, and $P_{\text{interp}}(R_i)$ is the interpolated precision corresponding to R_i.

The **Mean Average Precision (mAP)** metric averages AP over all classes:

$$mAP = \frac{1}{n} \sum_{k=1}^{n} AP_k \tag{3.6}$$

Here, **mAP@50** signifies the mAP computed at an Intersection over Union (IoU) threshold of 0.5, while **mAP@95** represents the average mAP evaluated over multiple IoU thresholds ranging from 0.5 to 0.95 in increments of 0.05.

The **Intersection over Union (IoU)** measure is defined as:

$$IoU(bb_p, bb_{gt}) = \frac{bb_p \cap bb_{gt}}{bb_p \cup bb_{gt}} \tag{3.7}$$

where bb_p is the predicted bounding box and bb_{gt} is the ground truth bounding box corresponding to the object.

These metrics collectively provide a comprehensive evaluation of the detection model's accuracy, precision, and robustness in identifying and localizing objects within images.

3.3 Google Cloud Vision API

The Google Cloud Vision API is a robust machine learning service provided by the Google Cloud Platform, enabling the integration of advanced image analysis capabilities into various applications. This API leverages state-of-the-art machine learning models to interpret the content of images and extract meaningful information. Its core functionalities include:

- **Object Detection:** Identifies and localizes multiple objects within an image, providing information about their presence and positions.
- **Labeling:** Assigns descriptive tags to images by recognizing various entities, scenes, and concepts depicted in the content.
- **Facial Recognition:** Detects faces and analyzes facial features, including expressions, landmarks, and emotions.
- **Explicit Content Detection:** Identifies and categorizes inappropriate or explicit content to ensure safer user experiences.
- **Logo Detection:** Recognizes logos in images and provides metadata related to detected brands.
- **Landmark Recognition:** Identifies famous landmarks, monuments, and notable locations present in images.

Google Cloud Vision API provides a cloud-based solution for image understanding, thereby minimizing the reliance on local hardware resources for computationally intensive tasks. At the core of this technology is the concept of **Vision AI**, which enables machine learning models to interpret and analyze visual content in a way that mimics human vision. This encompasses tasks such as *image classification*, *object recognition*, and *image analysis*.

The **Google Cloud Platform (GCP)** offers several services tailored for visual data processing within machine learning workflows. Notably, the **AutoML Vision API** allows users to train custom models with minimal expertise, while the **Google Vision API** provides access to a pre-trained model capable of identifying objects and extracting insights from images. These services are widely utilized within the GCP ecosystem for automating visual recognition tasks and enhancing the efficiency of intelligent applications.

3.4 YOLOV8 for traffic sings detection

3.4.1 YOLOV8 for Bounding Box and Identification

YOLOv8 introduces several architectural advances, including a pooling layer and a streamlined convolutional architecture to optimize computational efficiency. This model predicts three dimensional tensors comprising bounding box coordinates, confidence scores, and class identifiers, which are essential for robust object detection. YOLOv8 builds on the design principles of YOLOv5 and YOLOv7 ELAN to improve performance and flexibility. Its updated architecture includes a redesigned backbone network, an anchor-free detection head, and a novel loss function.

For distance estimation, YOLOv8 relies on depth data obtained from the RGB-D camera. However, the accuracy of depth measurements may vary due to external factors, such as lighting conditions and surface properties. To validate the depth estimation's reliability, tests were carried out under normal conditions. The results, compared to actual physical measurements, demonstrate the model's robustness in object detection and localization tasks.

3.4.2 Training the YOLOV8 Model

A critical phase in validating the object detection system involved the comprehensive training of the YOLOv8 model, which established a performance benchmark before real-world deployment. For this purpose, a custom dataset comprising 8,316 annotated images was utilized. The dataset featured 15 distinct road sign categories and was systematically partitioned into training, validation, and testing subsets to support a robust evaluation framework.

YOLOv8, the latest iteration in the "You Only Look Once" family of object detectors, integrates significant architectural and algorithmic improvements, enhancing its suitability for both object localization and distance-aware detection. The training process leveraged these enhancements to optimize model performance in various traffic environments.

Key Training Outcomes

1. **Model Accuracy**
   The trained model achieved an impressive overall classification accuracy of **92.6%**, confirming its reliability in detecting a diverse set of traffic signs with minimal misclassification.

2. **Mean Average Precision (mAP@0.5)**
   The model attained a **mean Average Precision (mAP) of 95%** at an

IoU threshold of 0.5, demonstrating competitive performance when benchmarked against state-of-the-art detection algorithms. This high mAP value confirms the model's effectiveness in identifying object boundaries and categories across varied conditions. The progression of training metrics is visualized in **Figure ??**, which reflects the model's learning stability and precision.

3. **Loss Convergence and Optimization Efficiency**
   The training loss curve exhibited rapid and stable convergence, indicating that the optimization algorithm effectively minimized error over iterations. This also reflects the high quality and consistency of the annotated training data.

Figures 3.6–3.7b illustrate the training dynamics, validation behavior, and detection performance of the proposed YOLOv8-based perception model.

As shown in Figure 3.6, the training losses, including bounding box regression loss, classification loss, and distribution focal loss, consistently decrease throughout the training epochs. This behavior indicates stable optimization and effective learning of both spatial localization and semantic classification features. The smooth convergence of these losses confirms that the model successfully captures the underlying distribution of the RGB-D traffic sign dataset.

This observation is further supported by the quantitative training results, where the model achieved a training accuracy of **95.73%** with a final training loss of **0.1379**. These values demonstrate strong learning capacity without signs of unstable convergence.

Figure 3.7a presents the corresponding validation losses, which closely follow the training loss trends. The validation performance reaches an accuracy of **99.35%** with a minimal loss of **0.0202**, while the test set yields an accuracy of **99.63%** and a loss of **0.0184**. The small discrepancy between training, validation, and testing metrics indicates excellent generalization capability and negligible overfitting, confirming the robustness of the learned representations.

From a detection performance perspective, Figure 3.7b highlights the evolution of precision, recall, and mean Average Precision metrics during training. Precision and recall rapidly increase and stabilize at high values, reflecting the model's ability to accurately detect traffic signs while minimizing both false positives and false negatives. Quantitatively, the detector achieves a precision of **93.45%**, a recall of **94.28%**, and an F1-score of **93.22%**, demonstrating a well-balanced trade-off between sensitivity and specificity.

Moreover, the high mAP@0.5 and consistent mAP@0.5:0.95 values shown in Figure 3.7b confirm reliable detection performance across varying IoU thresholds, which is critical for precise localization in autonomous navigation scenarios.

Collectively, the qualitative trends observed in Figures 3.6–3.7b, combined with the strong quantitative metrics, validate YOLOv8's capability to deliver accurate,

stable, and real-time object detection. The consistency across training, validation, and testing phases, along with high detection accuracy, makes the proposed approach particularly suitable for embedded deployment in AGV and autonomous robotic systems operating in structured indoor environments.

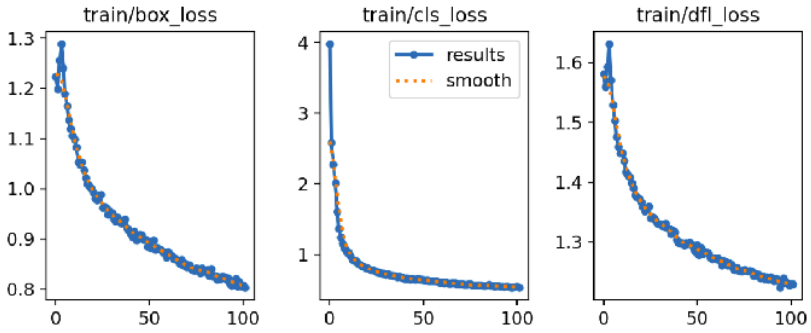
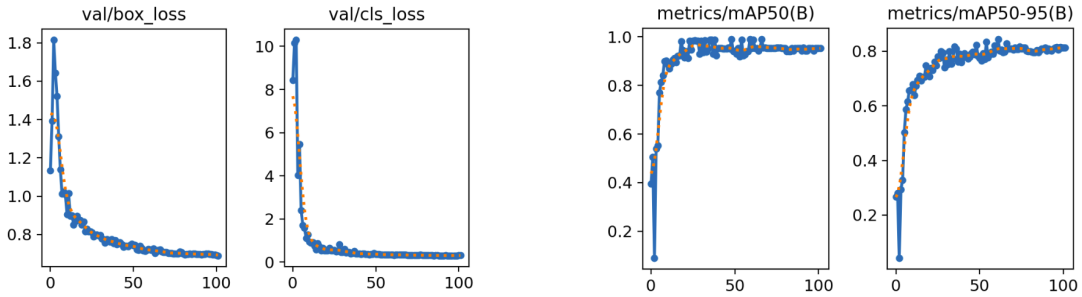


Figure 3.6: Train loss curves



(a) Validation loss curves

(b) Detection performance metrics

Figure 3.7: YOLO training convergence and detection performance

3.5 Real-Time Obstacle Detection and Avoidance for AGVs using YOLOv8 and RGB-D Sensors

3.5.1 System architecture

The design of a perception system can vary depending on the specific application and requirements. As illustrated in Figure 1 [77], the primary sensor used for data acquisition is an RGB-D camera, which captures both RGB and depth images. The depth images are particularly useful for obstacle identification, as they allow

the system to detect barriers by analyzing variations in distances. This information is essential for tasks such as path planning and collision avoidance. Additionally, the depth image can calculate the distance to objects within a defined area, such as within a bounding box. By analyzing depth data in the designated region, the system determines how far an object is from the camera, a capacity critical for applications such as grasping or object manipulation, where distance awareness is crucial. On the other hand, the RGB images captured by the camera are processed using the YOLO deep learning algorithm. This algorithm can recognize and classify objects in real time after being trained on extensive datasets. Consequently, the perception system performs object recognition tasks by identifying and labeling various items in the environment. The RGB image also facilitates the creation of bounding boxes around detected objects. These bounding boxes define the size and boundaries of recognized objects, allowing precise localization and tracking. Furthermore, the system can estimate an object's size or dimensions based on the bounding box, making it suitable for applications that require object measurements or size-based analysis.

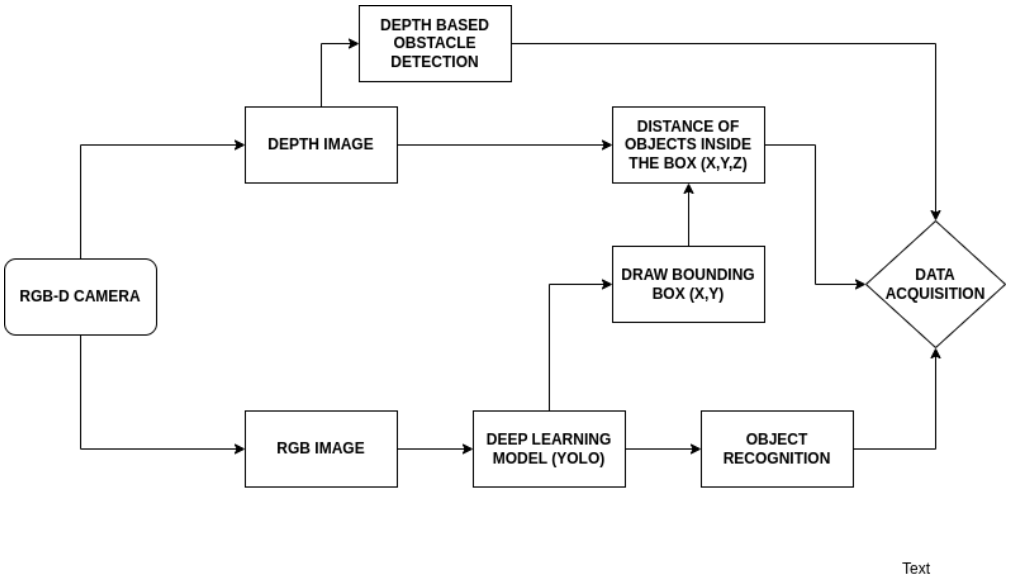


Figure 3.8: System architecture

In summary, the perception system integrates RGB and depth data to enhance its capabilities. RGB images, processed by YOLOv8, used for object recognition and bounding box generation, while depth images provide critical information for obstacle detection and distance estimation. Together, these components allow the perception system to collect and interpret data during the acquisition phase effectively. In this implementation, RGB and depth images were captured using

Microsoft Kinect V1, improving the system's ability to perform real-time perception and navigation tasks.

3.5.2 Experimental results

The first condition involved an experiment using the YOLOV8 object detection model, which achieved successful object detection outcomes. During the experiment, the YOLOV8 model demonstrated its ability to detect objects accurately within captured images. This model proved to be effective in identifying and localizing various objects of interest. Leveraging the advanced algorithms and architecture of YOLOv8, the system achieved satisfactory results in terms of object detection accuracy and speed. The model's ability to handle real-time object detection tasks ensured timely and reliable identification of objects in the given context. The successful performance of the YOLOv8 model confirms its suitability for object detection applications, highlighting its potential in various scientific and practical domains.

First Experiment Results (Figure 3.9)(Figure 3.10)

- The experiment utilized input from a Kinect camera, which provided visual data for further analysis and processing.
- The distance to the stop sign was accurately measured and recorded during the experiment. Utilizing the depth information obtained from the Kinect camera, the system was able to calculate the precise distance between the robot and the stop sign.
- The speed of the Pioneer 3DX robot was carefully controlled and monitored throughout the experiment. By adjusting the robot's velocity parameters, the system ensured controlled and consistent movement during data collection and subsequent analysis.
- After the robot was set in motion, new inputs were captured by the Kinect camera, providing updated visual data to the system. These new inputs allowed the system to continuously perceive the environment and adapt its operations accordingly.
- The updated distance from the stop sign was recalculated based on the new inputs received after the robot's movement. This allowed for real-time assessment and tracking of the distance between the robot and the stop sign as the robot traversed its environment.

- The speed of the Pioneer 3DX robot was dynamically adjusted in response to changing conditions and input. This allowed the system to regulate the robot's movement speed, ensuring safe and efficient navigation while maintaining the desired experimental parameters.

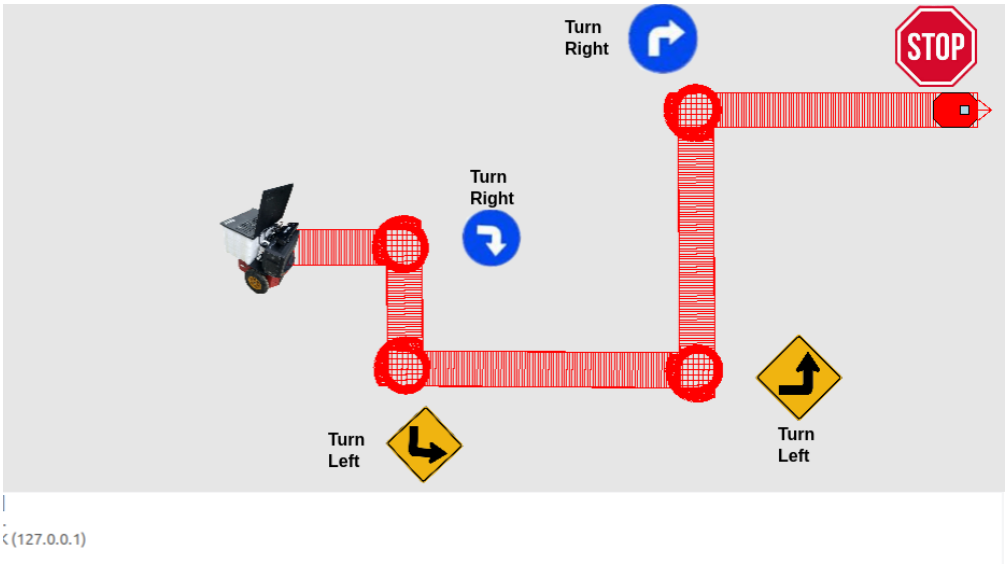


Figure 3.9: Experimental Mobile Robot Trajectories in ARIA-MobileSim

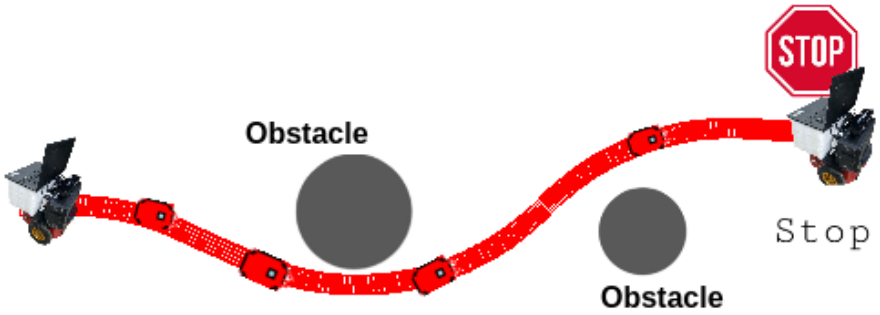


Figure 3.10: Experimental Obstacle Avoidance and Navigation Results in ARIA-MobileSim

Second Experiment Results

In this experiment, the system was tested against fast-moving objects and walls that cannot be detected by YOLOv8, as well as in low light conditions. The robot was kept safe and avoided endangering people or the environment.

In the experimental setup, the robot's behavior was programmed to respond to the proximity of obstacles within a range of 0.6 m. When an obstacle approached within this specified range, the robot initiated a stop command and a turning motion to find a clear path, as shown in Figure 3.11. This adaptive strategy aimed to facilitate obstacle avoidance, particularly in the context of navigating near walls.

Using this reactive approach, the robot effectively stopped its forward movement when it detected an obstacle within the predetermined range, as shown in Figure 3.11. This temporary pause allowed the robot to assess the environment and determine the optimal direction for evasive action. By actively searching for an unobstructed path through the turning motion, the robot aimed to identify a viable route to resume its intended trajectory.

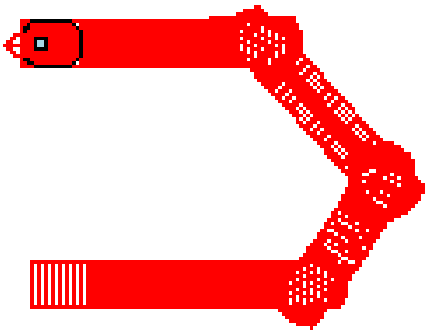


Figure 3.11: SExperimental Wall-Avoidance and Turning Behavior in ARIA-MobileSim.

3.6 Conclusion

3.7 Conclusion

This chapter examined the fundamental concepts and components of deep learning and computer vision, with particular emphasis on their application to object detection and recognition in autonomous robotic systems. The theoretical foundations of convolutional neural networks (CNNs) were introduced as the backbone of modern visual perception systems, highlighting their capacity to automatically learn hierarchical feature representations from visual data.

A comprehensive review of state-of-the-art object detection techniques was conducted, with a focus on **single-stage detectors** due to their suitability for real-time applications. Classical methods based on handcrafted features such as HOG, SIFT, and SURF were discussed to emphasize the limitations of traditional approaches. Subsequently, modern deep learning-based detectors were analyzed, including SSD [65], DenseBox [66], RetinaNet [67], RFB Net [68], and EfficientDet [69]. These methods were compared in terms of architectural design, detection accuracy, computational efficiency, and robustness to scale variation.

Among these techniques, particular attention was devoted to the **YOLO (You Only Look Once)** family of algorithms [12, 74, 70], which formulates object detection as a single regression problem and enables high-speed inference. The evolution of YOLO from early anchor-based versions to recent anchor-free designs was discussed, culminating in the selection of **YOLOv8** as the primary detection framework adopted in this research. YOLOv8 was chosen due to its anchor-free detection head, efficient backbone architecture, and advanced loss functions such as CIoU and Distributed Focal Loss (DFL), which jointly improve localization accuracy and convergence stability.

In addition to local detection, cloud-based vision techniques were introduced through the **Google Cloud Vision API**, which provides pre-trained deep learning models for object detection, labeling, and visual understanding. This approach highlights an alternative paradigm where computationally intensive perception tasks can be offloaded to cloud infrastructure, reducing onboard processing requirements.

To support reliable evaluation, standard object detection metrics were employed, including Precision, Recall, F1-score, Intersection over Union (IoU), Average Precision (AP), and Mean Average Precision (mAP@0.5 and mAP@0.5:0.95). These metrics provided a quantitative framework to assess detection accuracy, robustness, and generalization across datasets.

The chapter further demonstrated the practical adoption of these techniques through the integration of **YOLOv8 with RGB-D sensing**. RGB images were

used for semantic object recognition and bounding box generation, while depth data from the RGB-D camera enabled distance estimation and obstacle awareness. Experimental results confirmed that the proposed perception pipeline achieves high detection accuracy, stable convergence during training, and reliable real-time performance in structured environments.

Overall, this chapter establishes a solid methodological foundation for the perception layer of the proposed AGV system. By combining state-of-the-art deep learning techniques for object detection (YOLOv8), standardized evaluation metrics, and RGB-D sensor fusion, the chapter provides the necessary tools and justification for the navigation and decision-making strategies developed in the subsequent chapters.

Chapter 4

Cloud Vision-Based AGV Navigation

Chapter Overview

This chapter presents cloud-based vision approaches for AGV navigation. We discuss the integration of RGB-D sensors with point cloud processing, object clustering (DBSCAN), intensity-based methods, and 3D object representation for real-time navigation. The system architecture and practical implementation strategies are also highlighted.

Chapter Organization:

Contents

4.1 Introduction

In the past decade, numerous techniques have been developed and applied in AGV navigation. In [78], these approaches were classified into two main types: local navigation, also referred to as conventional navigation, and global navigation, often relying on heuristic methods for pathfinding. In [79] , an AGV motion control system was proposed based on magnetic tape navigation, where a magnetic tape on a floor serves as the guide. This system is notable for its ease of implementation, low cost, and high efficiency. Similarly, in [80], a navigation system was proposed for AGVs, using metal-line sensors for track guidance and RFID tags for localization. In [81], an AGV navigation approach used double magnetic nails, achieving reliable path tracking and stable accuracy through a fuzzy controller. In [82], a two-wheeled AGV with a laser-based (LIDAR) obstacle avoidance system was developed. Graph-based heuristic algorithms have also gained prominence in AGV navigation, particularly for pathfinding tasks [83]. Popular algorithms include A* (A-star), D* Lite, and Dijkstra's algorithm [84]. These methods have shown effectiveness in various environments, further solidifying their role in autonomous navigation. In recent years, image-based navigation and object detection have emerged as critical areas of research [85, 86, 87]. In [88], the Microsoft Kinect sensor was integrated with the YOLO algorithm for object detection and classification, using the depth sensor to measure object distances. Although effective, this approach relied heavily on object distance estimation and YOLO-based detection, which posed limitations. In [89], a monocular method was proposed that combined object detection and distance estimation using an R-CNN-based regression network. In [90], a collision warning system was capable of identifying specific objects and generating alerts when they were within hazardous proximity. This system relied on monocular cameras and deep learning models, but its performance was highly dependent onenvironmental factors such as lighting and weather conditions. Additionally, it required extensive training data and achieved only moderate accuracy (60%) in distance estimation. In contrast to these approaches, the proposed system leverages both RGB and depth sensors, enabling operation in diverse environments. It provides accurate distance measurements and precise object positions, allowing better navigation control. The detection of traffic signs has been an important focus in autonomous navigation systems, given their critical role in ensuring safe and efficient driving. In [91], an improved Faster R-CNN was used for traffic sign detection. This study enhanced recognition speed by simplifying the Gabor wavelet. through a regional suggestion algorithm. Similarly, in [92], Faster R-CNN was used as the base detection model while Generative Adversarial Networks (GANs) were incorporated for data augmentation, improving the robustness of the system under varying conditions. In [93], the Libra R-CNN algorithm was combined with a balanced feature pyramid to im-

prove the detection of traffic signs. This approach allowed the system to handle challenges such as occlusions and variations in sign appearance, demonstrating its effectiveness in diverse scenarios .

4.2 Robot Operating System

The Robot Operating System (ROS) [94] is an open-source middleware framework designed to streamline the development of robotics software.ROS serves as a basic software layer for robotics by providing tools and services that facilitate hardware abstraction, device driver management, and inter-process communication, despite not being an operating system in the traditional sense. Because of its adaptable and modular design, developers may effectively build and reuse code on a variety of robotic systems. ROS makes it easier to integrate and coordinate robotic operations by serving as a middleware layer that enables smooth communication between sensors, actuators, and software modules.The ability of ROS to abstract hardware specifics enables programmers to create code that is easily transferable between various robotic platforms, which is one of its main advantages. In addition to speeding up development, this abstraction promotes code reuse across a range of robotic applications. The communication architecture used by ROS is founded on the client-server and publish-subscribe concepts. This enables effective information sharing between distributed software components, or nodes. Nodes can broadcast information, like sensor readings, to particular topics, and other nodes can subscribe to these topics to get the pertinent information. Complex robotic systems benefit from increased modularity and scalability because to this asynchronous communication approach. Additionally, ROS comes with a full array of tools for analysis, simulation, visualization, and troubleshooting. For instance, Gazebo is a potent simulation environment, while Rviz is used to visualize sensor data and robot states. Additionally, command-line tools make it easier to administer nodes, topics, and services.

4.3 3D Object representation

4.3.1 RGB-D Point Cloud

Point clouds represent three-dimensional data structures composed of numerous spatial measurements. Each point typically contains Cartesian coordinates (X, Y, Z), and may also include supplementary attributes such as color, surface intensity, and orientation. These data points collectively capture the shape and structure of real-world environments or objects. Once collected, point cloud data undergoes a multi-stage processing workflow that includes importing into specialized

software, filtering to eliminate outliers and noise, aligning multiple scans through registration, and ultimately modeling or analyzing the resulting unified dataset. Due to their high level of spatial accuracy, point clouds are integral in diverse domains ranging from robotics and construction to autonomous navigation and digital preservation.

A primary tool for acquiring point clouds in real time is the RGB-D camera, a depth-sensing device capable of simultaneously capturing both color (RGB) and depth (D) information. This synergy enables direct 3D reconstruction of scenes. One of the pioneering commercial RGB-D devices was the Microsoft Kinect, which democratized access to range-sensing technology and spurred a surge in research and development by offering cost-effective, reliable tools for both static and dynamic 3D data capture [95, 96].

RGB-D sensing techniques can be broadly categorized into passive and active methods. Passive depth sensing often leverages stereo vision, where depth is inferred using geometric triangulation between two or more spatially separated cameras (monochrome or color), without requiring additional illumination [97]. On the other hand, active methods introduce structured patterns or emit light pulses to measure depth directly. Structured Light (SL) sensors, for instance, project known patterns onto a scene, making it easier to estimate depth even in textureless regions. Time-of-Flight (ToF) sensors determine depth by calculating the time it takes for emitted light to bounce back to the sensor.

Both SL and ToF approaches produce a depth map—an image where each pixel encodes distance information, often derived from infrared data. These depth maps are synchronized with RGB images to generate colored 3D reconstructions. Using pixel-to-point correspondence and intrinsic camera parameters, each depth pixel can be projected into a 3D point, forming the basis of the point cloud.

In real-time applications, accurate camera calibration is critical to ensure correct spatial alignment and to mitigate systematic errors. Once calibrated, the system detects and matches features between consecutive frames to estimate motion and scene structure. This frame-to-frame correspondence allows for incremental construction of a sparse point cloud. Subsequently, these local coordinates are transformed into a global reference frame using co-linearity constraints, yielding a denser point cloud whose resolution depends on the frame count and motion path.

Furthermore, by merging depth information with RGB data, a colored 3D model is formed. The end result is a dense, textured point cloud that effectively captures both the geometry and appearance of the scanned environment [98, 99].

Mathematically, a point cloud P is defined as a set of 3D vectors:

$$P = \{p_i \mid i = 1, 2, \ldots, N\}, \quad \text{where } p_i \in \mathbb{R}^3$$

This collection serves as a discrete, unordered, and permutation-invariant approximation of a continuous 3D surface. Figure 4.1 illustrates an example of such an

RGB-D point cloud, showing the distribution of 3D points in space and how RGB information is associated with each point to generate a colored 3D representation.

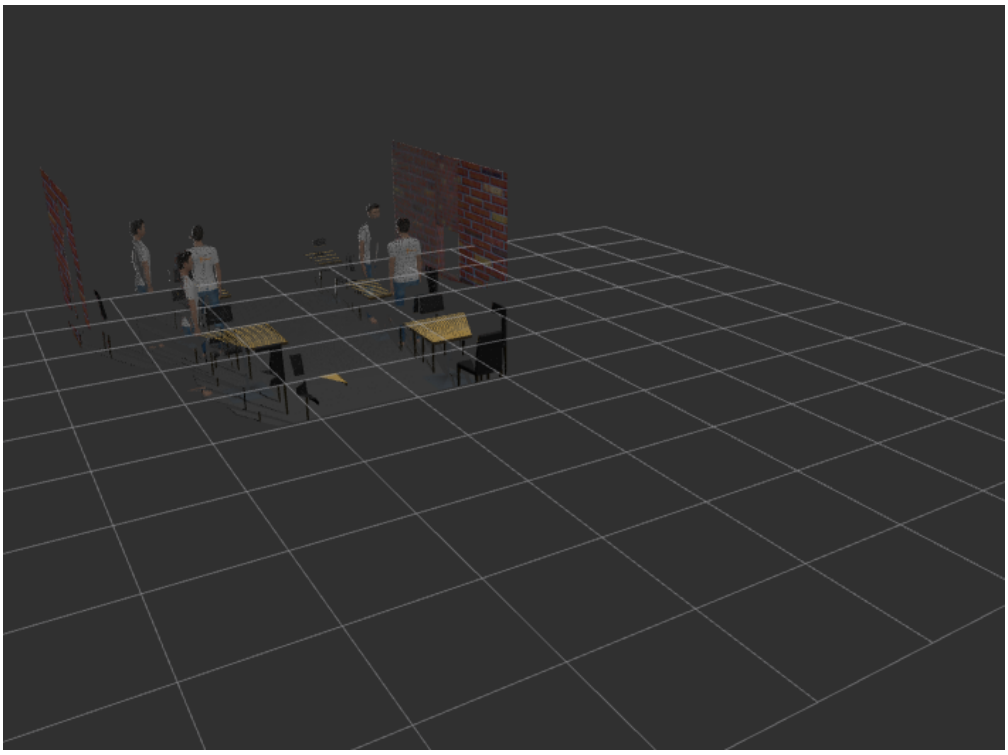


Figure 4.1: RGB-D Point cloud representation.

4.3.2 Radius-based point cloud filtering

Radius-based filtering is a method to retain only points within a specified distance from a reference center, often the origin. For each point $p_i = (x_i, y_i, z_i)$, the Euclidean distance from the center is calculated as:

$$d_i = \sqrt{x_i^2 + y_i^2 + z_i^2}$$

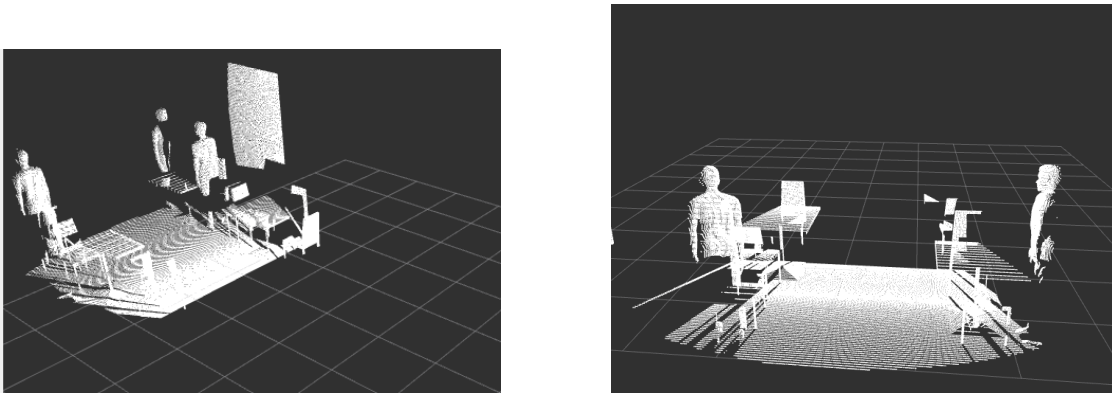
A threshold radius r is defined, and only points satisfying the condition:

$$d_i \leq r$$

are retained. This approach effectively isolates points located within a spherical region of interest and removes distant points.

Figure 4.2 illustrates an example of the RGB-D point cloud after applying radius-based filtering. Only points within the specified radius from the sensor

origin are retained, while more distant points are discarded, clearly showing the spherical region of interest.



(a) (b)

Figure 4.2: Filtered Point Cloud

As shown in Algorithm 2, the processing pipeline begins by initializing a ROS node responsible for subscribing to incoming point cloud messages and publishing filtered and transformed point clouds. A publisher is created to output `PointCloud2` messages so that downstream modules, such as navigation or mapping, can consume the processed data.

A `tf2` listener is also initialized to maintain the transformations between the sensor frame and the global `"map"` frame. These transformations are essential for aligning the point cloud in a consistent global reference, which is critical for tasks like obstacle avoidance .

Once a point cloud message is received, it is first converted from ROS format to an Open3D point cloud. Open3D provides efficient tools for 3D data processing, allowing points to be filtered based on a (detection radius of 2 meters). Only points within this radius are retained, which reduces computational load and focuses processing on relevant areas around the AGV.

The filtered point cloud is then converted back to ROS format. The algorithm attempts to look up the transformation from the sensor frame to the global map frame. If the transformation is available, it is applied to align the point cloud with the global reference. If the transform is unavailable, a warning is logged to ensure robust execution. Finally, the transformed point cloud is published to the appropriate ROS topic.

This loop continues indefinitely using `rospy.spin()`, enabling real-time processing of point cloud data. By combining radius-based filtering with global transformation, the algorithm ensures that only relevant points in the environment are

Algorithm 2 Point Cloud Transformation and Filtering Algorithm

```
Initialize ROS node
Initialize publisher to publish filtered and transformed PointCloud2 messages
Initialize tf2 listener and buffer
While running:
    When a PointCloud2 message is received:
    Convert the ROS point cloud to Open3D point cloud
    Filter points based on a detection radius of 2 meters
    Convert the filtered Open3D point cloud back to ROS PointCloud2 format
    Try to:
        Lookup the transform from the point cloud's frame to the "map" frame
        Transform the filtered point cloud to the "map" frame
    Catch:
        If transform is unavailable, log a warning
    Publish the transformed point cloud to the appropriate topic
Continue this process indefinitely using rospy.spin()
```

considered while maintaining consistent spatial alignment, which is essential for navigation and mapping in autonomous systems.

4.3.3 DBSCAN (Density-Based Spatial Clustering of Applications with Noise)

An unsupervised clustering approach called DBSCAN is especially helpful for datasets that contain noise and high-density regions. The radius ε (Eps-neighborhood of a point p) and MinPts (the minimum number of points in an Eps-neighborhood of p required to make p a core point of a cluster) are the two global input parameters that DBSCAN requires. A cluster is a maximal collection of points connected by density; in other words, the average density of points inside a cluster is significantly higher than that outside of it.

The neighborhood $N_\varepsilon(x_i)$ of the i^{th} object is defined as the set of objects x_j that are within a distance ε from x_i:

$$N_\varepsilon(x_i) = \{x_j \mid D(x_j, x_i) < \varepsilon\}$$

where $D(x_j, x_i)$ denotes the Euclidean distance:

$$D(x_j, x_i) = \sqrt{(x_j - x_i)^2 + (y_j - y_i)^2}$$

The number of objects in the neighborhood $N_\varepsilon(x_i)$ is denoted by $C(N_\varepsilon(x_i))$, and it represents the local density $\rho(x_i)$ of the i^{th} object:

$$\rho(x_i) = C(N_\varepsilon(x_i))$$

The second key parameter is MinPts, which distinguishes three types of objects: core objects, boundary objects, and noise. Core objects are defined as:

$$\rho(x_i) \geq \text{MinPts}$$

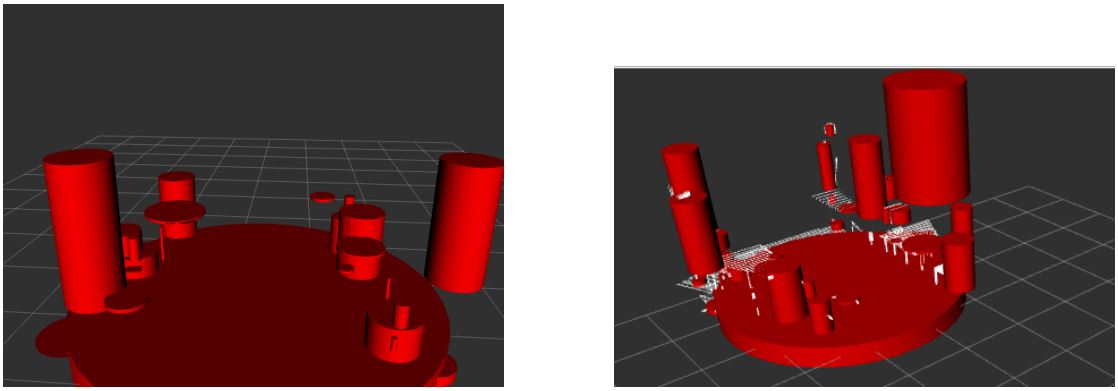
Objects satisfying this condition are labeled as core objects x_{core}. Boundary objects satisfy:

$$\rho(x_i) < \text{MinPts} \quad \text{and} \quad x_i \in N_\varepsilon(x_{\text{core}})$$

and are labeled as x_{border}. Noise objects are defined as:

$$\rho(x_i) < \text{MinPts} \quad \text{and} \quad x_i \notin N_\varepsilon(x_{\text{core}})$$

These relationships are illustrated in Figure 4.3.



(a) Core points

(b) Border and noise points

Figure 4.3: Density-Based Spatial Clustering illustrating core, border, and noise points.

The DBSCAN algorithm introduces two key concepts: *directly density-reachable* and *density-reachable*. A point x_j is *directly density-reachable* from another point x_i if it falls within the ε-neighborhood of x_i:

$$D(x_j, x_i) < \varepsilon$$

Points outside the ε-neighborhood of a core object x_i but within the ε-neighborhood of other core objects are still considered *density-reachable* from x_i. Based on

these definitions, each core object x_i, along with all points that are density-reachable from it (including directly density-reachable points), collectively form a cluster [100, 101].

Cluster Detection Pipeline

Algorithm 3 implements the DBSCAN-based cluster detection for point clouds captured by the RGB-D camera.

Algorithm 3 Cluster Detection Algorithm

```
Initialize ROS node and set up subscriber for PointCloud2 topic
Initialize visualization publisher for cluster markers
Upon receiving a PointCloud2 message:
        Convert it to Open3D point cloud format
        Downsample using uniform grid
        Remove non-finite points
if SEGMENTATION enabled then
    for up to 5 times do
                Remove planar components using RANSAC if plane > 80 points
    end for
end if
if points < threshold (e.g., 10) then
    Exit callback
end if
Apply DBSCAN clustering using EPS_CLUSTER and MIN_POINTS_CLUSTER
if no clusters found then
    Exit callback
end if
for each cluster D_k do
            Compute bounding box and centroid
            Publish Marker (CYLINDER) at centroid
end for
Clear old markers (DELETEALL)
```

As described in Algorithm 3, the pipeline begins by initializing a ROS node subscribing to the `PointCloud2` topic. A publisher is created for visualization markers to display detected clusters in RViz. Pre-processing includes downsampling and planar segmentation using RANSAC. After DBSCAN clustering, centroids and bounding boxes are calculated, and markers are published for visualization.

Centroid Marker Aggregation

To provide a temporally coherent and structured representation for downstream modules, individual centroid markers are aggregated using Algorithm 4.

Algorithm 4 Centroid Objects Marker Collection Algorithm

Initialize ROS node
Subscribe to `/centroid_objects` for individual Marker messages
Initialize publisher to publish MarkerArray messages to `/centroid_objects_list`
Maintain a global MarkerArray list
While running:
    Upon receiving a Marker:
if contains points **then** Append to MarkerArray list
end if
if empty marker (end of batch) **then**
    Publish MarkerArray
    Clear list
end if
Continue with `rospy.spin()`

Algorithm 4 ensures efficient collection and publishing of all detected object centroids as a single MarkerArray, which is essential for AGV navigation, obstacle avoidance, and downstream processing such as tracking or sensor fusion.

4.4 Object intensity density

The distinct color information contained in each extracted image is determined by the colors of the objects within the image. Each object for the same classes ID will provide a distinct colored image. each image will have a different grayscale representation. The grayscale conversion process retains some characteristics of the original color image, While grayscale conversion removes color information, it retains certain visual aspects related to color intensity, allowing some characteristics of the original image to persist in the grayscale representation :- Color Distribution: If an image has certain colors dominating specic areas, the resulting grayscale image will recreat varying shades of gray based on the dominance of those colors in those areas.-Contrast and Brightness: The contrast and brightness of the original color image influence the grayscale conversion. Areas of high contrast or brightness in the color image will often translate to more pronounced differences

in the grayscale version.-Color Combinations: Different combinations of colors in the original image will result in distinct grayscale representations.

4.4.1 Lightness Method

The Lightness method[102] computes the grayscale value as the average of the maximum and minimum of the red (R), green (G), and blue (B) components:

$$\text{Grayscale} = \frac{\min(R, G, B) + \max(R, G, B)}{2}$$

This method is simple but uses only two of the three color channels, which may result in loss of detail.

4.4.2 Average Method

The Average method[103] determines the grayscale value by computing the average of all three color channels:

$$\text{Grayscale} = \frac{R + G + B}{3}$$

In code implementations using `uint8` integers, overflow can occur when the sum exceeds 255. To avoid this, it's recommended to divide each channel before summation:

$$\text{Grayscale} = \frac{R}{3} + \frac{G}{3} + \frac{B}{3}$$

While straightforward, this method does not align well with human visual perception, which leads to the use of the Weighted method.

4.4.3 Weighted (Luminosity) Method

The Weighted method[104], or Luminosity method, accounts for the human eye's varying sensitivity to color wavelengths. It assigns different weights to each RGB component:

$$\text{Grayscale} = 0.299R + 0.587G + 0.114B$$

These coefficients are based on perceptual studies and yield a more accurate luminance representation in grayscale conversion.

4.4.4 Grayscale Description and Intensity Calculation

In an 8-bit image, each pixel's RGB components range from 0 to 255. The grayscale conversion applies a weighted or unweighted formula to each pixel to obtain a single intensity value. The histogram of a grayscale image illustrates the distribution of these values.

Even grayscale images of the same size may have different histograms, depending on their content, lighting, and texture.

The average intensity of a grayscale image or a region within it is calculated as:

$$\text{Average Intensity} = \frac{1}{N} \sum_{i=1}^{N} I_i$$

Where:

- N is the total number of pixels in the region.
- I_i is the intensity value of the i^{th} pixel.

This metric reflects the overall brightness level in the image or selected region.

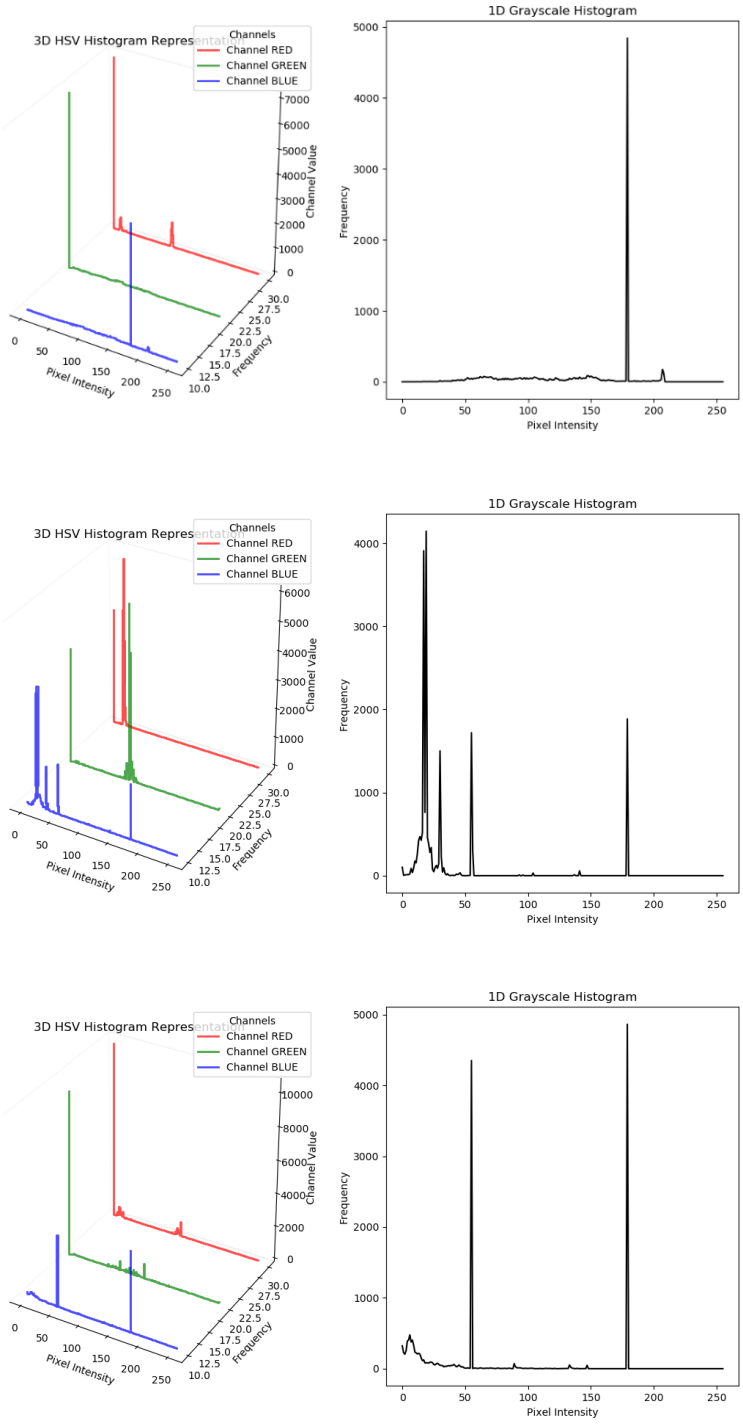
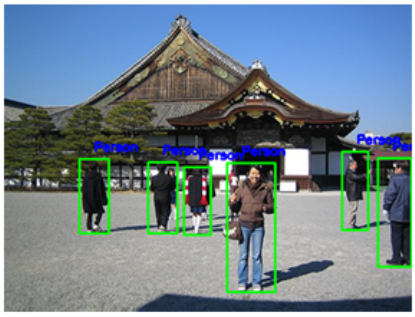


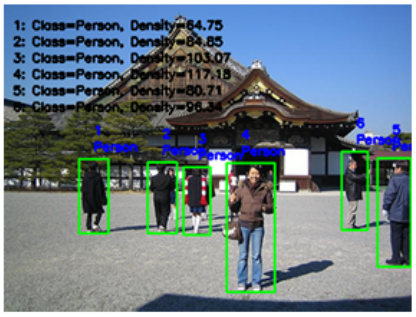
Figure 4.4: colored and grey scale representation of object with the same class id and different shape

As shown in Figure 4.4, we selected objects with the same class ID and measured their color density after converting the detected images to gray-scale. The results show that each object exhibits a unique density value, influenced by variations in shape and color. This provides an effective approach for creating a new object class that combines the original class ID with the associated color density, thereby enabling more precise identification of each detected object. This method improves the ability to track and differentiate objects within the same class ID, contributing to a deeper understanding of the environment.

(a) VOC test image.



(b) Deep learning alghorithms.



(c) Our detection results.

Figure 4.5: Object detection based on yolo and intensity density..

Algorithm 5 DetectedObjectsCallback (GCP + 3D Projection Fusion)

Wait for depth objects, camera info, RGB image, and robot pose
Compute robot rotation matrix R_{rob} using quaternion
Get projection matrix from camera info
try:
  Lookup transform from `map` to `xtion_rgb_optical_frame`
catch exception:
  return
Compute rotation matrix R and translation vector T from transform
Initialize intersection list for pointcloud markers
Get GCP detection bounding box $A = (x_{A1}, y_{A1}, x_{A2}, y_{A2})$
Compute GCP detection size and area S_A
for each depth object marker m_i **do**
    Get 3D bounding box (B_1, B_2) from marker dimensions
    Transform bounding box to camera frame using $R^T(B - T)$
    Project 3D corners to image space using intrinsics
    Compute projected box $B = (x_{B1}, y_{B1}, x_{B2}, y_{B2})$
    Compute area S_B and intersection S_I
    Compute union area S_U and IoU: $\text{IoU}_i = \frac{S_I}{S_U}$
end for
$i_{max} \leftarrow \arg\max(\text{IoU})$
if $\text{IoU}_{i_{max}} >$ threshold **then**
    Check for existing detected objects with overlapping volume
    if volume overlap is high **then**
        if new IoU is better **then**
            Replace old object with new detection
        end if
    else
        Add new detected object
    end if
    Assign scale based on object class
    Publish markers: POI, last object, label, label list
    if SPEECH enabled **then**
        Use TTS to say detection
    end if
end if

4.5 System architecture

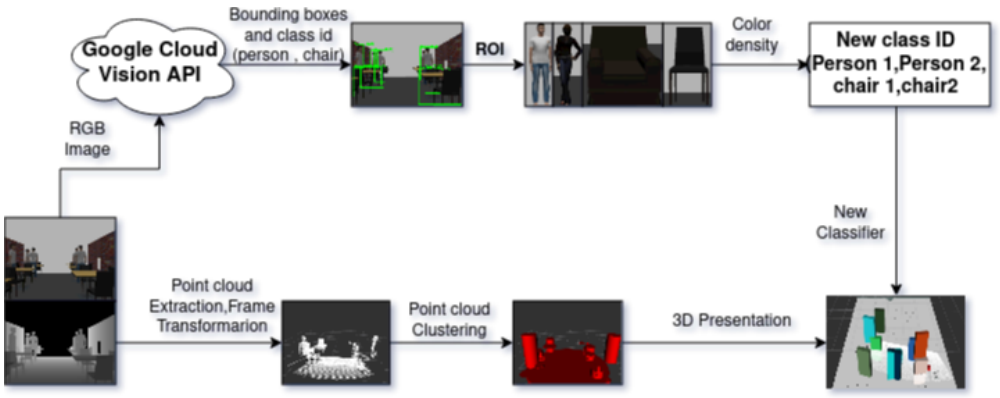


Figure 4.6: System architecture.

Figure 4.6 illustrates the proposed system architecture that integrates cloud-based vision services with RGB-D perception to enhance object detection, classification, and three-dimensional scene representation. The architecture combines two-dimensional object detection results obtained from the Google Cloud Vision API with depth-based point cloud processing, enabling robust object differentiation and accurate spatial localization in structured indoor environments.

The processing pipeline begins with the acquisition of an RGB image and its corresponding depth image from the RGB-D sensor. The RGB image is transmitted to the Google Cloud Vision API, which performs object detection and returns bounding box coordinates along with a pre-classification class ID for each detected object (e.g., person, chair). These outputs provide an initial semantic understanding of the scene.

A known limitation of cloud-based object detection arises when multiple objects share the same semantic class and similar physical characteristics. For instance, different individuals may be assigned the same class ID despite variations in appearance, size, or spatial location. To overcome this limitation, the proposed system introduces an additional object re-identification stage based on Region of Interest (ROI) analysis and intensity information.

Using the bounding box coordinates provided by the cloud service, a corresponding ROI is cropped from the RGB image for each detected object. The ROI processing algorithm performs three main operations:

1. Conversion of the cropped ROI to grayscale,

2. Computation of the object's intensity (color density),

3. Generation of a new refined class ID by combining the original class ID with the extracted intensity features.

This refinement process allows the system to distinguish between visually similar objects that were initially assigned identical class IDs, resulting in unique identifiers such as *Person 1*, *Person 2*, *Chair 1*, and *Chair 2*, as shown in Figure **??**.

In parallel, the depth image is processed to extract a point cloud representation of the scene. The point cloud undergoes coordinate transformation and clustering, enabling the segmentation of individual objects in three-dimensional space. By associating the ROI information obtained from the RGB image with the corresponding point cloud clusters, each detected object is accurately localized in the 3D environment.

The final output of the proposed architecture is a three-dimensional scene representation in which each object is uniquely identified and spatially positioned. By fusing cloud-based semantic detection with depth-based geometric information, the system improves object discrimination, robustness, and spatial awareness, making it well-suited for real-time robotic perception and navigation applications.

During experimental evaluation, the Google Cloud Vision API was invoked 21,971 times with zero recorded errors, as summarized in Table 4.1. A total of approximately 22,000 image analysis requests were successfully processed, demonstrating the reliability and robustness of the cloud vision service within the proposed framework.

Table 4.1: APIs and Services

Name	**Cloud Vision API**
Requests	21,971
Errors (%)	0
Latency, median (ms)	142
Latency, 95% (ms)	708

To test The system a 3D environment on GAZEBO simulation platforms was created which gives more flexibility in terms of moving and changing objects colors as shown in Figure 4.7

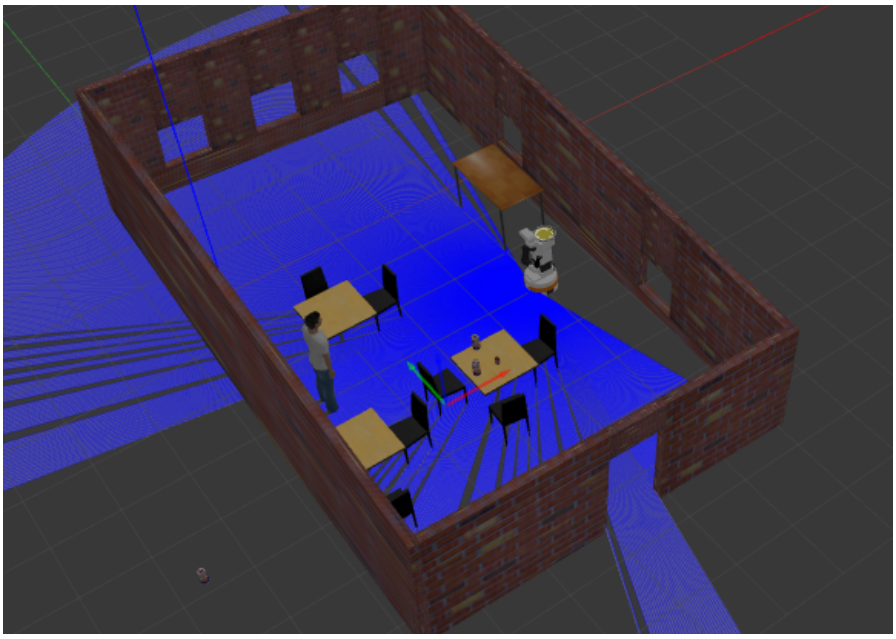


Figure 4.7: Simulation environment

4.5.1 3D Scene Reconstruction and Object Representation Results

Figures 4.8 and 4.9 present the qualitative results of the proposed cloud vision–based system architecture applied to three-dimensional scene reconstruction and object representation. These figures illustrate the progressive integration of detection, clustering, and visualization stages described in the system architecture.

Figure 4.8 shows the three-dimensional representation of detected objects generated solely from depth information, without incorporating color or intensity attributes. In this representation, each detected object is visualized as a vertical geometric primitive positioned according to its estimated centroid in the environment. Subfigures 4.8a and 4.8b present different viewpoints of the same reconstructed scene, highlighting the spatial distribution of objects relative to the global map frame. Although object localization is accurate, all detected entities appear visually identical, which limits the ability to distinguish between objects belonging to the same semantic class.

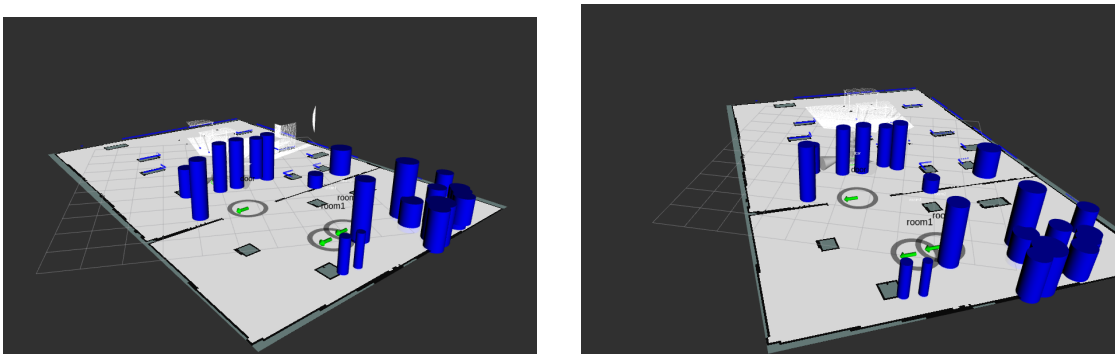
To overcome this limitation, the enhanced representation shown in Figure 4.9 incorporates the refined classification process introduced in the system architecture. By fusing cloud-based object detection results with ROI-based intensity analysis and point cloud clustering, each object is assigned a unique identifier and visual attribute. As a result, objects are rendered with distinct colors, allowing

clear differentiation between visually similar instances.

Subfigures 4.9a, 4.9b, and 4.9c of Figure 4.7 demonstrate the robustness of the proposed approach under different camera viewpoints and object configurations. The consistent color assignment confirms that the system successfully associates each RGB-based detection with its corresponding three-dimensional cluster. Subfigure 4.9c further illustrates the system's capability to handle a reduced number of objects while maintaining accurate spatial localization and identity preservation.

These results validate the effectiveness of the proposed cloud vision and 3D fusion architecture. By combining bounding box information from cloud-based detection, ROI intensity refinement, and depth-based point cloud processing, the system achieves reliable object discrimination and spatial awareness. This enriched 3D representation forms a solid foundation for downstream tasks such as AGV navigation, obstacle avoidance, and decision-making in structured indoor environments.

To evaluate the system in a controlled and flexible setting, a three-dimensional simulation environment was developed using the Gazebo platform. This environment enables dynamic modification of object positions and colors, facilitating comprehensive testing of the proposed architecture under various scenarios. The experimental results confirm that the system maintains consistent performance and accurate object representation across different configurations.

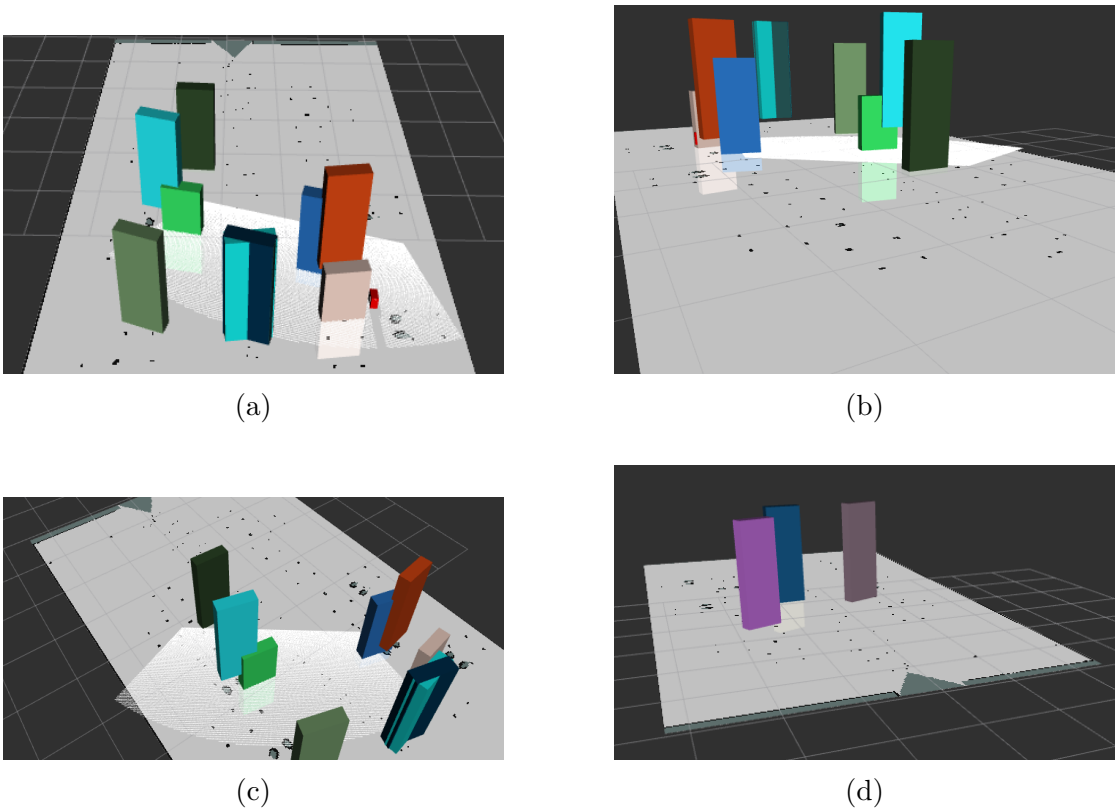


(a) (b)

Figure 4.8: 3D representation of the detected objects without Color information.

4.6 Conclusion

This chapter presented the development and implementation of a cloud vision-based autonomous guided vehicle (AGV) navigation system. The chapter introduced the concepts of 3D object representation using RGB-D point clouds cap-



(a) (b)

(c) (d)

Figure 4.9: 3D representation of the environment.

tured via the Kinect sensor. Through radius-based filtering and clustering with the DBSCAN algorithm, relevant objects in the robot's environment were accurately segmented and identified.

The chapter then detailed the intensity analysis of objects using multiple grayscale transformation methods These methods enabled the classification of objects based on their visual intensity, providing critical information for subsequent decision-making processes.

The system architecture was discussed following the visual processing stages, highlighting how the components integrated cloud vision APIs, point cloud processing, and robot control logic. This architecture allowed the AGV to perceive, process, and respond to its environment efficiently and in real-time.

General Conclusion

The rapid evolution of autonomous mobile robotics over the past decade has been largely driven by advances in artificial intelligence, computer vision, and sensing technologies. These developments have fundamentally transformed mobile robots from pre-programmed automated platforms into intelligent autonomous systems capable of perceiving, reasoning about, and interacting with complex and dynamic environments. Within this context, visual perception has emerged as a cornerstone for robust autonomy, particularly in navigation tasks that require semantic understanding of the environment.

The primary objective of this doctoral research was to enhance the visual perception capabilities of autonomous guided vehicles (AGVs) by integrating deep learning–based object detection with RGB-D sensing, thereby enabling reliable and real-time navigation in structured environments. To this end, state-of-the-art deep learning architectures, specifically YOLO-based models, were employed for traffic sign detection and semantic interpretation. By fusing RGB-based object detection with depth information extracted from an RGB-D sensor, the proposed approach achieved accurate distance estimation and spatial localization of relevant navigation cues, allowing the robot to make informed motion decisions without reliance on external localization infrastructures or remote supervisory control.

A major contribution of this work lies in the design and implementation of a perception-driven AGV navigation framework that tightly couples semantic vision, depth-based distance estimation, and real-time control. The system demonstrates how bounding box projections onto depth images can be effectively exploited to infer metric information, while three-dimensional point cloud generation and processing enable a richer and more structured representation of the surrounding environment. Through the application of filtering techniques, density-based clustering, and grayscale intensity analysis, the proposed method provides a robust mechanism for object representation and spatial reasoning in three-dimensional space.

Furthermore, this research investigates the integration of cloud-based vision services with local robotic perception, highlighting the potential of cloud intelligence to augment onboard processing capabilities. By offloading computationally

intensive vision tasks to cloud infrastructures while maintaining real-time interaction through ROS-based communication, the system illustrates a scalable approach to autonomous navigation that reduces hardware constraints on mobile robots. This hybrid architecture effectively bridges local autonomy and cloud-assisted intelligence, opening new perspectives for distributed robotic systems and smart industrial environments.

Experimental validation conducted in a simulated environment using a Pioneer 3-DX robot demonstrated the effectiveness of the proposed framework in real-time traffic sign detection, distance estimation, and navigation decision-making. The obtained results confirm that the fusion of deep learning–based semantic perception with RGB-D sensing significantly enhances robustness, accuracy, and adaptability when compared to purely vision-based or geometry-based approaches.

Despite these promising results, several challenges remain. Cloud-based perception systems are inherently subject to limitations related to communication latency, network reliability, data privacy, and bandwidth constraints. Addressing these issues will be critical for deploying such architectures in safety-critical and large-scale real-world applications. In addition, improving the generalization capability of deep learning models across diverse environments, illumination conditions, and sensor configurations remains an open research challenge.

Future research directions include the development of adaptive perception models capable of online learning, the integration of edge–cloud collaborative intelligence to mitigate latency constraints, and the extension of the proposed framework to cooperative multi-robot systems. In such systems, shared perception and collective decision-making enabled by cloud services could significantly enhance scalability, efficiency, and robustness. Moreover, transferring the proposed approach from simulation to real-world deployments will be essential to fully assess its practical applicability and performance.

In conclusion, this thesis contributes to the advancement of autonomous mobile robotics by proposing a comprehensive perception-driven navigation framework that combines deep learning, RGB-D sensing, and cloud-based intelligence. The results demonstrate that the intelligent fusion of semantic and geometric information constitutes a powerful enabler for robust AGV navigation and represents a significant step toward the realization of fully autonomous, scalable, and intelligent robotic systems capable of operating in complex real-world environments.

Bibliography

[1] I. Zamalloa, R. Kojcev, A. Hernandez, I. Muguruza, L. Usategui, A. Bilbao, and V. Mayoral, "Dissecting robotics—historical overview and future perspectives," *Robotics*, vol. 6, no. 4, pp. 1–26, 2017.

[2] R. Girshick, J. Donahue, and T. Darrell, "Rich feature hierarchies for accurate object detection and semantic segmentation," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2014, pp. 580–587. [Online]. Available: https://doi.org/10.1109/CVPR.2014.81

[3] F. Melgani and L. Bruzzone, "Classification of hyperspectral remote sensing images with support vector machines," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 42, no. 8, pp. 1778–1790, 2004. [Online]. Available: https://doi.org/10.1109/TGRS.2004.831865

[4] D. Lowe, "Distinctive image features from scale-invariant keypoints," *International Journal of Computer Vision*, vol. 60, pp. 90–110, 2004. [Online]. Available: https://doi.org/10.1023/B:VISI.0000029664.99615.94

[5] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," in *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2005, pp. 886–893. [Online]. Available: https://doi.org/10.1109/CVPR.2005.177

[6] H. Bay, T. Tuytelaars, and L. Van Gool, "Surf: Speeded up robust features," in *Computer Vision – ECCV 2006*, A. Leonardis, H. Bischof, and A. Pinz, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 404–417.

[7] K. He, X. Zhang, S. Ren, and J. Sun, "Spatial pyramid pooling in deep convolutional networks for visual recognition," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 37, no. 9, pp. 1904–1916, 2014. [Online]. Available: https://doi.org/10.1007/978-3-319-10578-9_23

[8] R. Girshick, "Fast r-cnn," in *2015 IEEE International Conference on Computer Vision (ICCV)*, 2015, pp. 1440–1448.

[9] S. Ren, K. He, and R. Girshick, "Faster r-cnn: Towards real-time object detection with region proposal networks," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 39, pp. 1137–1149, 2017.

[10] H. Kaiming, G. Georgia, D. Piotr, and G. Ross, "Mask r-cnn," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2017.

[11] J. Redmon, S. Divvala, and R. Girshick, "You only look once: Unified, real-time object detection," in *IEEE International Conference on Computer Vision (ICCV)*, 2016, pp. 779–788. [Online]. Available: https://doi.org/10.1109/CVPR.2016.91

[12] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, *SSD: Single Shot MultiBox Detector*. Springer International Publishing, 2016, p. 21–37. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-46448-0_2

[13] X. Zhou, D. Wang, and P. Krähenbühl, "Objects as points," 2019. [Online]. Available: https://arxiv.org/abs/1904.07850

[14] M. Tan, R. Pang, and Q. Le, "Efficientdet: Scalable and efficient object detection," in *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020, pp. 10 778–10 787. [Online]. Available: https://doi.org/10.1109/CVPR42600.2020.01089

[15] P. Kurrek, M. Jocas, F. Zoghlami, M. Stoelen, and S. Vahid, "AI Motion Control – Andirect adaptive tracking control of a nonholonomic mobile robot via neural networks. ," in *Proceedings of the Design Society: International Conference on Engineering Design (ICED)*, vol. 1, 2019, pp. 3561–3570.

[16] F. Rubio, F. Valero, and C. Llopis-Albert, "A review of mobile robots: Concepts, methods, theoretical framework, and applications," *International Journal of Advanced Robotic Systems*, vol. 16, no. 2, p. 1729881419839596, 2019.

[17] J. P. Laumond, *La robotique mobile*. Paris: Edition Hermes Science Publication, 2001.

[18] C. Samson, P. Morin, and R. Lenain, "Modeling and control of wheeled mobile robots," in *Springer Handbook of Robotics*, B. Siciliano and O. Khatib, Eds. Cham: Springer, 2016, ch. 49. [Online]. Available: https://doi.org/10.1007/978-3-319-32552-1_49

[19] D. Buccieri, D. Perritaz, P. Mullhaupt, Z. P. Jiang, and D. Bonvin, "Velocity-scheduling control for a unicycle mobile robot: theory and experiments," *IEEE Transactions on Robotics*, vol. 25, no. 2, pp. 451–458, 2009.

[20] G. Oriolo, A. D. Luca, and M. Vendittelli, "Wmr control via dynamic feedback linearization: design, implementation, and experimental validation," *IEEE Transactions on Control Systems Technology*, vol. 10, no. 6, pp. 835–852, 2002.

[21] O.Mohareri, R. Dhaouadi, and A. Rad, "Indirect adaptive tracking control of a nonholonomic mobile robot via neural networks," *Neurocomputing*, vol. 88, pp. 54–66, 2012, intelligent and Autonomous Systems.

[22] S. Bouzoualegh, E. Guechi, and R. Kelaiaia, "Model predictive control of a differential-drive mobile robot," *Acta Universitatis Sapientiae, Electrical and Mechanical Engineering*, vol. 10, no. 1, pp. 20–41, 2018. [Online]. Available: https://doi.org/10.2478/auseme-2018-0002

[23] J. I. Neimark and N. A. Fufaev, *Dynamics of Nonholonomic Systems*, ser. Translations of Mathematical Monographs. American Mathematical Society, 1972.

[24] A. M. Bloch, P. E. Crouch, J. Baillieul, and J. E. Marsden, *Nonholonomic Mechanics and Control*. Springer, 2003.

[25] R. Dhaouadi and A. A. Hatab, "Dynamic modelling of differential drive mobile robots, a unified framework," *Advances in Robotics & Automation*, vol. 2, no. 2, pp. 1–7, 2013.

[26] A. Mallem, "Algorithmes d'aide à la navigation d'un robot mobile," Ph.D. dissertation, Université de Batna 2, 2011.

[27] H.-K. L. K.-H. C. S.-B. Kim, J.-C. Bazin and S.-Y. Park, "Ground vehicle navigation in harsh urban conditions by integrating inertial navigation system, global positioning system, odometer and vision data," *IET radar, sonar & navigation*, vol. 5, no. 8, pp. 814–823, 2011.

[28] M. Ali, "Algorithmes d'aide à la navigation d'un robot mobile," Ph.D. dissertation, Université de Batna 2, 2011.

[29] M. O. A. Aqel, M. H. Marhaban, M. I. Saripan *et al.*, "Review of visual odometry: types, approaches, challenges, and applications," *SpringerPlus*, vol. 5, p. 1897, 2016.

[30] L. H. HS and P. G. KH, "Accelerometer for mobile robot positioning," *IEEE Transactions on Industry Applications*, vol. 37, no. 3, pp. 812–819, 2001.

[31] L. Jorge, M. Lino, D. Jorge, and N. U. ., "Sensors for mobile robot navigation," in *Autonomous Robotic Systems*, A. T. de Almeida and O. Khatib, Eds. London: Springer London, 1998, pp. 50–81.

[32] R. Siegwart, I. R. Nourbakhsh, and D. Scaramuzza, *Introduction to autonomous mobile robots*. MIT Press, 2011.

[33] N. Ahmad *et al.*, "Reviews on various inertial measurement unit (imu) sensor applications," *International Journal of Signal Processing Systems*, vol. 1, no. 2, pp. 256–262, 2013.

[34] F. M. Ortiz, M. Sammarco, L. H. M. K. Costa, and M. Detyniecki, "Applications and services using vehicular exteroceptive sensors: A survey," *IEEE Transactions on Intelligent Vehicles*, vol. 8, no. 1, pp. 949–969, 2023.

[35] V. A. Zhmud *et al.*, "Application of ultrasonic sensor for measuring distances in robotics," *Journal of Physics: Conference Series*, vol. 1015, no. 3, p. 032085, 2018.

[36] F. Duchon *et al.*, "Some applications of laser rangefinder in mobile robotics," *Journal of Control Engineering and Applied Informatics*, vol. 14, no. 2, pp. 50–57, 2012.

[37] L. Li, "Time-of-flight camera—an introduction," Technical white paper SLOA190B, 2014, accessed: 2025-05-08.

[38] M. Hansard *et al.*, *Time-of-flight cameras: principles, methods and applications*. New York: Springer Science & Business Media, 2012.

[39] R. A. Bolt, "Put-that-there: Voice and gesture at the graphics interface," *ACM SIGGRAPH Computer Graphics*, vol. 14, no. 3, pp. 262–270, 1980.

[40] S. S. Fisher, "Telepresence master glove controller for dexterous robotic end-effectors," in *Proceedings of SPIE - The International Society for Optical Engineering*, D. P. Casasent, Ed., vol. 726, 1987, p. 396.

[41] R. Marks, "3d spatial interaction for entertainment," in *2011 IEEE Symposium on 3D User Interfaces (3DUI)*. IEEE, 2011.

[42] K. Khoshelham and S. Elberink, "Accuracy and resolution of kinect depth data for indoor mapping applications," *Sensors*, vol. 12, no. 2, pp. 1437–1454, 2012. [Online]. Available: https://doi.org/10.3390/s120201437

[43] J. J. Hopfield, "Artificial neural networks," *IEEE Circuits and Devices Magazine*, vol. 4, pp. 3–10, 1988.

[44] A. P. James, "Threshnomics: An introduction to threshold logic in algorithms and circuits," *Journal of Computer Science & Systems Biology*, vol. 7, no. 6, 2014. [Online]. Available: https://doi.org/10.4172/jcsb.1000163

[45] J. F. Mas and J. J. Flores, "The application of artificial neural networks to the analysis of remotely sensed data," *International Journal of Remote Sensing*, vol. 29, no. 3, pp. 617–663, 2008.

[46] S. Sharma, S. Sharma, and A. Athaiya, "Activation functions in neural networks," *Towards Data Science*, vol. 6, no. 12, pp. 310–316, 2017.

[47] J. Feng and S. Lu, "Performance analysis of various activation functions in artificial neural networks," in *Journal of Physics: Conference Series*, vol. 1237, no. 2. IOP Publishing, 2019, p. 022030.

[48] J. Ren and H. Wang, "Chapter 3 - calculus and optimization," in *Mathematical Methods in Data Science*, J. Ren and H. Wang, Eds. Elsevier, 2023, pp. 51–89. [Online]. Available: https://www.sciencedirect.com/science/article/pii/B9780443186790000090

[49] A. L. Maas, A. Y. Hannun, and A. Y. Ng, "Rectifier nonlinearities improve neural network acoustic models," in *ICML Workshop on Deep Learning for Audio, Speech and Language Processing*, vol. 28, 2013. [Online]. Available: http://www.stanford.edu/~awni/papers/relu_hybrid_icml2013_final.pdf

[50] H. Robbins and S. Monro, "A stochastic approximation method," in *Herbert Robbins Selected Papers*. Springer, 1985, pp. 102–109.

[51] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[52] T. Tieleman and G. Hinton, "Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude," *COURSERA: Neural networks for machine learning*, vol. 4, no. 2, pp. 26–31, 2012.

[53] M. H. Sazlı, "A brief review of feed-forward neural networks," *Communications Faculty of Sciences University of Ankara Series A2-A3 Physical Sciences and Engineering*, vol. 50, no. 01, 2006.

[54] J. F. Mas and J. J. Flores, "The application of artificial neural networks to the analysis of remotely sensed data," *International Journal of Remote Sensing*, vol. 29, no. 3, pp. 617–663, 2008.

[55] M. Popescu, V. Balas, L. Perescu-Popescu, and N. Mastorakis, "Multilayer perceptron and neural networks," *WSEAS Transactions on Circuits and Systems*, vol. 8, no. 7, pp. 579–588, 2009.

[56] Z. C. Lipton, J. Berkowitz, and C. Elkan, "A critical review of recurrent neural networks for sequence learning," *arXiv preprint arXiv:1506.00019*, 2015.

[57] Y. I. Al Mashhadany, "Recurrent neural network with human simulator based virtual reality," in *Recurrent Neural Networks and Soft Computing*. InTech, 2012. [Online]. Available: https://doi.org/10.5772/35538

[58] B. Kumaraswamy, "Neural networks for data classification," in *Artificial Intelligence in Data Mining*. Elsevier, 2021, pp. 109–131. [Online]. Available: https://doi.org/10.1016/b978-0-12-820601-0.00011-2

[59] C. S. K. Dash, A. K. Behera, S. Dehuri, and S.-B. Cho, "Radial basis function neural networks: a topical state-of-the-art survey," *Open Computer Science*, vol. 6, no. 1, pp. 33–63, 2016.

[60] J. Ghosh and A. Nag, *An Overview of Radial Basis Function Networks*. Physica-Verlag HD, 2001, pp. 1–36.

[61] M. Törmä, "Kohonen self-organizing feature map in pattern recognition," *Photogrammetric Journal of Finland*, vol. 15, no. 1, 1995.

[62] D. Bhatt, C. Patel, H. Talsania, J. Patel, R. Vaghela, S. Pandya, K. Modi, and H. Ghayvat, "Cnn variants for computer vision: History, architecture, application, challenges and future scope," *Electronics*, vol. 10, no. 20, p. 2470, 2021. [Online]. Available: https://www.mdpi.com/2079-9292/10/20/2470

[63] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, pp. 436–444, 2015. [Online]. Available: https://doi.org/10.1038/nature14539

[64] R. Rajavel, S. Ravichandran, K. Harimoorthy *et al.*, "Iot-based smart healthcare video surveillance system using edge computing," *Journal of Ambient Intelligence and Humanized Computing*, vol. 13, pp. 3195–3207, 2022. [Online]. Available: https://doi.org/10.1007/s12652-021-03157-1

[65] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, "Ssd: Single shot multibox detector," in *European conference on computer vision*. Springer, 2016, pp. 21–37.

[66] L. Huang, Y. Yang, Y. Deng, and Y. Yu, "Densebox: Unifying landmark localization with end-to-end object detection," *arXiv:1509.04874 [cs.CV]*, 2015. [Online]. Available: https://arxiv.org/abs/1509.04874

[67] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollar, "Focal loss for dense object detection," in *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, Oct 2017.

[68] S. Liu and D. Huang, "Receptive field block net for accurate and fast object detection," in *Proceedings of the European Conference on Computer Vision (ECCV)*, Munich, Germany, September 2018, pp. 385–400.

[69] M. Tan, R. Pang, and Q. V. Le, "Efficientdet: Scalable and efficient object detection," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, Seattle, WA, USA, June 2020, pp. 10 781–10 790.

[70] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Las Vegas, NV, USA: IEEE, 2016, pp. 779–788.

[71] M. Mahasin and I. Dewi, "Comparison of cspdarknet53, cspresnext-50, and efficientnet-b0 backbones on yolo v4 as object detector," *International Journal of Engineering, Science and Information Technology*, vol. 2, pp. 64–72, 2022.

[72] H. Chen, Z. Chen, and H. Yu, "Enhanced yolov5: An efficient road object detection method," *Sensors*, vol. 23, no. 17, p. 8355, 2023.

[73] R. Li, X. Zeng, S. Yang, Q. Li, A. Yan, and D. Li, "Abyolov4: Improved yolov4 human object detection based on enhanced multi-scale feature fusion," *EURASIP Journal on Advances in Signal Processing*, vol. 2024, no. 1, p. 6, 2024.

[74] M. Ali and Z. Zhang, "The yolo framework: A comprehensive review of evolution, applications, and benchmarks in object detection," *Computers*, vol. 13, no. 12, p. 336, 2024. [Online]. Available: https://doi.org/10.3390/computers13120336

[75] J. Stallkamp, M. Schlipsing, J. Salmen, and C. Igel, "The german traffic sign recognition benchmark: A multi-class classification competition," in *Proceedings of the 2011 International Joint Conference on Neural Networks*, San Jose, CA, USA, 2011, pp. 1453–1460.

[76] R. Timofte, V. Prisacariu, L. Gool, and I. Reid, "Combining traffic sign detection with 3d tracking towards better driver assistance," in *Emerging Topics in Computer Vision and Its Applications*. Singapore: World Scientific, 2012, pp. 425–446.

[77] A. Medjaldi, Y. Slimani, and N. Karkar, "Cost-effective real-time obstacle detection and avoidance for agvs using yolov8 and rgb-d sensors," *Engineering, Technology & Applied Science Research*, vol. 15, no. 2, pp. 21 738–21 745, Apr. 2025. [Online]. Available: https://doi.org/10.48084/etasr.10135

[78] M. Aizat, A. Azmin, and W. Rahiman, "A survey on navigation approaches for automated guided vehicle robots in dynamic surrounding," *IEEE Access*, vol. 11, pp. 33 934–33 955, 2023.

[79] F. Liu, X. Li, and Y. Wang, "Design of automatic guided vehicle motion control system based on magnetic navigation," in *Chinese Control And Decision Conference (CCDC)*, Shenyang, China, 2018, pp. 4775–4779.

[80] V. Rubanov, D. Bushuev, E. Karikov, A. Bazhanov, and S. Alekseevsky, "Development a low-cost navigation technology based on metal line sensors and passive rfid tags for industrial automated guided vehicle," *Journal of Engineering and Applied Sciences*, vol. 15, no. 20, pp. 2291–2297, 2020.

[81] Q. F. Yan, H. Y. Hu, T. P. Hang, and Y. S. Fu, "Path tracking of ins agv corrected by double magnetic nails based on fuzzy controller," in *IEEE 3rd Advanced Information Management, Communicates, Electronic and Automation Control Conference (IMCEC)*, Chongqing, China, 2019, pp. 1732–1735.

[82] Z. Ma, O. Postolache, and Y. Yang, "Obstacle avoidance for unmanned vehicle based on a 2d lidar," in *International Conference on Sensing and Instrumentation in IoT Era (ISSI)*, Lisbon, Portugal, 2019, pp. 1–6.

[83] G. Fragapane, R. de Koster, F. Sgarbossa, and J. O. Strandhagen, "Planning and control of autonomous mobile robots for intralogistics: Literature review and research agenda," *European Journal of Operational Research*, vol. 294, no. 2, pp. 405–426, 2021.

[84] A. N. A. Rafai, N. Adzhar, and N. I. Jaini, "A review on path planning and obstacle avoidance algorithms for autonomous mobile robots," *Journal of Robotics*, vol. 2022, pp. 1–1, 2022.

[85] Y. Chen, W. Zheng, Y. Zhao, T. H. Song, and H. Shin, "Dw-yolo: An efficient object detector for drones and self-driving vehicles," *Arabian Journal for Science and Engineering*, vol. 48, no. 2, pp. 1427–1436, 2023.

[86] M. Ding *et al.*, "Learning depth-guided convolutions for monocular 3d object detection," in *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, Seattle, WA, USA, 2020, pp. 11 669–11 678.

[87] C. Reading, A. Harakeh, J. Chae, and S. L. Waslander, "Categorical depth distribution network for monocular 3d object detection," in *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, Nashville, TN, USA, 2021, pp. 8551–8560.

[88] D. H. D. Reis, D. Welfer, M. A. D. S. L. Cuadros, and D. F. T. Gamarra, "Mobile robot navigation using an object recognition software with rgbd images and the yolo algorithm," *Applied Artificial Intelligence*, vol. 33, no. 14, pp. 1290–1305, 2019.

[89] Y. Zhang *et al.*, "A regional distance regression network for monocular object distance estimation," *Journal of Visual Communication and Image Representation*, vol. 79, p. 103224, 2021.

[90] H. Saleh, S. Saleh, N. T. Toure, and W. Hardt, "Robust collision warning system based on multi objects distance estimation," in *IEEE Concurrent Processes Architectures and Embedded Systems Virtual Conference (COPA)*, San Diego, CA, USA, 2021, pp. 1–6.

[91] F. Shao, X. Wang, F. Meng, J. Zhu, D. Wang, and J. Dai, "Improved faster r-cnn traffic sign detection based on a second region of interest and highly possible regions proposal network," *Sensors*, vol. 19, no. 10, 2019.

[92] W. Huang, M. Huang, and Y. Zhang, "Detection of traffic signs based on combination of gan and faster-rcnn," *Journal of Physics: Conference Series*, vol. 1069, no. 1, 2018.

[93] Z. Zhao, H. Liu, and D. Cao, "Improved traffic sign detection algorithm based on libra r-cnn," *Journal of Mechanical Engineering*, vol. 57, no. 22, pp. 255–265, 2021.

[94] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Ng, "Ros: an open-source robot operating system," in *IEEE International Conference on Robotics and Automation Workshop on Open Source Software*, Jan. 2009.

[95] C. Malleson, J.-Y. Guillemaut, and A. Hilton, "3d reconstruction from rgb-d data," in *RGB-D Image Analysis and Processing*, P. Rosin, Y.-K. Lai, L. Shao, and Y. Liu, Eds. Cham: Springer International Publishing, 2019, pp. 87–115.

[96] M. Zollhöfer, "Commodity rgb-d sensors: data acquisition," in *RGB-D Image Analysis and Processing*, P. Rosin, Y.-K. Lai, L. Shao, and Y. Liu, Eds. Cham: Springer International Publishing, 2019, pp. 3–13.

[97] S. V. Alexandrov, J. Prankl, M. Zillich, and M. Vincze, "Calibration and correction of vignetting effects with an application to 3d mapping," in *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2016, pp. 4217–4223.

[98] T. Niemirepo, M. Viitanen, and J. Vanne, "Open3dgen: opensource software for reconstructing textured 3d models from rgb-d images," in *MMSys '21: 12th ACM Multimedia Systems Conference*. ACM, 2021, pp. 12–22.

[99] M. Zollhöfer, P. Stotko, A. Görlitz, C. Theobalt, M. Nießner, R. Klein, and A. Kolb, "State of the art on 3d reconstruction with rgb-d cameras," *Comput. Graph. Forum*, vol. 37, pp. 625–652, 2018.

[100] T. N. Tran, K. Drab, and M. Daszykowski, "Revised dbscan algorithm to cluster data with dense adjacent clusters," *Chemometrics and Intelligent Laboratory Systems*, vol. 120, pp. 92–96, 2013.

[101] Z. Guo, H. Liu, L. Pang, L. Fang, and W. Dou, "Dbscan-based point cloud extraction for tomographic synthetic aperture radar (tomosar) three-dimensional (3d) building reconstruction," *International Journal of Remote Sensing*, vol. 42, no. 6, pp. 2327–2349, 2021.

[102] S. Liu, "Two decades of colorization and decolorization for images and videos," 2022, arXiv:2204.13322 [cs.CV]. [Online]. Available: https://arxiv.org/abs/2204.13322

[103] C. Kanan and G. W. Cottrell, "Color-to-grayscale: Does the method matter in image recognition?" *PLoS ONE*, vol. 7, no. 1, p. e29740, 2012. [Online]. Available: https://doi.org/10.1371/journal.pone.0029740

[104] K. Padmavathi and K. Thangadurai, "Implementation of rgb and grayscale images in plant leaves disease detection: A comparative study," *Indian Journal of Science and Technology*, vol. 9, no. 6, p. 16, 2016.