الجمهورية الجزائرية الديمقراطية الشعبية

People's Democratic Republic of algeria

Minstry of Higher Education and scientific Research

UNIVERSITY FERHAT ABBAS - SETIF 1

FACULTY OF TECHNOLOGY

THESIS

Submitted to the Departement of Electronics

In Fulfilment 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 **XX / XX / 2025** 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 **XX / XX / 2025** 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

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, including **Mr.Sohaib Louda**and others.

On a personal note, I express my heartfelt gratitude to my family. I pray that Allah (*Subḥānahu wa Taᶜālā*) grants mercy and the highest level of Jannah to my beloved **mother** — her memory continues to guide and inspire me each day. I also thank my **father** for his constant support, strength, and sacrifices that have made this journey possible.

To my dear **brother Oussama** and my **sisters**, I am deeply grateful for your encouragement, presence, and sincere belief in me.

A heartfelt thank you to my beloved **fiancée** for her love, patience, and unwavering support throughout this journey. Your presence has been a source of balance and strength in my life.

Finally, I would like to thank my close friends **Fares** ,**Djallel Eddine**, **Yacine**, **Hakim**, **Isslam**, **Issam**, and many others for their friendship, encouragement, and positivity during this academic adventure.

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. , 1
IoU	Intersection over Union.
IR	Infrared.
LiDAR	Light Detection and Ranging.

mAP Mean Average Precision.

Mask R-CNN Mask Region-based Convolutional Neural Network.

MIT Massachusetts Institute of Technology.

MLP Multi-Layer Perceptron.

MSE Mean Squared Error.

R-CNN Region-based Convolutional Neural Network.

RADAR Radio Detection and Ranging.

RBF Radial Basis Function.

RBFN Radial Basis Function 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 Chinese Traffic Sign 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

Chapter 1

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” [14], 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 (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)[3], have enabled robots to interpret complex visual data with high accuracy. Early detection methods used classifiers like Support Vector Machines (SVMs)[6] along with manually crafted features such as SIFT[8], HOG[7],, and SURF[1]. 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[4], SPP-Net[2] , Fast R-CNN [11], Faster R-CNN [5], and Mask R-CNN [10], which first generate region proposals and then perform classification, and *one-stage models* YOLO algorithm [13], Single Shot MultiBox Detector (SSD)[15] , CenterNet[12], and EfficientDet[9]., 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** provides a general introduction to the research, outlining the motivation, objectives, and contributions of the study in the context of autonomous mobile robotics and intelligent navigation systems.
- **Chapter 2** offers a comprehensive overview of robotics, focusing on Autonomous Mobile Robots (AMRs). It discusses various locomotion types and provides detailed analysis of Differential Drive Mobile Robots (DDMRs), including their kinematic and dynamic modeling using methods such as the Lagrangian approach. The chapter also examines perception systems in mobile robotics, including proprioceptive and exteroceptive sensors, and the structure and calibration of Kinect RGB-D sensors.
- **Chapter 3** explores the intersection of deep learning and computer vision. It presents the structure of artificial neural networks, activation functions, and training methods. The chapter highlights object detection methods with emphasis on YOLO-based models, particularly YOLOv8 for traffic sign detection. It includes training procedures, dataset use, and evaluation metrics. The integration of RGB-D sensors for real-time obstacle detection and AGV navigation is also discussed through system architecture and experimental results.

- **Chapter 4** introduces a cloud vision-based approach for AGV navigation. It covers 3D object representation using RGB-D point clouds, filtering and clustering methods such as DBSCAN, and grayscale intensity calculation methods. A cloud-based system architecture is proposed to enhance real-time decision-making and perception capabilities.

- **Chapter 5** concludes the thesis by summarizing key findings and contributions. It also addresses limitations and suggests potential future directions for cloud-based intelligent navigation in mobile robotics.

Bibliography

[1] N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 886–893, 2005.

[2] R. Girshick. Fast r-cnn. *arXiv e-prints*, 2015.

[3] 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)*, pages 580–587, 2014.

[4] 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*, 37(9):1904–1916, 2014.

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

[6] Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. *Nature*, 521:436–444, 2015.

[7] D.G. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60:90–110, 2004.

[8] F. Melgani and L. Bruzzone. Classification of hyperspectral remote sensing images with support vector machines. *IEEE Transactions on Geoscience and Remote Sensing*, 42(8):1778–1790, 2004.

[9] R. Rajavel, S.K. Ravichandran, K. Harimoorthy, et al. Iot-based smart healthcare video surveillance system using edge computing. *Journal of Ambient Intelligence and Humanized Computing*, 13:3195–3207, 2022.

[10] J. Redmon, S. Divvala, and R. Girshick. You only look once: Unified, real-time object detection. In *IEEE International Conference on Computer Vision (ICCV)*, pages 779–788, 2016.

[11] 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*, 39:1137–1149, 2017.

[12] M. Tan, R. Pang, and Q. Le. Efficientdet: Scalable and efficient object detection. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 10778–10787, 2020.

[13] L. Wei, D. Anguelov, et al. Ssd: Single shot multibox detector. In *European Conference on Computer Vision*, 2016.

[14] Iñigo Zamalloa, Risto Kojcev, Ander Hernandez, Iker Muguruza, Luis Usategui, Aitor Bilbao, and Victor Mayoral. Dissecting robotics—historical overview and future perspectives. *Robotics*, 6(4):1–26, 2017.

[15] X. Zhou, D. Wang, and P. Krähenbühl. Objects as points. *CoRR*, 2019.

Chapter 2

An overview of robotics.

2.1 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

Figure 2.1: Shakey the Robot – 1966

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.

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

2.2 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 2.2[13].

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.

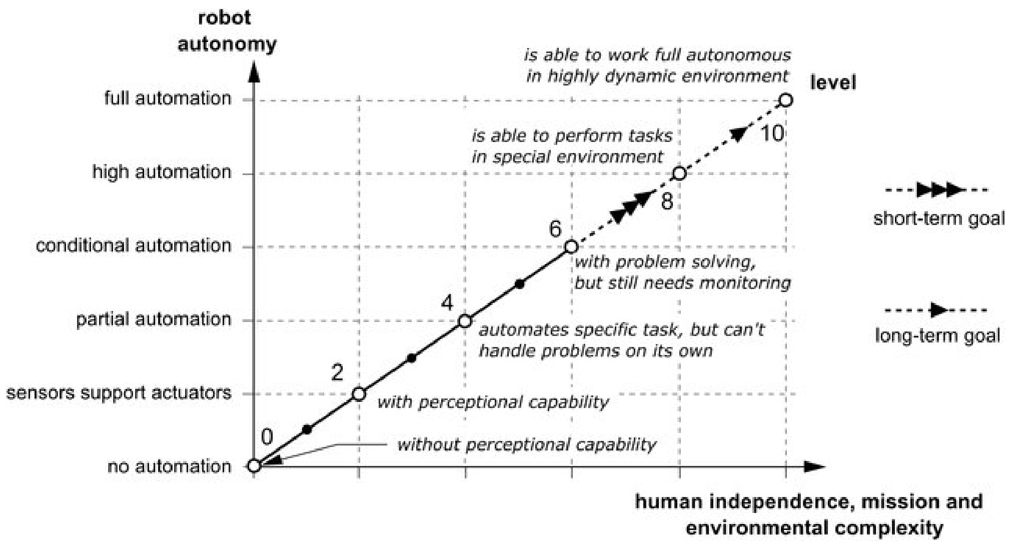

robot autonomy
full automation
high automation
conditional automation
partial automation
sensors support actuators
no automation
is able to work full autonomous in highly dynamic environment
level
is able to perform tasks in special environment
10
8
6
with problem solving, but still needs monitoring
4
automates specific task, but can't handle problems on its own
2
with perceptional capability
0
without perceptional capability
short-term goal
long-term goal
human independence, mission and environmental complexity

Figure 2.2: Robot autonomy based on ALFUS, the autonomy levels for unmanned systems.

2.3 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.

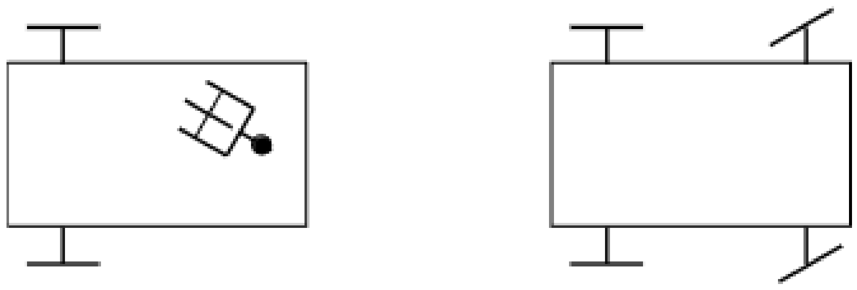


Figure 2.3: Unicycle-type and car-like mobile robots.

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[26].

2.4 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 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[14]. 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 2.3 [27]:

• 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.

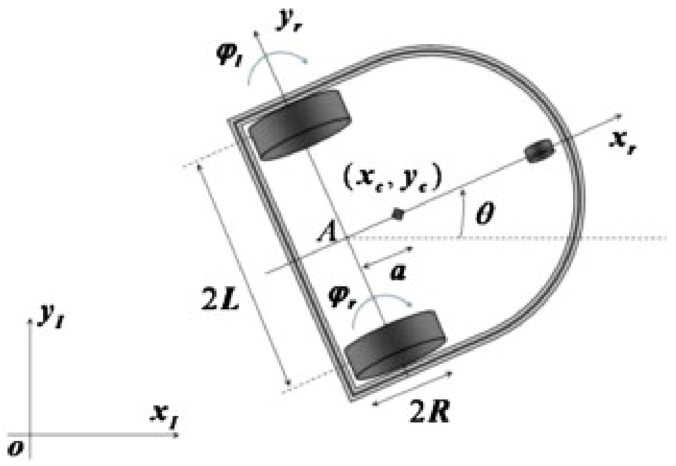


Figure 2.4: Differential-drive mobile robot model and reference frames

2.5 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* [6]. 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 [24, 22].

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 2.4, the robot's local coordinate system is fixed at 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 [22].

Reference Frames

Two primary coordinate systems are utilized for motion modeling:

- **Inertial Frame** $\{X_I, Y_I\}$: A fixed, global coordinate system attached to the environment. It serves as the reference for absolute positioning and orientation.
- **Robot Frame** $\{X_r, Y_r\}$: A body-fixed frame attached to the robot and moving with it. It is centered at point A and aligned with the robot's forward direction.

The robot's pose in the inertial frame is described by the vector:

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

where (x_a, y_a) represent the position of point A and θ denotes the robot's orientation with respect to the inertial X_I-axis.

Let $\mathbf{X}^r$ and $\mathbf{X}^I$ denote the coordinates of a point in the robot and inertial frames, respectively:

$$\mathbf{X}^r = \begin{bmatrix} x^r \\ y^r \\ \theta^r \end{bmatrix}, \quad \mathbf{X}^I = \begin{bmatrix} x^I \\ y^I \\ \theta^I \end{bmatrix}$$

These two representations are related through a rigid body transformation governed by a rotation matrix $\mathbf{R}(\theta)$, which accounts for the robot's heading:

$$\mathbf{X}^I = \mathbf{R}(\theta)\,\mathbf{X}^r$$

$$\mathbf{R}(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

This transformation facilitates the conversion of positions and orientations between the robot's local frame and the global inertial frame. Moreover, it extends to time derivatives (velocities), allowing us to relate the velocities in both frames as:

$$\dot{\mathbf{X}}^I = \mathbf{R}(\theta)\,\dot{\mathbf{X}}^r$$

This relation is critical in navigation, path planning, and control, as it provides a consistent way to interpret and transform motion across different reference systems.

2.5.1 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:

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

Using the orthogonal rotation matrix $\mathbf{R}(\theta)$, the velocity in the inertial frame yields:

$$-\dot{x}_a \sin\theta + \dot{y}_a \cos\theta = 0 \tag{I.2}$$

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 2.5[5].

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

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

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:

$$\dot{x}_{pR} = \dot{x}_a + L\dot{\theta}\cos\theta \tag{2.1}$$

$$\dot{y}_{pR} = \dot{y}_a + L\dot{\theta}\sin\theta \tag{2.2}$$

$$\dot{x}_{pL} = \dot{x}_a - L\dot{\theta}\cos\theta \tag{2.3}$$

$$\dot{y}_{pL} = \dot{y}_a - L\dot{\theta}\sin\theta \tag{2.4}$$

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

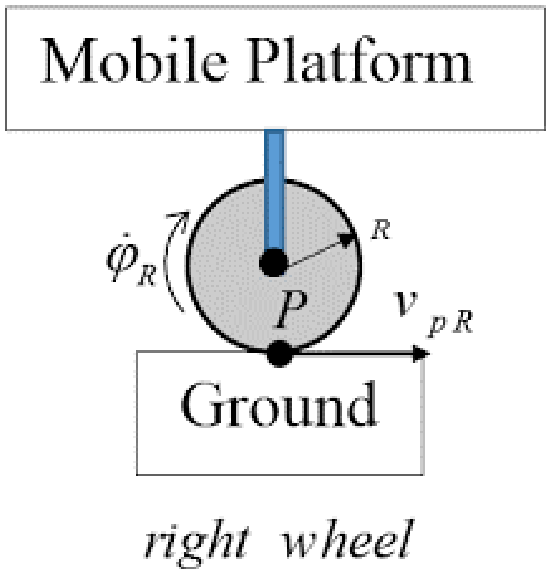


Figure 2.5: Rolling motion constraints

$$\dot{x}_{pR}\cos\theta + \dot{y}_{pR}\sin\theta = R\dot{\phi}_R \tag{2.5}$$
$$\dot{x}_{pL}\cos\theta + \dot{y}_{pL}\sin\theta = R\dot{\phi}_L \tag{2.6}$$

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

$$\Lambda(q)\,\dot{q} = 0 \tag{2.7}$$

where:

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

and the generalized velocity vector is:

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

Additionally, the wheel velocities are:

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

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

2.5.2 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:

$$v = \frac{v_R + v_L}{2} = \frac{R}{2}(\dot{\phi}_R + \dot{\phi}_L) \tag{2.8}$$

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:

$$\omega = \frac{v_R - v_L}{2L} = \frac{R}{2L}(\dot{\phi}_R - \dot{\phi}_L) \tag{2.9}$$

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

$$\begin{bmatrix} \dot{x}_a^r \\ \dot{y}_a^r \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} \frac{R}{2} & \frac{R}{2} \\ 0 & 0 \\ \frac{R}{2L} & -\frac{R}{2L} \end{bmatrix} \begin{bmatrix} \dot{\phi}_R \\ \dot{\phi}_L \end{bmatrix} \tag{I.8}$$

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:

$$\dot{q}^I = \begin{bmatrix} \dot{x}_a^I \\ \dot{y}_a^I \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} \frac{R}{2}\cos\theta & \frac{R}{2}\cos\theta \\ \frac{R}{2}\sin\theta & \frac{R}{2}\sin\theta \\ \frac{R}{2L} & -\frac{R}{2L} \end{bmatrix} \begin{bmatrix} \dot{\phi}_R \\ \dot{\phi}_L \end{bmatrix} \tag{I.9}$$

Equation (I.9) 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:

$$\dot{q}^I = \begin{bmatrix} \dot{x}_a^I \\ \dot{y}_a^I \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} \cos\theta & 0 \\ \sin\theta & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} v \\ \omega \end{bmatrix} \tag{2.10}$$

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.

2.5.3 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, 3].

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:

$$M(q)\ddot{q} + V(q,\dot{q})\dot{q} + F(\dot{q}) + G(q) + \tau_d = B(q)\tau - \Lambda^T(q)\lambda \tag{I.11}$$

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

2.5.4 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 [7].

The general Lagrange equation is:

$$\frac{d}{dt}\left(\frac{\partial L}{\partial \dot{q}_i}\right) - \frac{\partial L}{\partial q_i} = F - \Lambda^T(q)\lambda \tag{I.12}$$

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:

$$T_c = \frac{1}{2}m_c v_c^2 + \frac{1}{2}I_c\dot{\theta}^2 \tag{I.13}$$

$$T_{wR} = \frac{1}{2}m_w v_{wR}^2 + \frac{1}{2}I_m\dot{\theta}^2 + \frac{1}{2}I_w\dot{\phi}_R^2 \tag{I.14a}$$

$$T_{wL} = \frac{1}{2}m_w v_{wL}^2 + \frac{1}{2}I_m\dot{\theta}^2 + \frac{1}{2}I_w\dot{\phi}_L^2 \tag{I.14b}$$

Using geometric relationships:

$$v_c = \sqrt{\dot{x}_a^2 + \dot{y}_a^2 + d^2\dot{\theta}^2 + 2d(\dot{y}_a\cos\theta - \dot{x}_a\sin\theta)\dot{\theta}} \tag{I.15}$$

$$v_{wR} = \sqrt{\dot{x}_a^2 + \dot{y}_a^2 + L^2\dot{\theta}^2 + 2L(\dot{y}_a\cos\theta - \dot{x}_a\sin\theta)\dot{\theta}} \tag{I.16}$$

$$v_{wL} = \sqrt{\dot{x}_a^2 + \dot{y}_a^2 + L^2\dot{\theta}^2 - 2L(\dot{y}_a\cos\theta - \dot{x}_a\sin\theta)\dot{\theta}} \tag{I.17}$$

The total kinetic energy becomes:

$$T = \frac{1}{2}m(\dot{x}_a^2 + \dot{y}_a^2) - m_c d\dot{\theta}(\dot{y}_a\cos\theta - \dot{x}_a\sin\theta) + \frac{1}{2}I_w(\dot{\phi}_R^2 + \dot{\phi}_L^2) + \frac{1}{2}I\dot{\theta}^2 \tag{I.18}$$

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:

$$m\ddot{x}_a - md\ddot{\theta}\sin\theta - md\dot{\theta}^2\cos\theta = C_1 \tag{I.19a}$$

$$m\ddot{y}_a - md\ddot{\theta}\cos\theta - md\dot{\theta}^2\sin\theta = C_2 \tag{I.19b}$$

$$I\ddot{\theta} - md\ddot{x}_a\sin\theta + md\ddot{y}_a\cos\theta = C_3 \tag{I.19c}$$

$$I_w\ddot{\phi}_R = \tau_R + C_4 \tag{I.19d}$$

$$I_w\ddot{\phi}_L = \tau_L + C_5 \tag{I.19e}$$

In matrix form:

$$M(q)\ddot{q} + V(q,\dot{q})\dot{q} = B(q)\tau - \Lambda^T(q)\lambda \tag{I.20}$$

To remove λ, apply:

$$\dot{q} = S(q)\eta, \quad \ddot{q} = \dot{S}(q)\eta + S(q)\dot{\eta} \tag{I.21b, I.23}$$

Premultiplying by $S^T(q)$:

$$\bar{M}(q)\dot{\eta} + \bar{V}(q,\dot{q})\eta = \bar{B}(q)\tau \tag{I.26}$$

with:

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

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

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

$$\begin{bmatrix} m_0 & 0 \\ 0 & I_0 \end{bmatrix} \begin{bmatrix} \dot{v} \\ \dot{\omega} \end{bmatrix} + \begin{bmatrix} 0 & -m_c d\omega^2 \\ m_c d\omega & 0 \end{bmatrix} \begin{bmatrix} v \\ \omega \end{bmatrix} = \begin{bmatrix} \frac{1}{R} & 0 \\ 0 & \frac{L}{R} \end{bmatrix} \begin{bmatrix} \tau_R + \tau_L \\ \tau_R - \tau_L \end{bmatrix} \tag{I.27}$$

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.

2.6 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:

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

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:

$$\dot{y}\cos\theta - \dot{x}\sin\theta = 0 \tag{2.11}$$

$$\dot{x}\cos\theta + \dot{y}\sin\theta + \frac{l}{2}\dot{\theta} - r\dot{\phi}_R = 0 \tag{2.12}$$

$$\dot{x}\cos\theta + \dot{y}\sin\theta - \frac{l}{2}\dot{\theta} - r\dot{\phi}_L = 0 \tag{2.13}$$

Equation (2.11) enforces the non-slipping constraint: motion occurs only along the x_b axis. Equations (2.12) and (2.13) model pure rolling for each wheel.

Subtracting (2.13) from (2.12) gives a holonomic relation:

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

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

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

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

The relationship to the wheel velocities is:

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

2.6.1 Control Strategy

Table 2.1: 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

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:

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

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 $\omega$
Require: Wheel radius $r$, axle length $l$
Ensure: Right and left wheel angular velocities $\dot{\phi}_R$, $\dot{\phi}_L$
1: Input: $v$, $\omega$, $r$, $l$
2: Compute right wheel linear velocity:
3:         $v_R \leftarrow v + \frac{l}{2} \cdot \omega$
4: Compute left wheel linear velocity:
5:         $v_L \leftarrow v - \frac{l}{2} \cdot \omega$
6: Convert to angular wheel speeds:
7:         $\dot{\phi}_R \leftarrow \frac{v_R}{r}$
8:         $\dot{\phi}_L \leftarrow \frac{v_L}{r}$
9: Output: $\dot{\phi}_R$, $\dot{\phi}_L$
```

2.7 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.

2.7.1 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, like GPS and Galileo, are the most widely used ones here. IMUs, or inertial measuring units. The position, velocity, and occasionally heading of an ego vehicle are measured by GNSS receivers. 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. [19].

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[12, 18][2].

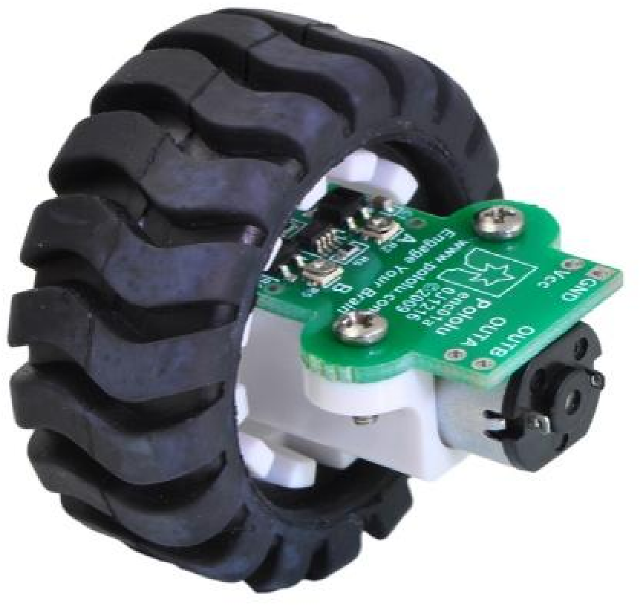


Figure 2.6: Wheel odometry with an optical encoder [Pololu Corporation 2016] frames

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 [16], [17]. 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 [20].

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. machine gyroscopes. A fast-spinning rotor's inertial characteristics are essential to

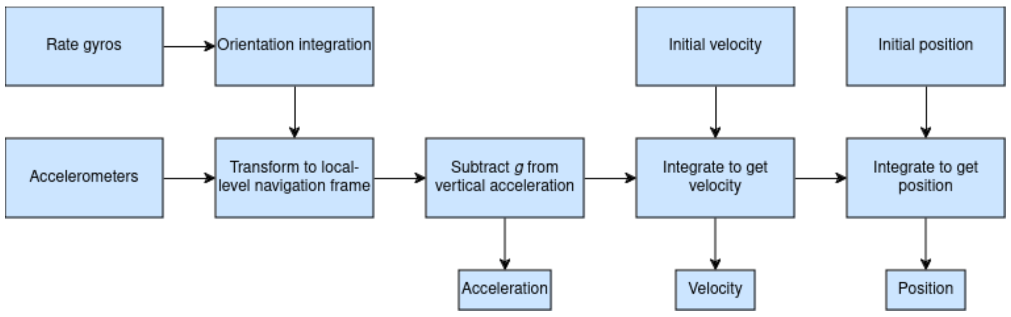

Rate gyros
Orientation integration
Initial velocity
Initial position
Accelerometers
Transform to local-level navigation frame
Subtract g from vertical acceleration
Integrate to get velocity
Integrate to get position
Acceleration
Velocity
Position

Figure 2.7: IMU Block diagram frames

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 [28].

Inertial Measurement Unit (IMU)

Devices that measure velocity, direction, and gravitational force mostly use IMU. It can be divided into two categories: the older technology, which uses gyroscopes and accelerometers as its two sensor kinds. While the gyroscope measures angular rotation, the accelerometer measures inertial acceleration. In order to measure from three axes, both sensors normally have three degrees of freedom. Later, IMU technology developed using a different kind of sensor. . [1].

2.7.2 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[25].

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 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.[29].

The distance d of the object causing the reflection can be calculated based on the propagation speed of sound C and the time-of-flight t:

$$d = \frac{c \cdot t}{2}$$

The speed of sound c in air is given by:

$$c = \sqrt{\gamma R T}$$

where:

- γ = ratio of specific heats,
- R = gas constant,
- T = temperature in degrees Kelvin.

Laser Rangefinder (Time-of-Flight, Electromagnetic)

Light Detection and Ranging (LIDAR) sensors, sometimes referred to as laser rangefinders, provide a notable benefit over ultrasonic rangefinders by using light rather than sound. Light beams are fired into the surroundings, and the reflected return is measured in LIDAR sensing. It is possible to estimate the intensity in range of the reflecting item by measuring the amount of light that is reflected and the beam's time-of-flight.

A rotating element with several stacked light sources is typically included in LIDAR, which produces a three-dimensional point cloud map that is excellent for evaluating scene geometry. LIDAR is not impacted by the lighting conditions of its surroundings because it is an active sensor with light sources. As a result, when using LIDAR in dim or fluctuating lighting, it does not encounter the same difficulties as cameras.[8].

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.

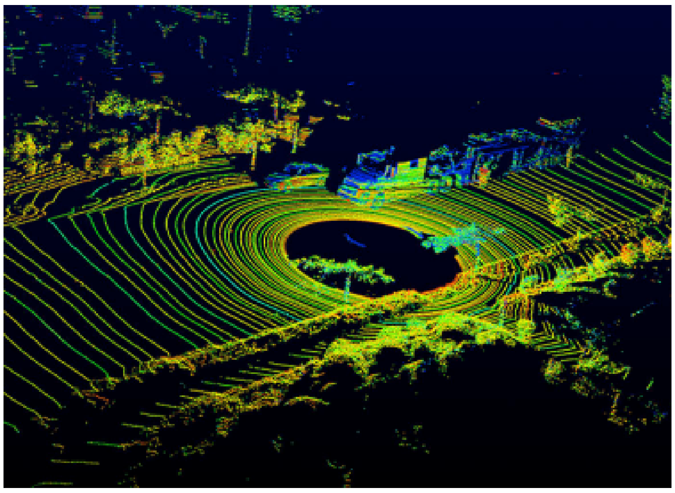


Figure 2.8: Detailed 3D scene geometry from LIDAR point cloud

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[15, 10].

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.[4]

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.[9]

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.[21]

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.

2.8 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 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

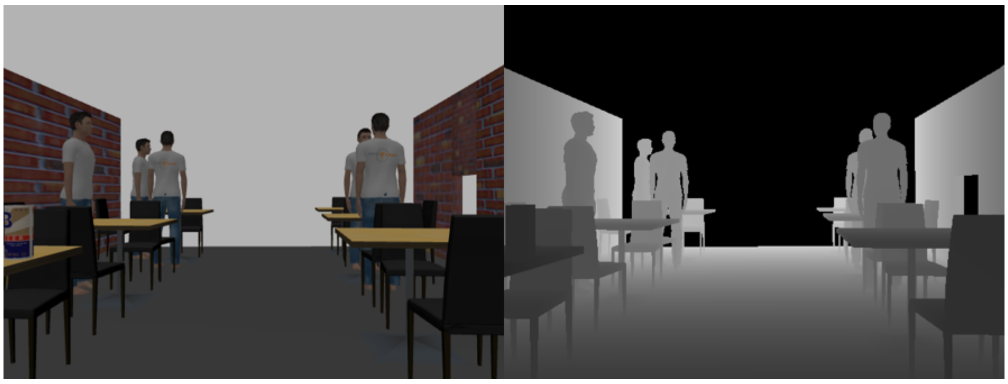


Figure 2.9: RGB along with depth Image.

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:

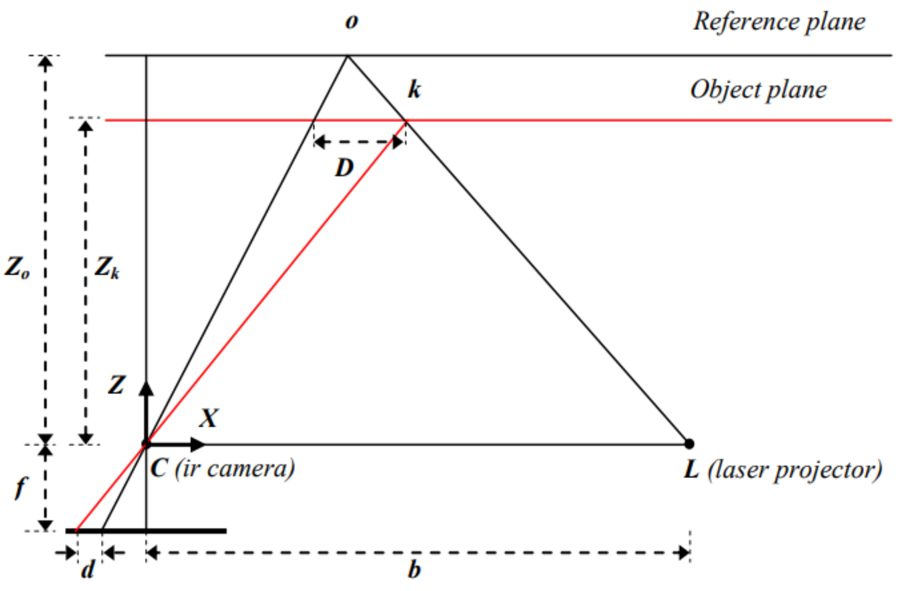


Figure 2.10: Relation between relative depth and measured disparity.

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

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

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.[11] 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.

2.8.1 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.

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 [11].

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.

2.9 Conclusion

This chapter provided an in-depth overview of autonomous mobile robots, emphasizing the mechanical and perceptual components essential to their operation. With an emphasis on wheeled mobile robots and their mathematical modeling, a variety of locomotional techniques were presented. In order to help robots understand their internal states and exterior surroundings, perception systems were further divided into proprioceptive and exteroceptive sensors. The implementation of successful autonomous navigation in organized environments is based on these fundamental ideas.Particular attention was given to microsoft Kinect RGB-Depth camera .

Bibliography

[1] Norhafizan Ahmad et al. Reviews on various inertial measurement unit (imu) sensor applications. *International Journal of Signal Processing Systems*, 1(2):256–262, 2013.

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

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

[4] Richard A. Bolt. Put-that-there: Voice and gesture at the graphics interface. *ACM SIGGRAPH Computer Graphics*, 14(3):262–270, 1980.

[5] Samir Bouzoualegh, El-Hadi Guechi, and Ridha Kelaiaia. Model predictive control of a differential-drive mobile robot. *Acta Universitatis Sapientiae, Electrical and Mechanical Engineering*, 10(1):20–41, 2018.

[6] 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*, 25(2):451–458, 2009.

[7] R. Dhaouadi and A. A. Hatab. Dynamic modelling of differential drive mobile robots, a unified framework. *Advances in Robotics & Automation*, 2(2):1–7, 2013.

[8] Frantisek Duchon et al. Some applications of laser rangefinder in mobile robotics. *Journal of Control Engineering and Applied Informatics*, 14(2):50–57, 2012.

[9] Scott S. Fisher. Telepresence master glove controller for dexterous robotic end-effectors. In David P. Casasent, editor, *Proceedings of SPIE - The International Society for Optical Engineering*, volume 726, page 396, 1987.

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

[11] K. Khoshelham and S.O. Elberink. Accuracy and resolution of kinect depth data for indoor mapping applications. *Sensors*, 12(2):1437–1454, 2012.

[12] S-B Kim et al. Ground vehicle navigation in harsh urban conditions by integrating inertial navigation system, global positioning system, odometer and vision data. *IET radar, sonar & navigation*, 5(8):814–823, 2011.

[13] Philip Kurrek, Mark Jocas, Firas Zoghlami, Martin Stoelen, and Salehi 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)*, volume 1, pages 3561–3570, 2019.

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

[15] Larry Li. Time-of-flight camera—an introduction. Technical white paper SLOA190B, 2014. Accessed: 2025-05-08.

[16] Hugh HS Liu and Grantham KH Pang. Accelerometer for mobile robot positioning. *IEEE Transactions on Industry Applications*, 37(3):812–819, 2001.

[17] Jorge Lobo et al. Sensors for mobile robot navigation. In *Autonomous Robotic Systems*, pages 239–267. Springer London, 1998.

[18] Ali Mallem. *Algorithmes d'aide à la navigation d'un robot mobile.* PhD thesis, Université de Batna 2, 2011.

[19] Ali Mallem. *Algorithmes d'aide à la navigation d'un robot mobile.* PhD thesis, Université de Batna 2, 2011.

[20] Ali Mallem. *Algorithmes d'aide à la navigation d'un robot mobile.* PhD thesis, Université de Batna 2, 2011.

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

[22] Omid Mohareri, Rached Dhaouadi, and Ahmad B. Rad. Indirect adaptive tracking control of a nonholonomic mobile robot via neural networks. *Neurocomputing*, 88:54–66, 2012. Intelligent and Autonomous Systems.

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

[24] G. Oriolo, A. De Luca, and M. Vendittelli. Wmr control via dynamic feedback linearization: design, implementation, and experimental validation. *IEEE Transactions on Control Systems Technology*, 10(6):835–852, 2002.

[25] 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*, 8(1):949–969, 2023.

[26] 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*, 16(2):1729881419839596, 2019.

[27] C. Samson, P. Morin, and R. Lenain. Modeling and control of wheeled mobile robots. In B. Siciliano and O. Khatib, editors, *Springer Handbook of Robotics*, chapter 49. Springer, Cham, 2016.

[28] Roland Siegwart, Illah Reza Nourbakhsh, and Davide Scaramuzza. *Introduction to autonomous mobile robots*. MIT Press, 2011.

[29] V. A. Zhmud et al. Application of ultrasonic sensor for measuring distances in robotics. *Journal of Physics: Conference Series*, 1015(3):032085, 2018.

Chapter 3

Deep learning and computer vision

3.1 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 Figure. 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[13].

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[14]. 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:

$$f(x) = G(wx^T + b) \tag{3.1}$$

In this expression, $w \in \mathbb{R}^d$ is the weight vector, $x \in \mathbb{R}^d$ is the input vector,

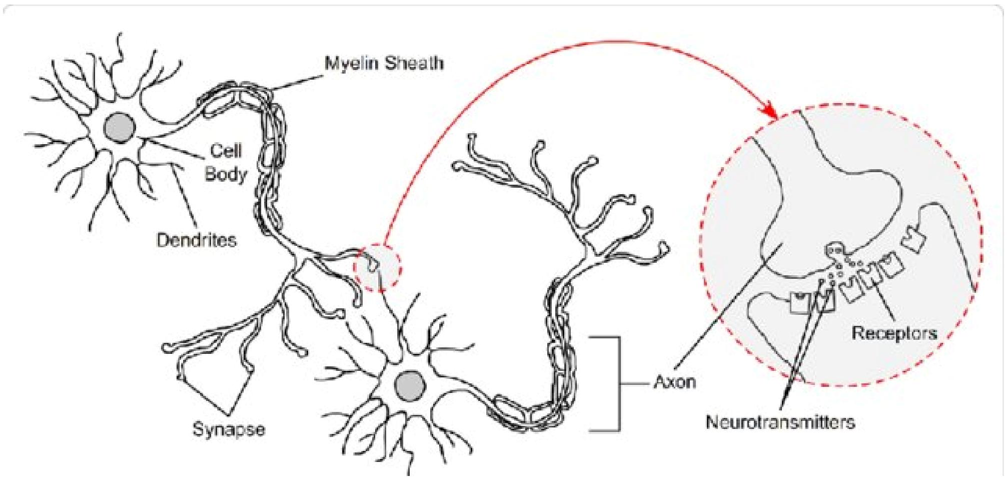


Figure 3.1: Simplified image of a biological neuron

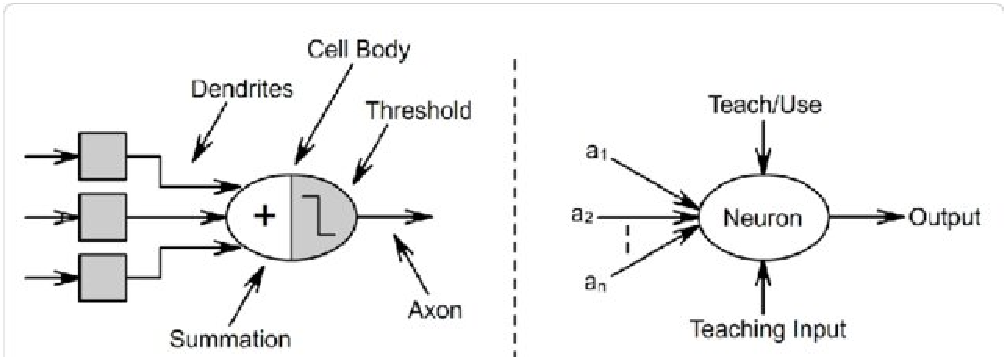


Figure 3.2: Biological neuron mapped to an artificial neuron.

and $b \in \mathbb{R}$ is the bias or intercept term. An artificial neuron performs a weighted 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, $wx^T + b$.

3.2 Component of artificial neural network

3.2.1 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.

3.2.2 Hidden layers

This layer resides in 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.

3.2.3 Output layers

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.

3.2.4 Weigh

This is a concept that the model grasps during training. As inputs are fed into the neural networks, the model assigns a value to each based on their significance, which represents a weight at a high level.

3.2.5 Biases

Constants are 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 make guarantee that neuron activation continues even when all inputs are zero.

3.3 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[27]. 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[37],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[8]. 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.

3.4 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

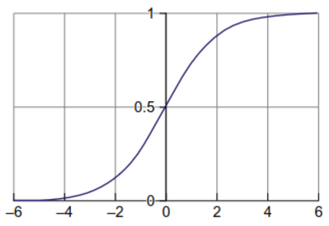


Figure 3.3: Sigmoid activation function.

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.

3.4.1 Sigmoid (Logistic) ACTIVATION FUNCTION

The following equations provide the sigmoid activation function[4]:

$$f_{\text{Sigmoid}}(x) = \frac{1}{1+e^{-x}} \tag{3.2}$$

When plotted, the function appears as shown in Figure 3.3.

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 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:

$$f'(x) = f_{\text{Sigmoid}}(x) \cdot (1 - f_{\text{Sigmoid}}(x)) \tag{3.3}$$

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.

3.4.2 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:

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

The derivative of the tanh function is:

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

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.

3.4.3 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 :

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

Its derivative is:

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

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.

3.4.4 LEAKY RELU FUNCTION

Leaky ReLU[25],represents an enhanced rendition of the ReLU function. The Learky ReLU defined as follows :

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

Its derivative is:

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

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.

3.4.5 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:

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

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.

3.4.6 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:

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

3.5 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:

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

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:

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

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)** [35], **Adam** [16], and **RMSprop** [41], each with adaptations for various scenarios in practice.

3.5.1 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.

3.5.2 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

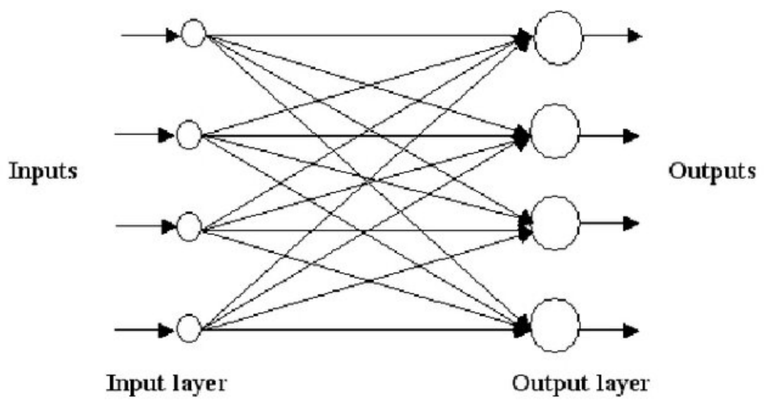


Figure 3.4: Single layer feedforward neural network

error landscape that lead to the minima.

3.6 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. [36]

3.6.1 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 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.[28]

3.6.2 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 [31], 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.

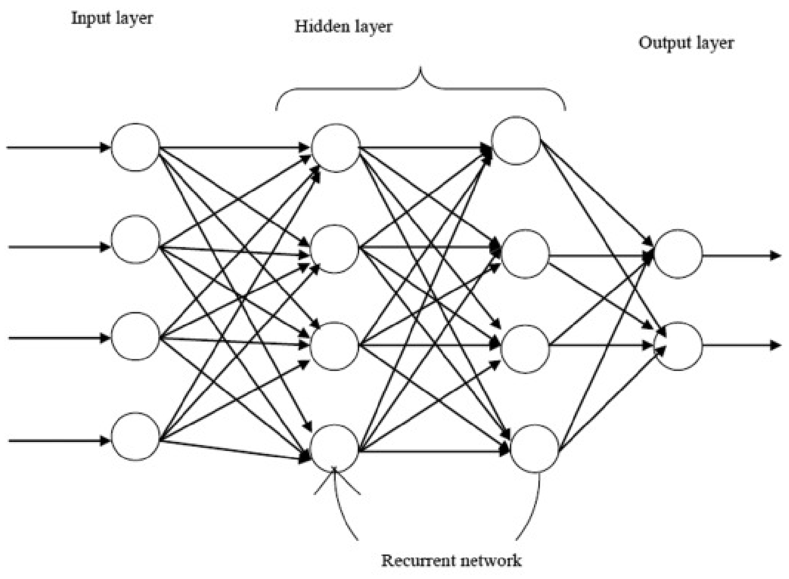


Figure 3.5: Architecture of RNN, recurrent neural networks.

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.

3.6.3 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[21]. 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[1, 17].

3.6.4 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.[7] 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.[11]

3.6.5 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 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.[43]

3.6.6 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 [3], enabling the network to detect edges, textures, and increasingly complex features as depth increases.

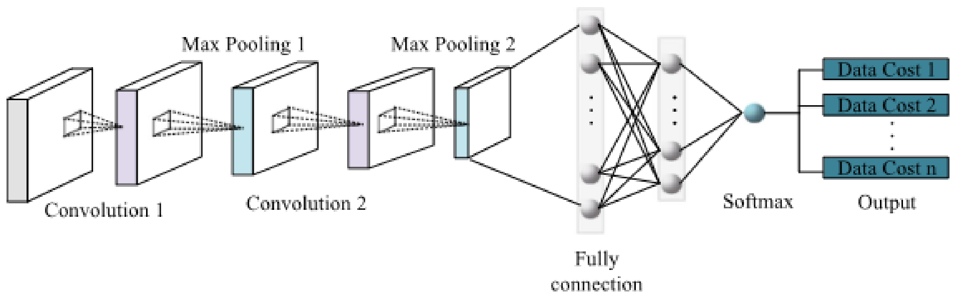


Figure 3.6: A Convolutional Neural Network Structure.

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 × W. Commonly utilized sizes 3 × 3, 5 × 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:

$$s(t) = \int x(a)w(t-a)\,da \tag{3.4}$$

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

$$s(t) = (x * w)(t) \tag{3.5}$$

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:

$$s(t) = (x * w)(t) = \sum_{a=-\infty}^{\infty} x(a)\, w(t-a) \tag{3.6}$$

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:

$$S(i,j) = (I * K)(i,j) = \sum_{m}\sum_{n} I(i-m, j-n)K(m,n) \tag{3.7}$$

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 3.7 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.

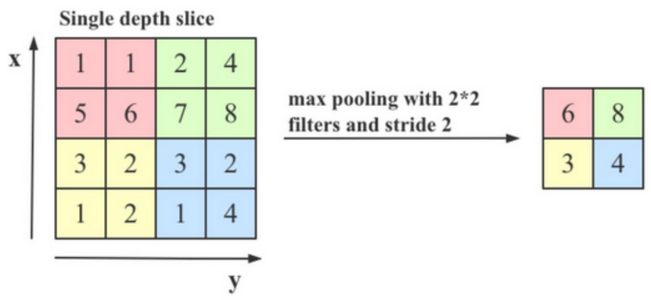


Figure 3.7: Illustration of the max pooling process: the highest activation in each local region is selected to form the downsampled output.

3.7 Deep learning for object detection and recognation

3.7.1 State-of-the-art

The field of computer vision is broad and extends beyond image processing. Its goal is to enable computers to comprehend what they see. Object detection stands as a cornerstone task in computer vision, serving as the linchpin for addressing more intricate vision challenges like 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 delves deeper into localized regions of an image and specific object categories. Although CNNs have been utilized in object detection since the 1990s, progress was initially hindered by limited training data and hardware resources such as computational power and storage capacity. However, the monumental breakthrough of CNNs in the 2012 ImageNet challenge reignited interest in deep CNN-based object detection, leading to substantial advancements 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 relied on manual design and the traditional sliding window approach. Feature extraction, such as Speeded Up Robust Features. (SURF)[6], Histogram of Oriented Gradients (HOG)[24],Scale Invariant Feature Transform (SIFT)[30]. Classifiers, including Support Vector Machine(SVM)[18] were commonly used to train individual shallow classifiers for different object classes. However, this manual feature design approach suffered from drawbacks like lack of robustness, poor generalization, and low detection accuracy, particularly in challenging environments. The limitations of traditional object detection algorithms stem from two main factors: the reliance on prior knowledge for feature extraction, which constrains the number of parameters and complicates manual parameter 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)[10] have gained widespread adoption across various domains for object detection, gradually replacing traditional detection techniques. In this context, deep learning models can be broadly categorized into two types: the two-stage model and the one-stage model. The two-stage model involves object detection being divided into two phases: firstly, generating region proposals, followed by classifying these proposals using a classifier and refining their positions. Noteworthy algorithms following this approach include R-CNN[12], SPP-Net[9] , Fast R-CNN [34], Faster R-CNN [15], and Mask R-CNN [32]. Conversely, the one-

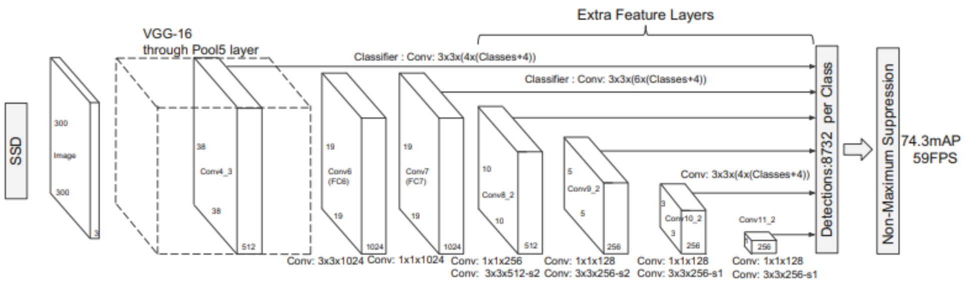


Figure 3.8: Architecture of the SSD model illustrating multi-scale feature prediction.

stage model directly predicts the object to generate the bounding box. Examples of such models include the YOLO algorithm [44], Single Shot MultiBox Detector (SSD)[45] , CenterNet[39], and EfficientDet [1ite63.

3.7.2 Single-Stage Object Detectors

Single-stage object detectors have undergone significant evolution, with each new model introducing architectural innovations and performance optimizations tailored to real-time object detection. Unlike two-stage detectors that separate the proposal and classification steps, single-stage detectors predict object locations and class labels in a unified forward pass, making them highly suitable for real-time applications. Below is a summary of several notable single-stage architectures, highlighting their core components and contributions to the field.

SSD (Single Shot MultiBox Detector)

The Single Shot MultiBox Detector (SSD) is a pioneering model that combines high accuracy with low latency. It utilizes a convolutional neural network (CNN) backbone to generate feature maps at multiple scales, enabling robust detection of objects of varying sizes. SSD assigns anchor boxes—also called default boxes—with different scales and aspect ratios to each spatial location in these feature maps. The network simultaneously predicts both the class probabilities and bounding box offsets for each anchor box. Post-processing involves non-maximum suppression (NMS) to eliminate redundant detections. The SSD detection head, built on top of the base CNN (e.g., VGG16), leverages a series of convolutional layers that make dense predictions across all feature levels [23].

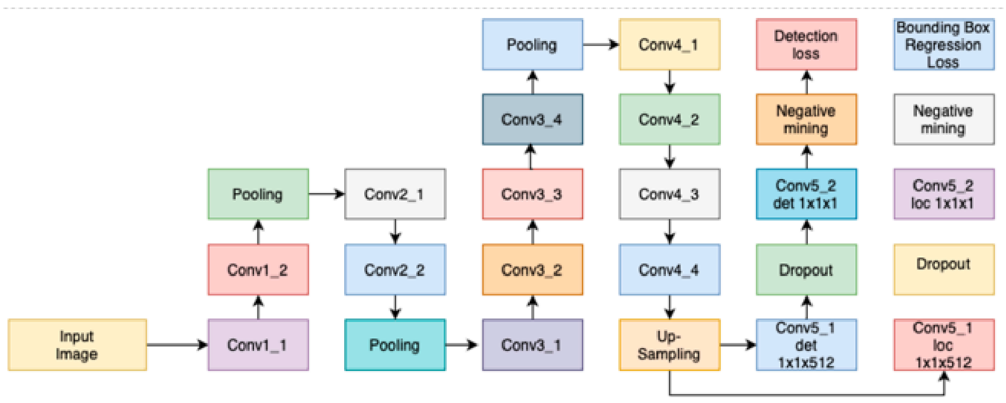


Figure 3.9: Detailed overview of the DenseBox architecture for dense prediction.

DenseBox

Proposed by Huang et al. [20], 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. [20], 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 [22], 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

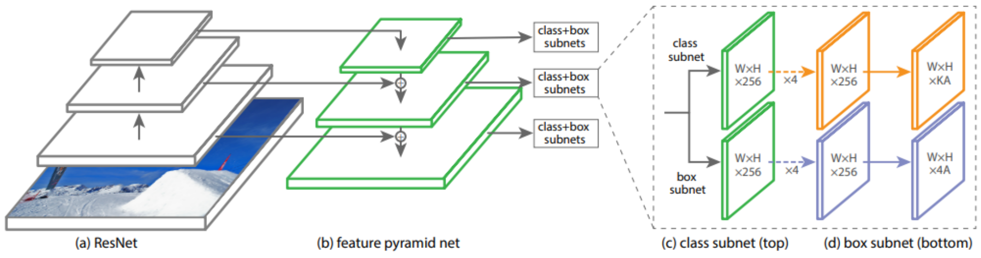


Figure 3.10: RetinaNet framework showcasing the integration of FPN and focal loss.

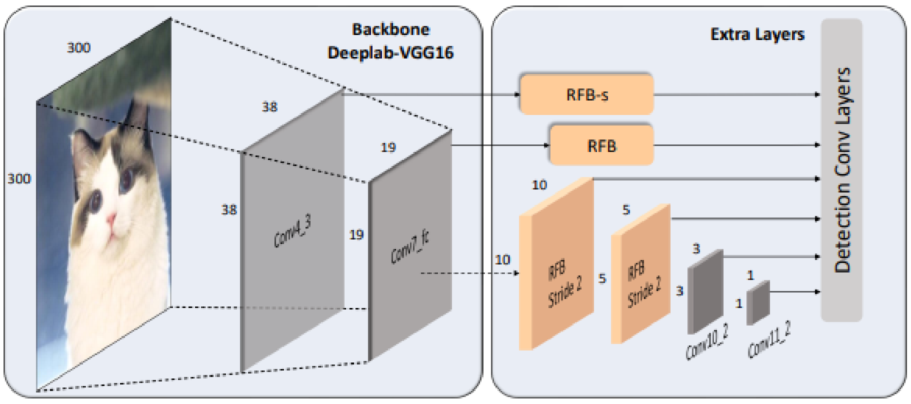


Figure 3.11: Architecture of the RFB Net with Receptive Field Blocks.

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 [40], 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.

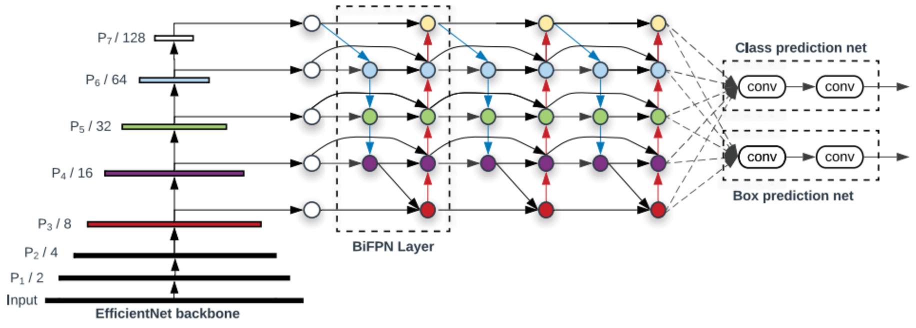


Figure 3.12: EfficientDet model architecture using BiFPN for feature fusion.

YOLO (You Only Look Once) Algorithm

YOLO (You Only Look Once) [33] 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 [26], 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 [5]. 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 [19].

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 [2].

3.8 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:

$$\begin{aligned}
&\lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 \right] \\
&+\lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[(\sqrt{w_i} - \sqrt{\hat{w}_i})^2 + (\sqrt{h_i} - \sqrt{\hat{h}_i})^2 \right] \\
&+\sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} (C_i - \hat{C}_i)^2 \qquad (3.8) \\
&+\lambda_{\text{noobj}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{noobj}} (C_i - \hat{C}_i)^2 \\
&+\sum_{i=0}^{S^2} \mathbb{1}_{i}^{\text{obj}} \sum_{c \in \text{classes}} (p_i(c) - \hat{p}_i(c))^2
\end{aligned}$$

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

3.8.1 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)[38]. 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)[42], 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

3.8.2 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:

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

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:

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

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

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

Average Precision (AP) is computed as:

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

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:

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

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:

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

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.

3.9 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.

3.10 YOLOV8 for traffic sings detection

3.10.1 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.

3.10.2 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 3.13**, 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.

During the training cycle, the model reached a training accuracy of **95.73%** with a final training loss of **0.1379**, suggesting strong learning of the input data distribution. On the validation set, an accuracy of **99.35%** and a minimal loss of **0.0202** were achieved. Similarly, test performance was robust, with an accuracy of **99.63%** and a loss of **0.0184**. The minimal discrepancies among the training, validation, and testing metrics point to excellent generalization with negligible overfitting.

From a detection perspective, the model excelled with a precision of **93.45%**, a recall of **94.28%**, and an F1-score of **93.22%**. These indicators reveal a balanced trade-off between sensitivity and specificity, confirming the model's robustness in minimizing both false positives and false negatives.

Collectively, these results validate YOLOv8's capability to deliver accurate and real-time object detection, offering high computational efficiency and reliability in unseen data environments. The model's consistency and performance make it a strong candidate for embedded applications in autonomous systems.

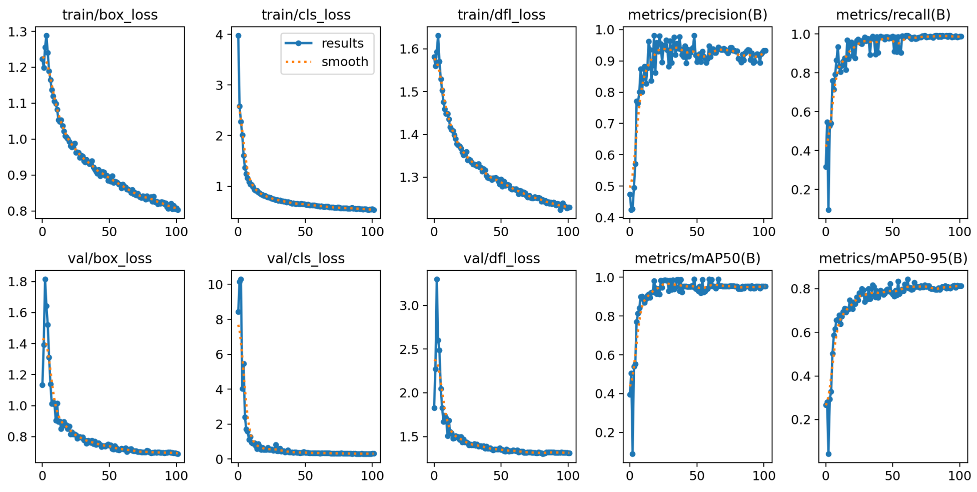


Figure 3.13: YOLOv8 model training performance metrics: accuracy, loss convergence, and mAP progression

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

3.11.1 Systeme architecture

The design of a perception system can vary depending on the specific application and requirements. As illustrated in Figure 1 [29], 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.

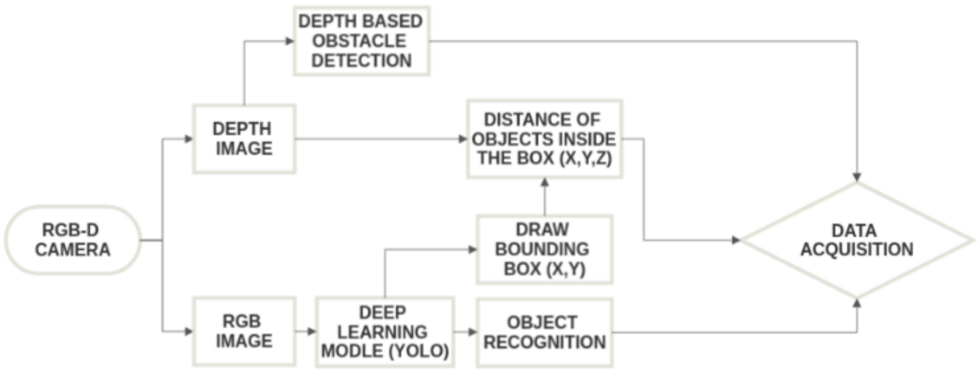

DEPTH BASED OBSTACLE DETECTION
DEPTH IMAGE
DISTANCE OF OBJECTS INSIDE THE BOX (X,Y,Z)
RGB-D CAMERA
DRAW BOUNDING BOX (X,Y)
DATA ACQUISITION
RGB IMAGE
DEEP LEARNING MODLE (YOLO)
OBJECT RECOGNITION

Figure 3.14: Systeme 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.

3.11.2 Experiments 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 4.5)(Figure ??)

- 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.

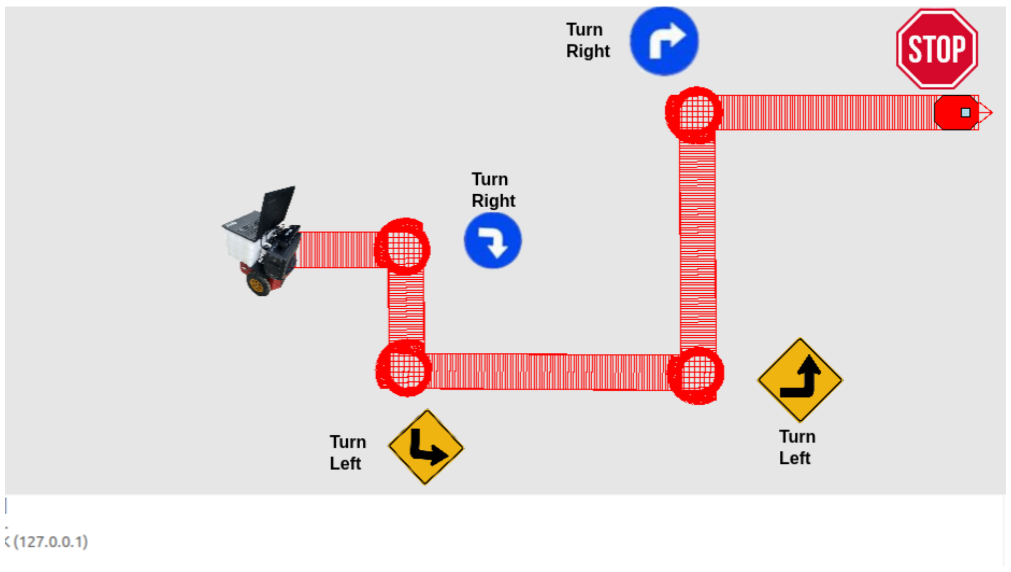


Figure 3.15: Systeme architecture

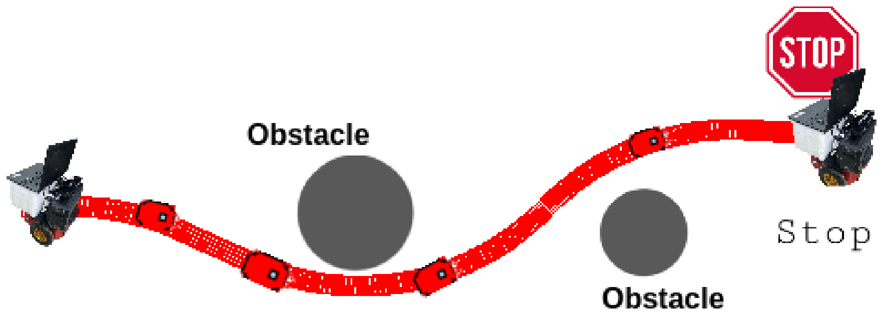


Figure 3.16: Systeme architecture

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 3.17. 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 3.17. 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.

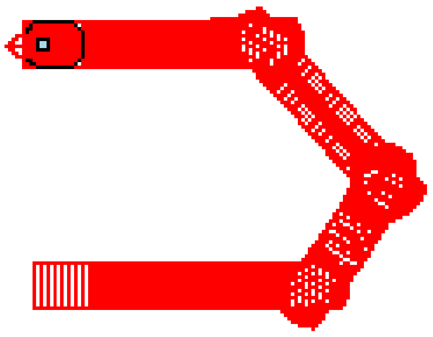


Figure 3.17: Systeme architecture

3.12 Conclusion

In summary ,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 tasks. The structure and operation of artificial neural networks were detailed.

The artificial neural network training process was then analyzed, highlighting key optimization techniques.

This chapter also reviewed state-of-the-art object detection methods, focusing on single-stage detectors like the YOLO family , and Cloud detection methods like Google Vision API . Particular attention was given to YOLOv8 due to its efficiency and accuracy. Traffic sign recognition datasets and evaluation metrics were introduced to support the validation process.This research highlights the remarkable performance of YOLOv8 when trained on a custom dataset for road sign detection, and the results of using both YOLOV8 and RGB-D for robot navigation.These results highlight the importance of employing adaptive techniques to enable autonomous systems, such as the robot in this experiment, to respond intelligently to environmental changes. By integrating obstacle detection, decision-making algorithms, and appropriate motion control, robots can navigate complex environments while minimizing the likelihood of collisions and achieving their intended goals.

Bibliography

[1] Y. I. Al Mashhadany. Recurrent neural network with human simulator based virtual reality. In *Recurrent Neural Networks and Soft Computing*. InTech, 2012.

[2] M.L. Ali and Z. Zhang. The yolo framework: A comprehensive review of evolution, applications, and benchmarks in object detection. *Computers*, 13(12):336, 2024.

[3] Dhaval Bhatt, Chintan Patel, Harsh Talsania, Jigar Patel, Raj Vaghela, Shwetank Pandya, Kushal Modi, and Hemant Ghayvat. Cnn variants for computer vision: History, architecture, application, challenges and future scope. *Electronics*, 10(20):2470, 2021.

[4] K. Chakraborty, S. Bhattacharyya, R. Bag, and A. A. Hassanien. Sentiment analysis on a set of movie reviews using deep learning techniques. In *Social Network Analytics*, pages 127–147. Elsevier, 2019.

[5] H. Chen, Z. Chen, and H. Yu. Enhanced yolov5: An efficient road object detection method. *Sensors*, 23(17):8355, 2023.

[6] N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 886–893, 2005.

[7] Ch Sanjeev Kumar Dash, Ajit Kumar Behera, Satchidananda Dehuri, and Sung-Bae Cho. Radial basis function neural networks: a topical state-of-the-art survey. *Open Computer Science*, 6(1):33–63, 2016.

[8] J. Feng and S. Lu. Performance analysis of various activation functions in artificial neural networks. In *Journal of Physics: Conference Series*, volume 1237, page 022030. IOP Publishing, 2019.

[9] R. Girshick. Fast r-cnn. *arXiv e-prints*, 2015.

[10] 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)*, pages 580–587, 2014.

[11] S. Hamedi, Z. Kordrostami, and A. Yadollahi. Artificial neural network approaches for modeling absorption spectrum of nanowire solar cells. *Neural Computing and Applications*, 31(Suppl 12):8985–8995, 2019.

[12] 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*, 37(9):1904–1916, 2014.

[13] John J. Hopfield. Artificial neural networks. *IEEE Circuits and Devices Magazine*, 4:3–10, 1988.

[14] Alex P. James. Threshnomics: An introduction to threshold logic in algorithms and circuits. *Journal of Computer Science & Systems Biology*, 7(6), 2014.

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

[16] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[17] B. Kumaraswamy. Neural networks for data classification. In *Artificial Intelligence in Data Mining*, pages 109–131. Elsevier, 2021.

[18] Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. *Nature*, 521:436–444, 2015.

[19] 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*, 2024(1):6, 2024.

[20] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. Focal loss for dense object detection. *arXiv preprint arXiv:1708.02002*, 2017.

[21] Zachary C. Lipton, John Berkowitz, and Charles Elkan. A critical review of recurrent neural networks for sequence learning. *arXiv preprint arXiv:1506.00019*, 2015.

[22] Shifeng Liu and Dianwei Huang. Receptive field block net for accurate and fast object detection. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 385–400, Munich, Germany, September 2018.

[23] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C. Berg. Ssd: Single shot multibox detector. In *European conference on computer vision*, pages 21–37. Springer, 2016.

[24] D.G. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60:90–110, 2004.

[25] Andrew L. Maas, Awni Y. Hannun, and Andrew Y. Ng. Rectifier nonlinearities improve neural network acoustic models. In *ICML Workshop on Deep Learning for Audio, Speech and Language Processing*, volume 28, 2013.

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

[27] 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*, 29(3):617–663, 2008.

[28] 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*, 29(3):617–663, 2008.

[29] 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*, 15(2):21738–21745, April 2025.

[30] F. Melgani and L. Bruzzone. Classification of hyperspectral remote sensing images with support vector machines. *IEEE Transactions on Geoscience and Remote Sensing*, 42(8):1778–1790, 2004.

[31] M.C. Popescu, V.E. Balas, L. Perescu-Popescu, and N. Mastorakis. Multilayer perceptron and neural networks. *WSEAS Transactions on Circuits and Systems*, 8(7):579–588, 2009.

[32] J. Redmon, S. Divvala, and R. Girshick. You only look once: Unified, real-time object detection. In *IEEE International Conference on Computer Vision (ICCV)*, pages 779–788, 2016.

[33] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 779–788, Las Vegas, NV, USA, 2016. IEEE.

[34] 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*, 39:1137–1149, 2017.

[35] Herbert Robbins and Sutton Monro. A stochastic approximation method. In *Herbert Robbins Selected Papers*, pages 102–109. Springer, 1985.

[36] 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*, 50(01), 2006.

[37] S. Sharma, S. Sharma, and A. Athaiya. Activation functions in neural networks. *Towards Data Science*, 6(12):310–316, 2017.

[38] 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*, pages 1453–1460, San Jose, CA, USA, 2011.

[39] M. Tan, R. Pang, and Q. Le. Efficientdet: Scalable and efficient object detection. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 10778–10787, 2020.

[40] Mingxing Tan, Ruoming Pang, and Quoc V Le. Efficientdet: Scalable and efficient object detection. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 10781–10790, Seattle, WA, USA, June 2020.

[41] Tijmen Tieleman and Geoffrey Hinton. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning*, 4(2):26–31, 2012.

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

[43] Markus Törmä. Kohonen self-organizing feature map in pattern recognition. *Photogrammetric Journal of Finland*, 15(1), 1995.

[44] L. Wei, D. Anguelov, et al. Ssd: Single shot multibox detector. In *European Conference on Computer Vision*, 2016.

[45] X. Zhou, D. Wang, and P. Krähenbühl. Objects as points. *CoRR*, 2019.

Chapter 4

Cloud Vision-Based AGV Navigation

4.1 Introduction

In the past decade, numerous techniques have been developed and applied in AGV navigation. In [1], 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 [9] , 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 [19], a navigation system was proposed for AGVs, using metal-line sensors for track guidance and RFID tags for localization. In [23], an AGV navigation approach used double magnetic nails, achieving reliable path tracking and stable accuracy through a fuzzy controller. In [11], 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 [5]. Popular algorithms include A* (A-star), D* Lite, and Dijkstra's algorithm [16]. 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 [3, 4, 17]. In [18], 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 [24], a monocular method was proposed that combined object detection and distance estimation using an R-CNN-based regression network. In [20], 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 [21], 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 [7], 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 [25], the Libra R-CNN algorithm was combined with a balanced feature pyramid to improve 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) [15] 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 [12, 26].

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 [2]. 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 [13, 27].

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.

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.

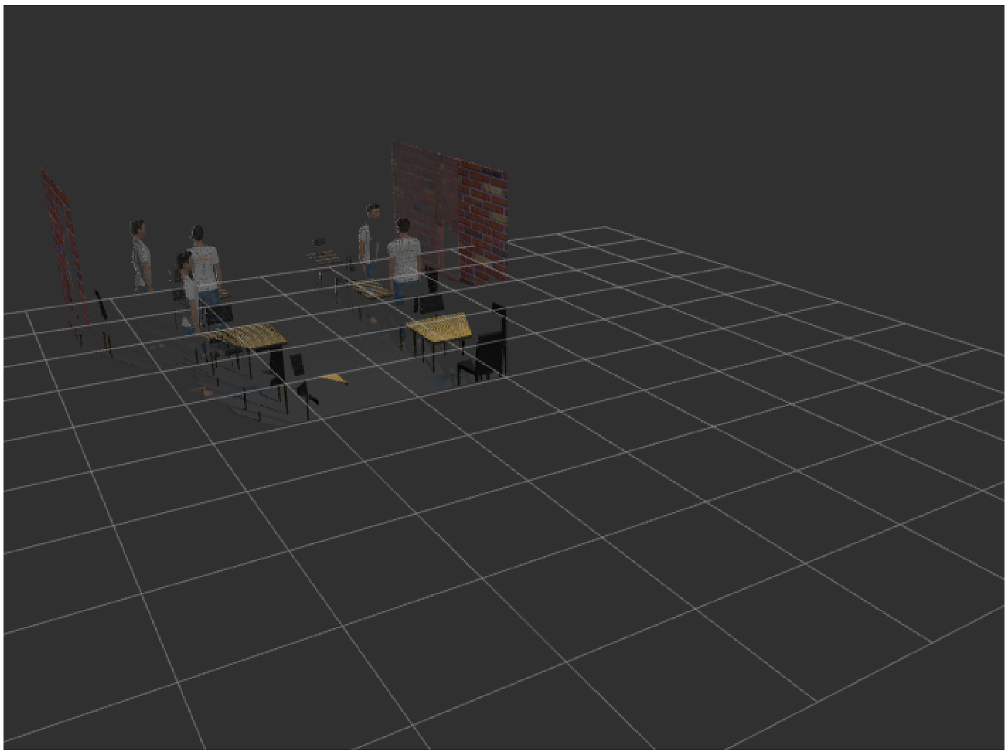


Figure 4.1: RGB-D Point cloud representation.

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

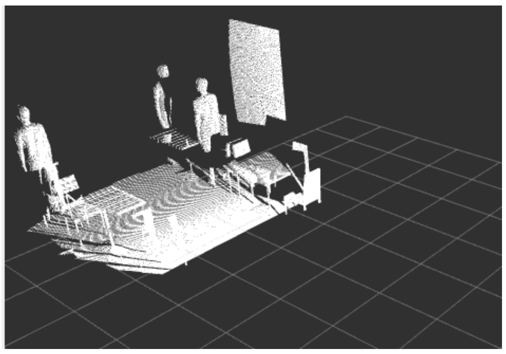
An unsupervised clustering approach called DBSCAN is especially helpful for datasets that contain noise and high-density regions. The radius ε (Eps-neighborhood of a point p) and MinPts (the minimum number of points in an Eps neighborhood of the point p required to make p a core point of a cluster) are 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.

Equation defines the neighborhood $N_\varepsilon(x_i)$ of the i^{th} object 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 distance function, which is expressed as:

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


(a)

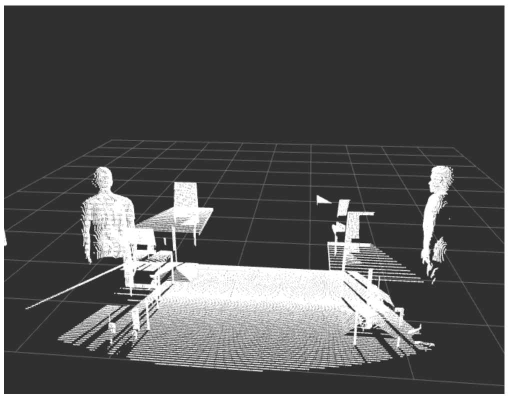

(b)

Figure 4.2: Filtred Point Cloud

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 MNOP, the minimum number of objects in the neighborhood. MNOP distinguishes three types of objects: core objects, boundary objects, and noise. Core objects are defined as follows:

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

Objects that satisfy this condition are labeled as core objects x_{core}. On the other hand, boundary objects are defined by the condition:

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

Objects that satisfy this condition are labeled as boundary objects x_{border}. Finally, noise objects are data points that do not meet the required criteria:

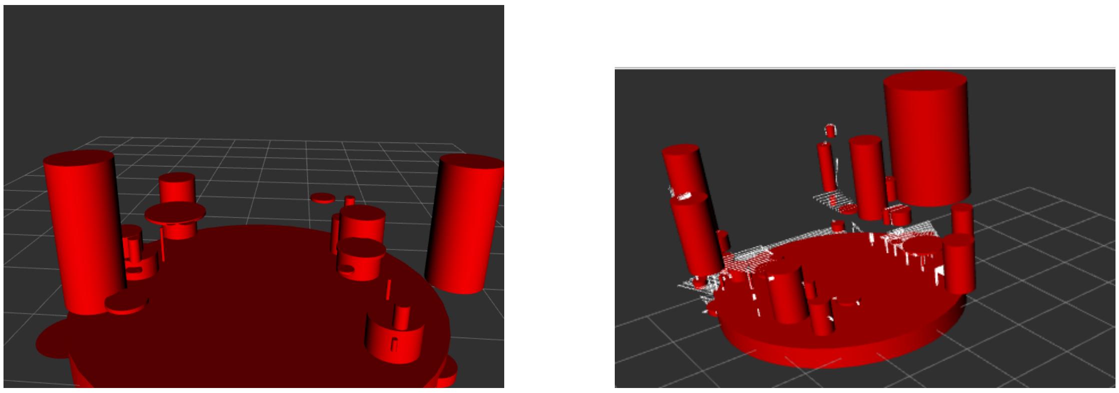
$$\rho(x_i) < \text{MNOP} \quad \text{and} \quad x_i \notin N_\varepsilon(x_{\text{core}})$$

These objects are denoted as x_{noise}.

The DBSCAN algorithm is designed to cluster data based on density. It 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, i.e.,

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



(a) (b)

Figure 4.3: Density-Based Spatial Clustering

where $D(x_j, x_i)$ represents the distance between points x_j and x_i.

In the case of core objects, an object chain can be formed such that $p_1, \ldots, p_N$ with $p_1 = x_i$ and $p_N = x_j$. When points lie outside the ε-neighborhood of a core object x_i but are within the ε-neighborhood of other core objects, those points are still considered *density-reachable* from x_i.Algorithm 1 shows clustering wking methodology .Based on these definitions, each core object x_i, along with all points that are density-reachable from it (including those directly density-reachable), collectively form a cluster. [22, 6]

For synchronized processing downstream modules, such as object tracking, sensor fusion with camera-based detectors, or semantic reasoning engines, which need a temporally coherent and structured representation of all clusters detected at a specific timestamp, aggregate individual cluster markers published one at a time and publish them collectively as a MarkerArray. We implement a Marker Aggregator Node that combines distinct Marker messages into a structured MarkerArray message in order to satisfy this need.

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

Algorithm 2 Point Cloud Transformation and Filtering Algorithm

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

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[10] 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[8] determines the grayscale value by computing the average of all three color channels:

Algorithm 3 Cluster Detection Algorithm

```
 1: Initialize ROS node and set up subscriber for PointCloud2 topic
 2: Initialize visualization publisher for cluster markers
 3: Upon receiving a PointCloud2 message:
 4:         Convert it to Open3D point cloud format
 5:         Down sample the point cloud using uniform grid
 6:         Remove non-finite points from the cloud
 7: if SEGMENTATION is enabled then
 8:     for up to 5 times do
 9:             Segment and remove planar components using RANSAC if plane > 80 points
10:     end for
11: end if
12: if number of points < threshold (e.g., 10) then
13:     Exit callback
14: end if
15: Apply DBSCAN clustering using EPS_CLUSTER and MIN_POINTS_CLUSTER
16: if no clusters found then
17:     Exit callback
18: end if
19: for each cluster $D_k$ do
20:         Extract cluster points and compute bounding box
21:         Determine max width (x or y) and height from bounding box
22:         Compute centroid of the cluster
23:         Create and publish a Marker of type CYLINDER at the centroid with calculated dimensions
24: end for
25: Clear old markers by publishing a DELETEALL Marker
```

Algorithm 4 Centroid Objects Marker Collection Algorithm

```
1: Initialize ROS node
2: Subscribe to /centroid_objects for individual Marker messages
3: Initialize publisher to publish MarkerArray messages to
   /centroid_objects_list
4: Maintain a global MarkerArray list to accumulate received markers
5: While running:
6:         When a Marker message is received:
7: if the marker contains points then
8:             Append the marker to the MarkerArray list
9: end if
10: if the marker contains no points (interpreted as end of batch) then
11:             Publish the collected MarkerArray to /centroid_objects_list
12:             Clear the MarkerArray list for the next batch
13: end if
14: Continue this process indefinitely using rospy.spin()
```

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

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[14], 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.

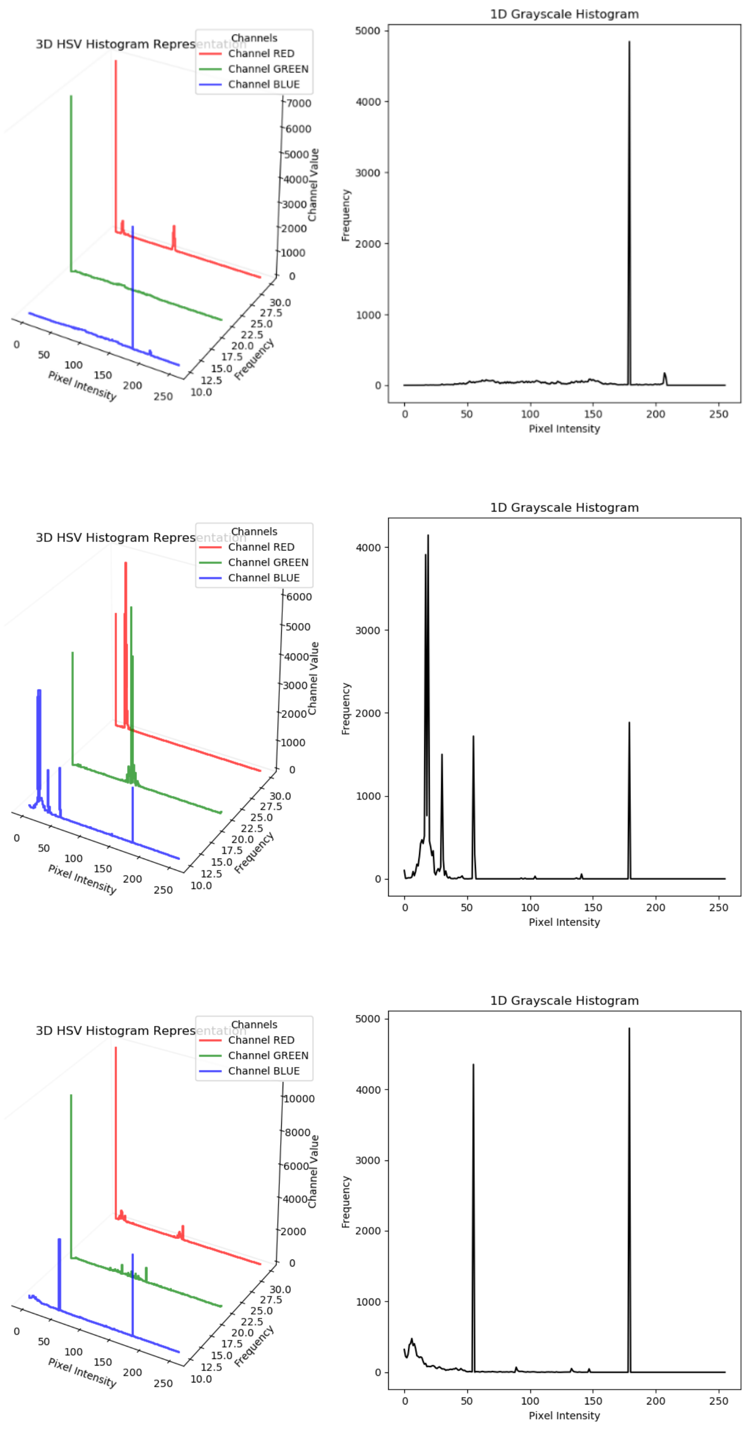
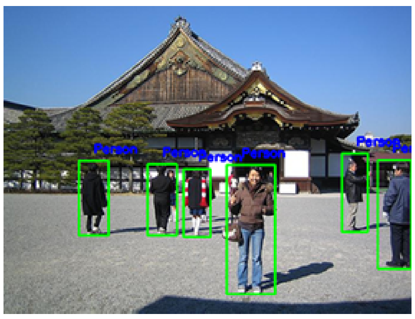


Figure 4.4: colored and grey scale representation of object with the same class id and diffrent shape

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 grayscale. 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.

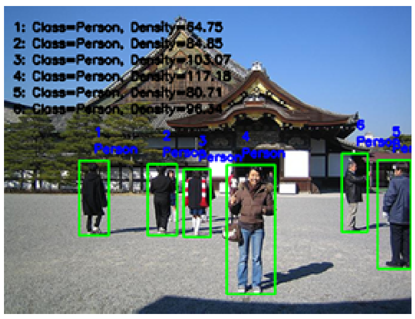

1: Class=Person, Density=64.75
2: Class=Person, Density=84.85
3: Class=Person, Density=103.07
4: Class=Person, Density=117.18
5: Class=Person, Density=80.71
6: Class=Person, Density=96.34
1 Person
2 Person
3 Person
4 Person
6 Person
5 Person
(c) Our detection results.

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

Algorithm 5 DetectedObjectsCallback (GCP + 3D Projection Fusion)

```
1: Wait for depth objects, camera info, RGB image, and robot pose
2: Compute robot rotation matrix $R_{rob}$ using quaternion
3: Get projection matrix from camera info
4: **try**:
5:    Lookup transform from map to xtion_rgb_optical_frame
6: **catch exception**:
7:    **return**
8: Compute rotation matrix $R$ and translation vector $T$ from transform
9: Initialize intersection list for pointcloud markers
10: Get GCP detection bounding box $A = (x_{A1}, y_{A1}, x_{A2}, y_{A2})$
11: Compute GCP detection size and area $S_A$
12: **for** each depth object marker $m_i$ **do**
13:     Get 3D bounding box $(B_1, B_2)$ from marker dimensions
14:     Transform bounding box to camera frame using $R^T(B - T)$
15:     Project 3D corners to image space using intrinsics
16:     Compute projected box $B = (x_{B1}, y_{B1}, x_{B2}, y_{B2})$
17:     Compute area $S_B$ and intersection $S_I$
18:     Compute union area $S_U$ and IoU: $\text{IoU}_i = \frac{S_I}{S_U}$
19: **end for**
20: $i_{max} \leftarrow \arg\max(\text{IoU})$
21: **if** $\text{IoU}_{i_{max}}$ > threshold **then**
22:     Check for existing detected objects with overlapping volume
23:     **if** volume overlap is high **then**
24:         **if** new IoU is better **then**
25:             Replace old object with new detection
26:         **end if**
27:     **else**
28:         Add new detected object
29:     **end if**
30:     Assign scale based on object class
31:     Publish markers: POI, last object, label, label list
32:     **if** SPEECH enabled **then**
33:         Use TTS to say detection
34:     **end if**
35: **end if**
```

4.5 Systeme architecture

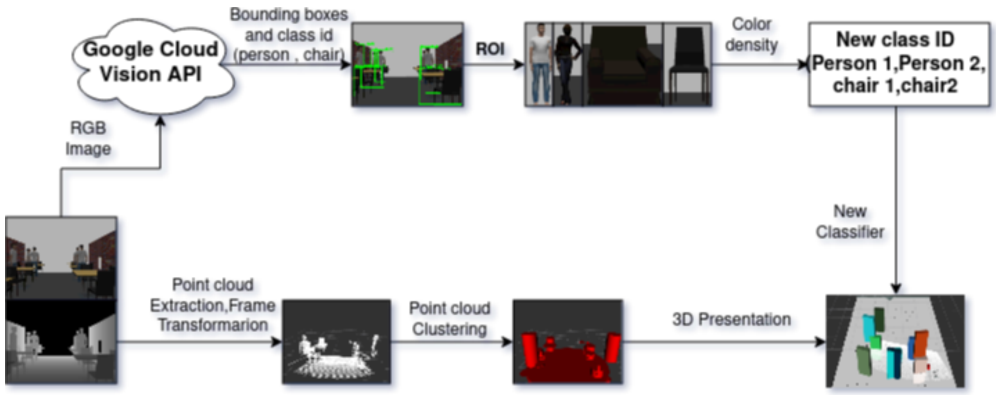

Google Cloud Vision API
Bounding boxes and class id (person , chair)
ROI
Color density
New class ID Person 1,Person 2, chair 1,chair2
RGB Image
New Classifier
Point cloud Extraction,Frame Transformarion
Point cloud Clustering
3D Presentation

Figure 4.6: Systeme architecture.

The system architecture integrates cloud-based vision processing to enhance the accuracy and effectiveness of object detection and classification. The core of this system is built around fundamental components of object detection, specifically focusing on bounding box coordinates and pre-classification IDs. The architecture is designed to improve the speed and accuracy of object recognition and categorization by leveraging cloud vision services, thereby optimizing the platform's overall vision processing capabilities for real-time applications.

One of the challenges in object classification arises when two objects have identical physical characteristics, such as height and weight, resulting in Google Cloud Platform (GCP) assigning the same pre-classification ID. For example, individuals of varying heights, weights, and clothing styles may be represented by the same class ID. To mitigate this, the system employs an additional step to re-identify objects by adding the intensity value of the object and representing its location in three-dimensional space. This approach results in more accurate and effective detection, ensuring that each object is correctly identified.

Figure 4.6 illustrates the overall flow diagram of the system architecture. The three-dimensional representation of the scene, derived from the depth image, and the object recognition provided by GCP from the RGB image, form the foundation for locating objects within the 3D space. GCP extracts the bounding box coordinates and pre-classification IDs for each detected object. The method for cropping the Region of Interest (ROI) from the image, involves extracting the bounding box coordinates and pre-classification IDs via GCP.

The key functions of the ROI algorithm are as follows:

1. Convert the cropped image to grayscale.

2. Calculate the object's intensity density.
3. Generate a new class ID for each object, based on the previous class ID.

The ROIs, along with their corresponding 3D representations produced through point cloud clustering (depicted in Figure 4.6), provide additional color information that is ultimately used to represent and locate objects within the 3D environment, as shown in Figure 4.6. The GCP service was called 21,971 times, with zero errors recorded, as shown in Table 4.1. A total of 22,000 individual image analysis requests were made to the API during this process, confirming the service's robustness and error-free performance during the evaluation.

Table 4.1: APIs and Services

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

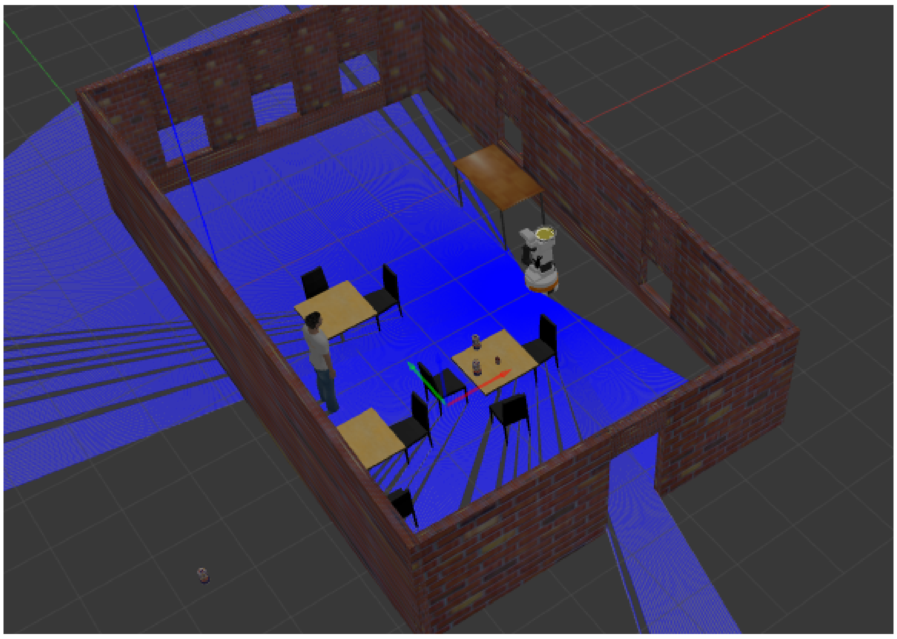
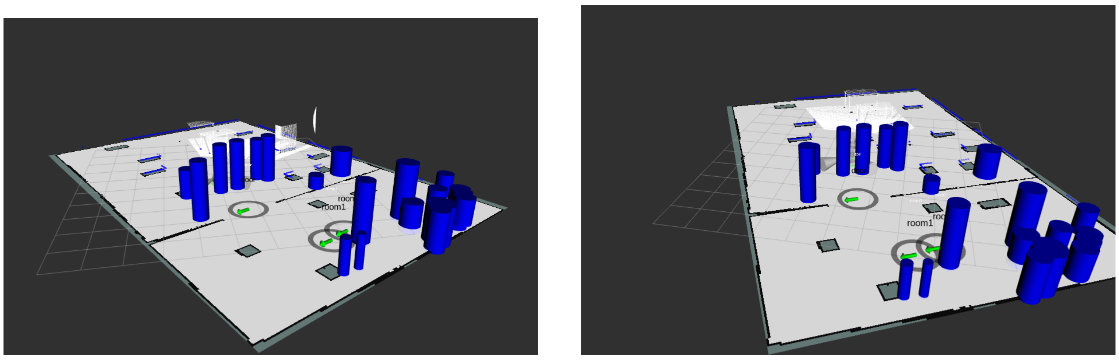
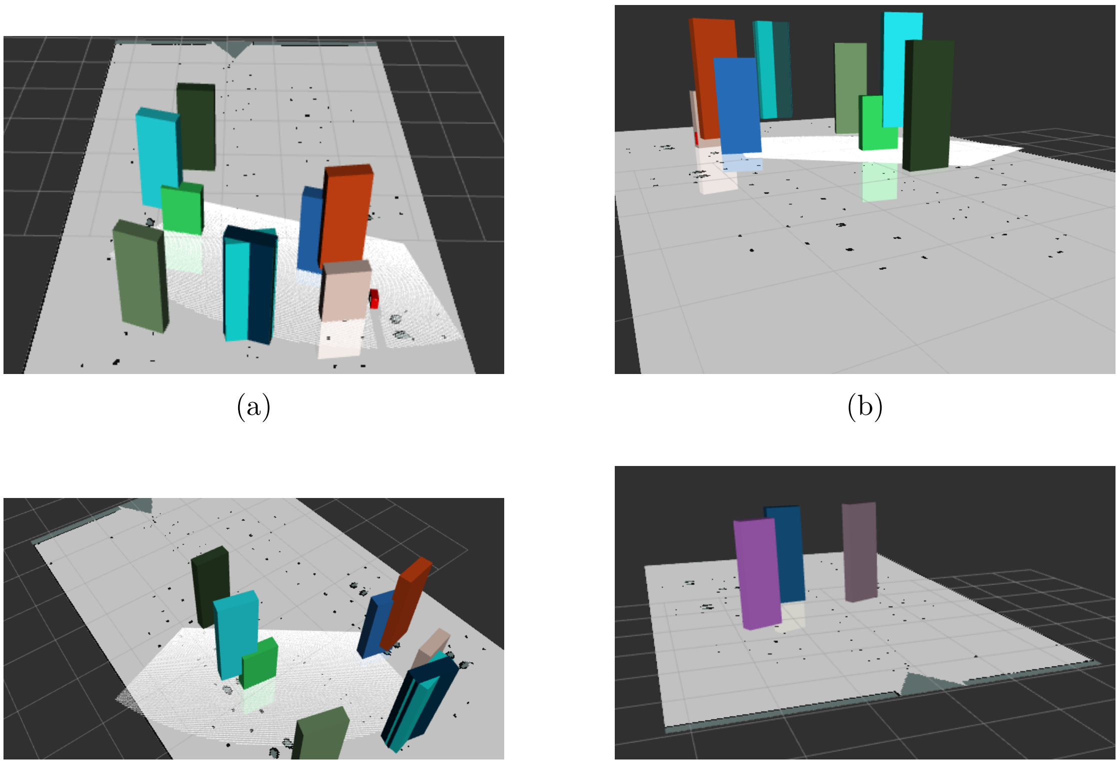


Figure 4.7: Simulation environment



(a) (b)

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



(a) (b)

(c) (d)

Figure 4.9: 3D representation of the envirenement.

To test The system a 3D environment on GAZEBO simulation platforms was created which gives more exibility in terms of moving and changing objects colors.

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 captured 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.

Bibliography

[1] M. Aizat, A. Azmin, and W. Rahiman. A survey on navigation approaches for automated guided vehicle robots in dynamic surrounding. *IEEE Access*, 11:33934–33955, 2023.

[2] 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)*, pages 4217–4223. IEEE, 2016.

[3] 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*, 48(2):1427–1436, 2023.

[4] M. Ding et al. Learning depth-guided convolutions for monocular 3d object detection. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 11669–11678, Seattle, WA, USA, 2020.

[5] 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*, 294(2):405–426, 2021.

[6] Ziye Guo, Hui Liu, Lei Pang, Li Fang, and Wenna Dou. Dbscan-based point cloud extraction for tomographic synthetic aperture radar (tomosar) three-dimensional (3d) building reconstruction. *International Journal of Remote Sensing*, 42(6):2327–2349, 2021.

[7] W. Huang, M. Huang, and Y. Zhang. Detection of traffic signs based on combination of gan and faster-rcnn. *Journal of Physics: Conference Series*, 1069(1), 2018.

[8] C. Kanan and G. W. Cottrell. Color-to-grayscale: Does the method matter in image recognition? *PLoS ONE*, 7(1):e29740, 2012.

[9] 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)*, pages 4775–4779, Shenyang, China, 2018.

[10] S. Liu. Two decades of colorization and decolorization for images and videos. *Journal Name*, 2022.

[11] 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)*, pages 1–6, Lisbon, Portugal, 2019.

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

[13] T.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*, pages 12–22. ACM, 2021.

[14] 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*, 9(6):16, 2016.

[15] Morgan Quigley, Ken Conley, Brian Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, and Andrew Ng. Ros: an open-source robot operating system. volume 3, 01 2009.

[16] 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*, 2022:1–1, 2022.

[17] 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)*, pages 8551–8560, Nashville, TN, USA, 2021.

[18] D. H. Dos Reis, D. Welfer, M. A. De Souza Leite Cuadros, and D. F. T. Gamarra. Mobile robot navigation using an object recognition software with rgbd images and the yolo algorithm. *Applied Artificial Intelligence*, 33(14):1290–1305, 2019.

[19] 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*, 15(20):2291–2297, 2020.

[20] 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)*, pages 1–6, San Diego, CA, USA, 2021.

[21] 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*, 19(10), 2019.

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

[23] Q. F. Yan, H. Yan Hu, T. P. Hang, and Y. Si 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)*, pages 1732–1735, Chongqing, China, 2019.

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

[25] Z. Zhao, H. Liu, and D. Cao. Improved traffic sign detection algorithm based on libra r-cnn. *Journal of Mechanical Engineering*, 57(22):255–265, 2021.

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

[27] 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*, 37:625–652, 2018.

Chapter 5

Conclusion General

The field of autonomous mobile robotics has witnessed remarkable advancements in recent years, driven particularly by the rapid development of artificial intelligence (AI) and related technologies. These innovations have enabled robots to evolve beyond basic automated devices into intelligent systems capable of perceiving, comprehending, and interacting with dynamic, complex environments in real time.

A primary goal of this research is to significantly enhance robot vision. To achieve this, recent developments in deep learning models such as YOLO were utilized and fused with depth information to ensure robustness and accuracy. The focus was on creating a real-time traffic sign detection system that incorporates depth data, which was then implemented to guide the robot autonomously through its environment without relying on remote information.

A significant contribution of this work is the integration of cloud-based vision processing with real-time 3D point cloud analysis for autonomous guided vehicle (AGV) navigation. Through advanced filtering, clustering, and grayscale intensity analysis, the system was capable of constructing a rich 3D representation of the environment and its surrounding objects. This approach opens new possibilities in the field of 3D environment modeling and demonstrates the potential of cloud services to reduce the need for onboard hardware and processing power.

Looking ahead, future research should investigate the limitations of current cloud services—such as latency, reliability, data privacy, and bandwidth constraints—and work towards enhancing these aspects to better support real-time, large-scale autonomous robotic systems. Additionally, expanding the generalization of AI models across diverse environments and exploring collaborative multi-robot systems enabled by cloud intelligence will be vital for the next generation of intelligent robots. Continued innovation in these areas promises to accelerate the deployment of autonomous systems capable of operating seamlessly in increasingly complex and interconnected worlds.