POLITECNICO DI TORINO

Master's Degree in Automotive Engineering - Autonomus and Connected Vehicles

Master's Degree Thesis

Road Actor Recognition and Classification for First-Person Real-Time 3D Rendering

Supervisors

Prof. Ezio SPESSA

Ing. Claudio RUSSO

Candidate

Stefano GARRONE

July 2024

Summary

This thesis introduces a cutting-edge approach to enhance the perception capabilities of autonomous vehicles in real-world driving scenarios. The research aims to recognize and classify various road actors, including vehicles, pedestrians, cyclists, and obstacles, employing advanced computer vision techniques, and leveraging tools like Qt Quick 3D and C++ programming language. The study initiates with an in-depth review of existing methodologies in road actor recognition and classification, outlining their strengths, limitations, and avenues for improvement. Building upon this foundation, the thesis proposes an approach integrating state-of-the-art deep learning algorithms with real-time 3D rendering techniques, using Qt Quick 3D. The key part of this thesis is the creation and implementation of an algorithm capable of extracting environmental information using a stereo camera. This algorithm can recognize and classify all road actors within its field of view. The system leverages convolutional neural networks (CNNs) for feature extraction and classification, enabling it to understand the position and orientation of vehicles. Additionally, Qt Quick 3D is used to develop a human-machine interface (HMI) that provides real-time 3D representations of the road environment. The use of C++ ensures seamless integration and efficient execution of the proposed solution. The research outcome will contribute to the ongoing efforts towards developing safer and more efficient autonomous vehicles capable of navigating complex real-world environments with enhanced perception capabilities.

Acknowledgements

At this significant moment in my life, I find myself writing these words of gratitude for the second time, a tangible sign of another milestone achieved. This complex journey, rich in emotions and satisfactions but also filled with effort and constant commitment, has finally come to an end. Therefore, with sincere appreciation and gratitude, I take this opportunity to express my thanks to those who made this path possible. I wish to express my profound gratitude to the entire Bylogix team and Professor Spessa. Their support, patience, availability, and professionalism have been fundamental during the months of elaboration and development of my thesis project. A special thank you goes to my family. With their constant support and great ability to never make me feel the weight of anything, they have made all this possible. Thank you, because without them, none of this would have been possible. A special mention to my ADV program companions, now my second family. From Vane to coach Tony, Matte, Vale, and everyone else without exception. You are extraordinary people with whom I share a great daily sporting passion, filled with joys, achievements, sweat, and effort. Parallel to my academic journey, you have been support and motivation both inside and outside the floor and have significantly shaped me into the person I am today, so thank you. A heartfelt thank you to all my friends, particularly Zop, Matte, Marco, and everyone who has always been by my side during these months. Being there always is not a given, but you have always demonstrated it without hesitation. Finally, thank you to those who have recently entered my life and to those who, for various reasons, are no longer present. Even if your role in this phase of my life was temporally limited for obvious reasons, your contribution has been precious and significant to me and also fundamental.

Thank you all from the bottom of my heart.

Stefano Garrone

In questo momento significativo della mia vita, mi ritrovo per la seconda volta a scrivere queste parole di ringraziamento, segno tangibile di un altro traguardo raggiunto. Questo viaggio complesso e ricco di emozioni, soddisfazioni, ma anche di fatica e impegno costante, è finalmente giunto alla sua conclusione. È quindi con sincero apprezzamento e gratitudine che colgo l'occasione per esprimere il mio grazie a coloro che hanno reso possibile questo percorso. Desidero esprimere la mia profonda gratitudine a tutto il team di Bylogix e al professor Spessa. Con il loro supporto, la loro pazienza, disponibilità e professionalità sono stati fondamentali durante i mesi di elaborazione e sviluppo del mio progetto di tesi. Un ringraziamento speciale va alla mia famiglia. Con il loro costante sostegno e la grande capacità di non farmi pesare ne mancare mai nulla, hanno reso tutto questo possibile. Grazie perché senza di loro, nulla di tutto ciò sarebbe stato possibile. Menzione speciale ai miei compagni di ADV program, ormai la mia seconda famiglia. Da Vane a coach Tony, Matte, Vale e tutti gli altri nessuno escluso. Siete persone straordinarie con cui condivido quotidianemnte una grande passione sportiva condita di gioe, traguardi, sudore e fatica. In modo parallelo al mio percorso universitario siete stati sostegno e motore fuori e dentro il floor e in modo significativo mi avete reso la persona che sono oggi, quindi grazie. Un sentito ringraziamento a tutti i miei amici, in particolare a Zop, Matte, Marco e a tutti coloro che sono stati sempre al mio fianco in questi mesi. Esserci sempre non è scontato, ma voi me l'avete sempre dimostrato senza esitazioni. Infine, un grazie a chi è entrato di recente nella mia vita e a chi, per diversi motivi, non è più presente. Anche se il vostro ruolo in questa fase della mia vita è stato temporalmente limitato per ovvie ragioni, il vostro contributo per me è stato prezioso e significativo e altrrsì fondamentale.

Grazie di cuore a tutti.

Stefano Garrone

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Road Actor Recognition and Classification for First-Person Real-Time 3D Rendering

Road actor recognition and classification for first-person real-time 3D rendering constitute a pivotal frontier in the realm of autonomous driving and computer vision. As vehicles navigate through complex urban environments, the ability to accurately detect, identify, and classify various entities sharing the road, including vehicles, pedestrians, cyclists, and obstacles, is paramount for ensuring safe and efficient navigation. At the heart of this endeavor lies the fusion of sensor data acquired from many different sources, including LiDAR, cameras, radar, and other environmental sensors. Through sophisticated sensor fusion techniques, the system integrates information from these diverse modalities to construct a comprehensive understanding of the vehicle's surroundings in real-time. Feature extraction emerges as a critical step in this process, wherein relevant features are extracted from the sensor data to facilitate robust classification. Geometric, appearance-based, and deep learning-based approaches are employed to discern salient characteristics of road actors, enabling the system to differentiate between various entities with precision. Classification algorithms play a pivotal role in the subsequent categorization of detected objects into predefined classes. Traditional machine learning methods such as Support Vector Machines and Random Forests, alongside deep learning architectures like Convolutional Neural Networks and Recurrent Neural Networks, are leveraged to classify road actors accurately and efficiently. Real-time 3D rendering serves as the interface through which the system provides timely and immersive visual feedback to autonomous driving systems. By seamlessly integrating classification results with sensor data, the system generates dynamic 3D representations of the vehicle's surroundings, enabling it to make informed decisions in real-time. However, the road to achieving robust road

actor recognition and classification is fraught with challenges. Occlusions, varying environmental conditions, and computational constraints impose significant hurdles that demand innovative solutions. Continued research and development efforts are essential to overcome these challenges and propel the field of autonomous driving forward, ultimately ushering in a future where vehicles navigate with unprecedented safety and efficiency.

1.2 The Goal of the Thesis

The central aim of this thesis revolves around the development of a Human-Machine Interface (HMI) tailored specifically for drivers, with a primary emphasis on providing a representation of the vehicle surroundings in real-time, especially focusing on the frontal area with all the information obtained by a stereocamera. This HMI system is meticulously crafted to elevate driver awareness and ensure safety through the precise detection, recognition, and classification of diverse objects encountered on the road. Moreover, it endeavors to compute the distance of these objects from the vehicle, delineate their spatial coordinates, and ascertain their orientations relative to the vehicle's vantage point, leveraging the capabilities of a meticulously designed stereo camera setup. In the pursuit of this multifaceted objective, the thesis embarks on a comprehensive exploration of the contemporary state-of-the-art technologies and methodologies underpinning object detection and classification. This journey entails a meticulous comparative analysis of various techniques to discern the most efficacious approaches conducive to real-time application within the dynamic context of driving environments. Through a discerning lens, the thesis scrutinizes the nuances of different methods, unraveling how each technique grapples with the challenges posed by the detection and recognition of objects across a spectrum of environmental conditions. Moreover, the thesis endeavors to unravel the intricate mechanisms governing distance estimation, drawing insights from the realms of disparity and depth perception intrinsic to pairs of stereo camera images. Delving deep into these fundamental processes, the research aims to elucidate how these mechanisms synergistically contribute to the precise calculation of object distances and spatial coordinates within a three-dimensional framework. This scholarly endeavor is underpinned by an exhaustive review of existing literature and cutting-edge technologies, thereby facilitating a nuanced understanding of the prevailing capabilities and limitations inherent to such systems. Through the culmination of this exhaustive study, the thesis not only endeavors to construct an advanced HMI tailored for drivers but also aspires to make substantive contributions to the broader domain of computer vision and autonomous driving. By meticulously evaluating and enhancing existing technologies, the research aims to foster the evolution of a robust system poised

to enhance road safety by furnishing drivers with real-time, granular insights into their immediate surroundings, thereby fortifying their capacity to navigate complex driving scenarios with heightened efficacy and confidence.

1.3 Thesis Organization

This thesis is structured into five distinct chapters, each contributing uniquely to the overarching research endeavor. The inaugural chapter serves as a gateway to the thesis, offering a comprehensive introduction that not only sets the stage for the subsequent discussions but also outlines the principal themes and objectives of the research journey. In the following second chapter, a detailed examination of the existing literature on Visual Perception, Ego-motion, Occupancy Grids, and Object Detection and Classification is conducted. This chapter aims to offer a thorough overview of the theoretical foundations underpinning the research. Furthermore, it explores the complexities of Stereo Cameras and Stereoscopic Vision, highlighting their importance in the study's context. Finally, it delves into various neural network technologies, providing an in-depth analysis. Following this exhaustive literature review, the third chapter is dedicated to the exposition of the project logic and a deep analysis of the code. Herein, the intricacies of the programming framework utilized in the implementation of the research objectives are meticulously elucidated, offering readers a detailed understanding of the technical underpinnings of the study. Chapter four serves as a detailed account of the tests and simulations conducted throughout the course of the research. Also the results obtained are analyzed and displayed in this chapter. Finally, in the fifth and concluding chapter, the culmination of the research endeavor is encapsulated. This chapter not only summarizes the goals achieved throughout the thesis but also reflects upon the journey undertaken and discusses potential avenues for future enhancements to further fortify the efficacy of the system under study. By offering critical reflections and insights, this final chapter adds depth and nuance to the overall discourse, thereby enriching the scholarly contribution of the thesis and an appendix with some part of the code.

Chapter 2

State of the Art

2.1 Analysis of Current Technologies

As the foundation for any project, a thorough examination of existing technologies is paramount to understanding the landscape and identifying potential avenues for innovation. In this endeavor, the initial step involved a meticulous review of pertinent literature, aimed to understanding the current state-of-the-art technologies directly relevant to the project's scope and objectives. The review process encompassed a comprehensive exploration of macro-areas pivotal to the project's domain, with a focus on delineating advancements and trends within each domain. Three primary macro-areas emerged as focal points of analysis, each offering unique insights and opportunities for integration within the project framework:

- Ego-Motion: This facet delves into the realm of motion estimation, particularly from the perspective of the ego vehicle viewpoint. Understanding the dynamics of ego-motion is crucial for tasks such as localization, navigation, and scene understanding within dynamic environments.
- Occupancy Grid: The concept of occupancy grid mapping lies at the intersection of robotics and perception, providing a systematic framework for modeling spatial environments based on sensor measurements. By discretizing space into a grid and modeling occupancy probabilities, occupancy grids facilitate robust environment representation and navigation in uncertain environments.
- Computer Vision: Central to many modern applications, computer vision encompasses a broad spectrum of techniques and methodologies aimed at extracting meaningful information from visual data. From object detection and tracking to image segmentation and scene understanding, computer vision techniques play a pivotal role in enabling intelligent perception and decision-making in autonomous systems.

Through an exhaustive examination of literature, each macro-area was scrutinized to glean insights into the latest advancements, emerging trends, and notable challenges. By synthesizing knowledge from diverse sources and disciplines, this comprehensive analysis serves as a cornerstone for informing subsequent phases of the project, guiding the selection of appropriate methodologies, algorithms, and technologies to achieve project objectives effectively.

2.1.1 Ego-Motion

Ego-motion refers to the process of estimating the movement of a camera relative to a static scene [1]. This capability is a critical component of autonomous navigation systems in driver-less cars. Accurate ego-motion estimation, along with visual simultaneous localization and mapping (SLAM), plays a pivotal role in the development of advanced artificial intelligence applications for modern vehicles [2]. One of the significant challenges in ego-motion estimation is the computational complexity involved, which poses limitations for implementation in embedded systems. To address this challenge, [1] propose a novel hardware architecture. This innovative approach leverages look-up tables and a feature matching algorithm, effectively reducing the algorithmic complexity without relying on iterative loops or complex geometrical constraints. Their experimental results demonstrate the practicality of a compact GPU-based implementation, which not only achieves higher accuracy but also increases processing speed compared to previous monocular visual odometry algorithms. All the documentations highlight the critical importance of ego-motion estimation in the realm of autonomous navigation and propose forward-thinking solutions to the associated challenges. While paper [1] focuses on optimizing hardware for use in embedded systems, paper [2] delves into the realm of unsupervised learning and visual data for depth estimation and ego-motion. This research underscores the potential of monocular cameras to enhance driverless car applications, showcasing significant advancements in the field. Overall, these studies make substantial contributions to the progress of autonomous navigation and robotics by presenting efficient and practical solutions for ego-motion estimation. Their innovative approaches not only enhance the accuracy and speed of these systems but also pave the way for more effective integration of such technologies in real-world applications.

2.1.2 Occupancy Grid

The occupancy grid map is a widely-used tool for representing the environments surrounding mobile robots and intelligent vehicles. Traditional occupancy grid mapping techniques must adapt to dynamic environments, requiring not only the detection of occupied areas but also the understanding of dynamic changes [3].

The basic concept is to create a map of the environment as an evenly spaced grid of variables that indicate whether a specific location is occupied by an obstacle. There are two main types of occupancy maps:

- Probability map: Uses probability values to provide a more detailed representation. Each cell in the occupancy grid contains a value indicating the likelihood that the cell is occupied.

- Binary map: Uses boolean values to create the map. A "true" value represents an occupied spot, while a "false" value represents a free space.

Most research in this field has primarily utilized sonar, radar, or LiDAR to represent the environment, as confirmed by nearly all the papers analyzed in this area. In contrast, the study presented in [3] offers a smart alternative using stereo vision. The papers reviewed span the domains of autonomous driving and robotics, addressing pivotal challenges critical to the safe and efficient navigation of complex environments. One of the primary focal points across these studies [4] lies in obstacle estimation and tracking [5]. Traditional methods often falter due to the dynamic and unpredictable nature of real-world driving scenarios. By introducing dynamic information through particle-based occupancy grid tracking and Bayesian occupancy filters, these papers [6] [7] offer novel solutions that transcend the limitations of conventional approaches. These methodologies enable autonomous systems to not only detect obstacles but also track them in real-time with enhanced accuracy and efficiency, fostering safer navigation through dynamic environments [8]. Furthermore, advancements in occupancy grid mapping techniques represent a significant leap forward in environmental perception for autonomous vehicles. Leveraging radar and LiDAR sensor data, these studies [9] [10] [11] [12] introduce sophisticated algorithms for estimating road courses, detecting parked vehicles, and generating detailed 3D object surface meshes. By harnessing the strengths of these sensor technologies, these methodologies empower autonomous systems to construct highly accurate representations of their surroundings, essential for informed decision-making and precise navigation. Moreover, integrating computer vision techniques with radar data heralds a new frontier in occupancy grid mapping, where machine learning algorithms are employed to learn inverse sensor models and perform semantic segmentation tasks [13]. These cutting-edge approaches [14] not only surpass the limitations of traditional filtering-based methods but also offer substantial improvements in mapping accuracy, paving the way for more reliable and robust autonomous navigation systems. In summation, the collective contributions of these papers represent a significant advancement in the fields of autonomous driving and robotics. By addressing critical challenges in obstacle estimation, object tracking, and occupancy grid mapping, these studies not only push the boundaries of technological innovation but also promise to revolutionize

how autonomous systems perceive and interact with their environment, ultimately fostering safer and more efficient transportation solutions for the future.

2.1.3 Computer Vision

Computer vision plays a pivotal role in autonomous driving, particularly in areas such as object detection, identification, depth estimation, and Simultaneous Localization and Mapping (SLAM). As autonomous vehicle (AV) technology rapidly advances, achieving accurate and efficient environmental perception is essential for safe and effective operation. Nearly all the analyzed papers emphasize the use of multiple sensors to gather environmental data, enabling the detection and classification of various objects. This underscores the importance of visual perception in autonomous driving and highlights the diverse methodologies and innovations aimed at enhancing object detection, identification, depth estimation, and SLAM in AV environments. In particular in the [15], the authors propose a novel object classification method tailored specifically for the fusion of vision and Light Detection and Ranging (LIDAR) data in AV environments. Leveraging Convolutional Neural Networks (CNNs) and image upsampling techniques, their approach integrates depth information from LIDAR point clouds with Red Green Blue (RGB) data to facilitate robust object classification across diverse driving scenarios. This fusion method combines the strengths of both RGB-D and fusion approaches, enabling effective long-distance object detection while overcoming limitations in depth perception and spatial density. Similarly, in [16] introduces an Object Detection mechanism that merges data from cameras and 3D LIDAR (OD-C3DL) for improved object detection in AVs. By employing CNNs, the proposed mechanism processes point clouds from LIDAR and images to recognize objects effectively, enhancing object classification accuracy and reducing extraction time. Notably, OD-C3DL demonstrates superior performance in identifying automobiles and pedestrians, addressing challenges such as sparse point clouds and the absence of depth information in images. In contrast, the [17] focuses on real-time vehicle detection using pure LiDAR point cloud data. Unlike previous methods, their approach employs a pre-Region of Interest (RoI) pooling convolution technique and pose-sensitive feature map design to achieve high accuracy in predicting vehicle location, orientation, and size. The proposed method significantly reduces processing time, making it suitable for real-time applications in AVs. Moving beyond object detection, the [18] addresses the challenging task of real-time vehicle type classification under varying conditions. The proposed method utilizes adaptive multi-class Principal Components Analysis (PCA) and self-clustering to classify vehicle types based on eigenvectors representing extracted vehicle fronts. By automatically extracting Regions of Interest (ROIs) and treating daytime and nighttime conditions separately, the system achieves promising performance in real-time classification. Finally, the [19] explores the

critical role of visual perception in AVs, covering object detection, identification, depth estimation, and SLAM methodologies. It categorizes vision sensors into monocular, stereo, and RGB-D cameras, discussing traditional methods and those based on deep learning. The integration of machine learning and deep learning enhances the capabilities of vision sensors for better detection results, contributing to the advancement of AV technology.

2.2 Stereo Camera

In our pursuit of gathering environmental data, we rely on the sophisticated capabilities of a stereo camera, specifically the ZED2 model manufactured by StereoLabs (depicted in Figure 2.1). A stereo camera, often referred to as a 3D camera, is a technological marvel equipped with multiple lenses meticulously positioned to replicate the interocular distance of human eyes. This unique configuration enables the camera to capture images from subtly divergent perspectives, akin to the mechanism of binocular vision observed in humans. The primary objective of employing a stereo camera revolves around the creation of depth perception within captured images or videos. This is accomplished by leveraging the inherent disparity between the perspectives provided by each lens. Through meticulous analysis of these disparities, stereo cameras can discern the relative distances to objects within the scene, thereby facilitating the generation of accurate depth maps and enabling a multitude of applications. Indeed, the depth information derived from stereo cameras finds application across a diverse array of domains. From the realms of 3D imaging and depth mapping to the realms of augmented reality, virtual reality, object tracking, and gesture recognition, the utility of this data knows no bounds. By harnessing the nuanced differences in perspectives captured by the stereo camera, these applications can create immersive and interactive experiences, blurring the lines between the digital and physical worlds. Furthermore, the ability of stereo cameras to accurately measure depth and spatial relationships within a scene has profound implications across various industries. In robotics, for instance, stereo cameras facilitate precise navigation and obstacle avoidance, enhancing the autonomy and efficiency of robotic systems. In medical imaging, stereo cameras aid in the creation of detailed anatomical models and assist in surgical planning. Similarly, in entertainment, stereo cameras contribute to the creation of captivating visual effects and immersive gaming experiences. In essence, the versatility and utility of stereo cameras make them indispensable tools in the fields of computer vision, robotics, medical imaging, entertainment, and beyond. By harnessing the power of binocular vision and sophisticated depth perception algorithms, stereo cameras empower us to explore and interact with our environment in unprecedented ways, paving the path for innovation and advancement across a multitude of disciplines.

Figure 2.1: ZED2

2.2.1 ZED 2

The ZED 2 stereo camera stands as a pinnacle of technological innovation, heralded as one of the most formidable devices in its class, particularly renowned for its unparalleled capabilities in depth perception and artificial intelligence integration (StereoLabs, [20]). This cutting-edge camera not only boasts an impressive field-of-view (FOV) but also delivers exceptional image quality, as evidenced by the specifications outlined in Table 2.1. What sets the ZED 2 apart from its counterparts is not just its hardware prowess, but also its meticulous design and engineering. Encased in a robust all-aluminum enclosure, the camera ensures durability and reliability in even the most demanding environments. Additionally, the incorporation of thermal control mechanisms within the enclosure serves to mitigate any potential biases arising from focal length variations or motion sensor irregularities (ZED2 DataSheet, [21]). The versatility of the ZED 2 extends far beyond its technical specifications, as it has been purposefully crafted to excel in a myriad of challenging applications. From facilitating autonomous navigation and mapping tasks to enhancing augmented reality experiences and enabling advanced 3D analytics, the ZED 2 emerges as a versatile tool capable of tackling diverse use cases with unparalleled precision and efficiency. In the realm of autonomous navigation, the ZED 2 empowers robots and unmanned vehicles with the ability to perceive and interpret their surroundings in three dimensions, facilitating safe and reliable navigation through complex environments. In augmented reality applications, the camera serves as a crucial component in overlaying virtual elements seamlessly onto the real world, blurring the lines between physical and digital realms. Moreover, in the domain of 3D analytics, the ZED 2 provides researchers and analysts with invaluable depth data, enabling detailed spatial analysis and modeling for a wide range of applications, from urban planning to industrial inspection. In essence, the ZED 2 represents a convergence of cutting-edge technology and practical design, poised to revolutionize industries ranging from robotics and computer vision to entertainment and beyond. With its exceptional depth perception capabilities, coupled with seamless integration of artificial intelligence, the ZED 2 sets a new standard for stereo cameras, paving the way for groundbreaking advancements in

various fields of science and technology.

Camera	
Output Resolution	2x(2208x1242)@15fps 2x(1920x1080)@30fps 2x(1280x720)@60fps 2x(672x376)@100fps
Field of View	Max. 110°(H) x 70°(V) x 120°(D)
Interface	USB 3.0/2.0 Integrated 1.2m cable
Depth Range	0.3 m to 20 m
Depth Accuracy	$< 1\%$ up to 3m $< 5\%$ up to 15m

Table 2.1: ZED 2 - Camera specifications

Sensors	
Motion	Gyroscope, Accelerometer, Magnetometer
Environmental	Barometer, Temperature

Table 2.2: ZED 2 - Sensors specifications

Beyond the fundamental attributes typically associated with stereo cameras, the Zed camera system offers a distinctive advantage in the form of its integrated suite of sensors (refer to Table 2.2). This comprehensive array of sensors encompasses a variety of functionalities, providing an additional layer of versatility and capability to the camera that can prove invaluable in specific applications. The inclusion of multiple sensors within the Zed camera package enhances its utility and effectiveness across a wide range of scenarios. By amalgamating various sensing modalities, such as inertial sensors, ambient light sensors, and temperature sensors, the Zed camera transcends the limitations of traditional stereo cameras, offering a holistic approach to environmental perception and data acquisition. In practical terms, the integration of these sensors empowers the Zed camera with enhanced situational awareness and adaptability. For instance, inertial sensors can provide valuable information about the camera's orientation and movement, facilitating more precise navigation and motion tracking in dynamic environments. Ambient light sensors, on the other hand, enable the camera to adjust its exposure settings autonomously, ensuring optimal image quality across diverse lighting conditions.

Physical	
Dimensions	174.9 x 29.9 x 31.9 mm
Weight	164g
Op. Temp.	-10°C to +45°C
Power	380 mA / 5V USB Powered

Table 2.3: ZED 2 - Physical specifications

Moreover, temperature sensors play a crucial role in thermal management, helping to regulate the camera's internal temperature and prevent overheating, thereby ensuring consistent performance and longevity. This comprehensive sensor package not only enhances the camera's performance but also extends its operational capabilities to a broader spectrum of applications. In certain use cases, such as robotics, autonomous vehicles, and industrial automation, the integration of diverse sensors within the Zed camera system can provide a distinct competitive advantage. By leveraging the rich data provided by these sensors, users can gain deeper insights into their environment, make more informed decisions, and achieve greater efficiency and precision in their operations. Furthermore, the modular nature of the Zed camera system allows for easy integration and customization, enabling users to tailor the camera to their specific needs and preferences. Whether deployed in research laboratories, manufacturing facilities, or outdoor environments, the Zed camera's integrated sensor package equips users with the tools they need to tackle complex challenges and push the boundaries of innovation. In summary, the integration of a comprehensive suite of sensors within the Zed camera system enhances its versatility, performance, and adaptability, offering users a distinct advantage in a wide range of applications. By harnessing the collective capabilities of these sensors, the Zed camera transcends the limitations of traditional stereo cameras, paving the way for new opportunities and advancements in the field of computer vision and beyond.

2.2.2 Stereoscopic Vision

Much like the intricate workings of the human brain in deciphering depth perception, the utilization of stereoscopic vision techniques offers a sophisticated means to ascertain the spatial distances between a camera and objects within its field of view (StereoCamera, [22]). This method harnesses the principles of binocular vision, mirroring the visual mechanism employed by humans to perceive depth and spatial relationships in the surrounding environment. At the core of stereoscopic vision lies the configuration of two lenses, each capturing a slightly different perspective of the scene. These lenses operate in tandem to create distinct planes of vision that are

co-planar to each other. Crucially, the projection points derived from these lenses align along the same level on the y-axis, as illustrated in Figure 2.2. In this visual representation, the points p_1 and p_2 serve as exemplary projections captured by the respective lenses. These points, despite originating from different viewpoints, are situated on a shared plane, reflecting the consistency in spatial alignment facilitated by stereoscopic vision techniques. The synchronized alignment of projection points on co-planar planes is instrumental in enabling accurate depth perception through stereoscopic vision. By analyzing the disparities between corresponding points captured by each lens, the system can infer the relative distances to objects within the scene. This process is akin to the human brain's interpretation of binocular cues, where variations in the perspectives of the left and right eyes provide vital depth cues for spatial perception. Furthermore, the depth information derived from stereoscopic vision techniques finds extensive application across a myriad of fields, ranging from robotics and autonomous navigation to virtual reality and 3D imaging. By leveraging the inherent capabilities of stereoscopic vision, these applications can achieve heightened precision and fidelity in spatial mapping, object recognition, and scene reconstruction. In essence, stereoscopic vision serves as a powerful tool in unlocking the mysteries of depth perception, mirroring the sophisticated mechanisms employed by the human visual system. Through meticulous analysis of co-planar projection points and disparities between viewpoints, stereoscopic vision techniques empower cameras to navigate and interact with the world with unprecedented depth and clarity, ushering in a new era of innovation and discovery in the realm of computer vision and beyond.

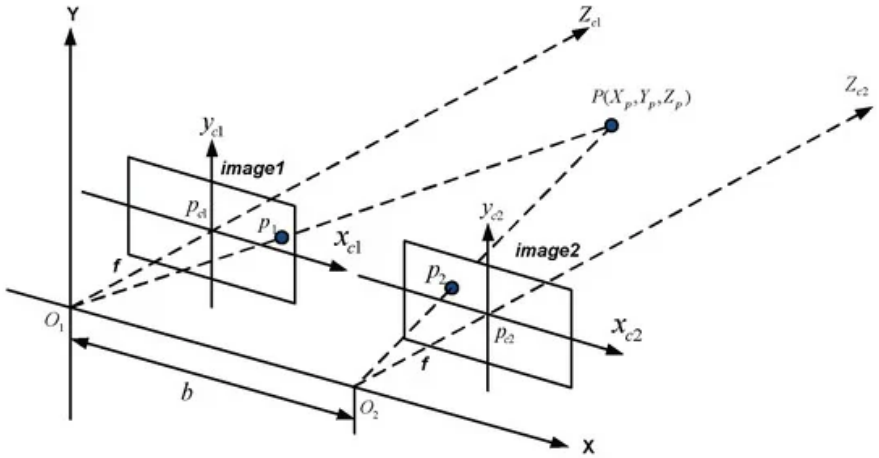


Figure 2.2: Geometry of Stereoscopic Vision

2.2.3 Image Rectification

Image rectification is a process that modifies projected images to align them on a shared plane. This technique is utilized for row-alignment, leveraging epipolar geometry to ensure that corresponding projection points align at the same pixel level. As depicted in figure 2.3, epipolar geometry can be represented by two rectangles that symbolize the image planes of the left and right cameras in a three-dimensional space. Since the image planes in reality are not perfectly parallel and coincident, they require adjustment. This adjustment is performed using epipolar geometry. In figure 2.3, X denotes the points captured by both cameras, and O_L and O_R are the optical centers. X is projected onto the left image plane as point X_L and onto the right image plane as point X_R. By connecting points O_R and O_L, a line known as the baseline is formed. The epipoles, e_l and e_r, are the intersection points of the baseline and the two image planes. An epipolar line is created by e_R and x_R, and another epipolar line is created from e_L and X_L. The match for each pixel in another image can only be located on the epipolar line. By making both epipolar lines parallel, X_R and X_L will be positioned on the same pixel row, reducing error propagation.

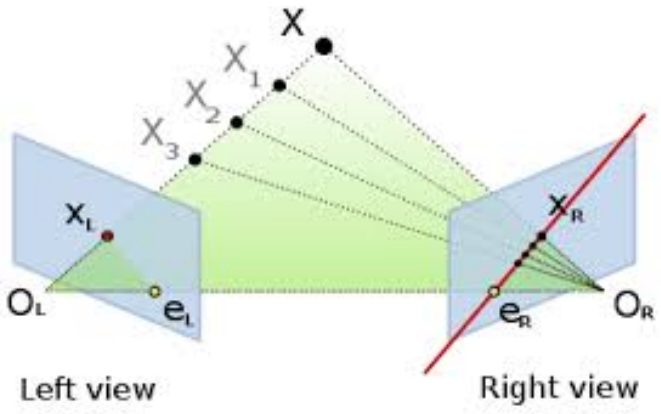


Figure 2.3: Epipolar Geometry

2.2.4 Disparity Map

As depicted in Figure 2.4, the phenomenon of stereoscopic vision unfolds through the intricate interplay of geometric principles and optical phenomena. At the heart of this process lies the projection of a tangible object, represented by point P, onto the image planes of two distinct cameras. However, due to the separation between the cameras, denoted as T and often referred to as the baseline, the projections

of point P onto these camera planes exhibit a perceptible shift. In the context of stereoscopic vision, this shift manifests as the displacement of projection points, symbolized as P_l and P_r, corresponding to the image of point P as captured by the left and right cameras, respectively. The magnitude of this displacement is directly influenced by several key factors: Z, representing the distance from the object to the camera; T, signifying the baseline or distance between the two cameras; and f, which denotes the focal length of the camera. The intricate dance between these parameters dictates the observed shift in projection points and forms the basis for determining the depth information inherent in stereoscopic imaging. As the distance Z between the object and the cameras varies, so too does the magnitude of the displacement, leading to nuanced variations in the perceived depth within the captured scene. Moreover, the separation between the cameras, represented by the baseline T, plays a pivotal role in shaping the disparity between projection points. A wider baseline typically results in a more pronounced shift in projection, enhancing the depth perception capabilities of the stereoscopic imaging system. Similarly, the focal length f of the camera influences the scale and distortion of the captured image, further contributing to the overall disparity observed between projection points. By carefully calibrating these parameters, practitioners can fine-tune the stereoscopic imaging system to achieve optimal depth perception and spatial accuracy. The comprehensive understanding of these geometric relationships and optical phenomena is paramount in leveraging the full potential of stereoscopic vision for various applications, including depth mapping, object recognition, and 3D reconstruction. By dissecting the intricate interplay of factors such as baseline separation, focal length, and object distance, researchers and engineers can unlock new avenues for innovation and discovery in the realm of computer vision and beyond.

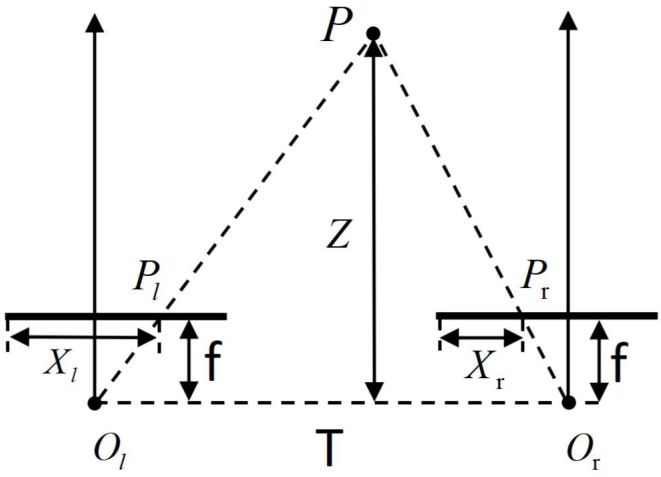


Figure 2.4: Disparity Scheme

Stereoscopic vision, a marvel of human perception, mimics the depth perception experienced by our eyes. Imagine shutting one eye: an object occupies a certain spatial position. Now, swap eyes, and suddenly, the object seems to shift to a different location. This apparent motion of objects between two stereo images is termed "disparity" (d). In essence, it quantifies the variation in distance of corresponding points between these paired images. Illustrated in Figure 2.4, this concept is depicted with P_l and P_r representing corresponding points in the left and right images respectively, each with its respective distances, X_l and X_r.

$$d = X_l - X_r \tag{2.1}$$

Utilizing this disparity information, we can derive the distance Z through the equation:

$$Z = \frac{T * f}{d} \tag{2.2}$$

Here, T denotes the baseline distance between the two cameras, and f represents the focal length. Notably, this equation highlights the inverse relationship between distance and disparity; when an observed point is nearer to the image planes, the disparity value tends to be greater, and vice versa. To transform these mathematical concepts into visual representations, techniques like semi-global block matching come into play. This method operates by scanning through pixels in one image, identifying their corresponding counterparts in the other, and computing the disparity between them. The resulting disparity values, typically ranging from 0 to 255, are then translated into a grayscale image, known as a "disparity map." In this map, darker regions represent points closer to the camera, appearing as black, while brighter areas denote objects farther away, resembling white. Thus, by iteratively applying this process to both camera images, a unified grayscale image is generated, vividly portraying the relative distances of objects in the scene.

2.3 Neural Network

2.3.1 Object Detection

Object detection is one of the main task in computer vision. It deals with identify and localizing different object in an image or video. One single image can include several region of interest (ROI) pointing to different objects. Objects detection algorithms can be divided into two main categories: single-shot detectors and two-stage detectors [23]. One of the earliest successful attempts to address the object detection problem using deep learning was the R-CNN (Regions with CNN features) model. It used a combination of region proposal algorithms and a convolutional

neural networks (CNNs) to detect and localize objects in images. Object detection algorithms are broadly classified into two categories based on how many times the same input image is passed through a network [23]. A single-shot object detection, such as YOLO, uses a single pass of the input image to detect the presence and the location of the objects in the image. Processing an entire image in a single shot, make the process very computationally efficient but generally a bit less accurate than the other method, especially is less effective to detect small objects. Two-shot object detection instead uses two passes of the input image to understand presence and location of objects. The first step is used to create a set of proposal or potential object location, and then with the second one is use to confirm or not the proposal and make the final prediction. This approach is more accurate but computationally expensive. Usually for real-time application the single-shot one is more used.

CNN

Convolutional Neural Networks (CNNs) represent a pivotal breakthrough in the realm of deep learning, revolutionizing tasks ranging from image analysis to natural language processing and beyond. At the core of their functionality lies a sophisticated architecture comprising distinct layers designed to extract, process, and interpret intricate patterns within complex data, particularly in image classification tasks. The CNN architecture, as illustrated in Figure 2.5, consists of several interconnected layers, each serving a specific purpose in the hierarchical processing of input data. At the forefront lies the input layer, where raw data is ingested and processed. In the context of image analysis, the input layer typically represents an image encoded as a matrix of pixel values, forming the foundational data structure upon which subsequent layers operate [24]. Moving beyond the input layer, we encounter the hidden layers, which constitute the crux of the CNN architecture. These hidden layers are composed of several key components, each contributing to the network's ability to discern meaningful features from the input data [24]. First and foremost among these components is the Convolutional Layer, which plays a pivotal role in feature extraction. At its essence, the convolutional layer employs filters or kernels, small matrices typically sized 3x3 or 5x5, which traverse the input image, systematically scanning for distinctive features such as edges, textures, and patterns [24]. This process, known as convolution, yields feature maps or activation maps, which highlight the presence of specific features detected by the filters. Accompanying the convolutional layer is the concept of stride, which dictates the step size at which the filter traverses the input image. Strides can vary, with values of 1 indicating movement one pixel at a time, while larger values skip pixels, influencing the granularity of feature detection [24]. Moreover, padding, the addition of extra pixels around the input image, serves to preserve spatial dimensions post-convolution, with 'valid' and 'same' being common padding types

[24]. An integral element linked with the convolutional layer is the activation function, typically ReLU (Rectified Linear Unit), applied element-wise to introduce non-linearity into the network. By replacing negative pixel values with zero, ReLU enables the network to learn complex patterns and relationships within the data, enhancing its capacity for feature discrimination [24]. Pooling layers represent another fundamental component of CNN architecture, tasked with reducing the spatial dimensions of feature maps. Through techniques such as max pooling and average pooling, pooling layers extract dominant features while mitigating computational burden, thereby enhancing the network's efficiency and robustness to spatial variations in the input data [24]. Lastly, the fully connected (dense) layer integrates the extracted features from preceding layers, serving as the nexus for final classification or regression tasks. Neurons in this layer are connected to all activations in the preceding layer, with the 2D feature maps flattened into a 1D vector before being fed into the fully connected layer [24]. Finally, the output layer, tailored to the specific task at hand, employs appropriate activation functions such as Softmax for multi-class classification or Sigmoid for binary classification [24]. In summary, Convolutional Neural Networks epitomize the pinnacle of deep learning architectures, offering unparalleled capabilities in extracting and interpreting intricate patterns within complex datasets. Through their hierarchical structure and interconnected layers, CNNs have revolutionized a myriad of domains, propelling the boundaries of artificial intelligence and paving the way for unprecedented advancements in machine perception and cognition.

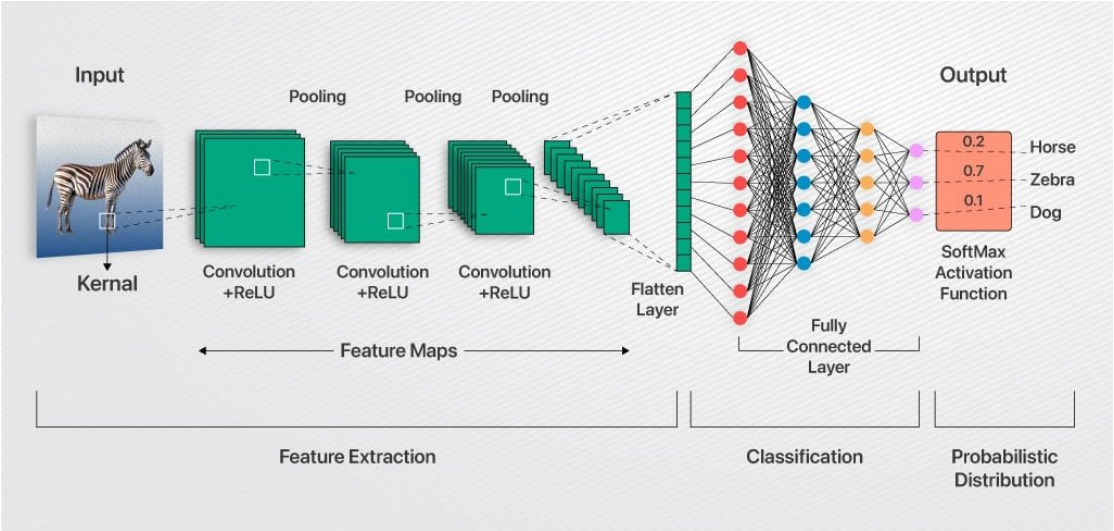


Figure 2.5: CNN Architecture [25]

Another crucial aspect concerning the training of a neural network is addressing the issues of underfitting and overfitting, which correspond to high bias and high

variance, respectively. Understanding and managing these problems is essential for the development of an effective neural network model [26]. When a neural network suffers from high bias, also known as underfitting, it indicates that the model is too simplistic to capture the underlying patterns in the data. This usually occurs when the model has not been trained for a sufficient number of epochs or when it lacks complexity in its architecture. To mitigate underfitting, it is necessary to extend the training duration and enhance the model's capacity by increasing the number of hidden layers and neurons. This allows the network to learn more intricate representations of the input data, thereby improving its performance [26]. Conversely, overfitting occurs when a neural network exhibits high variance. This means that while the model performs exceptionally well on the training data, it fails to generalize to new, unseen data. Overfitting typically arises when the model becomes overly complex, capturing noise and random fluctuations in the training data rather than the actual underlying patterns. To combat overfitting, several strategies can be implemented. Regularization techniques, such as L1 and L2 regularization, introduce a penalty to the loss function based on the magnitude of the model parameters. This discourages the network from becoming excessively complex by constraining the size of the parameters. Another effective regularization technique is dropout. In dropout, the network randomly deactivates a fraction of its nodes during training, as determined by a specified probability parameter. This prevents the network from becoming too reliant on any particular set of nodes, promoting more robust learning [26]. Furthermore, data augmentation methods can be employed to artificially enhance the diversity of the training dataset. By applying random transformations such as rotations, translations, flips, random cropping, and color shifting, the model is exposed to a wider variety of data scenarios. This encourages the network to learn more generalized and robust features, ultimately improving its ability to perform well on new, unseen data [26]. By carefully balancing the complexity of the neural network and employing appropriate regularization and data augmentation techniques, it is possible to reduce both underfitting and overfitting, leading to a model that performs well on both training and test datasets.

YOLO

YOLO (You Only Look Once) revolutionizes object detection by employing an end-to-end neural network architecture to predict objects within a scene simultaneously, marking a departure from conventional methods. This approach has yielded extraordinary results, positioning YOLO as a frontrunner in real-time object detection algorithms, surpassing its counterparts by a significant margin [23]. In contrast to traditional algorithms that rely on region proposal networks to detect regions of interest (ROI) before conducting recognition on these regions separately,

a process that often involves multiple iterations, YOLO streamlines the entire procedure into a single fully connected layer, thereby condensing the process into just one iteration. This unified approach not only simplifies the computational workflow but also enhances efficiency by reducing redundant computations, ultimately leading to faster and more accurate object detection. Since its inception in 2015, YOLO has continuously evolved, undergoing several enhancements with subsequent versions, as documented in the literature [23]. These iterations have introduced refinements and optimizations, further improving the algorithm's performance, robustness, and versatility. Through ongoing research and development efforts, YOLO continues to push the boundaries of object detection capabilities, solidifying its status as a cornerstone in the field of computer vision.

Each iteration of YOLO is driven by the overarching objective of enhancing the algorithm's performance in terms of accuracy detection and speed compared to its predecessors. The evolution of YOLO is characterized by a continuous quest to refine and optimize its capabilities, aiming to achieve superior results with each upgraded version. A comprehensive examination of the improvements introduced in successive iterations is provided in Table 2.4, where the key enhancements from the previous version are outlined. With each new release, YOLO undergoes a series of modifications and enhancements targeted at addressing specific shortcomings and capitalizing on emerging technological advancements. These improvements encompass a wide range of aspects, including feature extraction, model architecture, training methodology, and inference optimization. By leveraging insights gained from empirical evaluations and feedback from the research community, the developers of YOLO strive to push the boundaries of what is achievable in real-time object detection. The iterative refinement process inherent to YOLO's development cycle underscores its commitment to continuous improvement and innovation. Through systematic experimentation and rigorous evaluation, each version of YOLO aims to deliver tangible enhancements in terms of both performance metrics and practical utility. By meticulously documenting the evolution of YOLO and highlighting the key improvements introduced in each iteration, researchers and practitioners alike can gain valuable insights into the trajectory of advancements within the realm of object detection algorithms.

Figure 2.6 presents a comprehensive performance comparison between two notable iterations of the YOLO object detection framework: YOLOv8 and YOLOv5. This visual representation serves as a testament to the significant advancements made within the span of a year, underscoring the relentless pursuit of excellence in the field of computer vision. The comparison between YOLOv8 and YOLOv5 provides valuable insights into the evolution of the YOLO architecture over time, illustrating the tangible improvements achieved in terms of detection accuracy, computational efficiency, and overall effectiveness. By comparing the performance metrics of these

Version	Improvements
YOLO v2	- Different CNN backbone (Darknet-19) - Use of Anchor Boxes (to handle a wide range of object size) - Batch Normalization (to improve accuracy and stability) - Multi-scale training strategy (to improve small object detection) - New Loss Function
YOLO v3	- New CNN architecture (Darknet-53) - Anchor boxes with different scales and aspect ratios - Feature Pyramid Network (to improve multiple scale and small object detection)
YOLO v4	- New CNN architecture (CSPNet) - k-means clustering (anchor boxes more closely aligned) - GHM loss (to improve the model's performance on imbalanced datasets)
YOLO v5	- More complex architecture (EfficientDet) - Different training data (D5) with 600 object categories - Dynamic anchor boxes (new generating method) - Spatial pyramid pooling (to improve small object detection performance) - Introduction of CIoU loss (variant of IoU loss)
YOLO v6	- New CNN architecture (EfficientNet-L2) - Dense anchor boxes (new generating method)
YOLO v7	- Nine anchor boxes (to reduce the number of false positives) - Focal loss (To better detect challenging objects) - Higher resolution - Higher speed
YOLO v8	- Addition of the CSPDarknet53 backbone - Use of PANet as the neck network

Table 2.4: YOLO Verisions Imporvments

two versions, we gain a deeper understanding of the progress made and the challenges overcome in the pursuit of state-of-the-art object detection capabilities. The observed advancements in YOLOv8 compared to its predecessor, YOLOv5, signify a

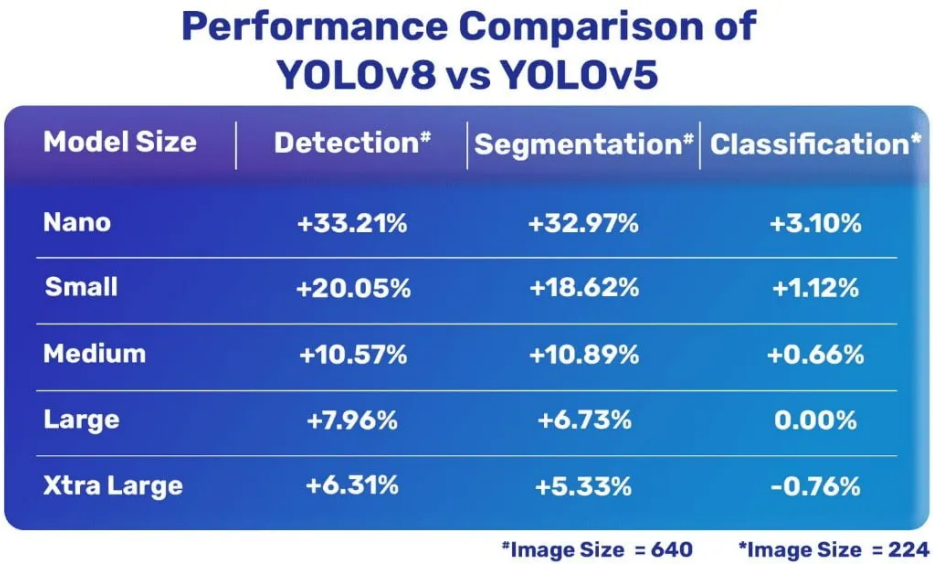
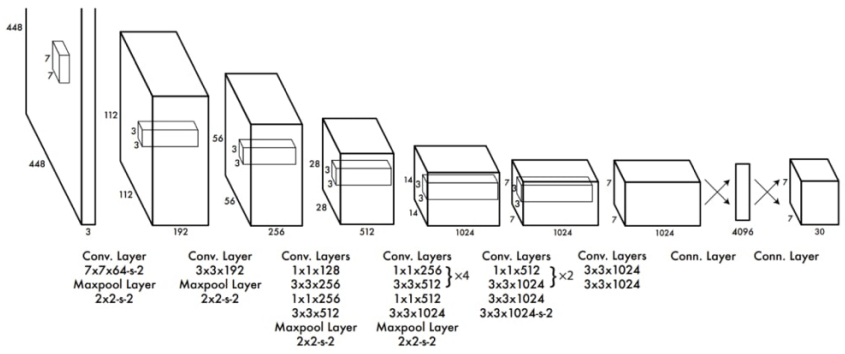


Figure 2.6: Perfromance Comparison of YOLOv8 vs YOLOv5. Source:[27]

culmination of extensive research, experimentation, and innovation. These improvements may encompass a multitude of factors, including algorithmic optimizations, architectural refinements, dataset enhancements, and training methodologies. Moreover, the performance gap highlighted in the comparison underscores the iterative nature of algorithm development, wherein each iteration builds upon the successes and learns from the limitations of its predecessors. By analyzing the performance gap between YOLOv8 and YOLOv5, researchers and practitioners gain valuable insights into the trajectory of advancements within the YOLO framework and the broader landscape of object detection algorithms. This comparative analysis serves as a guiding beacon for future research endeavors, providing a roadmap for further enhancements and breakthroughs in the realm of computer vision and artificial intelligence.

The YOLO algorithm represents a seminal advancement in the domain of object detection, characterized by its streamlined approach and remarkable efficiency. At its core, YOLO begins its journey by ingesting an image as input, which then undergoes meticulous scrutiny through a deep convolutional neural network (CNN). This CNN architecture, serving as the backbone of YOLO, forms the bedrock upon which object identification within the image is built, as vividly depicted in Figure 2.7. The architectural intricacies of YOLO are finely tuned to optimize performance and accuracy. The initial stages of the model's convolution layers undergo pre-training utilizing the vast image repository of ImageNet. This pre-training phase is pivotal, incorporating temporary average pooling and fully connected layers to imbue the model with a foundational understanding of image



The Architecture. Our detection network has 24 convolutional layers followed by 2 fully connected layers. Alternating 1×1 convolutional layers reduce the features space from preceding layers. We pretrain the convolutional layers on the ImageNet classification task at half the resolution (224×224 input image) and then double the resolution for detection.

Figure 2.7: YOLO Architecture. Source:[23]

features and semantics. Subsequently, this pre-trained model is seamlessly adapted for detection purposes, as empirical evidence from prior studies has showcased significant performance enhancements through the addition of convolutional and connected layers to pre-existing networks. In the heart of YOLO lies a distinctive approach to object localization and classification. The input image is meticulously partitioned into an S $\times$ S grid, where each grid cell assumes the responsibility of detecting objects within its domain. Within each grid cell, YOLO predicts multiple bounding boxes alongside confidence scores, which serve as indicators of the model's certainty regarding the presence of an object and the accuracy of the predicted bounding box. This multi-faceted prediction mechanism not only facilitates precise object localization but also enhances the overall robustness of the algorithm. Central to YOLO's efficacy is its ingenious strategy for bounding box assignment during training. While multiple bounding boxes may be predicted per grid cell, only one bounding box predictor is designated to be responsible for each object. This assignment of responsibility is governed by a dynamic process based on the predictor's current Intersection over Union (IOU) with the ground truth, fostering specialization among predictors and culminating in superior overall recall scores. An indispensable component of the YOLO pipeline is the application of non-maximum suppression (NMS), a post-processing technique designed to refine object detection results. Given the propensity for multiple bounding boxes to be generated for a single object, NMS plays a pivotal role in eliminating redundancy and enhancing the accuracy and efficiency of object detection. Through the judicious removal of redundant or erroneous bounding boxes, NMS ensures that only the most relevant and reliable bounding boxes are retained, thus producing a refined output with enhanced precision and clarity. In summary, the YOLO algorithm stands

as a testament to the ingenuity and innovation driving advancements in object detection methodologies. From its inception, YOLO has epitomized a paradigm shift in the field, offering a holistic solution characterized by efficiency, accuracy, and versatility. Through its intricate fusion of convolutional neural networks, bounding box prediction mechanisms, and post-processing techniques like non-maximum suppression, YOLO continues to push the boundaries of what is achievable in real-time object detection, reshaping the landscape of computer vision with its unparalleled efficacy and sophistication.

R-CNN

A Regional Proposal Convolutional Neural Network (R-CNN) is a traditional type of two-stage detector. The core concept involves the network generating region proposals from an input image—specifically, 2000 candidate regions where objects might be located. These proposals are then analyzed by a CNN to determine the presence of objects within them. To generate a manageable number of region proposals, a selective search algorithm is employed, which primarily consists of three steps [28]:

- 1. Generate initial sub-segmentation , we generate many candidate regions.
- 2. Use greedy algorithm to recursively combine similar regions into larger ones.
- 3. Use the generated regions to produce the final candidate region proposal.

Next, a CNN functions as a feature extractor, identifying features used to classify the presence of an object within each candidate region proposal. Finally, the algorithm refines the bounding box to improve its precision.

Fast R-CNN

Fast R-CNN is an improved version of the original R-CNN, designed to address some of its limitations. The basic concept remains similar, but instead of feeding region proposals into the CNN, the input image is processed by the CNN to produce a convolutional feature map [28]. From this map, region proposals are identified and warped into squares. These squares are then reshaped to a fixed size using a Region of Interest (RoI) pooling layer, allowing them to be fed into a fully connected layer. Using the RoI feature vector, a softmax layer predicts the class of each proposed region and its bounding box [28]. This approach is faster than the original R-CNN because it eliminates the need to process 2000 region proposals with the CNN for each image. Instead, the convolution operation is performed only once per image, generating a feature map that is used for all region proposals.

Faster R-CNN

Both R-CNN and Fast R-CNN use selective search to determine region proposals. This method is slow and time-consuming, reducing the network's speed and efficiency. To address this issue, Faster R-CNN was developed. Similar to Fast R-CNN, the input image is fed into a convolutional network to produce a convolutional feature map. However, instead of using a selective search algorithm on the feature map, a separate network generates the region proposals. These predicted regions are then reshaped using the RoI pooling layer, which is used to classify the image within the proposed regions and predict the bounding box offsets. This method is significantly faster than the previous two and can be used for real-time object detection, similar to YOLO.

2.3.2 TensorRT

In the expansive domain of deep learning model optimization, TensorRT emerges as a formidable force, harnessing the computational might of NVIDIA GPUs to elevate neural network performance to unprecedented levels. As depicted in Figure 2.8, this optimizer delves into a myriad of crucial aspects, profoundly enhancing the efficacy of trained neural networks and accelerating their integration across a diverse array of applications. At its core, TensorRT embarks on the journey of Mixed Precision Reduction, a sophisticated technique meticulously crafted to augment throughput while meticulously preserving the intrinsic accuracy of models. Through the meticulous quantization of models to INT8 precision, TensorRT adeptly navigates the delicate balance between computational efficiency and fidelity to the original model, laying the foundation for enhanced performance. Furthermore, TensorRT intricately weaves in Layer and Tensor Fusion mechanisms, strategically consolidating nodes within kernels to optimize the utilization of GPU memory and bandwidth. This intricate fusion process not only streamlines the computational pipeline but also serves as a potent remedy for potential bottlenecks, thereby fortifying overall performance with remarkable efficiency. A standout feature of TensorRT lies in its implementation of Kernel Auto-Tuning, an ingenious process that dynamically selects optimal data layers and algorithms tailored to the nuanced intricacies of the underlying GPU platform. This adaptive methodology ensures that neural networks operate at peak efficiency across a spectrum of hardware configurations, unlocking their full potential with unparalleled precision. Moreover, TensorRT integrates Dynamic Tensor Memory management, an astute strategy aimed at minimizing memory footprint while facilitating the judicious reuse of memory for tensors. This dynamic allocation scheme not only conserves precious GPU memory but also optimally distributes memory resources throughout the course of inference tasks, optimizing performance with unwavering efficacy. Additionally, TensorRT showcases its prowess through Multi-Stream Execution capabilities, employing a

scalable design to concurrently process multiple input streams in parallel. This parallelized approach significantly enhances throughput and responsiveness, particularly in scenarios necessitating real-time inference on data streams, thereby ensuring seamless operation even in the most demanding environments. Lastly, TensorRT distinguishes itself in its proficiency in optimizing recurrent neural networks (RNNs) over temporal sequences, leveraging dynamically generated kernels to adaptively refine network performance across successive time steps. This temporal optimization framework ensures that RNNs exhibit optimal performance across a diverse array of temporal contexts, accentuating their suitability for tasks such as time-series analysis and sequential data processing. In summation, TensorRT epitomizes a comprehensive suite of optimization techniques, meticulously tailored to leverage the computational prowess of NVIDIA GPUs and propel deep neural network performance to unprecedented heights. Through its adept fusion of precision reduction, memory optimization, and dynamic kernel generation, TensorRT stands as an indispensable cornerstone in the realm of deep learning model deployment and inference optimization, shaping the future landscape of AI with unparalleled innovation and efficiency.

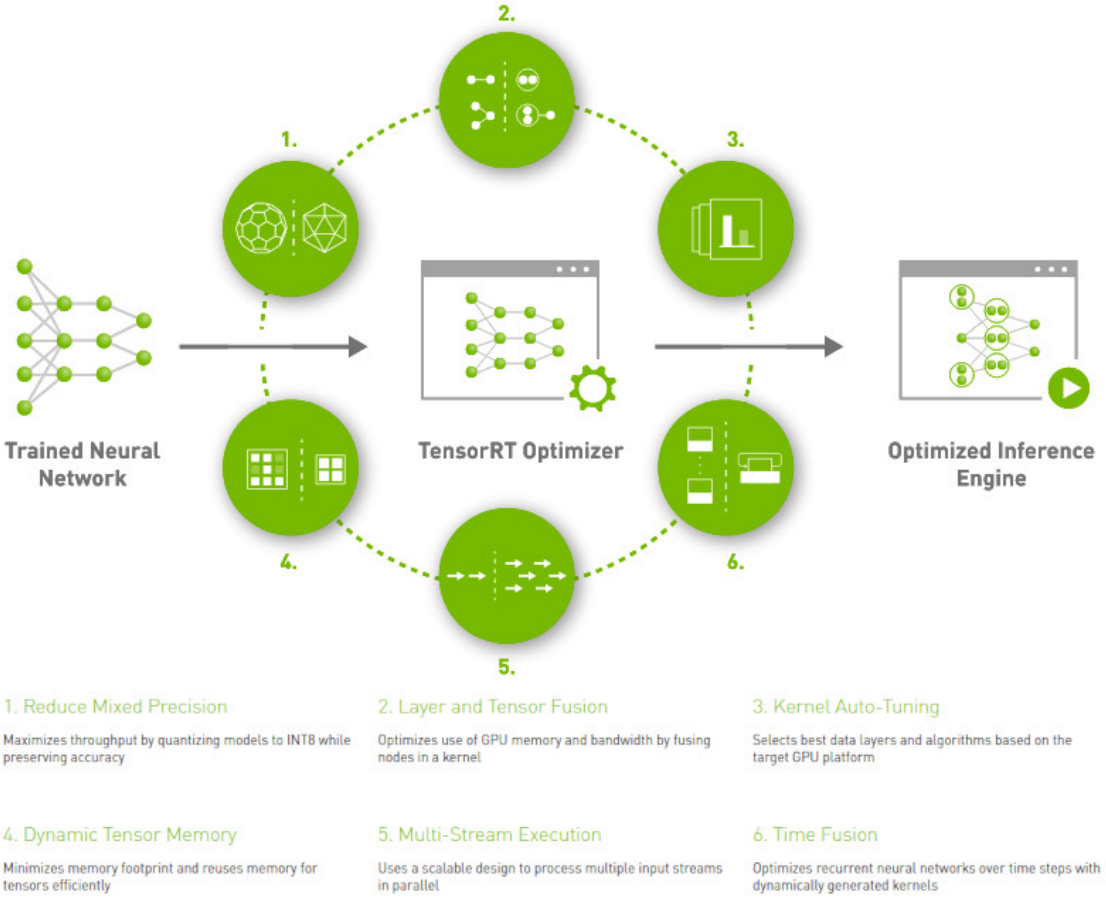


Figure 2.8: TensorRT [29]

Chapter 3

Project

3.1 Object Detection

The ZED software development kit (SDK) leverages the sophisticated functionalities of the 3D Object Detection API to perform highly accurate object detection tasks. By utilizing the advanced capabilities of artificial intelligence and neural networks, the SDK is able to adeptly identify a wide variety of objects depicted within images. This state-of-the-art technology is renowned for its exceptional precision, ensuring robust and reliable detection outcomes [29]. In the context of our specific application, the extensive range of object classes and their corresponding sub-classes detected by the API aligns seamlessly with our requirements. This alignment ensures that the system can effectively recognize and categorize a diverse array of objects, thereby meeting the stringent demands of our project with excellence. The versatility of the SDK in identifying both common and specialized objects underscores its applicability across various domains. By examining the contents of Table 3.1, we gain a comprehensive understanding of the object classes and their associated sub-classes. This table serves as an invaluable resource, providing detailed insights into the extensive range of objects that the SDK can accurately detect. From everyday items to specialized entities, the API's capabilities cover a broad spectrum, confirming its suitability for a wide variety of applications. The detailed categorization in Table 3.1 not only highlights the depth of the SDK's object detection capabilities but also demonstrates its potential to adapt to diverse use cases. Whether for general-purpose applications or highly specialized tasks, the SDK's robust framework and advanced AI algorithms ensure that it can deliver precise and reliable detection outcomes consistently. This level of detail and accuracy is crucial for the success of our project, as it guarantees that the system can handle the complexity and variability of real-world environments with confidence and efficacy [29].

Object class	**Object Sub-classes**
Person	Person
	Person head
Vehicle	Bicycle
	Car
	Motorbike
	Bus
	Truck
	Boat
Bag	Backpack
	Handbag
	Suitcase
Animal	Bird
	Cat
	Dog
	Horse
	Sheep
	Cow
Electronics	Cellphone
	Laptop
Fruit and Vegetable	Banana
	Apple
	Orange
	Carrot
Sport	Ball

Table 3.1: Object Detection Class and Sub-classes [29]

In addition to its primary function of object detection and classification, the object detection API offers a rich array of supplementary outputs, as we can see from table 3.2, each providing valuable insights into the characteristics and behavior of detected objects. Delving into these outputs enhances our understanding and utilization of the API's capabilities, enriching the analysis and application of detected objects. Firstly, the API furnishes each detected object with a unique identifier, facilitating tracking and continuity of analysis across frames or time intervals. This identification number serves as a crucial reference point for monitoring the object's trajectory and behavior over time. Furthermore, the API's

Object Data	Output
ID	Integer
Label	Class
Tracking state	Ok, Off, Searching, Terminate
Action State	Idle, Moving
Position	[x, y, z]
Velocity	$[v_x, v_y, v_z]$
Dimensions	[width, height, length]
Detection confidence	0 - 100
2D bounding box	Four pixel coordinates
3D bounding box	Eight 3D coordinates
Mask	Binary mask

Table 3.2: Object Data Output[29]

output includes detailed labeling for each detected object, specifying its type and subclass. This categorization enables precise classification, aiding in subsequent analysis and decision-making processes. Moreover, the object detection API provides information regarding the tracking state of each object. With four distinct states - 'Ok', 'Off', 'Searching', and 'Terminate' - this attribute offers insights into the current status of object tracking, enhancing situational awareness and facilitating adaptive response strategies. Additionally, the API offers insights into the dynamic behavior of detected objects through the 'Action State' parameter. This attribute delineates whether the object is stationary or in motion, providing valuable contextual information for further analysis. The API's output also includes spatial information such as the 3D position of the object's central point, represented as a vector in the Cartesian coordinate system. This positional data enables precise localization and spatial awareness, facilitating augmented reality applications and spatial analytics. Furthermore, the API furnishes velocity vectors for each detected object, capturing its movement dynamics in three-dimensional space. This velocity information enriches our understanding of object behavior and facilitates predictive modeling and motion analysis. Moreover, the API provides dimensional attributes for each object, including width, height, and length, enabling accurate spatial characterization and sizing of detected objects. Additionally, the API quantifies the confidence level of each detection with a value ranging from 0 to 100, reflecting the precision of localization and classification. This confidence score serves as a crucial metric for assessing the reliability of detection and guiding subsequent analysis and decision-making processes. Moreover, the API delineates the spatial extent of each

detected object through both 2D and 3D bounding boxes. These bounding boxes enclose the object within the image and spatial domains, respectively, providing a visual representation of its spatial footprint. Finally, the API offers pixel-level segmentation masks for detected objects, distinguishing between object pixels and background pixels. This mask facilitates precise delineation and extraction of object regions, enabling advanced image processing and analysis tasks. In conclusion, while the object detection API offers a broad range of output parameters, not all may meet the specific requirements of certain applications. Nonetheless, it serves as a robust foundation from which to build and tailor solutions to suit individual needs.

3.2 Distance and Position estimation

The stereo camera, designed to emulate the human binocular vision system, mirrors the natural separation between human eyes, which averages around 65 mm. This separation allows each eye to perceive a slightly different view of the surrounding environment. By comparing these distinct perspectives, the brain is able to gauge depth and understand the three-dimensional motion space. The ZED 2 stereo camera leverages this principle by having its viewpoints separated by 120 mm, thereby enhancing its depth perception capabilities. Depth perception is crucial for comprehending the spatial relationships between objects, enabling us to view the world in three dimensions. To fully exploit this capability, we utilize the depth map and 3D point cloud data provided by the ZED SDK [29]. The depth map records the distance (Z) for each pixel (X, Y) in the image, measured from the back of the left eye of the camera to the corresponding object in the scene. Visually, the depth map is represented as a grayscale image: pixels closer to the camera appear white (value 255), while those farther away appear black (value 0). With a depth estimation range extending up to 20 meters, the camera provides extensive distance data that meets the requirements of our application. By integrating depth estimation from the depth map with positional data from object detection, we achieve a comprehensive understanding of an object's spatial location and distance within the environment. This integration is essential for accurately interpreting the 3D positions of objects in a scene. Stereo vision relies on triangulation to infer depth from a disparity image. Depth resolution (Dr) is a function of distance (Z) and can be expressed by the formula:

$$Dr = Z^2 \times \alpha$$

where α is a constant. The accuracy of depth measurement diminishes quadratically with increasing distance, varying from approximately 1% of the distance in the near range to 9% in the far range [29]. Depth accuracy can be compromised

by anomalies, particularly on homogeneous or textureless surfaces such as white walls or green screens, leading to temporal instability in the measurements. The ZED 2 stereo camera's sophisticated design and advanced capabilities make it an invaluable tool for applications requiring precise depth perception and 3D spatial awareness. By combining these technologies, we are able to create a robust system capable of accurately mapping and understanding complex environments. This detailed understanding is essential for a wide range of applications, from autonomous navigation to advanced robotics, where precise spatial information is critical for performance and safety.

3.3 Object Orientation

StereoLabs offers a powerful algorithm capable of detecting objects within the environment as seen by the ZED2 stereo camera. This algorithm identifies various objects in the scene and generates precise 2D bounding boxes around each detected object. These bounding boxes are instrumental in isolating and analyzing the detected objects within the image. My algorithm leverages these bounding boxes by using them as regions of interest (ROI) for further image processing. By focusing on these ROIs, we can more effectively extract detailed information about the objects within them. Specifically, from each bounding box, we can derive a mask of the detected object. This process involves assigning a pixel value of 255 to areas within the bounding box where the object is present and a value of 0 to areas corresponding to the background. Consequently, this creates a binary mask where the object is rendered completely white against a black background, providing a clear and distinct representation of the object. The generation of these masks is crucial for subsequent image processing tasks. The binary mask not only simplifies the object segmentation process but also enhances the accuracy of object recognition and tracking. By isolating the object from its background, the algorithm can perform more precise analyses, such as calculating the object's exact dimensions, shape, and position within the scene. Furthermore, these masks can be used to improve the performance of machine learning models by providing clean, noise-free data for training and inference. The clear delineation between object and background ensures that models can learn and predict with higher accuracy. This is particularly beneficial in applications such as autonomous navigation, robotics, and augmented reality, where understanding the precise location and boundaries of objects is critical. The integration of StereoLabs' object detection algorithm with my custom processing techniques creates a robust system for detailed object analysis. By utilizing the bounding boxes and corresponding masks, we achieve a comprehensive understanding of the detected objects, enhancing the overall effectiveness of our image processing pipeline. This combination of advanced detection capabilities and

precise object segmentation underscores the versatility and power of the ZED2 stereo camera in real-world applications, making it an invaluable tool for cutting-edge technological solutions.

3.3.1 Strategy

The fundamental concept employed to detect the orientation of a vehicle relative to the camera involves analyzing the z distance for each point within the mask. This process identifies the point with the lowest distance, indicating the object closest to the camera, which serves as the starting point for orientation detection. In Figure 3.1, this starting point is denoted by the green circle. By maintaining the y value constant, we can determine the extreme left and right points (represented by the blue and red circles, respectively) at the same height as the starting point. Subsequently, we compute the z distance of these two points. With this information, along with the lateral distance relative to the starting point (x coordinate), we can calculate the angles representing the orientation of the vehicle. This methodological approach is pivotal for accurately determining the orientation of the vehicle in relation to the camera's perspective. By leveraging geometric principles and distance measurements, we can precisely ascertain the angular orientation of the vehicle with respect to the camera's viewpoint. This level of detail is crucial for various applications, such as autonomous navigation systems and driver assistance technologies, where understanding the spatial relationship between the vehicle and its surroundings is essential for making informed decisions and executing precise maneuvers. Furthermore, this approach offers flexibility and adaptability across different scenarios and environments. By dynamically analyzing the z distances and geometric relationships within the captured image, the system can effectively detect and interpret the orientation of vehicles under varying conditions, including changes in lighting, terrain, and vehicle size. The integration of this orientation detection technique into the overall framework enhances the functionality and utility of the camera system. By providing real-time insights into the orientation of vehicles within the camera's field of view, the system enables seamless integration with intelligent transportation systems, surveillance applications, and other advanced technologies aimed at improving safety, efficiency, and situational awareness in diverse settings. In summary, the method outlined above represents a robust and efficient approach to vehicle orientation detection, offering precise and reliable results that can be utilized across a wide range of applications and environments. Its effectiveness lies in its ability to leverage fundamental geometric principles and distance measurements to accurately infer the orientation of vehicles relative to the camera's viewpoint, thereby facilitating informed decision-making and enhanced functionality in various technological domains.

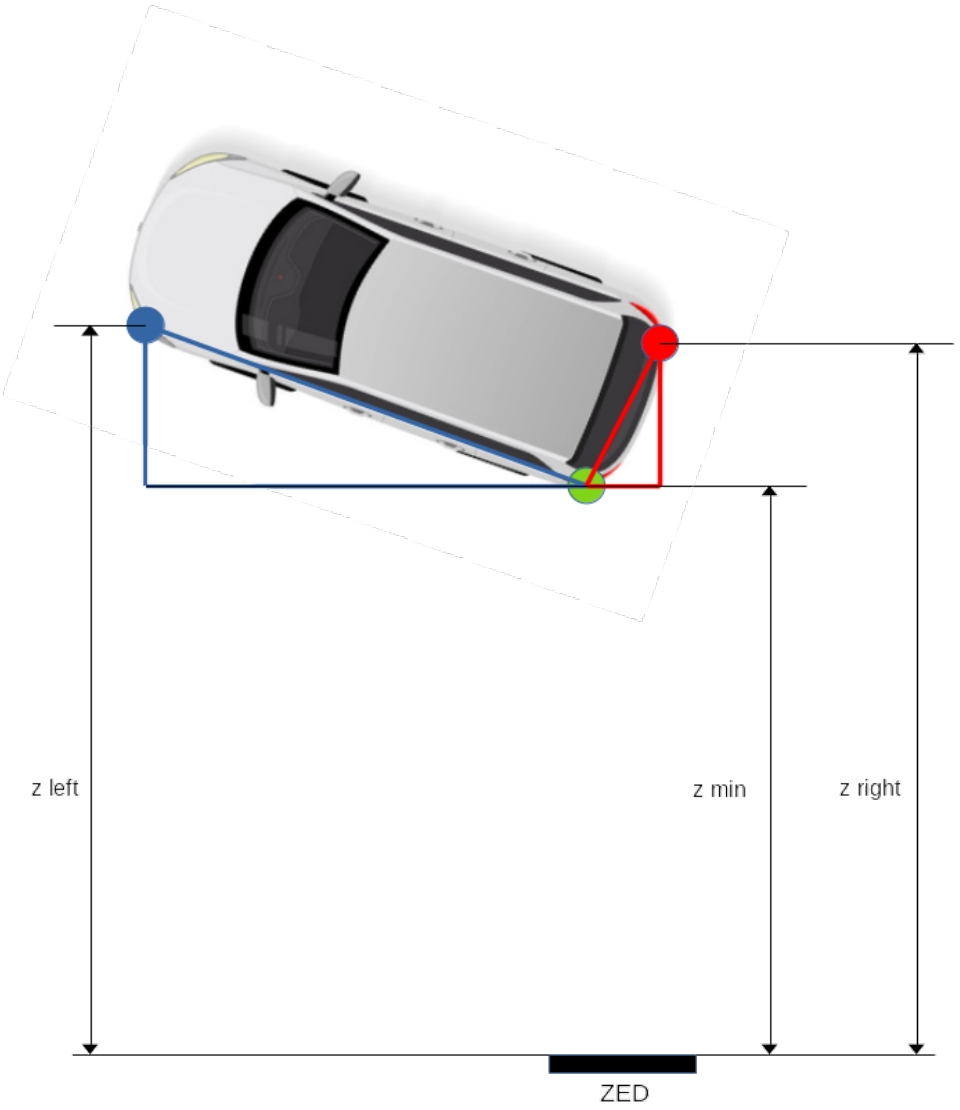


Figure 3.1: Vehicle points detection

3.3.2 Angles computation

Expanding upon the analysis depicted in Figure 3.2, which provides a zoomed-in view of the previous illustration, we observe the emergence of two distinct triangles originating from the closest point. These triangles, delineated by the blue and red lines connecting to the left and right points, respectively, form the basis for calculating the orientation angles of the vehicle relative to the camera. To initiate our calculations, we refer to a fundamental formula in trigonometry, illustrated in Figure 3.3. This formula serves as the cornerstone of our approach, enabling

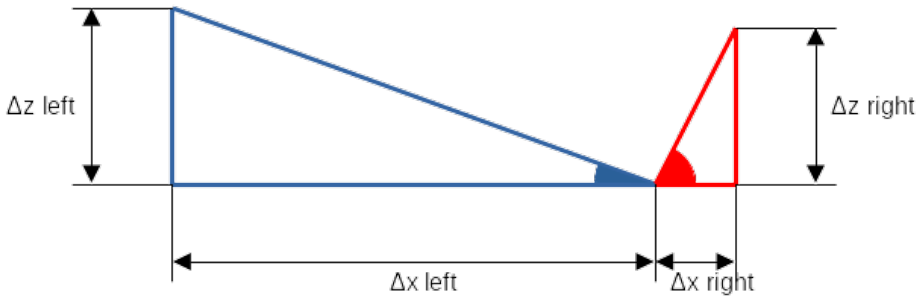


Figure 3.2: Orientation angles

us to establish the necessary relationships between the angles and sides of the triangles formed by the points of interest. By leveraging trigonometric principles, we can derive the precise orientation angles of the vehicle based on the geometric configuration of these triangles. The blue and red triangles, emanating from the closest point and extending towards the left and right points, respectively, provide crucial insights into the spatial orientation of the vehicle within the camera's field of view. Through careful analysis of the angles and distances within these triangles, we can infer the angular orientation of the vehicle relative to the camera's perspective with a high degree of accuracy. These expressions take into account factors such as the distances between points and camera and is not affected by error related to the height of the camera or the angle of inclination, providing a comprehensive framework for accurately determining the vehicle's orientation in three-dimensional space. The utilization of trigonometric principles in conjunction with geometric analysis enhances the robustness and accuracy of our orientation detection algorithm. By incorporating mathematical rigor into our methodology, we can ensure reliable and consistent results across various scenarios and environmental conditions. This level of precision is essential for applications requiring precise spatial awareness, such as autonomous navigation systems, augmented reality overlays, and object tracking technologies. In summary, the zoomed-in view provided by Figure 3.2 offers a detailed perspective on the geometric relationships underlying our orientation detection method. By leveraging trigonometric principles and geometric analysis, we can derive precise orientation angles for vehicles relative to the camera's viewpoint, enabling enhanced functionality and performance in a wide range of technological applications.

$$tan(\theta) = \frac{Opposite}{Adjacent} \tag{3.1}$$

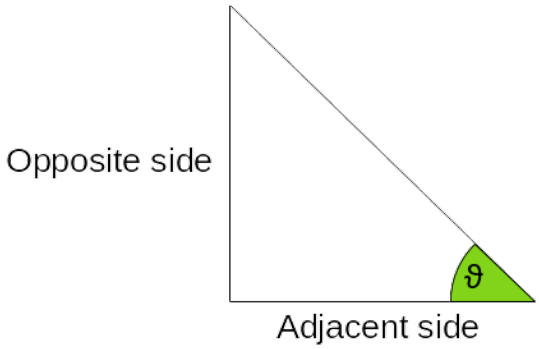


Figure 3.3: Basic Triangle

Equation 3.1 presents a fundamental trigonometric relationship, expressing the tangent of an angle (θ) as the ratio of the length of the side opposite to the angle to the length of the side adjacent to it. This relationship is illustrated in Figure 3.3, where the angle θ is defined within the context of a basic triangle. Expanding upon this fundamental concept, we apply it to our specific scenario involving triangles formed by variations in the z and x directions. In our case, the sides of the triangles correspond to the changes in the z direction (Δz) and the x direction (Δx). Consequently, Equation 3.1 is adapted to our context, resulting in Equation 3.2:

$$\tan(\theta) = \frac{\Delta z}{\Delta x} \tag{3.2}$$

In Equation 3.2, Δz represents the variation in the z direction, while Δx denotes the variation in the x direction. This equation captures the relationship between these variations and the angle θ, which characterizes the orientation of the vehicle relative to the camera. To determine the actual value of the angle θ, we utilize the inverse tangent function, denoted as tan^{-1}. By applying this function to the ratio $\frac{\Delta z}{\Delta x}$, we obtain the angle θ. This process enables us to calculate the precise orientation angle of the vehicle based on the observed variations in the z and x directions. The utilization of trigonometric principles in this manner facilitates the accurate determination of vehicle orientation, providing valuable insights for applications such as object tracking, navigation, and surveillance. By leveraging mathematical relationships derived from basic trigonometry, we can extract meaningful information about the spatial configuration of objects within the camera's field of view, enabling enhanced functionality and performance in a variety of technological contexts.

Filtering

After numerous attempts, we observed significant noise that rendered the vehicle's output and, consequently, the car's spatial orientation angle somewhat unstable. This noise primarily originates from the depth map. Despite applying various image filtering techniques and map stabilizers, the depth and distance values continuously fluctuate. This fluctuation results in constant changes to our green point (the point closest to the camera), which in turn causes persistent variations in the angle. To mitigate this issue, we introduced a filtering mechanism for the computed angle. This filter operates in a very straightforward manner: a queue stores a fixed number of computed angle values, and then an average of these values is calculated. Every new angle computed is added to the queue, replacing the oldest value, ensuring that the average is consistently updated. By adjusting the number of values stored in the queue, we can achieve an optimal balance between the responsiveness of angle changes over time (by reducing the number of stored values) and the stability of the orientation angle (by increasing the number of stored values). This method allows us to fine-tune the performance, providing a good trade-off between real-time responsiveness and the stability needed for accurate spatial orientation. Consequently, this approach significantly improves the overall stability and accuracy of the vehicle's spatial orientation, addressing the instability caused by the fluctuating depth map values.

3.4 Code Explanation

In this section, we delve deeply into all aspects of the C++ code. Specifically, in the appendices of this document, we provide some key sections of the code. In the following there is explanations of how the code are designed to give a comprehensive understanding of the underlying mechanisms and functionalities. We focus on the following parts:

- StereoCamera (ZED) Setup: We explore the configuration and initialization of the StereoCamera, detailing the steps required to properly set up the ZED camera for accurate data capture.
- Computation of Object Distance and Orientation: This part covers the algorithms and methodologies used to calculate the distance and orientation of objects.
- Data Sent to the QML Model: Here, we explain how the processed data is emitted and transmitted to the QML model.
- Vehicle Model: This section analyzes the vehicle model implemented in the code comparing it with the model for the other objects.

3.4.1 StereoCamera (ZED) Setup

As the first step of the code, it is essential to set up all the parameters of the ZED camera to ensure high-quality images under various light and weather conditions. Additionally, we need to enable all the necessary functionalities for our project. In Appendix A, we present the initial part of the code, which includes all the camera setup details. To delve deeper into the setup process, the first section of the code focuses on the initial parameter configuration. Here, we specify the depth mode accuracy to "neural" and set the minimum distance to 100cm. This adjustment helps enhance the medium-range precision of the detection. We also configure the unit of measurement to centimeters and align the coordinate system with the one used in the Qt interface, as illustrated in Figure 3.4. Next, we adjust all the video

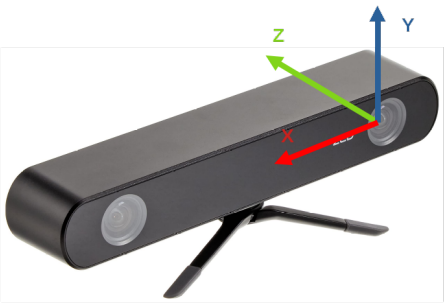


Figure 3.4: ZED Coordinate System

setting parameters of the camera, including gain, brightness, contrast, saturation, and other relevant settings, to ensure accurate video acquisition. This step is crucial for obtaining clear and precise images throughout the project's execution. Following the video settings, we proceed to activate the object detection functionalities. This involves enabling all the necessary features to detect objects effectively. We also set the confidence threshold to 60%, establishing a lower limit below which objects are ignored. This threshold helps filter out less reliable detection, ensuring that only significant objects are considered. Finally, we configure the settings related to object classes and filters. This involves specifying which types of objects the system should recognize and how to filter them based on the project's requirements. By fine-tuning these parameters, we can optimize the performance of the object detection system to meet our specific needs.

3.4.2 Computation of Object Distance and Orientation

In this section, we thoroughly examine the portion of the code responsible for computing the distance and relative position of various objects with respect to the camera, as well as determining the orientation of detected vehicles. The specific

code for this functionality is detailed in Appendix B. We begin by iterating through all objects detected in the scene. The initial step involves extracting the 2D bounding box of each detected object, which serves as the region of interest (ROI) for subsequent computations. Next, we focus on the mask of the object within the bounding box. By analyzing each point within this mask, we identify the point with the shortest distance with respect to the camera. Once this point is located, we store its coordinates and proceed to extract the outermost left and right points of the mask at the same height. These lateral distances, along with the depth from the camera, are crucial for computing the orientation of the vehicles. After calculating the left and right angles, we apply a filter to the right one (the one that we will use for the hmi angle of the cars) to mitigate the shaking effect observed in the car orientation, thereby enhancing the stability of the output image. This filtering process involves computing the average value of the last 30 angles measurements. By carefully following these steps, we ensure accurate and stable computation of object distances, positions, and vehicle orientations, providing a reliable basis for further processing and analysis in the system.

3.4.3 Data Sent to the QML Model

To establish a correlation between the information obtained from the camera and the data computed by our algorithm with the output we want to display on our Human-Machine Interface (HMI), we must systematically link all the necessary information to the HMI code in QML. The relevant code for this process is located in Appendix C. First, after detecting an object, we need to determine if it is a car or another type of object. We start by processing all objects that are not cars. For each non-car object, we collect its x, y, and z positions and store these coordinates in a QVector, which is used to continuously update the object's position. Additionally, we save all the relevant variables required to correctly position and size the object in the HMI. Next, we check whether this is the first time the object, identified by its unique ID number, is being detected. If it is the first detection, we append the new object to our data structures; otherwise, we update its position accordingly. For cars, the procedure is similar but includes an additional step. Besides the x, y, and z positions, we also capture and process the angle to correctly orient the car in the 3D space. This extra step ensures that cars are accurately represented in terms of both position and orientation within the HMI. By meticulously following these steps, we ensure that all objects, whether cars or not, are correctly represented and updated in our HMI.

3.4.4 Vehicle Model

In this section, we delve into the vehicle model used to process the information described in the previous section, manage it, and emit signals to send it to the QML code for the HMI interface. The detailed code for this process is provided in Appendix D. This specific code handles cars, while a similar but slightly simpler version is used for other objects. The simpler version for non-car objects does not need to manage orientation and angle parameters. Examining the code, we find a series of modules primarily designed to store and manage information before sending it via signals. The process begins by setting the properties of each vehicle. The initial "append" module creates a QVector to collect parameters for new cars. Subsequently, the second "append" module adds the new car to the vehicles list, setting all necessary parameters such as ID, coordinate positions, orientation angles, and a "canc" variable. The "canc" variable acts as a counter, used to remove objects from the scene when they are no longer detected. During each update, if the car (or object) is detected, this counter is reset to its initial value. Conversely, if the object is not detected, the counter decreases until it reaches zero, at which point the object is removed from the scene. The "update" module is then used to refresh the position of vehicles after each iteration. Additionally, there is a "freeupdate" module that continuously updates the "canc" parameter even if a specific car is no longer detected. This mechanism ensures that all vehicles are properly removed from the scene when they are no longer present. Overall, the vehicle model code is designed to efficiently manage the detection, updating, and removal of objects within the HMI interface. By emitting signals and handling various parameters, including position and orientation, it ensures accurate and dynamic representation of objects in the HMI.

3.5 Qt 3D

Qt 3D stands as a robust framework meticulously crafted to cater to the intricate needs of near-realtime simulation systems, boasting support for both 2D and 3D rendering within Qt C++ and Qt Quick applications. Rooted in the ethos of scalability, extensibility, and flexibility, Qt 3D represents a cornerstone in the realm of graphics programming, empowering developers to craft immersive and dynamic visual experiences with unparalleled ease [30]. Delving into the rich tapestry of features that Qt 3D has to offer unveils a myriad of tools and functionalities tailored to meet the diverse demands of modern application development. At the heart of Qt 3D lies its rendering capabilities, encompassing a fully configurable renderer that furnishes developers with the freedom to implement tailored rendering pipelines suited to their specific requirements. This inherent flexibility enables swift iteration and experimentation, facilitating the creation of visually stunning and performant

applications. Beyond its rendering prowess, Qt 3D distinguishes itself through its robust simulation framework, providing a generic platform for the development of near-realtime simulations across a spectrum of domains. From physics simulations to audio processing, collision detection to artificial intelligence (AI), Qt 3D offers a versatile canvas upon which developers can paint their simulation scenarios with unparalleled precision and fidelity. A hallmark of Qt 3D's design philosophy lies in its modular architecture, neatly organized into a core framework complemented by a myriad of interchangeable aspects. These aspects encapsulate specific functionalities, such as physics, audio, collision detection, AI, and pathfinding, allowing developers to mix and match components seamlessly to construct tailored solutions that align with their project's objectives. In the realm of materials and shaders, Qt 3D shines with its robust and highly flexible material system, offering developers multiple levels of customization to achieve their desired visual effects. Supporting all stages of the OpenGL programmable rendering pipeline, including vertex, tessellation control, tessellation evaluation, geometry, and fragment shaders, Qt 3D empowers developers to push the boundaries of graphical fidelity and realism. Shadow mapping, a critical component of many rendering pipelines, is seamlessly integrated into Qt 3D with simplicity and efficiency in mind. With Qt 3D, generating visually appealing shadows comes with minimal performance overhead, allowing developers to achieve stunning visual effects without compromising on runtime performance. In the context of academic research, Qt 3D presents a plethora of avenues for exploration and analysis within the realm of computer graphics and simulation. For instance, incorporating Qt 3D into a thesis project could involve the development of a small application or simulation as a case study, demonstrating the practical applications of the discussed concepts. Alternatively, one could delve into a performance analysis, examining the behavior of Qt 3D in various scenarios such as rendering complex scenes or handling large datasets. Moreover, conducting a comparative study between Qt 3D and other 3D frameworks in terms of ease of use, flexibility, and performance could yield valuable insights into the strengths and weaknesses of different approaches. Finally, exploring and discussing unique features of Qt 3D, such as its aspect-oriented design or material system, could shed light on the underlying principles driving its architecture and functionality.

3.5.1 3D Interface

The Human-Machine Interface (HMI) developed for this project is a 3D interface window designed to enhance the visualization and interaction experience for users. At the center of this interface is the ego car, depicted in blue, which serves as the primary reference point. This ego car is equipped with a camera, and all detected vehicles are represented in green. These vehicles are dynamically introduced into the scene as they are detected, reflecting their real-time orientation relative to the

camera's perspective. As the ego vehicle moves, the orientation of the detected vehicles updates accordingly, providing a realistic and fluid representation of the surrounding environment. Additionally, the interface allows users to navigate through the 3D space and manipulate the camera's viewpoint, offering a flexible and immersive experience. This feature is particularly useful for observing the environment from different angles and gaining a comprehensive understanding of the surroundings. For objects that are not cars, such as pedestrians, other types of vehicles, and animals, a different approach is used. These objects are represented by cylinders. This design choice simplifies the representation since the primary focus of the project is on cars. Cylinders are an ideal shape because they eliminate the need to depict the orientation of these objects, which is not the project's goals. Each cylinder is sized proportionately to the object's actual dimensions, both in terms of base area and height, ensuring an accurate and recognizable representation. An important aspect of the vehicle representation within the HMI is the standardized orientation of all detected vehicles. All vehicles, regardless of their actual direction, are oriented to face the same direction as the ego vehicle. This convention addresses a limitation of the software, which cannot reliably distinguish whether the detected vehicle is facing towards or away from the ego vehicle. By aligning all vehicles in the direction of the ego vehicle's movement, the interface maintains consistency and avoids potential confusion, thus enhancing the overall user experience. In summary, this HMI provides a detailed and interactive 3D visualization of the environment surrounding the ego vehicle. By focusing on cars and using simplified representations for other objects, the interface achieves a balance between detailed visualization and practical usability. The ability to navigate and adjust the camera view further enriches the user's interaction, making the HMI an effective tool for monitoring and analyzing the dynamic traffic scenario.

3.5.2 HMI Code

In Appendix E, we provide the QML code pertinent to the Human-Machine Interface (HMI) of our system. To offer a detailed explanation of how this code is structured and operates in practice, it is essential to understand the interaction between the QML interface and the C++ model, which is facilitated through signals. The C++ code emits signals, which the QML code then captures and processes. Initially, after creating the window, camera, and light components, we establish connections for these signals to transmit data regarding the position and orientation of the cars. For other objects, a cylinder representation is used. This means that every time a new car or object is detected, a corresponding new element is created within the interface. This element remains within the interface and continuously updates its position and orientation as long as the object stays within the camera's field of view. The mechanism ensures that all detected cars and objects are dynamically

tracked and rendered accurately in real-time within the HMI. This setup allows for a seamless and responsive user experience, where the interface reflects the current state of the environment with minimal latency. The continuous update of the position and orientation data ensures that the visual representation of the cars and other objects is always current, providing users with reliable and up-to-date information about the system's surroundings.

Chapter 4

Simulation and Tests

4.1 Simulation

Figure 4.1: Left view

The simulation and testing are conducted in and around the premises of Bylogix. The test area is designed to resemble a ring surrounding the company, featuring randomly parked cars and a few pedestrians, creating an ideal environment for testing. The stereocamera ZED2 is mounted on a test vehicle as we can see from figure 4.1 and figure 4.2, which is then driven around the circuit.
During these laps, the camera is connected to a PC running the necessary software to capture and analyze the data. From the resulting images, the outcomes are promising. The system demonstrates a robust capability to accurately recognize cars and other objects in its environment. The orientation system's precision is satisfactory, effectively displaying and positioning all encountered vehicles relative to the camera's viewpoint. Additionally, the system accurately identifies non-car

Figure 4.2: Front View

objects, and the cylindrical shapes used to represent these objects exhibit high fidelity, closely matching their real-world dimensions. Overall, the performance of the system is commendable. It successfully detects and accurately represents various objects, making it a reliable tool for further development and real-world application testing. The environment around Bylogix proves to be an excellent testing ground, providing diverse scenarios that contribute to the thorough evaluation of the system's capabilities.

4.2 Results

The results from our tests are highly promising. By comparing the images captured by the camera with the HMI (Human-Machine Interface) output presented to the user, we can see that discrepancies between the real world and the HMI are minimal and fall within an acceptable error range. Every vehicle within the camera's field of view is accurately detected and represented in the HMI, with their orientations correctly aligned relative to the camera. One minor issue observed is a slight difference in the height of the outputs. This variation arises because, in our simulation, the camera is mounted on the roof of the test vehicle, which lacks a hood. Conversely, in the HMI, the camera is depicted as being mounted on the hood. This positioning choice was made to minimize the presence of the "ego car" in the HMI view, allowing for more accurate distance estimation and representation between the test vehicle and the real world. To illustrate the functionality and accuracy of the HMI output, we captured pairs of frames at different points during our road test simulations. These pairs consist of the HMI interface and the corresponding

camera view of the environment, with each detected object in the latter being surrounded by a 2D red bounding box. This approach enables us to evaluate how the system performs under varying environmental conditions, providing insight into its precision and robustness. Each pair of images showcases the HMI's ability to consistently reflect the camera's view, demonstrating the system's reliability and effectiveness in real-world scenarios. In the first pair of images (Figure 4.3

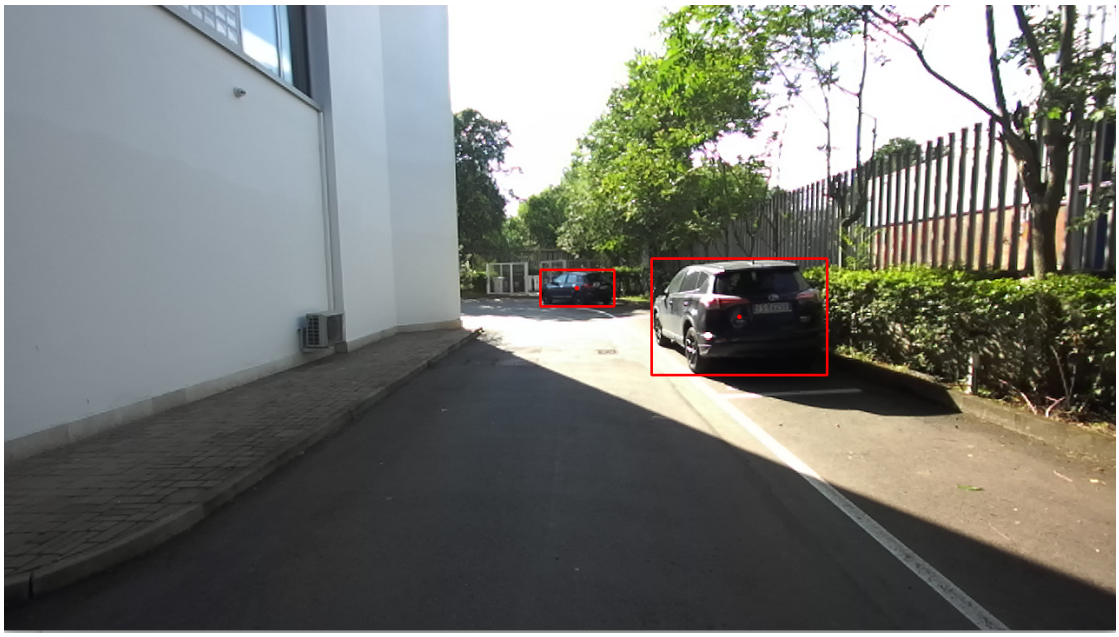


Figure 4.3: Example 1: Camera View

and Figure 4.4), we capture an initial snapshot from our test simulation. In these images, two cars are visible: one is parked laterally relative to our vehicle and is much closer, while the other is positioned farther away, almost at the edge of the camera's detection range. The HMI accurately represents both vehicles, showcasing their relative positions and orientations concerning the camera and each other. Notably, the system demonstrates its capability to precisely detect vehicles even when they are near the detection range limit. This accuracy is crucial for ensuring the reliability of the system in various real-world scenarios. Another important aspect to highlight is the lighting conditions in this segment of the testing track. The area includes a significant shadow zone that is quite dark, followed by a region with very high brightness. Despite these challenging lighting conditions, the camera and software perform quite well, maintaining good detection capabilities across varying light levels. This ability to handle extreme variations in lighting further underscores the robustness and reliability of our system in different environmental conditions. The HMI's performance in these conditions indicates a strong potential

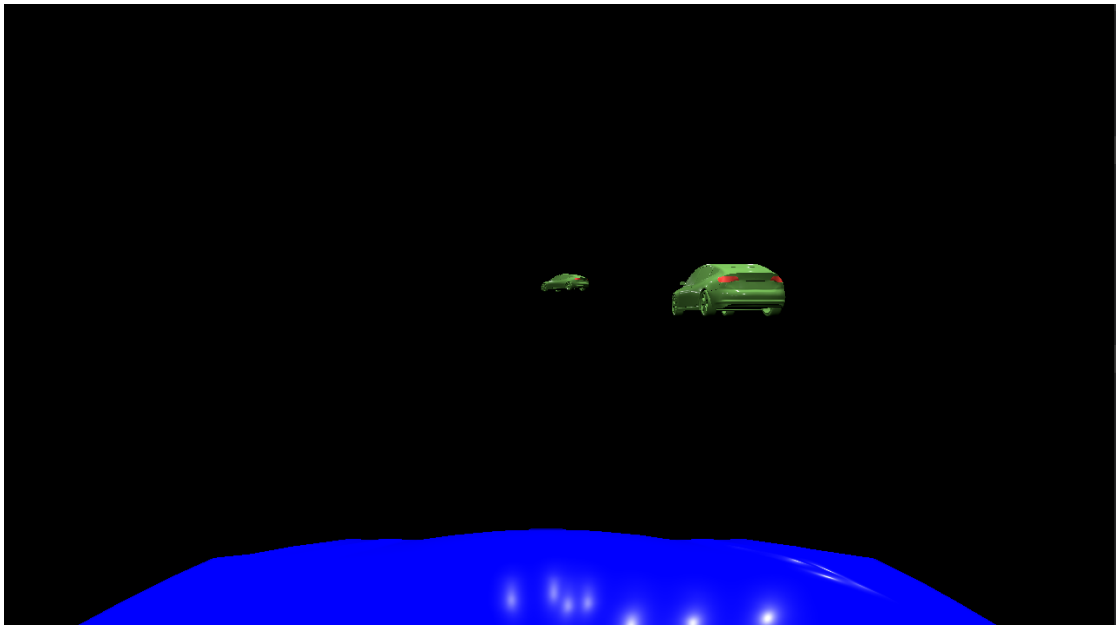


Figure 4.4: Example 1: HMI View

for deployment in various applications, offering confidence in its ability to handle complex and dynamic situations like challenging scenarios.

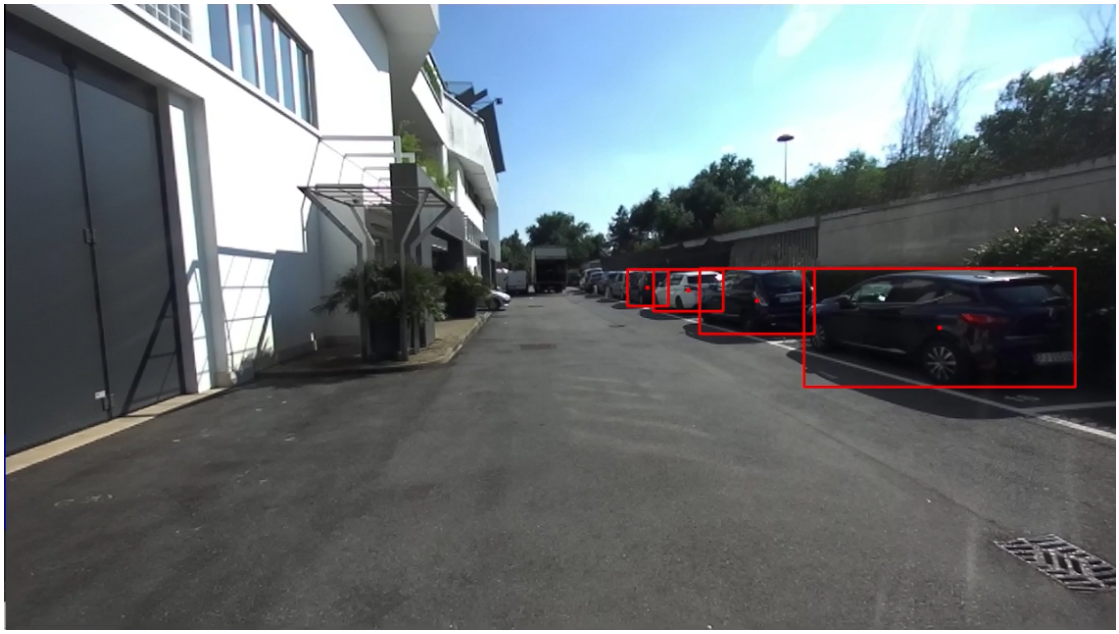


Figure 4.5: Example 2: Camera View

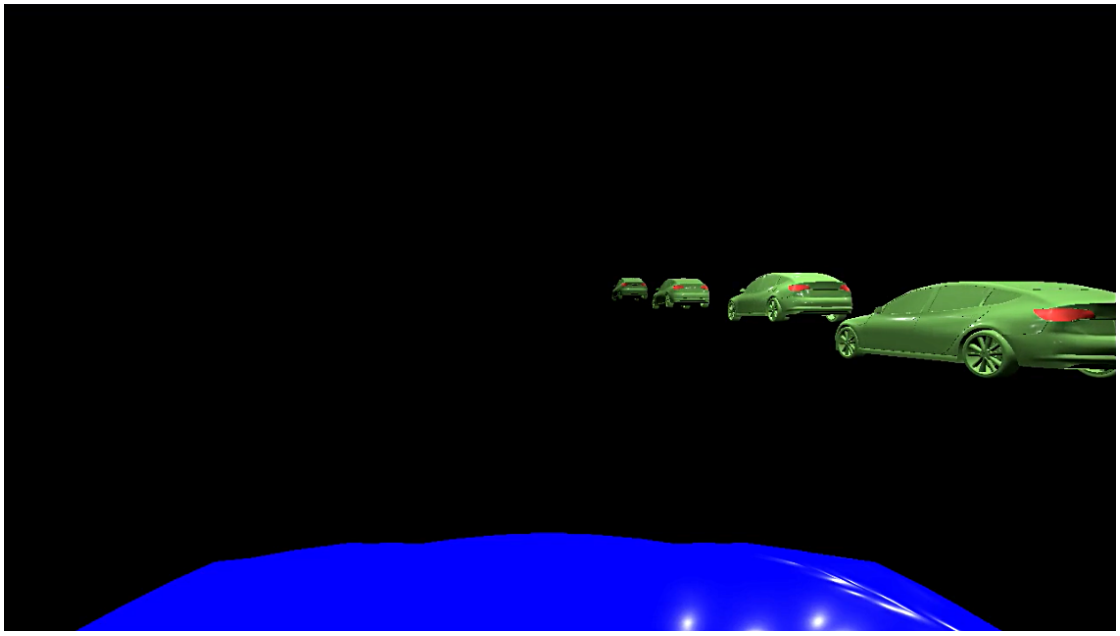


Figure 4.6: Example 2: HMI View

In the second pair of images (Figure 4.5 and Figure 4.6), we observe a high-traffic scenario, with a substantial number of vehicles represented by a row of cars on the right side of the image. The system successfully detects vehicles up to a certain distance, identifying a total of four cars, all of which are accurately represented in our HMI, as shown in Figure 4.6. This segment of the testing track is characterized by direct sunlight shining almost frontally and directly onto the camera. Despite these challenging lighting conditions, the camera performs exceptionally well, maintaining clear visibility and accurately detecting cars even under very high brightness. The system's ability to function effectively in such conditions highlights its robustness and reliability. In the HMI, our ego car is consistently represented in blue. Additionally, the row of detected cars on the right is displayed by the classical green cars, with their orientations accurately depicted. This clear and precise representation ensures that the user has an accurate understanding of the environment, even in complex and brightly lit scenarios. The performance observed in this high-traffic, high-brightness test scenario underscores the system's capability to handle diverse and challenging environments. The accurate detection and representation of multiple vehicles, despite direct sunlight interference, demonstrate the effectiveness of the camera and HMI integration in providing reliable and detailed real-time information to the user.
From the third example (Figure 4.7 and Figure 4.8) onward, the weather conditions differ from the first two examples as the test was conducted on a different day. The weather was very cloudy, almost raining, which allowed us to evaluate performance

Figure 4.7: Example 3: Camera View

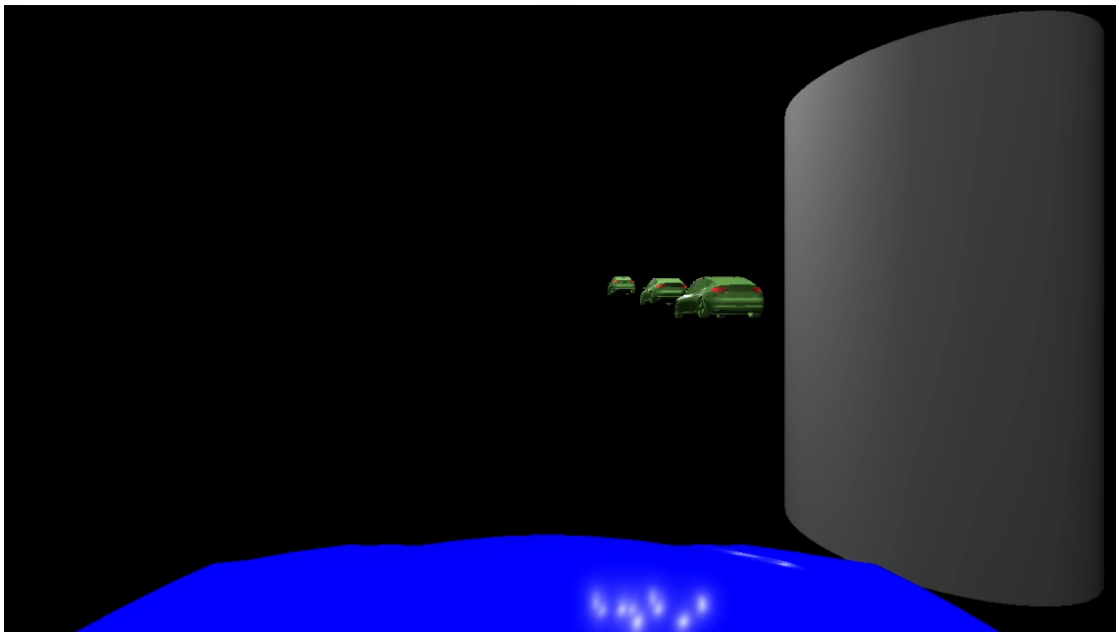


Figure 4.8: Example 3: HMI View

under varied weather conditions. Nevertheless, cloudy conditions are ideal for camera-related work. In this scenario, we detect a light-duty vehicle that obviously

is not classified as a car, and it is therefore represented as a cylinder with the dimensions of the bounding box around it. The scene also includes three cars lined up behind the heavy-duty vehicle, with the first two oriented in the opposite direction of our motion. However, aligned with what was mentioned in the previous section, all vehicles are depicted in the HMI with the same orientation as the ego vehicle's direction of motion. The output, in this case, remains accurate and precise, demonstrating consistent performance across different conditions.

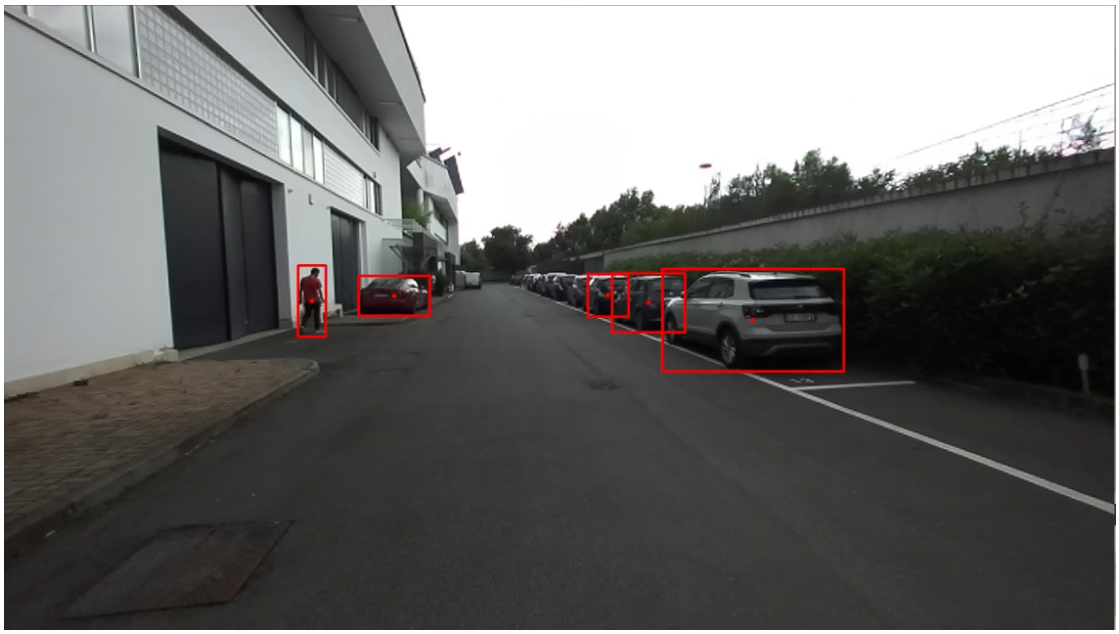


Figure 4.9: Example 4: Camera View

The next example (Figure 4.9 and Figure 4.10) takes place in nearly the same location as the second example. The primary difference in this scenario is the presence of a person on the left side of the image, accompanied by several cars. As depicted, the person is represented by a cylinder, consistent with the previous example, maintaining the same size as the bounding box around them. Additionally, there is a car situated to the left in front of the person. On the right side of the image, up to three cars are visible, with any additional vehicles being too distant to fall within the maximum detection range. This example highlights the system's ability to accurately identify and represent different types of objects within the scene, maintaining consistency in the representation of human figures as cylinders. The inclusion of multiple vehicles and a pedestrian provides a comprehensive overview of the detection capabilities in a more complex environment. By comparing this example to the previous ones, we can observe how the system adapts to varied arrangements and densities of objects, ensuring reliable performance even when

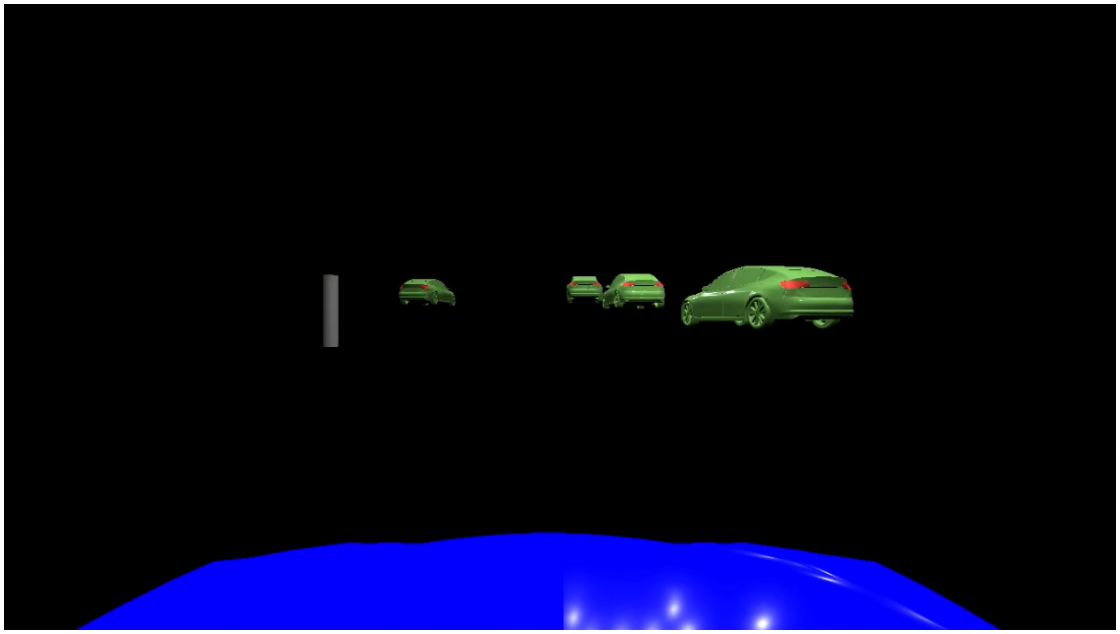


Figure 4.10: Example 4: HMI View

the objects are positioned at different distances and have different size.

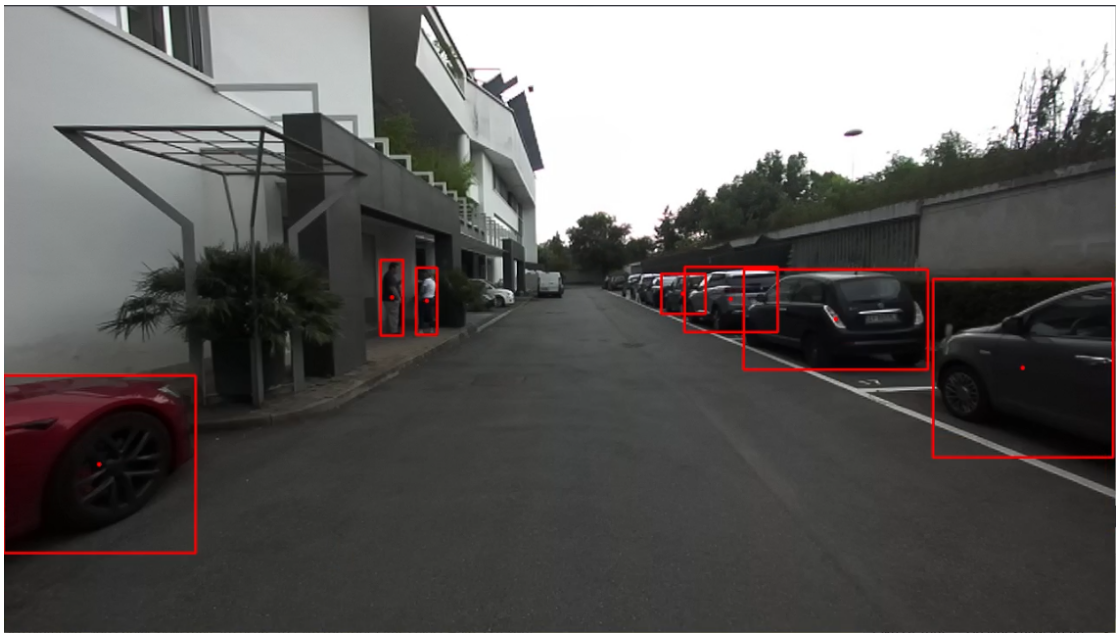


Figure 4.11: Example 5: Camera View

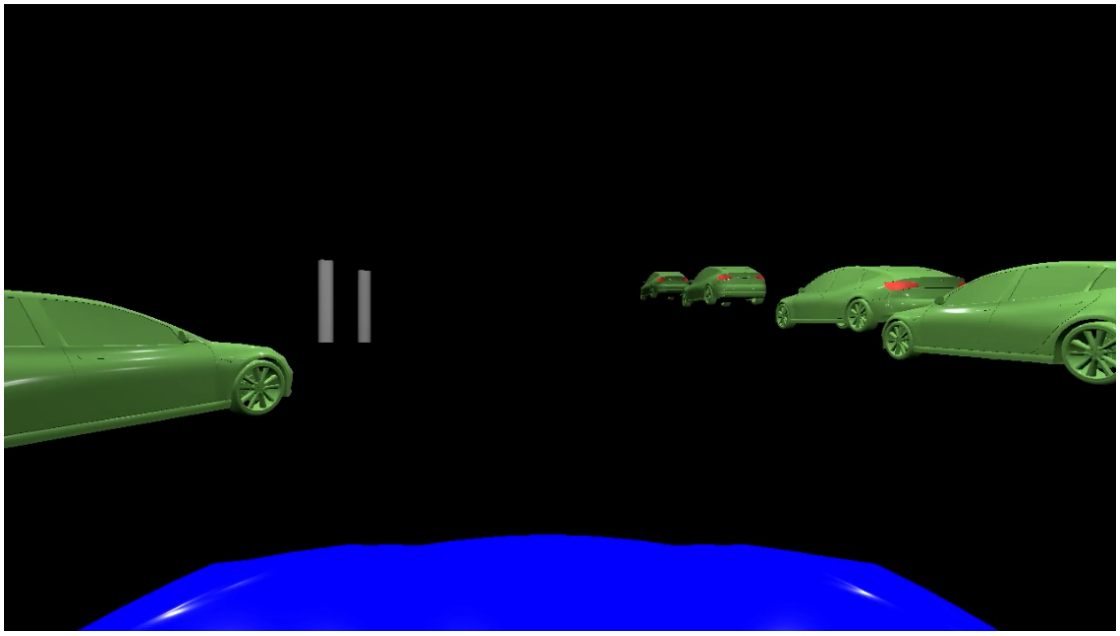


Figure 4.12: Example 5: HMI View

The last pair of images (Figure 4.11 and Figure 4.12) showcases a scenario similar to the previous example, but situated slightly further along the testing track. The column of cars on the right side remains consistent with the earlier scene. On the left side, the same car from the previous scene is gradually disappearing from view, while two people are seen conversing together. In this scenario, the results displayed on the HMI are very accurate, offering a perfect representation of the captured scene. The cars are oriented correctly, reflecting their real-world positions and directions. The two individuals are depicted as distinctly separate cylinders, emphasizing the system's capability to differentiate and accurately represent multiple human figures within the same frame also if they are quite close to each other. This example further highlights the robustness of the detection and representation system in handling dynamic and slightly altered scenarios within the same environment. The ability to consistently track and represent both stationary and moving objects, including vehicles and pedestrians, underscores the system's precision.

Chapter 5

Conclusion

Given the time constraints of the project, the results obtained are quite commendable. It is acknowledged that our output is subject to a certain degree of error, particularly in the orientation angle detection. This is due to error propagation, where multiple small errors accumulate, affecting the overall accuracy. One source of error stems from the depth map, which inherently contains inaccuracies in depth computation. These inaccuracies are influenced by factors such as lighting conditions and image quality. Additionally, the three-point estimation process contributes to the overall error, leading to inaccuracies in angle computation. To mitigate this issue, we implemented a strategy that involves filtering the angle values by computing the average. This approach helps to reduce noise and minimize oscillations in the vehicle representations, resulting in a more stable output. Despite these errors, the results are still satisfactory and acceptable. However, there is room for improvement. By focusing on reducing these errors and enhancing the stability and robustness of the system, we can achieve even better results.

5.1 Further Improvements

A series of potential enhancements can be made to the system to reduce errors and improve precision, stability, and robustness. The primary strategy involves implementing a neural network for the detection and classification of road actors, which would provide greater control and flexibility. A promising approach is the development of a neural network classifier capable of simultaneously identifying the type of vehicle and its primary orientation, such as front, rear, and side views [31] [32] [33]. This would enable a single network to perform an initial powerful classification, filtering objects in the environment and ensuring a more accurate representation of the detected vehicles. The use of advanced neural network architectures, such as convolutional neural networks (CNNs) or recurrent neural

networks (RNNs), can significantly enhance the system's ability to process and classify visual data with high accuracy. To achieve this, a comprehensive dataset of vehicle images from all possible viewpoints is necessary, similar to the dataset created and used in [34] [31]. For instance, in [34], the Car Full View (CFW) dataset was introduced, which involves recording individual cars from every angle by circling the vehicle, starting from the top front and walking clockwise until returning to the initial position. This method ensures a diverse and thorough collection of vehicle images that can significantly enhance the neural network's training and performance. The dataset should include various lighting conditions, weather scenarios, and different vehicle types to ensure the network is robust and versatile. Another significant improvement could be achieved by increasing the precision of distance estimation and depth mapping through sensor fusion. By integrating data from the stereo camera with information from other sensors, such as a LiDAR point cloud, we could obtain more accurate and precise measurements. Sensor fusion combines the strengths of multiple sensing technologies, compensating for the weaknesses of each individual sensor and resulting in more reliable and robust data. This approach can help in creating a more comprehensive understanding of the environment, allowing for better detection and classification of road actors. The integration of sensor fusion would involve sophisticated algorithms to merge the data from various sensors. Techniques such as Kalman filtering, particle filtering, or deep learning-based fusion methods can be employed to effectively combine the sensor data. These methods help in reducing the uncertainty and noise inherent in individual sensor measurements, leading to improved overall system performance. Additionally, from an application standpoint, it is crucial to enhance the system's real-time processing capabilities to ensure practical usability. This can be achieved by optimizing the software and hardware components, ensuring that the neural network and sensor fusion processes can operate efficiently within the constraints of the vehicle's onboard systems. Techniques such as hardware acceleration, parallel processing, and efficient code optimization can play a significant role in achieving real-time performance. Moreover, continuous learning and adaptation of the system can further improve its robustness. Implementing mechanisms for the system to learn from new data collected during operation can help it adapt to changing environments and improve its accuracy over time. This can be facilitated through online learning algorithms or periodic updates to the neural network model based on newly acquired data.

In conclusion, while the current system performs well within the given constraints, there are several avenues for improvement that can significantly elevate its accuracy and reliability. Implementing a neural network for vehicle detection and orientation classification, utilizing a comprehensive dataset, and employing sensor fusion for enhanced depth mapping and distance estimation are key strategies to enhance the system. These enhancements will not only reduce errors but also bolster the

overall performance and robustness of the system. Continuous improvement and refinement of the system will lead to more precise and reliable outcomes, ultimately enhancing the overall effectiveness of the project. By optimizing and adapting the system, it will remain effective and efficient in real-world applications, contributing to safer and more reliable autonomous vehicle technologies.

Appendix A

Appendix

```
#include <sl/Camera.hpp>

{
Camera zed;

InitParameters init_params;
init_params.depth_mode = DEPTH_MODE::NEURAL;
init_params.depth_minimum_distance = 100;
init_params.coordinate_units = UNIT::CENTIMETER;
init_params.depth_stabilization = 45;
init_params.coordinate_system = sl::COORDINATE_SYSTEM::
    RIGHT_HANDED_Y_UP;
init_params.sdk_verbose = true;

auto returned_state = zed.open(init_params);
if (returned_state != sl::ERROR_CODE::SUCCESS) {
    std::cout << "Error " << returned_state << ", exit program.\n";
    return EXIT_FAILURE;
}

zed.setCameraSettings(VIDEO_SETTINGS::GAIN, 1);
zed.setCameraSettings(VIDEO_SETTINGS::BRIGHTNESS, 4);
zed.setCameraSettings(VIDEO_SETTINGS::CONTRAST, 4);
zed.setCameraSettings(VIDEO_SETTINGS::HUE, 0);
zed.setCameraSettings(VIDEO_SETTINGS::SATURATION, 4);
zed.setCameraSettings(VIDEO_SETTINGS::SHARPNESS, 4);
zed.setCameraSettings(VIDEO_SETTINGS::GAMMA, 5);
zed.setCameraSettings(VIDEO_SETTINGS::WHITEBALANCE_AUTO, 1);
zed.setCameraSettings(VIDEO_SETTINGS::AEC_AGC, 1);

ObjectDetectionParameters detection_parameters;
```

```
31 detection_parameters.enable_tracking = true;
32 detection_parameters.enable_segmentation = true;
33 detection_parameters.detection_model = OBJECT_DETECTION_MODEL::
       MULTI_CLASS_BOX_ACCURATE;
34
35 int detection_confidence = 60;
36 ObjectDetectionRuntimeParameters detection_parameters_rt(
       detection_confidence);
37 detection_parameters_rt.object_class_filter = {sl::OBJECT_CLASS::
       VEHICLE, sl::OBJECT_CLASS::PERSON, sl::OBJECT_CLASS::ANIMAL, sl::
       OBJECT_CLASS::SPORT}; detection_parameters_rt.
       object_class_detection_confidence_threshold[sl::OBJECT_CLASS::
       PERSON] = detection_confidence; detection_parameters_rt.
       object_class_detection_confidence_threshold[sl::OBJECT_CLASS::
       VEHICLE] = detection_confidence; detection_parameters_rt.
       object_class_detection_confidence_threshold[sl::OBJECT_CLASS::
       ANIMAL] = detection_confidence; detection_parameters_rt.
       object_class_detection_confidence_threshold[sl::OBJECT_CLASS::
       SPORT] = detection_confidence;
38
39 if (detection_parameters.enable_tracking) {
40     PositionalTrackingParameters positional_tracking_parameters;
41     zed.enablePositionalTracking(positional_tracking_parameters);
42 }
43
44 std::cout << "Object Detection: Loading Module..." << std::endl;
45 returned_state = zed.enableObjectDetection(detection_parameters);
46 if (returned_state != sl::ERROR_CODE::SUCCESS) {
47     std::cout << "Error " << returned_state << ", exit program.\n";
48     zed.close();
49     return EXIT_FAILURE;
50 }
51
52  zed.close();
53 }
```

Appendix B

Appendix

```
for (const auto& object : objects.object_list) {
    // Define the ROI around the object using the 2D bounding box
    from the object detection
    std::vector<cv::Point2f> vertices;
    for (const auto& point : object.bounding_box_2d) {
        vertices.emplace_back(cv::Point2f(point.x, point.y));
    }

    // Calculate the top-left vertex, width, and height
    float x_min = std::min({vertices[0].x, vertices[1].x, vertices
    [2].x, vertices[3].x});
    float y_min = std::min({vertices[0].y, vertices[1].y, vertices
    [2].y, vertices[3].y});
    float x_max = std::max({vertices[0].x, vertices[1].x, vertices
    [2].x, vertices[3].x});
    float y_max = std::max({vertices[0].y, vertices[1].y, vertices
    [2].y, vertices[3].y});

    float width = x_max - x_min;
    float height = y_max - y_min;

    // Create the rectangle
    cv::Rect2f roi(x_min, y_min, width, height);

    //Center of rectangle
    int x = x_min + width/2;
    int y = y_min + height/2;

    auto ID = std::to_string(object.id);

    sl::float4 point_cloud_value;
```

```
27      point_cloud.getValue(x, y, &point_cloud_value);
28
29      float distance_mask_min = 10000.0;
30      float x_pos = 0.0;
31      float y_pos = 0.0;
32      float left_angle = 0.0;
33      float right_angle = 0.0;
34      float yl = 0.0;
35      float yr = 0.0;
36      float d_x_pos = 0.0;
37      float D_to_right = 0.0;
38      float D_to_left = 0.0;
39
40      if (object.mask.isInit()){
41
42          std::cout << "Mask is available" << std::endl;
43          Mat mask = object.mask;
44          cv::Mat mask_cv = cv::Mat(mask.getHeight(), mask.getWidth(),
    CV_8UC1, mask.getPtr<sl::uchar1>(sl::MEM::CPU));
45
46          for(int i=0; i<int(mask.getHeight()); i++)
47          {
48              for (int j=0; j<int(mask.getWidth()); j++)
49              {
50                  if(mask_cv.at<sl::uchar1>(i,j) == 255)
51                  {
52                      sl::float4 point_cloud_value_mask;
53                      point_cloud.getValue(j+x_min, i+y_min, &
    point_cloud_value_mask);
54
55                      if(std::isfinite(point_cloud_value_mask.z))
56                      {
57                          float distance_mask = abs(
    point_cloud_value_mask.z);
58
59                          if(distance_mask < distance_mask_min)
60                          {
61                              distance_mask_min = distance_mask;
62                              x_pos = j+x_min;
63                              y_pos = i+y_min;
64                              d_x_pos = abs(point_cloud_value_mask.x);
65                          }
66                      }
67                  }
68              }
69          }
70
71          cv::Point MiddlePoint(x_pos, y_pos);
```

```
72          cv::circle(depth_image_ocv, MiddlePoint, 2, cv::Scalar(0,
    255, 0), 2); // Green
73
74          // External points
75          float xl = x_min+1;
76          yl = y_pos;
77          for (int i = 0; i<10; i++) {
78              xl += i;
79              sl::float4 point_cloud_value_check_l;
80              point_cloud.getValue(xl, yl, &point_cloud_value_check_l);
81              if (std::isfinite(point_cloud_value_check_l.z)) {
82                  break;
83              }
84          }
85          cv::Point LeftPoint(xl, yl);
86          cv::circle(depth_image_ocv, LeftPoint, 2, cv::Scalar(255, 0,
    0), 2);
87
88          float xr = x_max-1;
89          yr = y_pos;
90          for (int i = 0; i<10; i++) {
91              xr -= i;
92              sl::float4 point_cloud_value_check_r;
93              point_cloud.getValue(xr, yr, &point_cloud_value_check_r);
94              if (std::isfinite(point_cloud_value_check_r.z)) {
95                  break;
96              }
97          }
98          cv::Point RightPoint(xr, yr);
99          cv::circle(depth_image_ocv, RightPoint, 2, cv::Scalar(0, 0,
    255), 2);
100
101         sl::float4 point_cloud_value_mask_extLeft;
102         point_cloud.getValue(xl, yl, &point_cloud_value_mask_extLeft)
    ;
103         float d_xl = abs(point_cloud_value_mask_extLeft.x);
104         float distance_mask_extLeft = abs(
    point_cloud_value_mask_extLeft.z);
105         D_to_left = abs(d_x_pos - d_xl);
106
107         if (D_to_left != 0) {
108             float left_angle_rad = atan((distance_mask_extLeft -
    distance_mask_min)/D_to_left);
109             left_angle = (180*left_angle_rad)/M_PI;
110         }
111         else if(abs(D_to_left+D_to_right) < 250)
112         {
113             left_angle = 0;
114         }
```

```
115         else
116         {
117             left_angle = 90;
118         }
119 
120         sl::float4 point_cloud_value_mask_extRight;
121         point_cloud.getValue(xr, yr, &point_cloud_value_mask_extRight
    );
122         float d_xr = abs(point_cloud_value_mask_extRight.x);
123         float distance_mask_extRight = abs(
    point_cloud_value_mask_extRight.z);
124         D_to_right = abs(d_x_pos - d_xr);
125         if (abs(D_to_left+D_to_right) < 200)
126         {
127             right_angle = 0;
128         }
129         else
130         {
131             if (D_to_right != 0) {
132                     float right_angle_rad = atan((
    distance_mask_extRight - distance_mask_min)/D_to_right);
133                     right_angle = (180*right_angle_rad)/M_PI;
134                 }
135             else
136             {
137                right_angle = 90;
138             }
139         }
140     }
141     else
142     {
143         std::cout << "Mask NOT available" << std::endl;
144     }
145 
146     // Orintentation check
147     if (D_to_right > D_to_left) {
148         right_angle = - right_angle;
149     }
150 
151     // Filter on the angle
152     float filtered_right_angle = 0.0;
153 
154     if (object.id > last_id || first_v) {
155         QQueue<float> qQ;
156         qQ.push_back(right_angle);
157         angle_storage[object.id] = qQ;
158     }
159     else
160     {
```

```
161         if (angle_storage[object.id].size() >= 30) {
162             angle_storage[object.id].pop_front();
163         }
164         angle_storage[object.id].push_back(right_angle);
165 
166         float sum = 0;
167         for(int i=0; i<angle_storage[object.id].size(); i++) {
168             sum += angle_storage[object.id][i];
169         }
170         filtered_right_angle = sum/angle_storage[object.id].size();
171     }
```

Appendix C

Appendix

```
// Check all object different from cars
if (object.label != OBJECT_CLASS::VEHICLE || (object.label ==
    OBJECT_CLASS::VEHICLE && object.sublabel != OBJECT_SUBCLASS::CAR))
    {

     QVector3D posUP(object.position.x, object.position.y, object.
    position.z);
     float positionX = object.position.x;
     float positionY = object.position.y;
     float positionZ = object.position.z;
     float width = object.dimensions.x/2;
     float height = object.dimensions.y;
     int canc_p = 15;

     if (object.id > last_id || first_p){

         pModel.append(object.id, canc_p, positionX, positionY,
    positionZ, width, height);
         first_p = false;
         last_id = object.id;
     }
     else
     {
         pModel.update(object.id, posUP, width, height);
         vModel.freeupdate();
     }
}

// For cars
else
{
```

```
28 // Data for qml model
29     QVector3D posUP(object.position.x, object.position.y, object.
       position.z);
30     QVector3D rotUP(0, 180 + filtered_right_angle, 0);
31     float positionX = object.position.x;
32     float positionY = object.position.y;
33     float positionZ = object.position.z;
34     float angleX = 0.0;
35     float angleY = 180 + filtered_right_angle;
36     float angleZ = 0.0;
37     int canc = 15;
38
39     if (object.id > last_id || first_v) {
40         vModel.append(object.id, canc, positionX, positionY,
       positionZ, angleX, angleY, angleZ);
41         first_v = false;
42         last_id = object.id;
43     }
44     else
45     {
46         vModel.update(object.id, posUP, rotUP);
47         pModel.freeupdate();
48     }
49 }
50 }
51
52 // Free update to delete the scene if no detection
53 vModel.freeupdate();
54 pModel.freeupdate();
```

Appendix D

Appendix

```
#include "Vmodel.h"
#include <qqml.h>
#include <QRandomGenerator>

VModel::VModel(QObject *parent)
    : QObject{parent}
{}

QQmlListProperty<VehicleModel> VModel::vehicles()
{
#if QT_VERSION < QT_VERSION_CHECK(5, 15, 0)
    return QQmlListProperty<VehicleModel>(this, m_vehicles);
#else
    return QQmlListProperty<VehicleModel>(this, &m_vehicles);
#endif
}

void VModel::append(int id, int canc, float positionX, float
    positionY, float positionZ, float angleX, float angleY, float
    angleZ)
{
    QVector3D position = { positionX, positionY, positionZ };
    qDebug() << "Vehicle position:" << position;
    QVector3D eulerRotation = { angleX, angleY, angleZ };
    qDebug() << "Vehicle rotstion:" << eulerRotation;
    int ID = id;

    append(ID, canc, position, eulerRotation);
}

```

```
29 void VModel::append(int &ID, int &canc, QVector3D &position,
       QVector3D &eulerRotation)
30 {
31     VehicleModel *vehicle = new VehicleModel(this);
32     vehicle->setID(ID);
33     vehicle->setCanc(canc);
34     vehicle->setPosition(position);
35     vehicle->setEulerRotation(eulerRotation);
36     m_vehicles.append(vehicle);
37     qDebug() << "Vehicles list" <<  m_vehicles.size();
38     emit vehiclesChanged();
39     qDebug() << "Signal vehiclesChanged emitted (Append)";
40 }
41
42 void VModel::update(int ID, QVector3D posUp, QVector3D rotUp)
43 {
44     for (int i = 0; i < m_vehicles.size(); ++i) {
45         auto vehicle = m_vehicles.at(i);
46         if (vehicle->canc() < 1) {
47             m_vehicles.removeAt(i);
48             --i;
49         }
50     }
51
52     for (auto& vehicle : m_vehicles) {
53         if (vehicle->ID() == ID) {
54             vehicle->setPosition(posUp);
55             qDebug() << "Vehicle position update:" << posUp;
56             vehicle->setEulerRotation(rotUp);
57             qDebug() << "Vehicle rotation update:" << rotUp;
58             qDebug() << "Update vehicle with ID:" << ID;
59             int v = 15; //Inizialization value
60             vehicle->setCanc(v);
61         }
62         int reducedCanc = vehicle->canc() - 1;
63         vehicle->setCanc(reducedCanc);
64     }
65
66     emit vehiclesChanged();
67     qDebug() << "Signal vehiclesChanged emitted (Update)";
68 }
69
70 void VModel::freeupdate()
71 {
72     for (int i = 0; i < m_vehicles.size(); ++i) {
73         auto vehicle = m_vehicles.at(i);
74         if (vehicle->canc() < 1) {
75             m_vehicles.removeAt(i);
76             --i;
```

```
77          }
78      }
79
80      for (auto& vehicle : m_vehicles) {
81          int reducedCanc = vehicle->canc() - 1;
82          vehicle->setCanc(reducedCanc);
83      }
84      emit vehiclesChanged();
85  }
```

Appendix E

Appendix

```
import QtQuick
import QtQuick.Controls.Basic
import QtQuick3D
import QtQuick3D.Helpers
import QtQuick.Window

import "assets/tesla_blender"
import "assets/tesla_ego"

ApplicationWindow {
    id: window
    width: 1280
    height: 720
    visible: true
    color:"black"

    Node {
        id:sceneRoot

        PerspectiveCamera {
            id: camera
            position: Qt.vector3d(0, 0, 0)
            eulerRotation.y: 0
            eulerRotation.x: 0
            fieldOfView: 70
        }

        DirectionalLight {
            eulerRotation.y: 10
            eulerRotation.x: -90
        }
```

```
32
33          Connections {
34              target: vModel
35              function onVehicleChanged() {
36                  console.log("Manual connection signal")
37              }
38          }
39
40          Repeater3D {
41              id: vehicle3dRepeater
42              model: vModel.vehicles
43
44              delegate: Tesla {
45                  position: model.position
46                  scale: Qt.vector3d(0.5,0.5,0.5)
47                  eulerRotation: model.eulerRotation
48              }
49          }
50
51          Connections {
52              target: pModel
53          }
54
55          Repeater3D {
56              id: person3dRepeater
57              model: pModel.people
58
59              delegate: Model {
60                  id: myCylinder
61                  source: "#Cylinder"
62                  position: model.position
63                  property real cylinderRadius: model.width
64                  property real cylinderHeight: model.height
65                  scale: Qt.vector3d(cylinderRadius/100, cylinderHeight
    /100, cylinderRadius/100)
66                  materials: PrincipledMaterial {
67                      baseColor: "#eeeeee"
68                  }
69              }
70          }
71
72          Ego_car {
73              id:ego
74              position: Qt.vector3d(0,-150,100)
75              eulerRotation: Qt.vector3d(0,180,0)
76          }
77      }
78
79      View3D {
```

```
80          id: view
81          anchors.fill: parent
82          camera: camera
83          renderMode: View3D.Overlay
84
85          environment: SceneEnvironment {
86              id: sceneEnvironment
87              antialiasingMode: SceneEnvironment.MSAA
88              antialiasingQuality: SceneEnvironment.VeryHigh
89          }
90          importScene: sceneRoot
91
92      }
93
94      WasdController {
95          controlledObject: camera
96      }
97  }
```

Bibliography

[1] Abiel Aguilar-González, Miguel Arias-Estrada, and François Berry. «The Fastest Visual Ego-motion Algorithm in the West». In: *Microprocessors and Microsystems: Embedded Hardware Design* (Mar. 2019). DOI: 10.1016/j.micpro.2019.03.005. URL: https://hal.science/hal-02080630 (cit. on p. 5).

[2] Usman Arif, Waleed Razzaq, Azib Akram, and Wasif Muhammad. «Self-Driving Car Control Using Visual Ego-Motion Estimation». In: *2019 15th International Conference on Emerging Technologies (ICET)*. 2019, pp. 1–4. DOI: 10.1109/ICET48972.2019.8994357 (cit. on p. 5).

[3] You Li and Yassine Ruichek. «Occupancy Grid Mapping in Urban Environments from a Moving On-Board Stereo-Vision System». In: *Sensors* 14.6 (2014), pp. 10454–10478. ISSN: 1424-8220. DOI: 10.3390/s140610454. URL: https://www.mdpi.com/1424-8220/14/6/10454 (cit. on pp. 5, 6).

[4] Radu Gabriel Danescu. «Obstacle Detection Using Dynamic Particle-Based Occupancy Grids». In: *2011 International Conference on Digital Image Computing: Techniques and Applications*. 2011, pp. 585–590. DOI: 10.1109/DICTA.2011.104 (cit. on p. 6).

[5] C. Chen, C. Tay, C. Laugier, and Kamel Mekhnacha. «Dynamic Environment Modeling with Gridmap: A Multiple-Object Tracking Application». In: *2006 9th International Conference on Control, Automation, Robotics and Vision*. 2006, pp. 1–6. DOI: 10.1109/ICARCV.2006.345399 (cit. on p. 6).

[6] Sang-Il Oh and Hang-Bong Kang. «Fast Occupancy Grid Filtering Using Grid Cell Clusters From LIDAR and Stereo Vision Sensor Data». In: *IEEE Sensors Journal* 16.19 (2016), pp. 7258–7266. DOI: 10.1109/JSEN.2016.2598600 (cit. on p. 6).

[7] Çagan Önen, Ashish Pandharipande, Geethu Joseph, and Nitin Jonathan Myers. «LiDAR-Based Occupancy Grid Map Estimation Exploiting Spatial Sparsity». In: *2023 IEEE SENSORS, Vienna, Austria, October 29 - Nov. 1, 2023*. IEEE, 2023, pp. 1–4. ISBN: 979-8-3503-0387-2. DOI: 10.1109/SENSOR

S56945.2023.10325050. URL: https://doi.org/10.1109/SENSORS56945.2023.10325050 (cit. on p. 6).

[8] Ting Yuan, Dominik S. Nuss, Gunther Krehl, Michael Maile, and Axel Gern. «Fundamental properties of dynamic occupancy grid systems for vehicle environment perception». In: *2015 IEEE 6th International Workshop on Computational Advances in Multi-Sensor Adaptive Processing (CAMSAP)*. 2015, pp. 153–156. DOI: 10.1109/CAMSAP.2015.7383759 (cit. on p. 6).

[9] Robert Prophet, Henriette Stark, Marcel Hoffmann, Christian Sturm, and Martin Vossiek. «Adaptions for Automotive Radar Based Occupancy Gridmaps». In: *2018 IEEE MTT-S International Conference on Microwaves for Intelligent Mobility (ICMIM)* (2018), pp. 1–4. URL: https://api.semanticscholar.org/CorpusID:52129554 (cit. on p. 6).

[10] Klaudius Werber, Matthias Rapp, Jens Klappstein, Markus Hahn, Jürgen Dickmann, Klaus Dietmayer, and Christian Waldschmidt. «Automotive radar gridmap representations». In: *2015 IEEE MTT-S International Conference on Microwaves for Intelligent Mobility (ICMIM)*. 2015, pp. 1–4. DOI: 10.1109/ICMIM.2015.7117922 (cit. on p. 6).

[11] Renaud Dubé, Markus Hahn, Markus Schütz, Jürgen Dickmann, and Denis Gingras. «Detection of parked vehicles from a radar based occupancy grid». In: *2014 IEEE Intelligent Vehicles Symposium Proceedings*. 2014, pp. 1415–1420. DOI: 10.1109/IVS.2014.6856568 (cit. on p. 6).

[12] Frederik Sarholz, Jens Mehnert, Jens Klappstein, Juergen Dickmann, and Bernd Radig. «Evaluation of different approaches for road course estimation using imaging radar». In: *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2011, pp. 4587–4592. DOI: 10.1109/IROS.2011.6094623 (cit. on p. 6).

[13] Liat Sless, Gilad Cohen, Bat El Shlomo, and Shaul Oron. «Road Scene Understanding by Occupancy Grid Learning from Sparse Radar Clusters using Semantic Segmentation». In: *CoRR* abs/1904.00415 (2019). arXiv: 1904.00415. URL: http://arxiv.org/abs/1904.00415 (cit. on p. 6).

[14] Xu Jie and Zha Hongbin. «3D object surface meshing based on occupancy grids». In: *2012 IEEE International Conference on Computer Science and Automation Engineering (CSAE)*. Vol. 2. 2012, pp. 572–576. DOI: 10.1109/CSAE.2012.6272837 (cit. on p. 6).

[15] Hongbo Gao, Bo Cheng, Jianqiang Wang, Keqiang Li, Jianhui Zhao, and Deyi Li. «Object Classification Using CNN-Based Fusion of Vision and LIDAR in Autonomous Vehicle Environment». In: *IEEE Transactions on Industrial Informatics* 14.9 (2018), pp. 4224–4231. DOI: 10.1109/TII.2018.2822828 (cit. on p. 7).

[16] K. S. Arikumar, A. Deepak Kumar, Thippa Reddy Gadekallu, Sahaya Beni Prathiba, and K. Tamilarasi. «Real-Time 3D Object Detection and Classification in Autonomous Driving Environment Using 3D LiDAR and Camera Sensors». In: *Electronics* 11.24 (2022). ISSN: 2079-9292. DOI: 10.3390/electronics11244203. URL: https://www.mdpi.com/2079-9292/11/24/4203 (cit. on p. 7).

[17] Yiming Zeng, Yu Hu, Shice Liu, Jing Ye, Yinhe Han, Xiaowei Li, and Ninghui Sun. «RT3D: Real-Time 3-D Vehicle Detection in LiDAR Point Cloud for Autonomous Driving». In: *IEEE Robotics and Automation Letters* 3.4 (2018), pp. 3434–3440. DOI: 10.1109/LRA.2018.2852843 (cit. on p. 7).

[18] Yu Peng, Jesse S. Jin, Suhuai Luo, Min Xu, and Yue Cui. «Vehicle Type Classification Using PCA with Self-Clustering». In: *2012 IEEE International Conference on Multimedia and Expo Workshops*. 2012, pp. 384–389. DOI: 10.1109/ICMEW.2012.73 (cit. on p. 7).

[19] Fei Liu, Zihao Lu, and Xianke Lin. *Vision-Based Environmental Perception for Autonomous Driving*. 2022. arXiv: 2212.11453 [cs.CV] (cit. on p. 7).

[20] *StereoLabs ZED 2*. URL: https://www.stereolabs.com/products/zed-2 (cit. on p. 9).

[21] *ZED 2 Datasheet*. URL: https://cdn.sanity.io/files/s18ewfw4/production/105d20a6752c5e4da3700b13d270db6bb4d5b9ba.pdf/ZED%202%20Datasheet%20Mars%202023.pdf (cit. on p. 9).

[22] Demetria Ma, Alexander Tran, Nick Keti, Ryan Yanagi, Peter Knight, Kedar Joglekar, Nicholas Tudor, Burt Cresta, and Subodh Bhandari. «Flight Test Validation of Collision Avoidance System for a Multicopter using Stereoscopic Vision». In: June 2019, pp. 989–995. DOI: 10.1109/ICUAS.2019.8798023 (cit. on p. 11).

[23] Rohit Kundu. *YOLO: Algorithm for Object Detection Explained*. Jan. 2023. URL: https://www.v7labs.com/blog/yolo-object-detection (cit. on pp. 15, 16, 18, 19, 22).

[24] Pragati Baheti. *A Comprehensive Guide to Convolutional Neural Networks*. June 2021. URL: https://www.v7labs.com/blog/convolutional-neural-networks-guide (cit. on pp. 16, 17).

[25] Nidhi Sahai. *Convolutional Neural Networks – Definition, Architecture, Types, Applications, and more*. Jan. 2024. URL: https://www.analytixlabs.co.in/blog/convolutional-neural-network/ (cit. on p. 17).

[26] Rahul Chauhan, Kamal Kumar Ghanshala, and R.C Joshi. «Convolutional Neural Network (CNN) for Image Detection and Recognition». In: *2018 First International Conference on Secure Cyber Computing and Communication (ICSCCC)*. 2018, pp. 278–282. DOI: 10.1109/ICSCCC.2018.8703316 (cit. on p. 18).

[27] Mujtaba Raza. *Yolo V8: A Deep Dive Into Its Advanced Functions and New Features*. Oct. 2023. URL: https://medium.com/@mujtabaraza194/yolo-v8-a-deep-dive-into-its-advanced-functions-and-new-features-f008599fe604 (cit. on p. 21).

[28] Rohith Ganghi. *R-CNN, Fast R-CNN, Faster R-CNN, YOLO — Object Detection Algorithms*. July 2018. URL: https://towardsdatascience.com/r-cnn-fast-r-cnn-faster-r-cnn-yolo-object-detection-algorithms-36d53571365e (cit. on p. 23).

[29] StereoLabs. *StereoLabs Documentation*. URL: https://www.stereolabs.com/docs (cit. on pp. 26–30).

[30] The Qt Company. *Qt 3D Overview*. 2024. URL: https://doc.qt.io/qt-6/qt3d-overview.html (cit. on p. 39).

[31] Sekilab. *Vehicle Orientation Dataset*. URL: https://github.com/sekilab/VehicleOrientationDataset (cit. on pp. 52, 53).

[32] Ashutosh Kumar, Takehiro Kashiyama, Hiroya Maeda, Hiroshi Omata, and Yoshihide Sekimoto. «Real-time citywide reconstruction of traffic flow from moving cameras on lightweight edge devices». In: *ISPRS Journal of Photogrammetry and Remote Sensing* 192 (2022), pp. 115–129 (cit. on p. 52).

[33] Ashutosh Kumar, Takehiro Kashiyama, Hiroya Maeda, and Yoshihide Sekimoto. «Citywide reconstruction of cross-sectional traffic flow from moving camera videos». In: *2021 IEEE International Conference on Big Data (Big Data)*. IEEE. 2021, pp. 1670–1678 (cit. on p. 52).

[34] Andy Catruna, Pavel Betiu, Emanuel Tertes, Vladimir Ghita, Emilian Radoi, Irina Mocanu, and Mihai Dascalu. «Car Full View Dataset: Fine-Grained Predictions of Car Orientation from Images». In: *Electronics* 12.24 (2023). ISSN: 2079-9292. DOI: 10.3390/electronics12244947. URL: https://www.mdpi.com/2079-9292/12/24/4947 (cit. on p. 53).