HOCHSCHULE MITTWEIDA
University of Applied Sciences

Faculty of **Applied Computer Sciences and Biosciences**

THESIS

Optimization of Neural Networks for Autonomous Driving Applications Using Synthetic Data Generation

Mr.
Kedharnath Kuruba Basvaraj

Course of Study:
Applied Mathematics for Network and Data Sciences

Seminar Group:
MA21w1-M

E-Mail:
kkurubab@hs-mittweida.de

Supervisor:
Prof. Dr. Ing. Alexander Lampe
M.Sc. Thomas Davies

Mittweida, 14. September 2024

Contents

Additional Lists

List of Figures

List of Tables

Listings

Acronyms

AD . . . . . Autonomous driving

Adam . . . . . Adaptive Moment Estimation

AdamW . . . . . Adaptive Moment Estimation with Weight Decay

ADAS . . . . . Advanced Driver Assistance Systems

AI . . . . . Artificial Intelligence

AV . . . . . Autonomous Vehicle

CNN . . . . . Convolutional Neural Network

FOV . . . . . Field of View

FSOCO . . . . . Formula Student Objects in Context

IMU . . . . . Inertial Measurement Unit

IOU . . . . . Intersection over Union

LiDAR . . . . . Light Detection and Ranging

NMS . . . . . Non-Maximum Suppression

ODE . . . . . Open Dynamics Engine

RGB-D . . . . . Red, Green, Blue with Depth channel

RGB-D-M . . . . . Red, Green, Blue with Depth and Mask channels

ROS . . . . . Robot Operating System

SAV . . . . . Shared Autonomous Vehicle

SDF . . . . . Simulation Description File

SGD . . . . . Stochastic Gradient Descent

URDF . . . . . Unified Robot Description Format

WBF . . . . . Weighted Boxes Fusion

XACRO . . . . . XML Macro

YOLOv8 . . . . . You Only Look Once v8

Acknowledgment

I would like to express my heartfelt gratitude to Prof. Alexander Lampe for providing the thesis topic. His unwavering support and guidance have been invaluable, and his trust in my abilities has been a significant source of motivation, pushing me to exceed my limits. I also wish to extend my thanks to Mr.Thomas Davies for his guidance and feedback throughout the thesis. Finally, I am deeply grateful to my friends and family for their constant support and to Mittweida University for the opportunity to pursue my master's degree.

Abstract

The development of Autonomous Vehicles (AVs) holds significant potential to transform society by reducing accidents, improving travel efficiency, and contributing to environmental sustainability. However, the advancement of AV technology is constrained by the challenges associated with generating large quantities of high-quality, annotated real-world data, which are costly and time-consuming to produce. This thesis addresses these challenges by exploring methods for synthesizing high-quality training data using simulation software, specifically focusing on automating the annotation process. It investigates the use of early fusion and late fusion approaches to optimize neural networks trained on synthetic data, aiming to enhance their generalization to real-world scenarios. The research leverages the Gazebo simulation tool to generate synthetic data, including RGB images and Light Detection and Ranging (LiDAR) point clouds, and employs automated labeling techniques based on transformation, rotation, and perspective projection. During the thesis, a cone detection model was built and trained on synthetic images and then tested on real images to evaluate its performance. A particular focus is placed on training the Red, Green, Blue with Depth channel (RGB-D) model using the early fusion technique, with a modified You Only Look Once v8 (YOLOv8) architecture, specifically its Nano and Small variants. Evaluations were conducted on real images from the Formula Student Objects in Context (FSOCO) dataset, and with each iteration, improvements in the model's predictions and its ability to generalize to real images were demonstrated. The thesis specifically provides metrics for foreground and background cone classification, illustrating the effectiveness of the approach.

1 Introduction

1.1 Motivation and Background

In recent times, the development of AVs has accelerated, impacting society in numerous ways, including economic, environmental, and social aspects. AVs are expected to significantly reduce the number of accidents; according to the World Health Organization, 1.35 million people die worldwide each year due to accidents [1]. AVs will also improve travel times and reduce congestion. According to the INRIX Global Traffic Scorecard, citizens in the US lost an average of 97 hours annually due to congestion [2]. Furthermore, Autonomous driving (AD) technology is projected to enable a new Passenger Economy worth $7 trillion by 2050 [3]. AVs could also significantly improve air quality, reducing CO_2 emissions by up to 8.2% and NO_x emissions by up to 8.9%, thus contributing to a healthier environment [4].

Despite these benefits and recent advancements in generating large quantities of high-quality data, real-world datasets face challenges in keeping up with evolving requirements due to their costly and time-intensive experimental and labeling processes. The development of AD systems relies heavily on the ability to replicate complex and diverse traffic scenarios. Consequently, more research is being directed toward methods for generating high-quality synthetic data using simulation tools or game engines. These tools help create environments, objects, and material properties that closely mimic the real world, collecting data through Cameras, LiDAR, and Radar sensors to approximate real-world situations. AVs, which heavily rely on machine learning algorithms to perceive and respond to their environment, can be optimized to perform on synthetic data that can be produced and scaled at a fraction of the cost and time required for collecting, annotating, and processing real-world datasets, which often involve expensive equipment, labor-intensive labeling, and lengthy experimental setups.

1.2 Problem Statement

In real-time scenarios, recreating the diverse and dynamic situations necessary for capturing comprehensive real-world data is often challenging. This difficulty extends to generating annotated training data for AV research, which traditionally relies on labor-intensive manual annotation of real-world images, consuming significant time and resources. Simulation software, such as Gazebo, offers a promising solution by enabling the efficient generation of synthetic data, including sensor data like point clouds and RGB imagery, which can be annotated using the concepts of transformation, rotation, and perspective projection. However, despite advancements, optimizing neural networks to learn effectively from synthetic data and generalize to real-world scenarios remains challenging.

This thesis aims to address these challenges by investigating methods for synthesizing high-quality training data using simulation software, with a focus on automating the process of annotation. Additionally, it aims to optimize neural networks for training on this synthetic

data through early fusion and late fusion approaches, evaluating their performance on real images from the FSOCO dataset [5]. By exploring both data generation techniques and neural network optimization strategies, this thesis seeks to advance our understanding of leveraging synthetic data to enhance the development and deployment of AD systems.

1.3 Thesis Structure

The structure of this thesis is divided into seven chapters. After the introductory chapter, the remainder of the thesis is organized as follows:

Chapter 2: Provides an overview of AVs and their associated challenges. It emphasizes the critical role of data in AVs development, explores various data annotation techniques, and discusses tools related to simulation, 3D creation, and ROS. The chapter concludes with an examination of sensor fusion techniques.

Chapter 3: Introduces fundamental Machine Learning concepts essential for comprehending the implementation of object detection models.

Chapter 4: Details the simulation environment employed, the sensors integrated into the vehicle, and the process of generating RGB-D and LiDAR images. It also covers the automated label generation process.

Chapter 5: Covers the implementation of early fusion, including multiple training sessions and evaluations of the RGB-D model, with a focus on classification metrics for both foreground and background cones.

Chapter 6: Explores the implementation of late fusion, detailing the development of both the RGB and Depth models. It also focuses on the Weighted Boxes Fusion (WBF) ensemble method to achieve the final prediction.

Chapter 7: Concludes the thesis by summarizing the synthesis of training data, automation of annotation, and optimization of neural networks for improved generalization from synthetic to real-world scenarios, along with key findings and possible improvements.

2 Overview of Data Annotation, Simulation Tools, and Fusion Techniques

2.1 Overview of AVs

The development of automated driving technology dates back to the early 20th century, initially emphasizing autonomous control of speed, braking, lane-keeping, and other basic cruise control functions. Vehicle automation was first envisioned in 1918 [6], with General Motors showcasing the initial concept in 1939. Research and development began in the 1950s through a collaboration between General Motors and the Radio Corporation of America Sarnoff Laboratory. However, it wasn't until the past decade, fueled by the advancements of the Digital and 4th Industrial Revolutions, that significant technological progress was made, resulting in numerous prototype AVs being tested on the roads. Companies like Volvo, Google, Tesla, Audi, BMW, Mercedes-Benz, and Nissan have been actively developing AVs, with many already introducing models with self-driving capabilities and others planning to bring fully AVs to market soon [7] [8].

AVs hold significant importance for both private and commercial transportation, offering transformative benefits. They promise enhanced flexibility by enabling shared use among families and can revolutionize commercial services such as taxis, buses, and freight. Shared Autonomous Vehicles (SAVs) [9] are expected to blend car-sharing and taxi services, potentially reducing costs and improving efficiency. Unlike traditional taxis, SAVs with centralized control can optimize operations, cut empty travel costs, and improve service levels. They also support dynamic ridesharing and the concept of Mobility-as-a-Service [10]. Furthermore, AVs could alleviate parking demand in urban areas, potentially increasing urban density, while also possibly contributing to urban sprawl. The integration of platooning features in freight and buses could further enhance road capacity. Overall, AVs are poised to significantly impact transportation efficiency, urban planning, and economic activity.

The driving software platforms sector is experiencing significant expansion, with an anticipated annual growth rate of 43.2% from 2023 to 2030. This growth is expected to result in a market value of $109.8 billion [11]. These software platforms are crucial for enabling vehicles to navigate and interpret their environments, utilizing technologies such as Artificial Intelligence (AI), sensor fusion, and machine learning. The popularity of subscription-based services for AVs is increasing, driven by consumer interest and readiness to pay for extra features. According to McKinsey's 2021 ACES survey [12], which included over 25,000 participants, approximately 25% of respondents are likely to opt for advanced AD features in their next vehicle purchase. Of these interested buyers, two-thirds are willing to pay $10,000 or a comparable subscription fee for an L4 highway pilot, highlighting significant market potential.

2.2 Challenges of AVs

The challenges of AVs with respect to neural networks are multifaceted and have been extensively discussed in the literature [13] [14]:

1. **High Accuracy Requirement**: AV systems must achieve extremely high detection and prediction accuracy, performing reliably under all environmental conditions.
2. **Complexity of Autonomous Driving Systems**: AD systems involve interconnected decision-making processes where the output of one stage serves as the input for the next. Despite improvements in certain areas, the performance of the entire AD system is dependent on the integration of its components. Balancing energy-intensive controllers with efficient motion planners remains a significant challenge.
3. **Dynamic Road Environment**: Modern cities are increasingly dynamic due to digital advertisements and changing lighting conditions. Although various sensor-based solutions, such as Radar, vision, and lasers, have been proposed, maintaining high accuracy in such environments is challenging. The growing number of personal luxury vehicles further complicates detection, tracking, and recognition tasks for AVs due to their distinct designs, reflective surfaces, and varied sensor signatures, which differ from more common vehicle models, making accurate identification more difficult.
4. **Big Data and Real-Time Processing**: AVs continuously generate vast amounts of data through sensors, cameras, and LiDAR, necessitating real-time processing. Managing this big data while ensuring accuracy, power efficiency, and cost-effectiveness poses a significant challenge.
5. **Scale and Computational Demand**: Training neural networks for AVs requires immense computational power and vast datasets, necessitating significant resources for data capture, storage, processing, and human annotation.
6. **Intelligent Data Prioritization**: With the enormous volume of data captured by AVs, it is impractical to process everything. Intelligent data prioritization mechanisms are needed to filter and process only the most relevant information, discarding unnecessary data.
7. **Robustness and Adaptability**: AV systems must be robust and adaptable to various environmental conditions. Many AI techniques are trained on data from specific environments and struggle with cross-weather reliability. Adapting to unpredictable conditions like snow, rain, and fog is inherently challenging.
8. **Integration/Fusion of Sensory Data for Dynamic Decision Making**: Achieving optimal performance with a single sensor is difficult; hence, AVs rely on data fusion from multiple sensors. The fusion of environment perception and localization data, through methods like machine learning and Kalman filters, aims to improve accuracy. However, practical implementation and lightweight, robust fusion frameworks remain a challenge.
9. **Safety**: Neural networks in AVs are susceptible to adversarial perturbations, where minor changes in input data can lead to incorrect classifications. Current safety standards do not fully address the safety of self-learning algorithms, making standardization challenging.
10. **Debugging and Interpretation**: The complexity of neural network models makes it difficult to interpret and debug them when failures occur, emphasizing the need for thorough and accurate training data.

2.3 Annotated Training Data for AVs

AVs rely on deep neural networks trained with extensive labeled datasets, making precise data annotation essential for their development. These vehicles use various sensors, including cameras, LiDAR, Radar, and ultrasonic sensors, to continuously capture and process data about their surroundings, enabling informed navigation decisions. Modern systems typically feature 15-20 sensors for comprehensive environmental perception [15].

Data annotation involves labeling raw data to provide meaningful context, which is crucial for training AI models to improve their accuracy and decision-making. Manual annotation creates the labeled datasets needed for developing and validating autonomous systems. It ensures that raw sensor data is structured, supports supervised learning, enhances model accuracy, and allows testing against verified real-world data. Reliable self-driving technology fundamentally depends on large volumes of annotated data.

2.3.1 How Data Annotation Helps AVs

High-quality data annotation is essential for the development of accurate AI models and robust self-driving capabilities. Below are key areas where data annotation plays a crucial role that are adapted from [16]:

- **Navigating the Road with Autonomous Systems**: Data annotation supports the creation of multi-frame sensor data from cameras, LiDAR, Radar, and other sensors. This annotated data trains sensory fusion algorithms to construct a 3D representation of the environment, which enhances perception under various weather and lighting conditions. It also facilitates semantic segmentation of road features such as lane markings and curbs, enabling vehicles to stay within lanes and follow the correct routes.
- **Analyzing Driving Scenes**: Advanced Driver Assistance Systems (ADAS) systems utilize annotated data to analyze driving scenes and detect potential hazards. Bounding boxes for pedestrians, cyclists, and other threats train ADAS algorithms to identify these hazards promptly. Annotated lane markings and road edges enable lane departure warnings and blind spot alerts, improving overall driving safety.
- **Recognizing Road Signs and Traffic Lights**: Annotated images of road signs and traffic lights are critical for compliance with traffic rules. These annotations help train AI models to accurately detect and respond to stop signs, speed limits, and traffic lights, ensuring appropriate actions such as stopping at red lights or adjusting speed according to the limit.
- **Monitoring Driver Focus**: In semi-autonomous systems, monitoring driver attentiveness is crucial. Annotated data of driver eye movements and blinking patterns train models to detect signs of drowsiness or distraction, triggering alerts to maintain driver engagement.
- **Predicting Accidents**: Annotated data aids AI models in predicting potential hazards and preventing accidents. By analyzing surroundings and tracking movements, these models can anticipate collisions and take preemptive actions like braking or changing lanes to enhance road safety.

In summary, data annotation is fundamental to advancing AV capabilities, enabling everything from navigation and hazard detection to identity verification and accident prediction.

2.3.2 Data Annotation Techniques

Manual Annotation

Manual data annotation refers to the process of manually labeling or tagging data by human annotators. This involves reviewing and classifying data according to predefined criteria or labels, and it is crucial for creating high-quality training datasets for machine learning and AI models.

Advantages

- *High Accuracy:* Human annotators can understand complex contexts, nuances, and subtleties that automated systems may miss.
- *Flexibility:* Human annotators can adapt to a wide range of annotation tasks and can handle data that might be challenging for automated tools.

Disadvantages

- *Time-Consuming:* It can be slow and labor-intensive, particularly for large datasets.
- *Costly:* Requires significant human resources, which can be expensive.
- *Consistency Challenges:* Ensuring consistency across different annotators can be difficult, especially in large-scale projects.

Semi-Automated Annotation

Semi-automated annotation combines automation with human input to enhance data analysis, prediction, and labeling processes. This approach is especially valuable for complex tasks where automated methods require expert oversight. The process typically involves:

- **Automation**: Utilizing algorithms or machine learning models to perform initial data labeling or predictions.
- **Human Input**: Engaging human annotators to review, refine, and validate the results produced by automated systems.

Semi-automated annotation improves efficiency and accuracy by leveraging the speed of automated tools while incorporating the nuanced understanding of human experts.

Advantages

- *Increased Efficiency:* Automates the initial stages of annotation, reducing the overall time and effort required compared to fully manual processes.

- *Enhanced Accuracy:* Combines the precision of automated tools with the nuanced judgment of human experts, improving overall annotation quality.
- *Scalability:* Facilitates handling large datasets by automating repetitive tasks while utilizing human input for complex cases.
- *Flexibility:* Adaptable to various types of data and tasks, allowing for adjustments and improvements based on human feedback.

Disadvantages

- *Initial Setup Costs:* Requires investment in developing or acquiring automated tools and setting up systems for human review.
- *Complexity in Integration:* Combining automated processes with human input can be complex and may require careful management and coordination.
- *Quality Control Challenges:* Ensuring consistency and accuracy can be challenging, as the effectiveness of the process depends on both the automated system and human reviewers.
- *Dependence on Expertise:* Requires skilled human annotators to review and refine automated outputs, which may not be readily available or may incur additional costs.

Fully-Automated Annotation

Fully automated annotation refers to the process of using algorithms and machine learning models to label or tag data without human intervention. This approach relies entirely on computational methods to perform tasks such as data labeling, prediction, or classification. It is typically used in scenarios where large volumes of data need to be processed quickly and efficiently. The process involves:

- **Algorithms and Models**: Utilizing pre-trained machine learning models or custom algorithms to automatically generate labels or predictions for the data.
- **Automation**: Conducting the entire annotation process without manual input, relying solely on automated systems.

Fully automated annotation offers significant efficiency and scalability benefits, but may require careful management to ensure accuracy and effectiveness.

Advantages

- *High Efficiency:* This process handles large volumes of data quickly, reducing the time required compared to manual or semi-automated methods.
- *Cost-Effective:* Reduces the need for human labor, potentially lowering costs associated with data annotation.
- *Consistency:* Ensures uniformity in labeling, as the same algorithm or model applies consistent rules across the entire dataset.
- *Scalability:* Easily handles expanding datasets or increasing volumes of data without a proportional increase in resources.

Disadvantages

- *Lower Accuracy:* May struggle with complex or ambiguous cases that require human judgment, leading to potential inaccuracies.
- *Limited Adaptability:* Algorithms may not adapt well to new or evolving data patterns without retraining or updates.
- *Dependence on Model Quality:* The effectiveness of annotation is highly dependent on the quality and performance of the underlying models or algorithms.
- *Lack of Nuance:* Automated systems may miss subtle details or context that a human annotator might catch, potentially affecting the overall quality of annotations.

Fully Automated Annotation Using Simulation Tools

Fully automated annotation using simulation tools involves leveraging simulated environments and scenarios to automatically generate and label data. This approach uses computer-generated simulations to produce data and apply labels without human intervention. It is particularly useful for creating large datasets for training machine learning models where real-world data might be limited or difficult to obtain. The process typically includes:

- **Simulation Environments**: Using virtual environments to create realistic data, such as simulated images, videos, or sensor data.
- **Automated Labeling**: Applying labels automatically based on the parameters and rules defined within the simulation environment.

Advantages

- *High Efficiency:* Quickly generates large volumes of data and labels, which accelerates the training process for machine learning models.
- *Controlled Data Generation:* Allows for precise control over data characteristics and conditions, including rare or challenging scenarios that might be hard to capture in real-world data.
- *Cost-Effective:* Reduces the need for expensive data collection and manual annotation processes by automating both data generation and labeling.
- *Scalability:* Easily scales to produce vast amounts of data, accommodating large-scale machine learning projects.

Disadvantages

- *Model Dependence:* The quality of the generated data and labels depends on the accuracy and fidelity of the simulation models. Poorly designed simulations can lead to unrealistic or biased data.
- *Lack of Real-World Nuance:* Simulated data may not fully capture the complexities and variability of real-world data, which can limit the effectiveness of models trained exclusively on synthetic data.

- *Validation Challenges:* Verifying that the simulated data accurately reflects real-world conditions can be difficult, requiring additional steps to ensure its relevance and reliability.
- *Potential for Overfitting:* Models trained primarily on synthetic data might overfit to the specific characteristics of the simulation, potentially reducing their performance on real-world data.

2.3.3 Open-source Autonomous Driving Annotated Datasets

- **KITTI**: The KITTI dataset [17], created by the Karlsruhe Institute of Technology and Toyota Technological Institute in Chicago, offers over 15,000 annotated frames captured in Karlsruhe, Germany. It includes stereo camera images, LiDAR scans, GPS/Inertial Measurement Unit (IMU) data, and object bounding boxes.
- **Berkeley DeepDrive BDD100k**: The BDD100K dataset [18], developed by UC Berkeley, comprises over 100,000 diverse driving images annotated in detail. With more than one million high-quality annotations, it captures varied US environments and conditions.
- **Cityscapes**: The Cityscapes dataset [19], produced by MMCV at TU Darmstadt, includes 5,000 meticulously annotated images and videos of urban street scenes from 50 German cities. The dataset provides high-quality pixel-level annotations, ideal for training semantic segmentation models.
- **Waymo Open Dataset**: Released in 2019, the Waymo Open Dataset [20] offers extensive sensor data from Waymo's AVs. It includes LiDAR point clouds, camera images, and labels across a wide range of environments, weather conditions, and driving scenarios.
- **NuScenes**: The NuScenes dataset [21], developed by Motional, features 1,000 driving scenes from Boston and Singapore with multimodal sensor data, including LiDAR, Radar, camera, IMU, and GPS. It supports the development of 3D object detection and tracking models.
- **FSOCO Dataset**: The FSOCO dataset [5] includes images that have been manually labeled by the Formula Student Driverless community. It offers accurate data for detecting cones and supports both bounding box annotations and instance segmentation.

In this thesis, FSOCO dataset is used specifically for validation and inference of real RGB images for cone detection model.

2.4 Tools for Robotics, Simulation, and 3D Creation

2.4.1 Gazebo

Gazebo [22] [23] [24] is a powerful, open-source 3D simulation tool designed for modeling and testing robotic systems. It supports a wide range of applications from simple robot models to complex multi-robot environments. Gazebo operates across multiple platforms, including Linux and Windows, and can recreate both indoor and outdoor environments with high fidelity. The simulated environment using Gazebo is illustrated in Fig. 2.1a

Key Features

- **High-Fidelity Simulation:** Gazebo provides a realistic 3D environment where users can simulate complex robotic scenarios. It includes realistic rendering of objects, environments, and sensors, enabling accurate representation of real-world conditions.
- **Advanced Physics Engines:** The simulator incorporates powerful physics engines, such as Open Dynamics Engine (ODE), Bullet, Simbody, and DART. These engines allow for accurate simulation of real-world physics, including gravity, friction, and collision detection.
- **Comprehensive Sensor Simulation:** Gazebo can simulate a variety of sensors used in robotics, including RGB and Depth cameras, LiDAR, Radar, GPS, IMU, and contact sensors. This feature supports the collection of synthetic data that closely mimics real-world sensor outputs.
- **Flexible Robot Modeling:** Users can create and manipulate robot models using Simulation Description File (SDF) or Unified Robot Description Format. Gazebo provides a library of pre-built robot models and supports custom model creation.
- **Integration with ROS:** Gazebo is tightly integrated with the Robot Operating System (ROS), facilitating seamless control of robots through ROS nodes. This integration enhances the development and testing of robotics applications by leveraging ROS's extensive framework.
- **Plugin Support:** Gazebo supports plugins that extend its functionality. Users can develop plugins to interact with the core libraries—physics, rendering, and communication—allowing for customized simulation scenarios and controls.

Core Components

Gazebo is built around three main libraries:

- **Physics Library:** Simulates realistic object behavior by defining physical properties such as mass, friction coefficient, velocity, and inertia. The default physics engine is ODE, but users can choose from other engines like Bullet, Simbody, and DART.
- **Rendering Library:** Utilizes the Object-Oriented Graphics Rendering Engine for high-quality visualization of dynamic 3D objects and scenes.
- **Communication Library:** Manages communication between various components of simulation, enabling interactions and data exchange between different elements of Gazebo.

Simulation Elements

In Gazebo, two core elements define any 3D scene:

- **World:** Represents a 3D environment, either indoor or outdoor, and is described in a SDF format. The world file includes one or more models and defines the overall scene.

- **Model:** Refers to any 3D object within the world, such as a static table or a dynamic robot. Models are defined with visual, inertial, and collision properties in SDF format. Users can create models from scratch or use existing models from the community.

Gazebo also supports plugins for additional control, such as world plugins for world properties and model plugins for model properties.

Community and Support

Gazebo benefits from a wide community of users who contribute models and share knowledge. The tool is well-documented with numerous tutorials, making it accessible for both new and experienced users. Although Gazebo operates as a standalone simulator, it is commonly used in conjunction with ROS to model a wide variety of robots and complex scenarios.

2.4.2 Simulink

Simulink [22] is a graphical programming environment within MATLAB designed for modeling, simulating, and analyzing dynamic systems across multiple domains. It features a block diagramming interface and customizable block libraries and integrates seamlessly with MATLAB. Simulink is commonly used for automatic control, digital signal processing, and model-based design [25]. The simulated environment using Simulink is illustrated in Fig. 2.1b

MATLAB/Simulink offers the Automated Driving Toolbox™, which provides specialized tools for the design, simulation, and testing of ADAS and automated driving systems. Key features of this toolbox include:

- **Core Functionalities Testing:** Facilitates the testing of core functionalities such as perception, path planning, and vehicle control.
- **Map Data Integration:** Allows users to import HERE HD live map data [26] and OpenDRIVE® road networks [27] into MATLAB for design and testing purposes.
- **3D Scenario Building:** Enables the creation of photo-realistic 3D scenarios and modeling of various sensors.
- **Built-in Visualizer:** Provides a built-in visualizer to view live sensor detections and tracks.
- **Automated Labeling:** Features the Ground Truth Labeler app [28] for automating the labeling of objects, which can be used for training or sensor performance evaluation.
- **ADAS Examples:** Includes examples for simulating various ADAS features such as Adaptive Cruise Control, Automatic Emergency Braking, and Automatic Parking Assistance.
- **Hardware In the Loop Testing:** Supports Hardware In the Loop testing and C/C++ code generation, enabling faster prototyping.

2.4.3 CARLA

CARLA [29][22] is an open-source simulator based on the Unreal Engine, designed to advance AD research. Its modular platform and robust API support various applications, such as training perception algorithms and developing driving policies. The simulated environment using CARLA is illustrated in Fig. 2.1c.

Key features of CARLA include:

- **Modular and Flexible Design:** CARLA's architecture is highly modular, allowing users to customize and extend its functionalities. It is developed from scratch using the Unreal Engine, ensuring high-quality simulations.
- **OpenDRIVE Integration:** The simulator utilizes the OpenDRIVE standard to model roads and urban environments, providing detailed and realistic maps for simulation.
- **Customizable API:** The CARLA API, based on Python and C++, allows users to control various aspects of the simulation, including vehicle behavior, environmental conditions, and sensor setups.
- **Client-Server Architecture:** CARLA employs a scalable client-server model where the server handles simulation tasks such as world-state updates, sensor rendering, and physics computations. The client modules manage agent logic, including pedestrians and vehicles, and set up environmental conditions.
- **Realistic Simulation:** The server requires a dedicated GPU for accurate simulation results. CARLA enhances realism with advanced vehicle dynamics, including wheel friction, suspension, and center of mass, as well as automatic placement of traffic lights and stop signs based on OpenDRIVE data.
- **Safety Assurance Module:** CARLA integrates a safety assurance module using the RSS library, which governs vehicle control based on sensor information. This module evaluates various driving scenarios and determines appropriate responses to ensure safety at intersections and junctions.

CARLA's extensive feature set and adaptability make it a powerful tool for researchers and developers working in AD and ADAS.

2.4.4 LGSVL

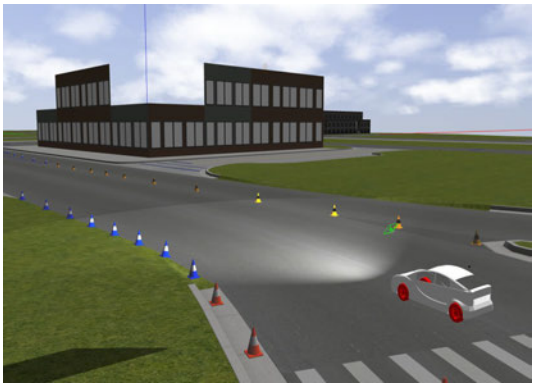
LGSVL [30], developed by LG Electronics America R&D Center, is a versatile open-source AV simulator built on the Unity game engine. It provides a comprehensive platform for testing and validating AV algorithms and integrates seamlessly with various development frameworks. The simulated environment using LGSVL is illustrated in Fig. 2.1d.

Key features of LGSVL include:

- **Integration and Compatibility:** LGSVL supports ROS1, ROS2, and Cyber RT for message passing, facilitating connections with popular AD stacks like Autoware and Baidu Apollo. Multiple simulators can communicate concurrently via ROS and ROS2 bridges, or a customized bridge for Baidu Apollo.

- **Photo-Realistic Environments:** Utilizing Unity's High Definition Render Pipeline, LGSVL creates highly realistic virtual environments for accurate simulation of traffic, physical surroundings, and vehicle dynamics.
- **Customizable Simulation:** The simulator offers a Python API for controlling various environmental entities. It supports a range of sensors, including cameras, LiDAR, IMU, GPS, and Radar, and allows users to define custom sensors via JSON files.
- **Simulation Components:** LGSVL provides functionality for environment simulation (traffic and physical environment), sensor simulation, and vehicle dynamics. It also includes options for segmentation and semantic segmentation.
- **Flexible Integration:** The simulator supports the Functional Mockup Interface for integrating external vehicle dynamics models, enhancing its adaptability to different modeling platforms.
- **Dynamic Environment Features:** Users can specify weather conditions, time of day, traffic agents, and dynamic actors within the 3D environment. Additionally, LGSVL can export HD maps from these environments.

LGSVL's robust feature set and integration capabilities make it an effective tool for developing and testing AV technologies.



(a) Gazebo

(b) Simulink [31]

(c) CARLA [29]

(d) LGSVL [30]

Figure 2.1: Examples of simulated environments used in autonomous vehicle development: (a) Gazebo, (b) Simulink, (c) CARLA, and (d) LGSVL.

2.4.5 Robot Operating System

ROS [32] is an open-source set of libraries and tools for software developers to create robotic applications. It provides essential features such as hardware abstraction, device drivers, software libraries, visualizers, communication mechanisms, and package management.

Core Features of ROS:

- It offers a built-in messaging system that utilizes a publish/subscribe mechanism to enable communication between various distributed nodes.
- Through its publish/subscribe system, ROS supports seamless data capture and replay across processes, facilitating the development of flexible and modular systems.
- ROS provides powerful visualization tools, such as RViz, which allows developers to visualize sensor data, debug robot states, and monitor the robot's environment and behavior in real-time.
- It integrates with simulation tools like Gazebo, enabling developers to test and validate their robotic applications in a simulated environment before deploying them on actual hardware.

2.4.6 Blender

Blender [33] is a comprehensive 3D creation suite that is free and open-source. It covers every aspect of the 3D workflow, including modeling, rigging, animation, simulation, rendering, compositing, motion tracking, video editing, and game development. Key features of Blender include:

- It offers a comprehensive set of modeling tools for creating, transforming, and editing 3D models.
- It has a built-in powerful rendering engine called Cycles, which supports ray tracing, global illumination, and GPU rendering. It also supports real-time rendering through its Eevee engine, suitable for interactive and preview rendering, FreeStyle- an edge and line-based non-photorealistic rendering engine.
- It includes a full-featured animation suite with keyframe animation, rigging, skinning, and inverse kinematics. Blender's animation tools also support simulations like cloth, fluids, and particles.
- It supports various file formats for import and export, making it compatible with other 3D modeling and animation software.

In this thesis, the traditional version of Gazebo is employed to create simulations of environments necessary for generating AD scenarios akin to those in the FSOCO dataset. ROS1 is utilized for communication, facilitating the transfer of information related to simulation states, sensor data, and control commands between Gazebo and other components. Blender is used to design objects, which are then imported into the simulated environment. The workflow, as shown in Fig.2.2, illustrates the process of creating and using 3D models in robotics simulation. It begins with Blender, where a 3D model (such as the orange traffic cone shown)

is created and exported as a .dae file. This model is then imported into Gazebo, which integrates with ROS. The workflow concludes with saving sensor data generated from simulations in Gazebo and processed through ROS. This process enables the creation of realistic virtual driving environments and simulation of vehicle behaviors in various traffic scenarios before real-world deployment on AVs.

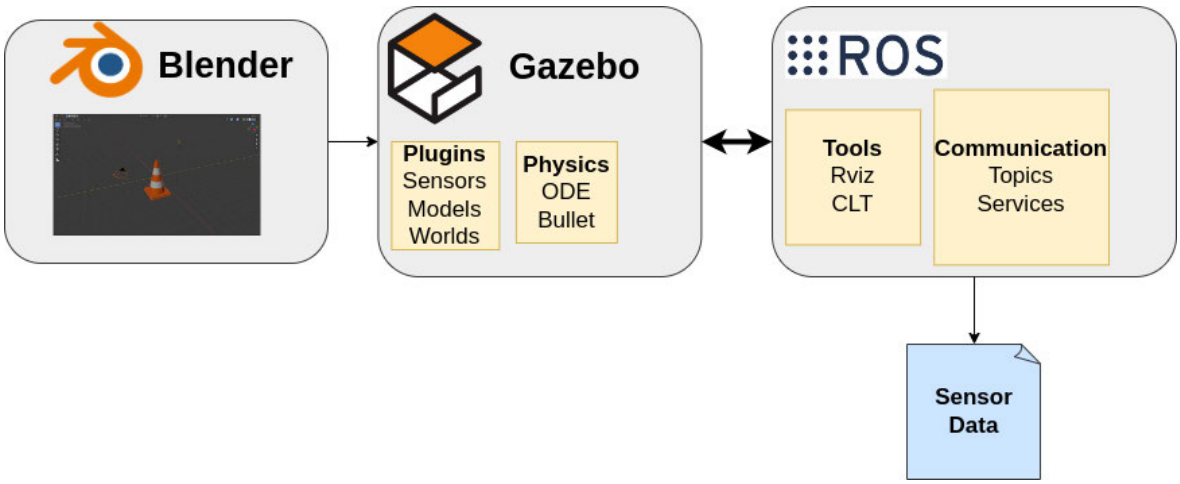


Figure 2.2: Workflow Integration of Blender, Gazebo, and ROS

2.5 Fusion Techniques for RGB-D Data

In autonomous systems and computer vision applications, different sensors are often required to capture various aspects of the environment, each with its own strengths and limitations. RGB cameras excel at capturing color and texture information, while depth sensors provide crucial spatial data. Fusing these complementary data sources can potentially overcome the limitations of individual sensors, leading to more robust and accurate perception systems. This fusion approach is particularly relevant when dealing with RGB-D data, where color images are combined with depth information. To effectively combine these diverse sensor inputs, two main approaches have emerged: early fusion, where data is integrated at the beginning of the processing pipeline, and late fusion, where integration occurs after initial processing of each modality. Fusion techniques for RGB-D data aim to fuse RGB images and Depth maps to enhance the understanding and analysis of scenes and objects.

2.5.1 Early Fusion

Early fusion is a method in multimodal machine learning that involves integrating different types of data (modalities) early in the processing pipeline. This approach merges raw data or features extracted from each modality before extensive processing occurs, creating a unified representation that allows for the exploitation of cross-correlations between modalities, potentially improving performance [34].

Types of Early Fusion

- **Input Fusion:** In this method, as shown in Fig 2.3, raw data from different modalities, such as RGB images and Depth maps, are directly integrated to form a combined input. For instance, RGB and Depth data can be merged into a four-channel input where the channels represent red, green, blue, and depth information respectively. This unified input is then used for further processing in a joint framework [34].
- **Early Feature Fusion:** This method as shown in the Fig.2.4 processes each modality (RGB and Depth) separately through specialized networks to extract low-level features. These features are then combined to form a joint representation, which is used for further processing in tasks like saliency map prediction. This approach leverages initial modality-specific processing while integrating information for enhanced performance in complex tasks [34].

Advantages:

- *Cross-Correlation Exploitation:* By combining modalities early, the system can leverage relationships between different data types, potentially leading to better performance as the integrated representation captures more comprehensive information from the beginning [35].
- *Unified Representation:* Early fusion allows the system to work with a cohesive data structure, which can be advantageous for tasks requiring a holistic view of the data [35].

Challenges:

- *Alignment and Integration:* For effective early fusion, it is crucial to ensure that data from different modalities are properly aligned and merged. Poor alignment can lead to ineffective fusion and reduced performance [35].
- *Feature Relevance:* In some cases, low-level features obtained from early fusion might not be directly relevant to the specific task, potentially decreasing the effectiveness of the fusion approach [35].

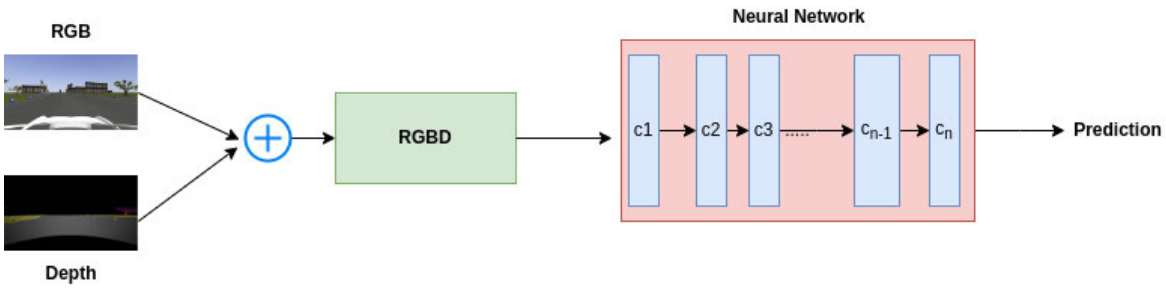


Figure 2.3: Input Fusion

Overall, early fusion provides a powerful mechanism for leveraging multiple types of data from the outset, offering the potential for improved system performance by integrating complementary information. In this thesis, we focus on implementing the input fusion technique to combine the RGB and D to form RGB-D as input to a single model.

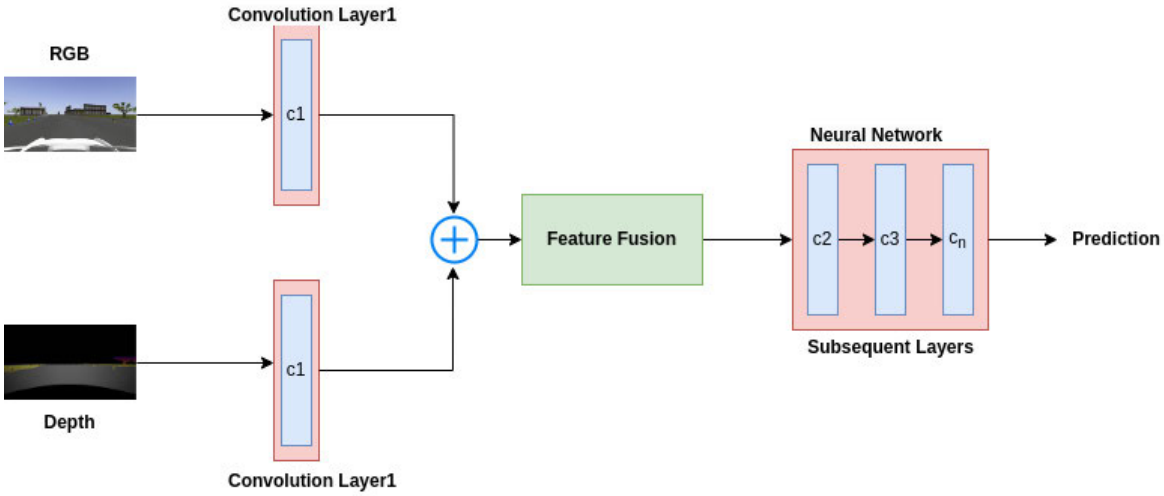


Figure 2.4: Early Feature Fusion

2.5.2 Late Fusion

Late fusion is a method in multimodal machine learning where data from different modalities are processed separately through distinct pipelines before being merged at a later stage. This approach typically involves full processing of each modality individually, followed by a final integration of the processed outputs. Late fusion allows each modality to be analyzed with specialized methods tailored to its specific characteristics, but it has limitations in utilizing cross-correlations between modalities.

Types of Late Fusion

- **Late Feature Fusion:** In this approach as shown in the Fig.2.5, two parallel network streams learn high-level features separately from RGB and depth data. These features are then concatenated and passed through additional layers to generate the final prediction. This method allows the model to integrate information from both modalities effectively, enhancing the accuracy of the prediction. [34].
- **Result Fusion:** In this approach as shown in the Fig.2.6, each modality is processed independently to produce separate predictions. These individual predictions are then combined to generate a final, unified prediction. This method allows for the integration of outcomes from different modalities, improving the overall accuracy of the final result [34].

Advantages:

- *Specialized Processing:* Each modality can be processed using methods specifically designed for its characteristics, which can enhance the performance of the individual processing steps[35].
- *Flexibility in Fusion Operations:* Various fusion operations, such as summation, averaging, or concatenation, can be employed to combine results, allowing for flexibility in the final integration process[35].

Challenges:

- *Limited Cross-Correlation Utilization:* Late fusion has a major drawback in that it does not effectively exploit cross-correlations between different modalities, potentially missing out on complementary information that could enhance performance [35].
- *Integration Complexity:* The process of merging independently processed data can become complex, particularly when dealing with large or diverse modalities, which may affect the overall system performance [35].

Overall, the late fusion method allows for the separate processing of modalities followed by their integration. While it provides flexibility in combining results, it may not fully leverage the interdependencies between different types of data. In this thesis, we focus on implementing the result fusion technique to process RGB and Depth images separately and then combine the predictions from each network using an ensemble approach to obtain the final prediction.

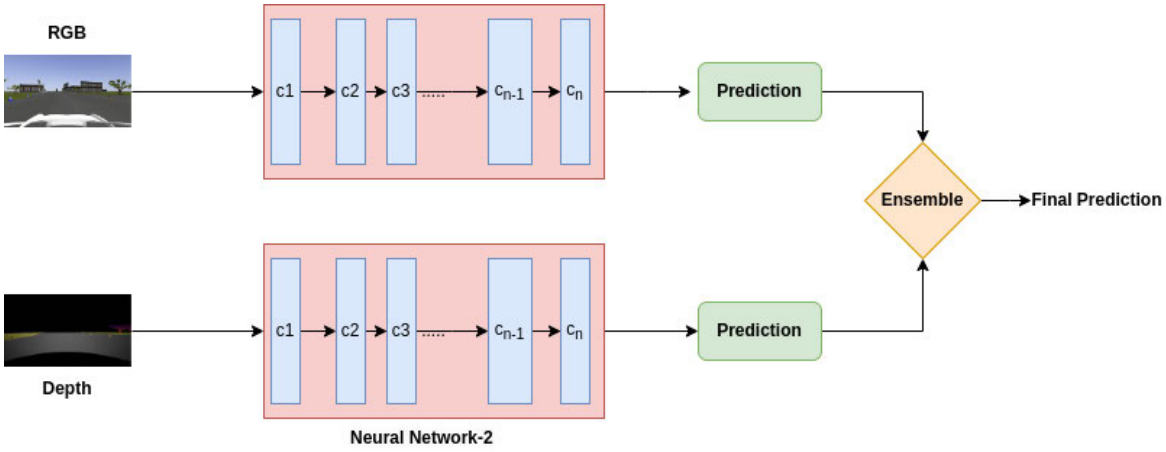


Figure 2.5: Result Fusion

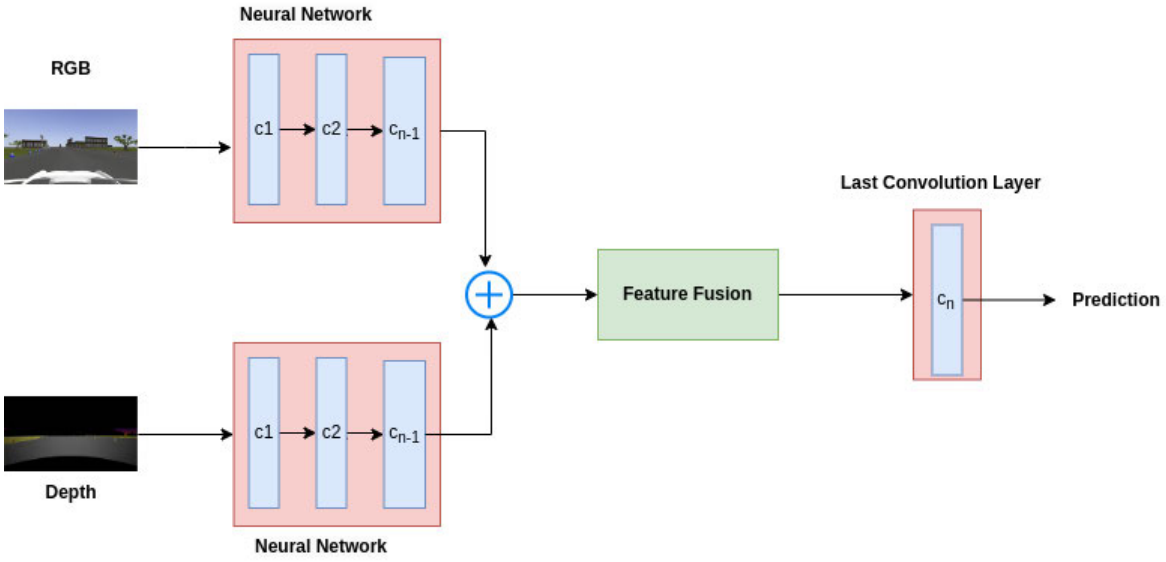


Figure 2.6: Late Feature Fusion

To summarize, this study employs the Gazebo simulation tool, combined with ROS1 and Blender, to create a simulation environment for capturing synthetic RGB and depth images using camera and LiDAR sensors. A fully automated labeling technique is utilized for the data captured. This data is then used to train cone detection models with both input fusion and result fusion techniques. Specifically, the first model is evaluated using real RGB images from the FSOCO dataset [5], aiming to study and optimize the neural network's performance based on these evaluations.

3 Machine Learning Concepts

3.1 Neural Networks

An artificial neuron is a mathematical model designed to mimic aspects of its biological counterpart. It processes inputs, applies an activation function, and produces an output. For a given neuron, let there be n inputs, where the input signals are $x_1, x_2, \ldots, x_n$ and the corresponding weights are $w_1, w_2, \ldots, w_n$. Additionally, a bias term $b \in \mathbb{R}$ is introduced to account for the bias in the model. Typically, the inputs are $\mathbf{x} \in \mathbb{R}^n$ and the weights are $\mathbf{w} \in \mathbb{R}^n$.

The output of the neuron is given by:

$$y = f\left(\sum_{j=1}^{n} w_j x_j + b\right)$$

Here, f represents the activation function applied to the weighted sum of the inputs plus the bias term.

A neural network is a system of layers of interconnected artificial neurons. It usually has an input layer, one or more hidden layers, and an output layer. Neurons in each layer are connected to neurons in the next layer through weighted connections. The network learns by adjusting these weights and biases to improve its performance on tasks.

Activation functions

An activation function is a mathematical function commonly used in artificial neurons to introduce non-linearity into the model. This non-linearity enables the model to solve tasks involving non-linearly separable data, which linear activation functions alone cannot achieve. Linear activation functions can only separate linearly separable data, regardless of the number of neurons used. It transforms the weighted sum of inputs plus bias into the output of the neuron. This can be expressed mathematically as:

$$f : \mathbb{R} \to \mathbb{R}$$

Activation functions are assumed to be squashing functions; these functions are monotonically increasing with $\lim_{z \to -\infty} f(z) = a$ and $\lim_{z \to \infty} f(z) = b$, where $a < b$.
Table 3.1 shows commonly used activation functions.

Activation Function	Definition	Range
Perceptron (Heaviside)	$h(z) = \begin{cases} 1 & \text{if } z \geq 0 \\ 0 & \text{if } z < 0 \end{cases}$	$\{0, 1\}$
Identity	$f(z) = z$	$\mathbb{R}$
Sigmoid	$\sigma(z) = \frac{1}{1+e^{-z}}$	$(0, 1)$
Hyperbolic Tangent	$\tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$	$(-1, 1)$
ReLU (Rectified Linear Unit)	$\text{ReLU}(z) = \max(0, z)$	$[0, \infty)$
Leaky ReLU	$\text{Leaky ReLU}(z) = \begin{cases} z & \text{if } z \geq 0 \\ \alpha z & \text{if } z < 0 \end{cases}$	$(-\infty, \infty)$
Softmax	$\text{Softmax}(z_i) = \frac{e^{z_i}}{\sum_j e^{z_j}}$	$(0, 1)$ (with outputs summing to 1)

Table 3.1: Common Activation Functions

3.2 Network Structures

FeedForward Network

A feedforward network is a type of artificial neural network in which data moves in one direction only: from the input layer, through any hidden layers, and to the output layer. This uni-directional flow ensures there are no cycles or loops in the network, forming a directed acyclic graph (DAG). The topological structure of a feedforward neural network can be described mathematically as a set of layers where neurons are connected without any feedback loops. The structure is characterized by:

$$N \subseteq \bigcup_{i=0}^{h-1} \bigcup_{j=i+1}^{h} N_i \times N_j$$

where:

- N_i denotes the set of neurons in layer i,
- N_j denotes the set of neurons in layer j,
- h denotes the depth of the network

Convolutional Neural Network

Convolutional Neural Network (CNN) is a type of neural network designed to efficiently handle spatially spatially structured data such as images, text, and audio. CNNs leverage convolutional layers to automatically learn and extract features from input data by optimizing filters (or kernels). This approach helps in capturing local patterns and hierarchical features efficiently.

Architecture

A CNN is composed of an input layer, a set of hidden layers, and an output layer. The architecture of a CNN is designed to effectively process spatially structured data. The hidden layers of a CNN include several types of layers:

- **Convolutional Layers:** These layers apply convolutional filters to the input data to extract features. The convolution operation is defined as:

$$\mathbf{y}_{i,j} = \sum_{m=1}^{M} \sum_{n=1}^{N} \mathbf{x}_{i+m,j+n} \cdot \mathbf{w}_{m,n} + b$$

  where
  - $\mathbf{x} \in \mathbb{R}^{H \times W \times C}$: Input matrix, where H denotes height,W denotes width, and C denotes the number of channels.
  - $\mathbf{w} \in \mathbb{R}^{K \times K \times C}$: Convolution kernel (or filter) with spatial dimensions $K \times K$ and depth C.
  - $b \in \mathbb{R}$: Bias term.
  - $\mathbf{y} \in \mathbb{R}^{H' \times W' \times F}$: Output feature map, where F is the number of filters, and H' and W' are the spatial dimensions after convolution.
- **Pooling Layers:** These layers reduce the spatial dimensions of feature maps. For max pooling:

$$\mathbf{y}_{i,j} = \max(\mathbf{x}_{i:i+k,j:j+k})$$

  where k is the pooling window size.
- **Fully Connected Layers:** These layers flatten the feature maps and perform classification or regression tasks.

Topological Structure: The topological structure of a CNN is characterized by layers organized as follows:

$$N \subseteq \bigcup_{l=1}^{L} (\text{Conv}_l \cup \text{Pool}_l \cup \text{FC}_l)$$

where:

- Conv_l denotes the set of convolutional layers at level l,
- Pool_l denotes the set of pooling layers at level l,
- FC_l denotes the set of fully connected layers at level l,
- L denotes the depth of the network.

Feature Extraction and Classification: Convolutional layers create feature maps representing different aspects of the input data, while fully connected layers interpret these features for prediction or classification.

3.3 Supervised vs. Unsupervised Learning

Supervised Learning

Supervised learning is a machine learning approach in which an algorithm is trained on labeled data to predict outcomes or identify relationships between input and output variables.

In supervised learning, the training data is represented as a set of pairs $(\mathbf{x}_i, y_i)$, where:

- $\mathbf{x}_i \in X \subseteq \mathbb{R}^n$ is the feature vector for the i-th instance, with X being the input feature space.
- $y_i \in Y$ is the corresponding output for the i-th instance.

The goal is to learn a function f that maps inputs to outputs:

$$f : X \to Y$$

where:

- For **classification tasks**, $Y = \{1, \ldots, C\}$ represents the output label space, with C being the number of possible discrete class labels.
- For **regression tasks**, $Y \subseteq \mathbb{R}$ represents a continuous output space.

The function f is trained to generalize from the given labeled examples, allowing it to predict the output for new, unseen instances based on the patterns learned from the training data.

Unsupervised Learning

Unsupervised learning is a type of machine learning where an algorithm learns from unlabeled data to identify patterns, structures, or relationships within the data. Unlike supervised learning, there are no predefined output labels for the training data.

In unsupervised learning, the training data is represented as a set of feature vectors $\mathbf{x}_i$, where:

- $\mathbf{x}_i \in X \subseteq \mathbb{R}^n$ is the feature vector for the i-th instance, with X being the input feature space.

The goal is to learn a structure or representation from the data itself without explicit guidance. Common tasks in unsupervised learning include:

- **Clustering:** Grouping similar instances together based on their feature vectors. For instance, identifying clusters in the data where members of the same cluster are more alike compared to those in different clusters. Mathematically, the goal of clustering can be expressed as finding a function f that assigns each $\mathbf{x}_i$ to a cluster C_j, where:

$$f(\mathbf{x}_i) = C_j, \quad \text{where} \quad C_j \subseteq X, \quad C_j \cap C_k = \emptyset \text{ for } j \neq k$$

- **Dimensionality Reduction:** Reducing the number of features while retaining the essential structure of the data. This can be achieved through methods like Principal Component Analysis or t-Distributed Stochastic Neighbor Embedding. In dimensionality reduction, the task is to find a lower-dimensional representation Z of the original data X, such that:
$$f(X) = Z, \quad Z \in \mathbb{R}^{n \times m}, \quad m < n$$
- **Anomaly Detection:** Identifying rare or unusual instances that differ significantly from the majority of the data.

Unsupervised learning does not require a mapping function f from inputs to outputs as in supervised learning. Instead, it focuses on discovering the inherent structure in the data, such as clusters, reduced dimensions, or anomalies:

$$\text{Unsupervised Learning: } X \rightarrow \text{Patterns or Structures}$$

In these tasks, the algorithm aims to find patterns, relationships, or structures that can either be useful for understanding the data or for other downstream machine learning tasks.

3.4 Types of Learning in Supervised Learning

Classification

Classification is a type of supervised learning where the goal is to predict the categorical class label of an instance based on its input features. The mathematical formulation can be represented as:

$$f : X \rightarrow Y^c$$

In this formulation:

- f represents the learner function.
- X represents the input space or feature space, where each instance is an n-dimensional feature vector.
- $Y^c = \{1, 2, \ldots, k\}$ represents the set of possible class labels, with k being the number of categories.

The learner function f maps an input instance from the input space X to one of the possible class labels Y^c. The output of the learner function is a categorical variable representing the predicted class label.

Regression

Regression is another type of supervised learning where the goal is to predict continuous or numeric values. The mathematical formulation for regression can be represented as:

$$f : X \rightarrow Y^r$$

In this formulation:

- f represents the learner function.
- X represents the input space or feature space, where each instance is an n-dimensional feature vector.
- $Y^r \subseteq \mathbb{R}$ represents the set of possible continuous or numeric values (output space).

The learner function f maps an input instance from the input space X to a continuous or numeric value from the output space Y^r. The output of the learner function is a numerical prediction.

3.5 Object Detection

Object detection is a field in computer vision focused on recognizing and localizing objects within images or video frames. Unlike classification, which only predicts the category of objects present, object detection also determines the spatial location of each object. This involves both classification (identifying what the object is) and localization (determining where the object is in the image).

Mathematically, object detection can be defined as a problem of simultaneously solving two tasks:

1. **Classification**: Given an image I, the task is to assign a label y from a set of classes C to each detected object. Formally, this can be represented as a function $f_{\text{cls}}(I)$ where $f_{\text{cls}}(I)$ outputs the class label y for each object.
2. **Localization**: Given an image I, the task is to determine the location of objects within the image. This is often represented by bounding boxes, which can be described using coordinates $(x_{\text{min}}, y_{\text{min}}, x_{\text{max}}, y_{\text{max}})$. Formally, this can be represented as a function $f_{\text{loc}}(I)$ where $f_{\text{loc}}(I)$ outputs the coordinates of the bounding boxes.

The object detection task can thus be expressed as finding the best combination of class labels and bounding box coordinates that maximizes the probability of detecting objects correctly. Mathematically, if BBox denotes the bounding box coordinates and Class denotes the class label, the goal is to optimize the following objective function [36]:

$$\text{Objective} = \sum_{i=1}^{N} \left[\text{Loss}_{\text{cls}}(p_i, p_i^*) + \beta \tilde{I}(t_i) \text{Loss}_{\text{loc}}(t_i, t_i^*) \right]$$

- $p_i \in [0,1]^{|C|}$ represents the predicted class probabilities for the i-th object, where $|C|$ is the number of classes.
- $p_i^* \in [0,1]^{|C|}$ represents the ground truth class probabilities for the i-th object.
- $t_i \in \mathbb{R}^4$ represents the predicted bounding box for the i-th object, described by coordinates $(x_{\text{min}}, y_{\text{min}}, x_{\text{max}}, y_{\text{max}})$.
- $t_i^* \in \mathbb{R}^4$ represents the ground truth bounding box for the i-th object.
- Loss_{cls} is the classification loss that quantifies the error in predicting the object class.

- Loss_{loc} is the localization loss that quantifies the error in predicting the bounding box coordinates.
- $\beta \in \mathbb{R}$ is a weighting factor that balances the importance of the classification and localization losses.
- $\tilde{I}(t_i)$ is an indicator function defined as:

$$\tilde{I}(t_i) = \begin{cases} 1 & \text{if IoU}(t_i, t_i^*) > \eta \\ 0 & \text{else} \end{cases}$$

where $\text{IoU}(t_i, t_i^*)$ denotes the Intersection over Union (IOU) between the predicted bounding box t_i and the ground truth bounding box t_i^*, and η is a threshold value. This function ensures that the localization loss is applied only when the predicted bounding box has sufficient overlap with the ground truth.

This objective function integrates classification and localization losses, ensuring that the model is optimized for both aspects of object detection. The sum is over all objects N detected in the image.

Intersection over Union

IOU is a metric used to evaluate the overlap between the predicted bounding box and the ground truth bounding box. It is defined as the ratio of the area of overlap to the area of union of the two boxes. Mathematically, IOU is given by:

$$\text{IoU}(t_i, t_i^*) = \frac{\text{Area of } (t_i \cap t_i^*)}{\text{Area of } (t_i \cup t_i^*)}$$

where t_i is the predicted bounding box and t_i^* is the ground truth bounding box. A higher IOU indicates a better alignment between the predicted and ground truth boxes. Fig.3.1 shows the pictorial representation of how IOU is calculated.

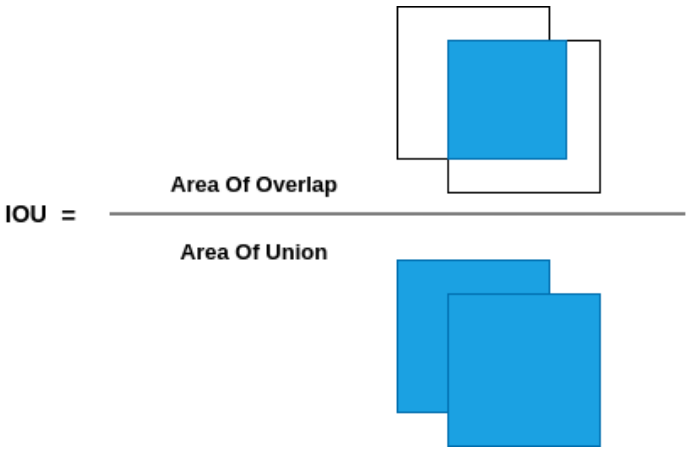


Figure 3.1: Intersection over Union

Non-Maximum Suppression

In object detection, a single object may be detected by multiple overlapping bounding boxes. Non-Maximum Suppression (NMS) is a method for removing duplicate bounding boxes. The NMS algorithm works as follows:

1. **Score Thresholding**: Select bounding boxes with scores above a certain threshold.
2. **Sorting**: Sort the selected bounding boxes based on their confidence scores.
3. **Suppression**: For each bounding box with the highest score, suppress all other bounding boxes that have an IOU above a specified threshold with the selected box.

Mathematically, NMS can be described using a threshold η for IOU:

$$\text{NMS}(B) = \{b \in B \mid \forall b' \in B, \text{IoU}(b, b') \leq \eta\}$$

where B represents the set of all bounding boxes, and b is a bounding box retained after applying NMS. This function ensures that only the bounding boxes with the highest confidence and minimal overlap are selected. In the In Fig.3.2, we can see how one of the blue boxes, denoting the predicted boxes, is selected in comparison to the red box, denoting the ground truth.

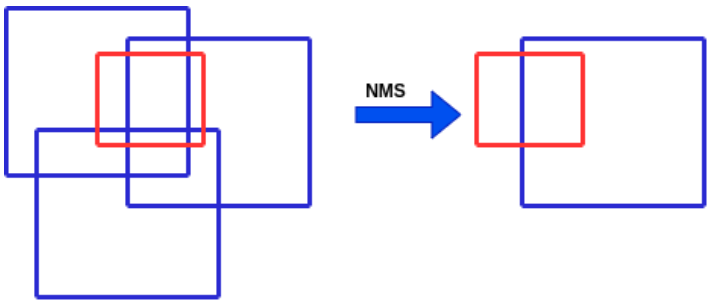


Figure 3.2: Non-Maximum Suppression

Incorporating IOU and NMS in the object detection framework ensures that the model's predictions are refined and optimized for both accuracy and relevance.

3.6 Stochastic Gradient Descent

Stochastic Gradient Descent (SGD) [37] is an optimization technique used in machine learning to minimize the error of a model by updating its weight vector $\mathbf{w} \in \mathbb{R}^n$. Gradient Descent, the broader method, iteratively adjusts the weights based on the gradient of an error function E, which measures how well the model predicts its targets.

SGD simplifies this by updating $\mathbf{w}$ using only a single sample or a small subset of data at each step, rather than the entire training set. The error function E captures the total loss across all samples and must be differentiable to compute gradients effectively. The total loss is often defined as the sum of local losses l, each representing the error for an individual

data sample.

$$E(X, W) = \sum_{i=1}^{N} l(\mathbf{x}_i, W)$$

where:

- $\mathbf{x}_i \in X \subseteq \mathbb{R}^n$: Data sample taken from the training set.
- $l(\mathbf{x}_i, W)$: Local loss for the training sample $\mathbf{x}_i$.
- N: Total number of training samples.
- W is a set of trainable parameters.

The goal of SGD is to minimize the error by updating the weight vector $\mathbf{w}$ in the direction that reduces the error. This is achieved by adjusting $\mathbf{w}$ using the gradient of the error function with respect to the weights. Specifically, for some weight vector $\mathbf{w} \in W$ at iteration t, the update is performed as follows:

$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \alpha \frac{\partial E(X, W)}{\partial \mathbf{w}^{(t)}}$$

where:

- $\alpha > 0$: Learning rate, determining the size of each update step.
- $\frac{\partial E(X,W)}{\partial \mathbf{w}^{(t)}}$: Gradient of the error function with respect to the weight vector at iteration t.

SGD's incremental updates make it efficient for large datasets, allowing quicker adjustments to $\mathbf{w}$ and enabling the model to learn continuously with each new data point.

In Mini-Batch Gradient Descent [37], the update rule is applied to a small subset of data, or mini-batch, rather than the entire dataset. The update for the weight vector $\mathbf{w} \in W$ at iteration $t + 1$ is given by:

$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \alpha \frac{\partial E_B(X_B, W)}{\partial \mathbf{w}^{(t)}}$$

This approach helps in efficiently updating the weights by averaging the gradients computed over multiple data samples, balancing between computational efficiency and the convergence properties of the algorithm.

3.7 AdamW Optimization Algorithm

Adaptive Moment Estimation with Weight Decay (AdamW) [38] is an advanced optimization algorithm that builds upon the principles of SGD but introduces several modifications to enhance convergence and generalization, especially in deep learning models. Unlike traditional SGD, which relies on a constant learning rate α, AdamW incorporates adaptive learning rates and weight decay to improve the optimization process.

AdamW extends the Adaptive Moment Estimation (Adam) optimizer, which adjusts the learning rate based on the first and second moments of the gradients. The Adam algorithm computes adaptive learning rates for each parameter by maintaining two running averages of the gradients:

$$\mathbf{m}^{(t)} = \beta_1 \mathbf{m}^{(t-1)} + (1 - \beta_1) \nabla E(X, W^{(t)})$$

$$\mathbf{v}^{(t)} = \beta_2 \mathbf{v}^{(t-1)} + (1 - \beta_2) \left(\nabla E(X, W^{(t)}) \right)^2$$

where:

- W is a set of trainable parameters.
- $\mathbf{m}^{(t)}$: Exponential moving average of the gradients (first moment).
- $\mathbf{v}^{(t)}$: Exponential moving average of the squared gradients (second moment).
- β_1 and β_2: Decay rates for the first and second moments, typically set to $\beta_1 = 0.9$ and $\beta_2 = 0.999$.

Since $\mathbf{m}^{(t)}$ and $\mathbf{v}^{(t)}$ are initialized as vectors of zeros, they are biased towards zero, particularly during early training steps. To address this, Adam computes bias-corrected estimates:

$$\hat{\mathbf{m}}^{(t)} = \frac{\mathbf{m}^{(t)}}{1 - \beta_1^t}, \quad \hat{\mathbf{v}}^{(t)} = \frac{\mathbf{v}^{(t)}}{1 - \beta_2^t}$$

These corrections adjust the moment estimates, mitigating the initial bias effect and allowing the algorithm to perform better during the initial iterations.

Adam uses the corrected estimates to update the model parameters $\mathbf{w} \in \mathbf{W}$:

$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \alpha \frac{\hat{\mathbf{m}}^{(t)}}{\sqrt{\hat{\mathbf{v}}^{(t)}} + \epsilon}$$

where:

- α: Learning rate, controlling the step size of updates.
- ϵ: A small constant (usually 10^{-8}) added for numerical stability to prevent division by zero.

AdamW modifies the traditional Adam optimizer by explicitly decoupling weight decay from the gradient-based updates. This adjustment addresses the over-regularization issue in Adam, where weight decay is applied indirectly through the gradient estimates. In AdamW, weight decay is directly applied to the weights, enhancing performance by reducing overfitting. The update rule for AdamW is:

$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \alpha \frac{\hat{\mathbf{m}}^{(t)}}{\sqrt{\hat{\mathbf{v}}^{(t)}} + \epsilon} - \lambda \mathbf{w}^{(t)}$$

where λ is the weight decay coefficient that controls the regularization strength.

AdamW offers several advantages, including adaptive learning rates for each parameter, improved generalization through decoupled weight decay, and faster convergence. These features make AdamW highly effective for training deep neural networks, balancing the simplicity of SGD with the advanced adaptive mechanisms of Adam.

4 Synthetic Dataset Generation and Annotation

This chapter discusses the steps involved in setting up the simulation environment in Gazebo and collecting data from sensors such as cameras and LiDAR, along with the vehicle's position. This data can then be used to generate fused RGB-D images and LiDAR images with automated labeling using concepts of transformations, rotations, and perspective projection.

Remark 4.1
This chapter primarily focuses on the generation of RGB-D images using point cloud data. A similar approach is used for generating LiDAR images and their corresponding labels, with slight modifications in the process.

4.1 Setting Up the Simulation Environment

The first step involves in cloning the "vehicle_sim" repository [39] from GitHub and install the required dependencies, and follow the instructions provided in the README.md file. The Gazebo world, as shown in Fig. 4.1, included various elements critical for AD scenarios:

Road Networks

- The simulated environment included variety of road layouts, including straight roads, intersections, and roundabouts using Gazebo's built-in tools.
- Buildings and other infrastructure elements were added to mimic urban and suburban environments.

Simulating Sensors

The repository supports the simulation of several key sensors:

- Velodyne models (VLP-16, HDL-32E) to simulate point cloud data.
- Monocular Camera were mounted on the simulated vehicle to capture RGB imgaes.
- An IMU was simulated to provide data on vehicle orientation and movement.

Customization and Scenario Variations

To ensure the robustness of our synthetic dataset, we customized the simulation scenarios by:

- Varying weather conditions and lighting to simulate different times of day.
- Creating multiple world files to cover a wide range of driving conditions and environments.

Creating Custom Objects with Blender

To create detailed and specific scenarios similar to those in the FSOCO dataset [5], we customized the simulated environment by designing traffic cones using Blender. This approach enhanced the realism and variety of our simulations. The following types of cones were designed:

- Blue cones
- Orange cones
- Yellow cones
- Large orange cones

These cones were modeled in Blender and exported as COLLADA files (.dae). The exported files were then imported into the Gazebo environment, allowing for precise placement within the simulation. An example of an orange cone modeled in Blender is shown in Fig. 4.2.

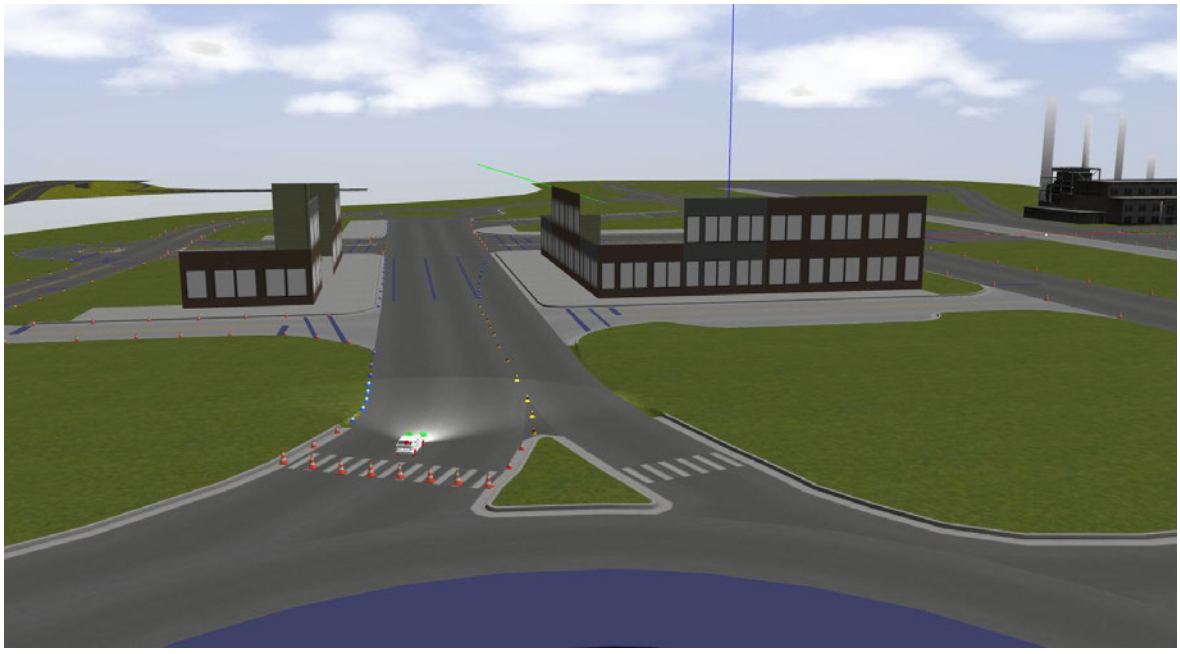


Figure 4.1: Simulated World

4.2 Vehicle and Sensors

In the simulated environment, the vehicle spawned into the world, as shown in Fig. 4.3, is represented by a set of interconnected links and joints. Each link corresponds to a part of the vehicle (e.g., base, wheels), and joints allow movement between these links. The GitHub repository "vehicle_sim" provides a streamlined workflow for defining, generating, and simulating the vehicle model within the development environment.

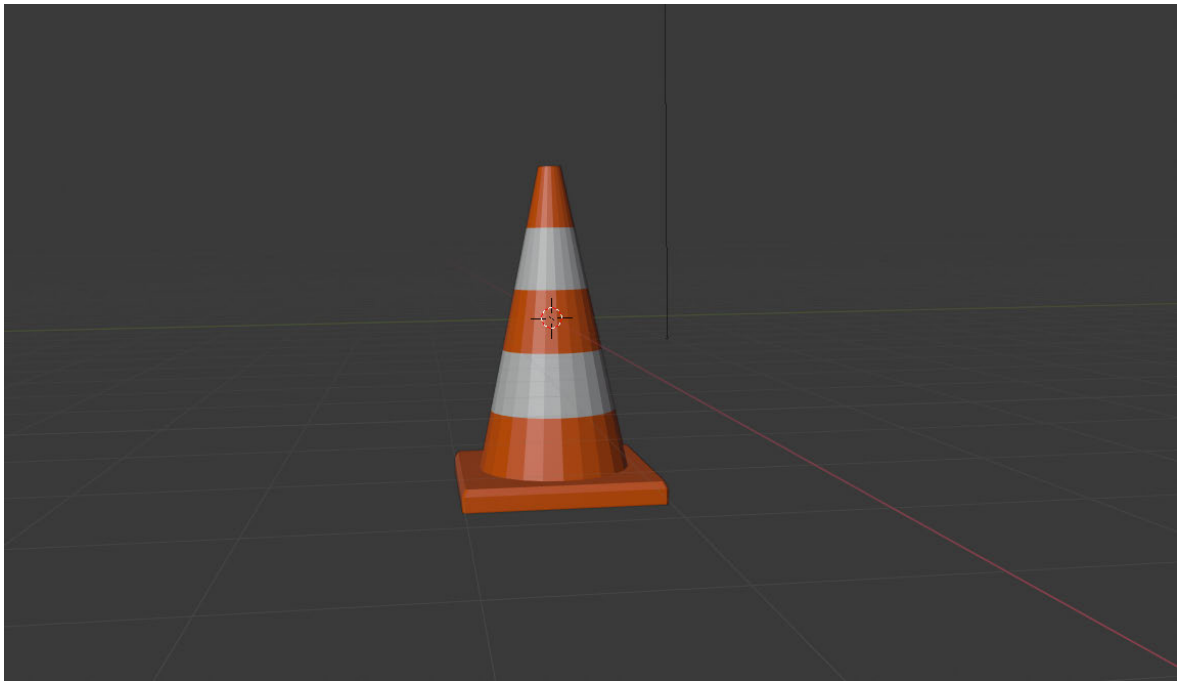


Figure 4.2: Traffic Cone Modelled Using Blender

The vehicle model is initially described using a detailed XML Macro (XACRO) file, which allows for the modular and parameterized description of the robot. During runtime, this XACRO file is processed to generate a complete Unified Robot Description Format (URDF) model using the XACRO tool. The generated URDF model is then dynamically loaded into the Gazebo environment. After the Gazebo simulation environment is initialized, setting up the context for the vehicle's operation, the vehicle model is spawned into the simulation at a specified position within the simulated world.

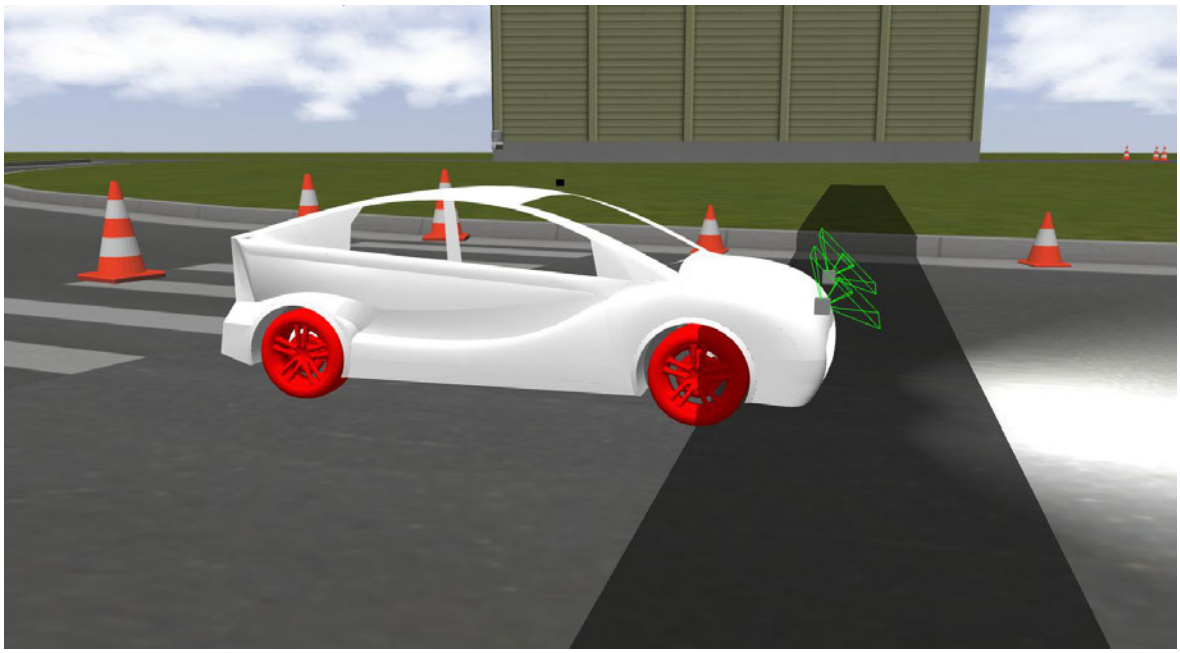


Figure 4.3: Spawned Vehicle

4.2.1 Camera

The camera integrated is a monocular type used for capturing visual data in a simulated environment, crucial for tasks like object detection, navigation, and visual perception in robotics applications. It boasts an image resolution of 1920x1080 pixels in RGB format (R8G8B8), ensuring detailed visual information for analysis. Its horizontal Field of View (FOV) is programmable, enabling wide-angle or narrow-angle views as needed. Clipping planes set at 0.02 units for near and 300 units for far determine the camera's visibility range within the simulation. Gaussian noise with a mean of 0.0 and a standard deviation of 0.007 adds realism to the captured images. It publishes two important types of data:

- `images_raw`: This topic provides the raw image data captured by the camera. This data is typically in its unprocessed form and includes the pixel values of each image frame.
- `camera_info`: This topic contains calibration information about the camera. It includes parameters such as intrinsic parameters (focal length, principal point, and distortion coefficients) and extrinsic parameters (camera position and orientation) that are crucial for accurately interpreting the raw image data.

The camera information is encapsulated in the following parameters:

- **Intrinsic Camera Matrix (K)**: This matrix, $\mathbf{K} \in \mathbb{R}^{3\times3}$, contains the intrinsic parameters of the camera, which are necessary for projecting 3D points into the 2D image plane. It is defined as:

$$\mathbf{K} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \tag{4.1}$$

- **Extrinsic Rotation Matrix (R)**: This matrix, $\mathbf{R} \in \mathbb{R}^{3\times3}$, represents the rotation from the vehicle's coordinate system to the camera's coordinate system.

$$\mathbf{R} = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix} \tag{4.2}$$

- **Extrinsic Translation Vector (t)**: This vector, $\mathbf{t} \in \mathbb{R}^3$, represents the translation from the vehicle's origin to the camera center.

$$\mathbf{t} = \begin{bmatrix} t_x \\ t_y \\ t_z \end{bmatrix} \tag{4.3}$$

- **Projection Matrix ($\mathbf{P}$)**: This matrix, $\mathbf{P} \in \mathbb{R}^{3\times4}$, combines both the intrinsic camera matrix $\mathbf{K}$ from equation 4.1 and the extrinsic parameters (rotation matrix $\mathbf{R}$ and translation vector $\mathbf{t}$) defined in from equations 4.2 and 4.3. It is used for projecting 3D points

into the 2D image plane. The projection matrix is defined as:

$$\mathbf{P} = \mathbf{K} \cdot \left[\mathbf{R} \mid \mathbf{t}\right] \tag{4.4}$$

where $\mathbf{K}$ is the intrinsic camera matrix, and $\left[\mathbf{R} \mid \mathbf{t}\right]$ is the extrinsic matrix combining rotation and translation.
In this case, the projection matrix $\mathbf{P}$ is given by:

$$\mathbf{P} = \begin{bmatrix} f_x & 0 & c_x & 0 \\ 0 & f_y & c_y & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

This matrix again includes the focal lengths and optical centers but also has a zero column to account for no translation along the x-axis.

The image shown in Figure 4.4 was captured from the data published on the topic `images_raw`.

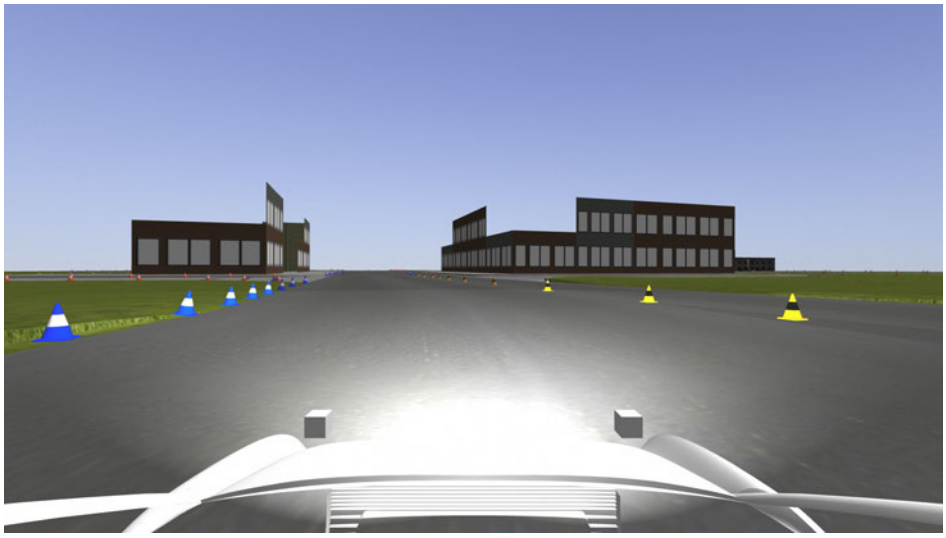


Figure 4.4: Camera Image

4.2.2 LiDAR

The LiDAR sensor is used for capturing detailed 3D point cloud data within a simulated environment. It is essential for tasks such as mapping, navigation, obstacle detection, and environment perception in robotics applications. In this thesis, HDL-32E Velodyne LiDAR sensor is mounted on top of the vehicle which provides high-resolution 3D point cloud data within the range of 50 meters from the vehicle. The LiDAR sensor can cover in a horizontal sweep starting directly ahead (0 radians) and extending to a little over half a full circle (3.14 radians). This means it can detect objects from all the way to the right, straight ahead, and all the way to the left, effectively covering a half-circle view. It publishes raw point cloud data to (`points_raw`) topic. The screenshot of the LiDAR rays emitted in the simulated world is shown in Fig 4.5.

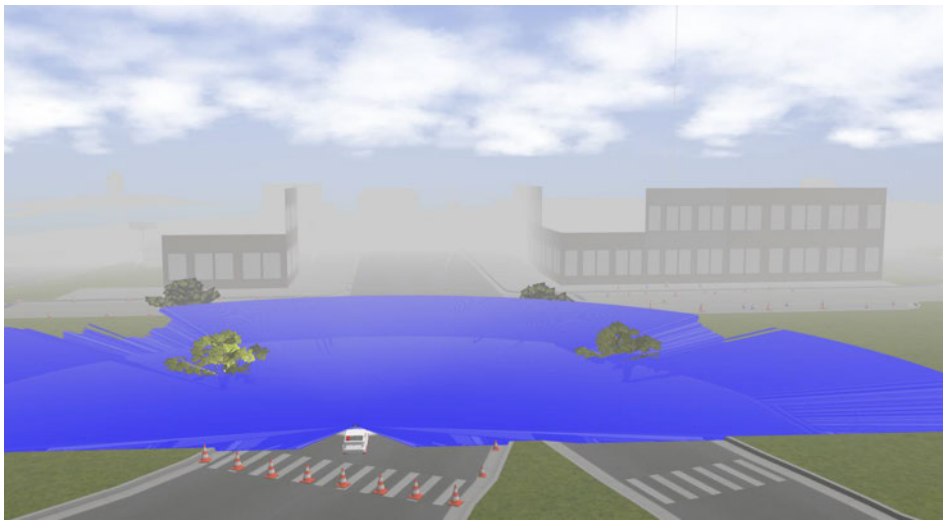


Figure 4.5: Laser rays emitted from LiDAR sensor

4.2.3 Vehicle Position

In Gazebo simulations, services such as `/gazebo/get_link_state` are used to retrieve the current pose (position and orientation) of specific links within the simulated environment. In this thesis, the simulated vehicle's origin is represented by the link `base_footprint`. The state of this link, which denotes the vehicle's position and orientation, is continuously published to the topic `/vehicle_info/pose`. This information is crucial for ROS applications, enabling them to access and utilize the precise pose of the vehicle for various robotic tasks. The pose data of a vehicle consists of position and orientation of vehicle in the world coordinates.

The below example vehicle pose is represented using ROS messages from the `geometry_msgs` package.

It consists of:

- **Position** (`geometry_msgs.msg.Point`):

$$\text{Position} \in \mathbb{R}^3 = (x, y, z)$$

  where x, y, and z denote the vehicle's position along the x-axis, y-axis, and z-axis in the world coordinate system, respectively.

- **Orientation** (`geometry_msgs.msg.Quaternion`):

$$\text{Orientation} \in \mathbb{R}^4 = (x, y, z, w)$$

  where x, y, z, and w denote the components of the quaternion that describes the vehicle's rotation in the world coordinate system.

4.3 Data Collection

As the vehicle is driven through the simulation environment, Gazebo publishes sensor data such as camera images, LiDAR point clouds, and vehicle position data to predefined ROS topics. A data synchronizer node subscribes to these topics, collects the sensor data, and synchronizes them based on their timestamps. The synchronized data is then saved to files for further analysis or processing. The overview of the data collection process is described in the Fig 4.6.

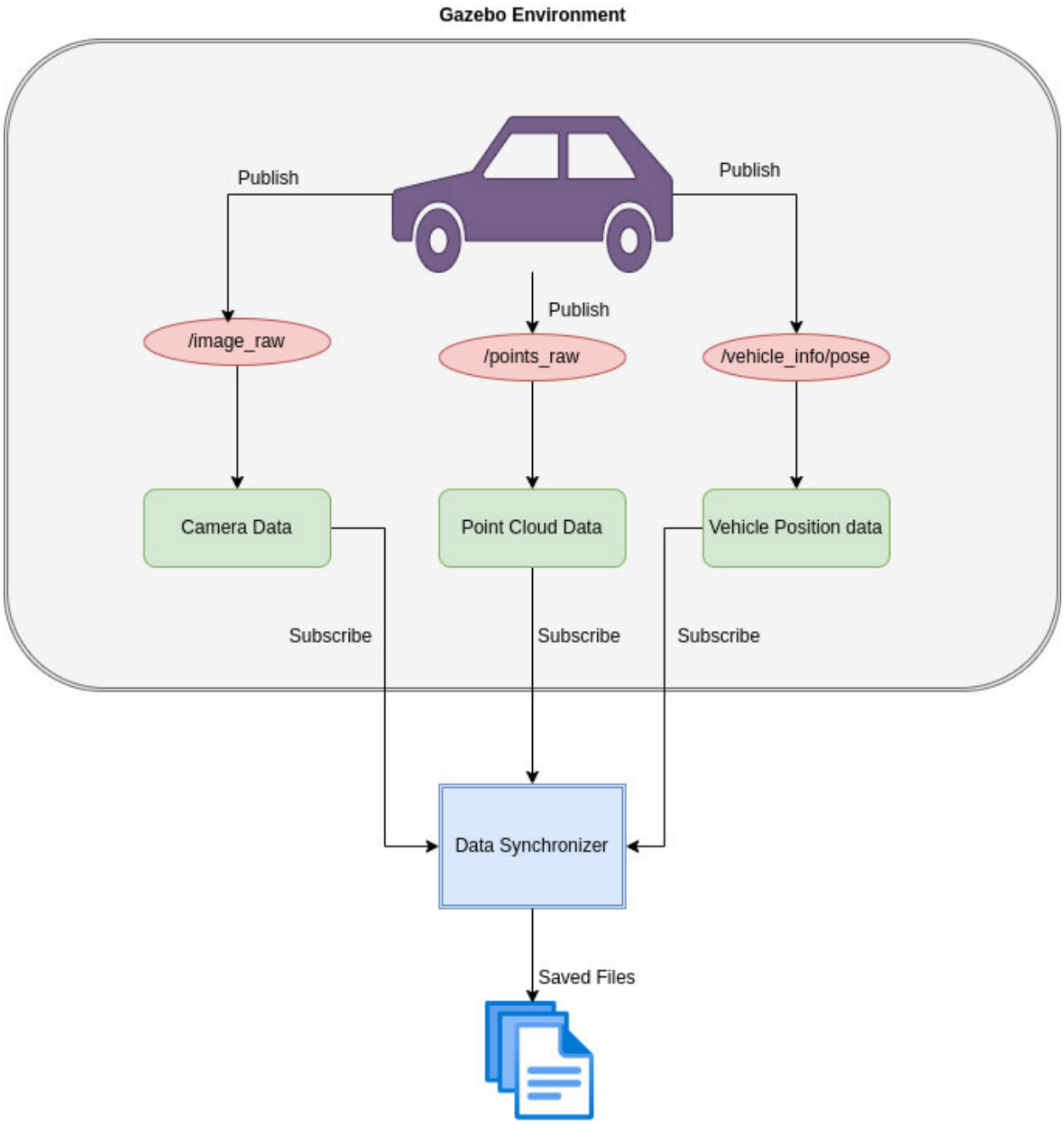


Figure 4.6: Data Collection from different sensors

4.4 RGB-D Data Generation with Labeling

Until this point, we have covered how to simulate the world and collect data from the sensors mounted on the vehicle. In this section, we will explore the process of generating RGB-D data and the associated labels. This involves a series of steps using the generated camera images, point cloud data, vehicle pose data, and the transformations between the vehicle and the camera. The following section explains this process in detail.

4.4.1 Point Cloud Transformation to Gazebo World Coordinates

Transformation Matrix

A transformation matrix [40] in Gazebo (and robotics in general) is a 4x4 matrix that represents both the translation and rotation of a rigid body in 3D space. It is used to transform a point from one coordinate system (reference frame) to another. This matrix is essential for tasks such as moving between world coordinates and camera coordinates.

The structure of the matrix is:

$$\mathbf{T} \in \mathbb{R}^{4\times4} = \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

where:

- $r_{ij} \in \mathbb{R}$: Elements of the 3×3 rotation matrix, which defines the orientation of the body in $\mathbb{R}^3$ space.
- $t_x, t_y, t_z \in \mathbb{R}$: Elements of the translation vector, representing the position of the body along the x-axis, y-axis, and z-axis, respectively.

A rotation matrix is a sub-part of the transformation matrix (the top-left 3x3 portion). It describes the orientation of a rigid body without considering its position. There are different ways to represent rotations, including:

- **Euler angles:** These are three angles (roll, pitch, yaw) that define rotations around the x-axis, y-axis, and z-axis in a specific order (the order can vary depending on the convention used).
- **Quaternions:** A quaternion is a four-element vector with a scalar rotation and a three-element vector. Quaternions are beneficial as they bypass the singularity problems that can occur with other rotational representations. The first element, w, is a scalar to normalize the vector, with the three other values, $[x, y, z]$, defining the axis of rotation.

The translation vector $\mathbf{t} \in \mathbb{R}^3$ indicates the shift from the origin of the reference frame to the origin of the transformed frame.

To transform a point $\mathbf{p} \in \mathbb{R}^4 = [x, y, z, 1]^T$ from one frame to another, you multiply it by the transformation matrix:

$$\mathbf{p}' \in \mathbb{R}^4 = \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

This operation will yield the coordinates of the point in the new reference frame, incorporating both rotation and translation.

Understanding and applying transformation matrices is crucial for tasks such as camera calibration, 3D reconstruction, and motion planning in robotics.

Steps to Convert Point Cloud into World Coordinates

Step1: Understanding the Inputs:

- *Vehicle to World Transformation*: Let $\mathbf{T}_{\text{vw}}$ be the transformation matrix representing the vehicle's pose in world coordinates. It includes rotation $\mathbf{R}_{\text{vw}}$ and translation $\mathbf{t}_{\text{vw}}$:

$$\mathbf{T}_{\text{vw}} = \begin{bmatrix} \mathbf{R}_{\text{vw}} & \mathbf{t}_{\text{vw}} \\ \mathbf{0}^T & 1 \end{bmatrix} \in \mathbb{R}^{4\times4} \tag{4.5}$$

- *Sensor to Vehicle Transformation*: Let $\mathbf{T}_{\text{sv}}$ represent the transformation matrix from sensor coordinates to vehicle coordinates. It consists of rotation $\mathbf{R}_{\text{sv}}$ and translation $\mathbf{t}_{\text{sv}}$:

$$\mathbf{T}_{\text{sv}} = \begin{bmatrix} \mathbf{R}_{\text{sv}} & \mathbf{t}_{\text{sv}} \\ \mathbf{0}^T & 1 \end{bmatrix} \in \mathbb{R}^{4\times4} \tag{4.6}$$

Step2: Transformation to Vehicle Frame:

- *Transforming Sensor Points to Vehicle Frame*: Sensor points $\mathbf{P}_{\text{s}} \in \mathbb{R}^{4\times N}$ are initially in sensor coordinates. To convert them to vehicle coordinates, multiply by $\mathbf{T}_{\text{sv}}$ from equation 4.6:

$$\mathbf{P}_{\text{v}} = \mathbf{T}_{\text{sv}} \cdot \mathbf{P}_{\text{s}} = [\mathbf{T}_{\text{sv}} \cdot \mathbf{p}_{\text{s},i} \mid i = 1, \ldots, N] \in \mathbb{R}^{4\times N} \tag{4.7}$$

  Here, $\mathbf{P}_{\text{v}}$ represents the sensor points transformed into the vehicle's coordinate system.

Step3: Transformation to World Frame:

- *Transforming Vehicle Points to World Frame*: Now, transform $\mathbf{P}_{\text{v}} \in \mathbb{R}^{4\times N}$ from equation 4.7 into world coordinates using $\mathbf{T}_{\text{vw}}$ from equation 4.5:

$$\mathbf{P}_{\text{w}} = \mathbf{T}_{\text{vw}} \cdot \mathbf{P}_{\text{v}} = [\mathbf{T}_{\text{vw}} \cdot \mathbf{p}_{\text{v},i} \mid i = 1, \ldots, N] \in \mathbb{R}^{4\times N} \tag{4.8}$$

  Here, $\mathbf{P}_{\text{w}}$ represents the sensor points converted into world coordinates.

Step4: Resulting World Coordinates:

- *Output*: The final output is $\mathbf{P}_{\text{w}}$ from equation 4.8, which represents the sensor points transformed into the global coordinate system of the world.

4.4.2 Calculation of distance from camera to world points

We need to calculate the distances from each point in a point cloud to the camera, taking into account the position and orientation of both the vehicle and the camera.

Step 1: Rotate Camera Position by Vehicle's Orientation:

The camera's position relative to the vehicle is given by $\mathbf{t}_{\text{cv}}$. We rotate this position by the vehicle's orientation using the rotation matrix $\mathbf{R}_{\text{v}}$:

$$\mathbf{t}_{\text{cv}}^{\text{rot}} = \mathbf{R}_{\text{v}} \cdot \mathbf{t}_{\text{cv}} \in \mathbb{R}^3 \tag{4.9}$$

Step 2: Translate Camera Position by Vehicle's Position:

The vehicle's position in the world frame is given by $\mathbf{t}_{\text{vw}}$ from equation 4.9. We translate the rotated camera position to the world frame:

$$\mathbf{t}_{\text{cw}} = \mathbf{t}_{\text{vw}} + \mathbf{t}_{\text{cv}}^{\text{rot}} \in \mathbb{R}^3 \tag{4.10}$$

Step 3: Calculate Euclidean Distances:

For each point in the point cloud $\mathbf{P}_{\text{w}}$, we calculate the Euclidean distance to the camera position in the world frame. The distance for each point is given by:

$$d_i = \|\mathbf{p}_{\text{w},i} - \mathbf{t}_{\text{cw}}\|_2 \mid i \in \{1, \ldots, N\} \in \mathbb{R} \tag{4.11}$$

where $\|\cdot\|_2$ denotes the Euclidean norm calculated along the appropriate axis, and d_i is the distance from the i-th point in the point cloud to the camera position.

4.4.3 Transforming World Coordinates into Camera Coordinates

We need to create a transformation matrix $\mathbf{T}_{\text{cv}}$ that converts coordinates from the camera's frame to the vehicle's frame. This transformation encompasses both rotational and translational adjustments.

Let $\mathbf{T}_{\text{cv}}$ represent the transformation matrix from camera coordinates to vehicle coordinates. It consists of rotation $\mathbf{R}_{\text{cv}}$ and translation $\mathbf{t}_{\text{cv}}$:

$$\mathbf{T}_{\text{cv}} = \begin{bmatrix} \mathbf{R}_{\text{cv}} & \mathbf{t}_{\text{cv}} \\ 0 & 1 \end{bmatrix} \in \mathbb{R}^{4\times4} \tag{4.12}$$

Inverse Transformation from Vehicle to Camera Coordinates: To convert points from the vehicle frame back to the camera frame, we need the inverse of $\mathbf{T}_{\text{cv}}$ defined in equation 4.12, denoted as $\mathbf{T}_{\text{cv}}^{-1}$. The inverse of a homogeneous transformation matrix can be computed as:

$$\mathbf{T}_{\text{cv}}^{-1} = \begin{bmatrix} \mathbf{R}_{\text{cv}}^{T} & -\mathbf{R}_{\text{cv}}^{T}\mathbf{t}_{\text{cv}} \\ 0 & 1 \end{bmatrix} \in \mathbb{R}^{4\times 4} \tag{4.13}$$

Inverse Transformation from World to Vehicle Coordinates: To convert points from the world frame back to the vehicle frame, we need the inverse of $\mathbf{T}_{\text{vw}}$ defined in equation 4.5, denoted as $\mathbf{T}_{\text{vw}}^{-1}$. The inverse of a homogeneous transformation matrix can be computed as:

$$\mathbf{T}_{\text{vw}}^{-1} = \begin{bmatrix} \mathbf{R}_{\text{vw}}^{T} & -\mathbf{R}_{\text{vw}}^{T}\mathbf{t}_{\text{vw}} \\ 0 & 1 \end{bmatrix} \in \mathbb{R}^{4\times 4} \tag{4.14}$$

Transforming the Point Cloud:

Step 1: World to Vehicle Coordinates: Multiply the point cloud $\mathbf{P}_{\text{w}}$ from equation 4.8 by $\mathbf{T}_{\text{vw}}^{-1}$ from equation 4.14 to transform it to the vehicle coordinates:

$$\mathbf{P}_{\text{v}} = \mathbf{T}_{\text{vw}}^{-1} \cdot \mathbf{P}_{\text{w}} = \left[\mathbf{T}_{\text{vw}}^{-1} \cdot \mathbf{p}_{\text{w},i} \mid i = 1, \ldots, N\right] \in \mathbb{R}^{4\times N} \tag{4.15}$$

Step 2: Vehicle to Camera Coordinates: Multiply the resulting point cloud $\mathbf{P}_{\text{v}}$ from equation 4.15 by $\mathbf{T}_{\text{cv}}^{-1}$ from 4.13 to transform it to the camera coordinates:

$$\mathbf{P}_{\text{c}} = \mathbf{T}_{\text{cv}}^{-1} \cdot \mathbf{P}_{\text{v}} = \left[\mathbf{T}_{\text{cv}}^{-1} \cdot \mathbf{p}_{\text{v},i} \mid i = 1, \ldots, N\right] \in \mathbb{R}^{4\times N} \tag{4.16}$$

4.4.4 Transforming Camera Coordinates into Pixel Coordinates

Perspective projection is a method used in computer graphics and computer vision to project 3D points onto a 2D plane (the image plane) as seen by a camera.

Remark 4.2
The following explanation pertains to the conversion of a single 3D point to 2D coordinates. For a more in-depth explanation, please refer to [41].

Mathematical Background

Given:

$$\mathbf{p_w} = [x, y, z, 1]^T \in \mathbb{R}^4$$

is a 3D point in homogeneous coordinates in the world coordinate system from equation 4.8.

$\mathbf{K}$ is the camera intrinsic matrix, $\mathbf{R}$ is the rotation matrix, and $\mathbf{t}$ is the translation vector as defined in the equations 4.1, 4.2 and 4.3 .

The relationship between the 3D world point $\mathbf{x}_w$ and its corresponding 2D image point $\mathbf{x}_i = [u, v, 1]^T$ can be described by:

$$s\mathbf{x}_i = \mathbf{K}\left[\mathbf{R} \mid \mathbf{t}\right]\mathbf{x}_w \in \mathbb{R}^3 \tag{4.17}$$

where s is a scaling factor due to the use of homogeneous coordinates.

The camera intrinsic matrix from equation 4.1, projects 3D points given in the camera coordinate system, defined in equation 4.16 to 2D pixel coordinates i.e.

$$\mathbf{p} = \mathbf{K}\mathbf{p}_c \in \mathbb{R}^3$$

Hence,

$$s\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}\begin{bmatrix} x_c \\ y_c \\ z_c \end{bmatrix} \in \mathbb{R}^3$$

The intrinsic parameter matrix is scene-independent. Once estimated, it can be reused if the focal length remains fixed (e.g., with a zoom lens). If the image is scaled, these parameters must be scaled by the same factor.

The joint rotation-translation matrix

$$[\mathbf{R} \mid \mathbf{t}]$$

is the product of a projective transformation and a homogeneous transformation. The 3-by-4 projective transformation maps 3D point in camera coordinates to 2D point in the image plane, represented in normalized camera coordinates.

$$z_c\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}\begin{bmatrix} x_c \\ y_c \\ z_c \\ 1 \end{bmatrix} \in \mathbb{R}^3 \tag{4.18}$$

Thus, given the representation of a point in point cloud $\mathbf{p}_\mathrm{w}$, we can also obtain representation in the camera coordinate system defined in 4.16 as:

$$\mathbf{p}_c = \begin{bmatrix} \mathbf{R} & \mathbf{t} \\ 0 & 1 \end{bmatrix}\mathbf{p}_w \in \mathbb{R}^4 \tag{4.19}$$

This homogeneous transformation consists of $\mathbf{R}$, a 3-by-3 rotation matrix, and $\mathbf{t}$, a 3-by-1 translation vector:

$$\begin{bmatrix} \mathbf{R} & \mathbf{t} \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \in \mathbb{R}^{4\times 4} \tag{4.20}$$

Hence by using equation 4.20 in the equation 4.19 we get

$$\begin{bmatrix} x_c \\ y_c \\ z_c \\ 1 \end{bmatrix} = \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_w \\ y_w \\ z_w \\ 1 \end{bmatrix} \in \mathbb{R}^4$$

Combining the projective transformation from equation 4.18 and homogeneous transformations from equation 4.20 maps 3D world coordinates to 2D image plane coordinates:

$$\begin{aligned} z_c \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} &= [\mathbf{R} \mid \mathbf{t}] \begin{bmatrix} x_w \\ y_w \\ z_w \\ 1 \end{bmatrix} \\ &= \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_w \\ y_w \\ z_w \\ 1 \end{bmatrix} \in \mathbb{R}^3 \end{aligned} \tag{4.21}$$

with $x' = x_c/z_c$ and $y' = y_c/z_c$, equating the equations for intrinsic matrix from 4.1 and extrinsics from 4.21 together in 4.17

$$s \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \end{bmatrix} \begin{bmatrix} x_w \\ y_w \\ z_w \\ 1 \end{bmatrix} \in \mathbb{R}^3$$

if $z_c \neq 0$, the transformation above is equivalent to the following,

$$\begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} f_x x_c/z_c + c_x \\ f_y y_c/z_c + c_y \end{bmatrix} \in \mathbb{R}^2$$

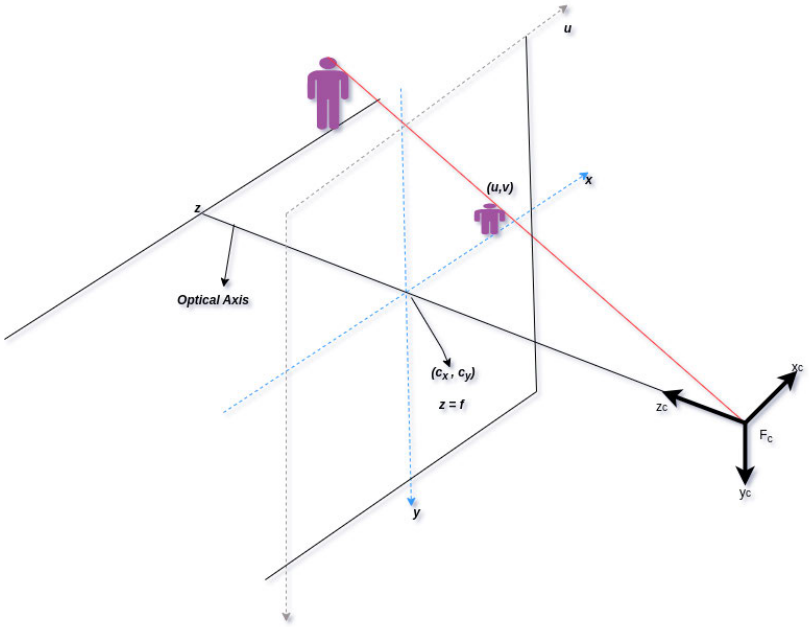


Figure 4.7: 3D to 2D projection [40]

The `projectPoints` method in OpenCV [42], which leverages the aforementioned principles, was used to project the 3D LiDAR points onto the 2D image, thereby determining the pixel coordinates [41]. The visual representation of projecting world objects onto the image plane is illustrated in Fig 4.7.

4.4.5 Adding Depth and Mask Channel to RGB

Using the camera image and pixel coordinates obtained from perspective projection, we enhance the RGB image with additional channels. Specifically, the depth information, which represents the distance from each point in the point cloud to the camera calcuated in the subsection 4.4.2 , is added as a new channel. Additionally, a mask channel is created to indicate the presence of depth information for each pixel.

The depth channel adds an additional layer to the RGB image by providing depth information for each pixel, which represents the distance from the camera to the corresponding point in the scene. This depth information is determined based on the 3D coordinates of the points. After calculating the distance from the camera to each world point, each 3D point is projected onto the 2D image plane. The resulting depth information is then stored in the corresponding pixel of the Depth map, reflecting the distance from the camera to the point in the scene.

The mask channel is a binary image where each pixel is marked as 1 if there is corresponding depth information available (i.e., there is a point in the point cloud that projects to this pixel), and 0 otherwise. This channel helps in distinguishing between pixels with valid depth information and those without.

By combining these additional channels with the original RGB channels, we obtain a richer representation of the scene that includes both color and depth information, enhancing the capabilities for various computer vision tasks.

4.4.6 Identification of Cones and Label Generation

In this section, the goal is to identify traffic cones within a given environment captured by a camera and generate labels associated with their positions in the image. This process involves several key steps to ensure accurate detection and localization of the cones, as illustrated in Fig. 4.8.

- **Cone Detection and Filtering:** The positions and colors of traffic cones are read from the Gazebo world file, and a filtering process is applied to select cones within a 50-meter range from the vehicle. This ensures that only relevant cones are considered for further processing.
- **Perspective Transformation:** Once candidate cones are identified, their positions are transformed from the world coordinate system to the camera's coordinate system as explained in the subsection 4.4.3. This transformation accounts for the vehicle's position and orientation relative to the cones and the camera. It enables accurate projection of the 3D cone positions onto the 2D image plane captured by the camera.
- **Projection onto Image Plane:** Using the intrinsic parameters of the camera, such as focal length and principal point, the 3D positions of the cones are projected onto the 2D image plane as explained in the subsection 4.4.4. This projection calculates the pixel coordinates where each cone is expected to appear in the image.
- **Generation of Labels:** Labels are generated for each detected cone based on its projected position in the image, often including details such as the cone's color and its location within the image (represented by a bounding box).
- **Output:** The final output includes a labeled file saved in text format with all detected cones, including their corresponding bounding boxes and class labels. These labels follow the YOLOv8 format, where each line represents an object with its class, and normalized bounding box coordinates (center x, center y, width, height).

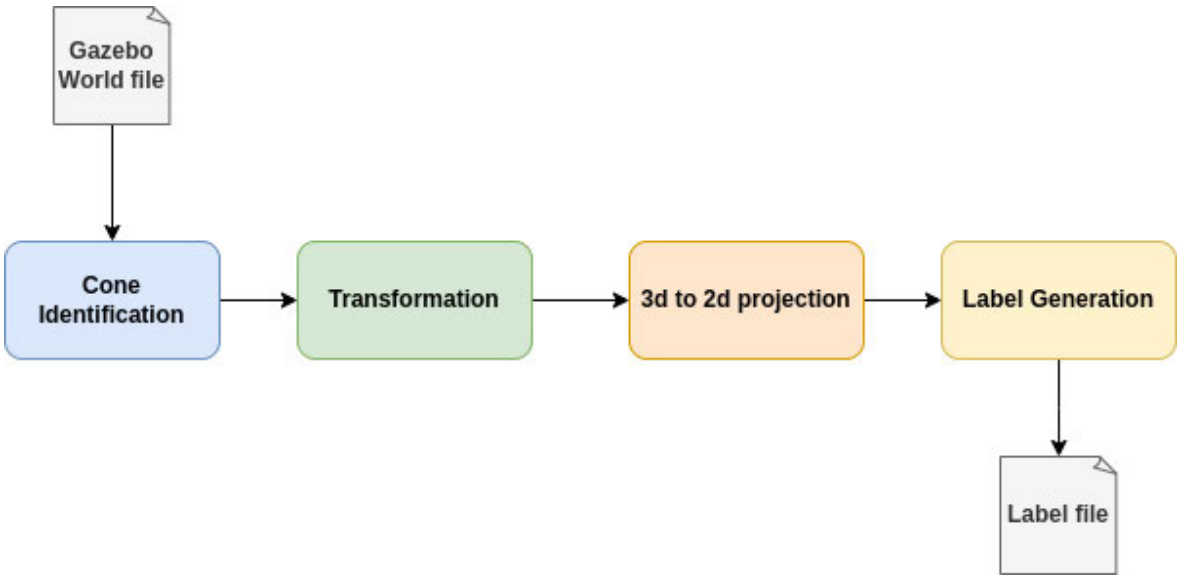


Figure 4.8: Label Generation Process

4.5 LiDAR Image and Label Generation

LiDAR images are used as input for the Depth model, which is part of the late fusion technique. The process for generating LiDAR images is similar to that used for RGB-D data, with a few key differences. As shown in Fig.4.9, in the case of LiDAR, pixel coordinates are derived by transforming the point cloud into camera coordinates using perspective projection, and then mapping these coordinates onto an empty RGB image with default settings (0,0,0). This differs from the RGB-D method, where projections are made directly onto the RGB image plane. Additionally, during the labeling process, cones in RGB-D or RGB images are selected based on their distance of 50 meters in front of the vehicle, whereas in LiDAR images, the point cloud is used to identify cones in the immediate vicinity. Finally, the labels for the LiDAR images contain just one class since there is no color information for the cones in these images. The remainder of the process remains the same.

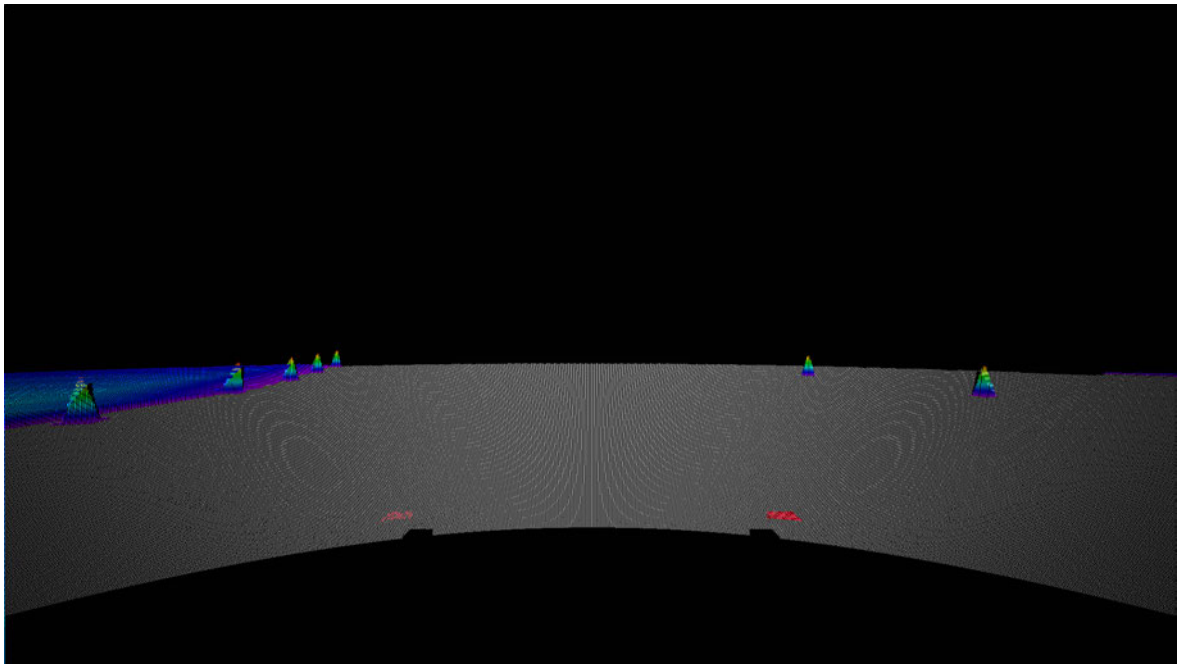


Figure 4.9: LiDAR Image Generated Using Point Cloud and Camera Parameters

5 Early Fusion Model Implementation

In the previous chapter, we covered the setup of the simulation environment and the generation of images and labels for cone detection. This chapter focuses on the implementation of the cone detection model using early fusion techniques. We will discuss the experiments conducted and present the results obtained.

Remark 5.1
In this thesis, multiple datasets were iteratively generated, including RGB images, RGB-D images with and without a mask channel, LiDAR images, and their corresponding labels, following the process defined in Chapter 4.

Remark 5.2
In this thesis, the YOLOv8 architecture was used to develop all the models. As there is no official paper on YOLOv8, details of the architecture will be omitted. For a detailed explanation, please refer to the sources [43], [44], and [45].

5.1 Objective and Scope

The primary objective of this thesis is to develop and evaluate an early fusion model for object detection using the YOLOv8 architecture. The model is trained on RGB-D synthetic data and tested on real RGB images. The key aims of this approach are:

- **Enhancement of Object Detection Capabilities:** By incorporating depth information into the training process, the model is expected to improve its ability to detect and classify objects more accurately.
- **Evaluation of Model Generalization:** The model's performance will be assessed on real RGB images to determine how well it generalizes from synthetic RGB-D data to practical, real-world scenarios. This evaluation is crucial to understanding the effectiveness of using depth information during training when only RGB data is available for testing.

5.2 Training & Validation

Remark 5.3
All models discussed in this section were evaluated with an IOU threshold of 0.6 for NMS and a confidence threshold of 0.5.

5.2.1 RGB-D Model

Initially, the RGB-D model was trained on a dataset of 4172 synthetic images to detect and classify yellow, blue, orange and large orange traffic cones. The data required to generate the RGB-D images, along with labels, was captured by driving the car through the simulated world as described in Chapter 4. For this training, the YOLOv8 architecture, modified to accommodate the additional depth channel, was inspired by the suggestions outlined in a relevant GitHub issue [46]. The training parameters are shown in the Table 5.1. For evaluation, real RGB images were used, with the additional depth channel defaulted to 0.

Parameter	Value
Training Images	4172
Background Images(%)	11%
Epochs	50
Batch Size	16
Image Size	640
Optimizer	AdamW
Model Architecture	Nano
Learning Rate	0.01
Momentum	0.937
Weight Decay	0.0005
Validation Images	1500
Input Channels	4 (RGB-D)

Table 5.1: Training Parameters RGB-D Model

Model Evaluation

The confusion matrix displayed in Fig. 5.1 provides an overview of the model's performance on the validation set. This matrix highlights that the model struggles with real-world images where the depth information is defaulted to zero. The RGB-D model was trained on synthetic images where depth data was provided for pixels within a specific range, specifically, within 50 meters from the vehicle. However, when exposed to real images lacking this depth information, the model's accuracy diminishes significantly.

The primary challenge arises because the model was heavily reliant on depth information for detecting traffic cones, and it performs poorly when such information is absent. This is reflected in the high rate of false negatives, averaging 93%, which indicates that the model frequently misclassifies cones as background even though the cones are present. In real-world scenarios, where depth values are either not available or inaccurately represented (defaulting to zero), the model's predictions become unreliable. This suggests that the model's training on synthetic data did not adequately prepare it to handle scenarios where depth information is missing or significantly different from the training conditions.

Further analysis of the confusion matrix reveals specific difficulties in distinguishing between similar objects, such as different types of traffic cones, under variable lighting and atmospheric conditions. For example, the model often confuses orange cones with large orange

cones, indicating a potential flaw in the feature extraction layers of the architecture. These issues underscore the need for the model to better understand the depth channel and for the inclusion of realistic training samples that mimic real-world scenarios.

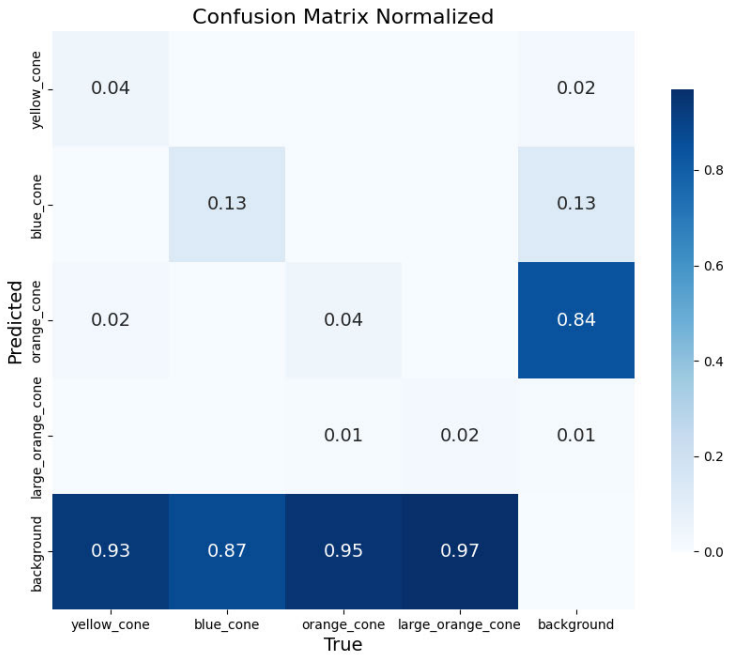


Figure 5.1: Confusion Matrix for RGB-D Model Validation

5.2.2 RGB-D Model With Additional Mask Channel

Since the RGB-D model struggled to generalize depth information—due to the fact that only a few pixels had meaningful depth values while others defaulted to zero—a new approach was necessary. To address this limitation, an additional channel, known as the mask channel, was introduced in the training images. This new format, referred to as Red, Green, Blue with Depth and Mask channels (RGB-D-M), includes a binary mask channel that provides information about the presence of valid depth data:

- **Value of 1:** Indicates that the corresponding pixel has a depth value greater than zero.
- **Value of 0:** Indicates that the corresponding pixel has a depth value of zero, meaning no meaningful depth information is available.

This mask channel allows the model to distinguish between pixels with valid depth information and those without. By incorporating this additional channel, the model can more effectively learn from depth information during training, leading to improved performance when processing real-world images where depth data is absent. In addition, data was gathered under different lighting and atmospheric settings to mimic the real-world images used in the validation set. The training parameters are shown in the Table 5.2.

Remark 5.4
The RGB-D-M model is a variant of the original RGB-D model, with the additional mask channel being its key distinction. It still falls within the broader category of RGB-D models, as it primarily operates on the same principles with an added feature for depth information refinement.

Parameter	Value
Training Images	5122
Background Images(%)	10%
Epochs	50
Batch Size	16
Image Size	640
Optimizer	AdamW
Model Architecture	Nano
Learning Rate	0.01
Momentum	0.937
Weight Decay	0.0005
Validation Images	1500
Input Channels	5 (RGB-D-M)

Table 5.2: Training Parameters RGB-D-M Model

Model Evaluation

The confusion matrix presented in Fig.5.2 offers a summary of the model's performance on the validation set. The inclusion of the additional mask channel has led to a modest increase in true positives across various labels, indicating that the model is now slightly better at distinguishing between pixels with valid depth information and those without. However, the model still faces significant challenges when processing real images. This is primarily because the validation images have a depth value of zero across all pixels, exacerbating the domain gap between the synthetic training data and real-world images.

Additionally, the model continues to struggle with misclassification, particularly in distinguishing between similar traffic cone types, such as orange and large orange cones. The high rate of false negatives, where cones are frequently classified as background, suggests that the model might still be overly reliant on depth information or not sufficiently robust in extracting and utilizing other relevant features.

The limited diversity in the synthetic training images may also contribute to the model's underperformance. The synthetic data might not adequately capture the variety of lighting, atmospheric conditions, and object appearances found in real-world scenarios, leading to poor generalization on the validation set. This highlights the need for further refinement in both the training data and model architecture, focusing on enhancing the model's ability to handle variations and reducing its dependency on depth information.

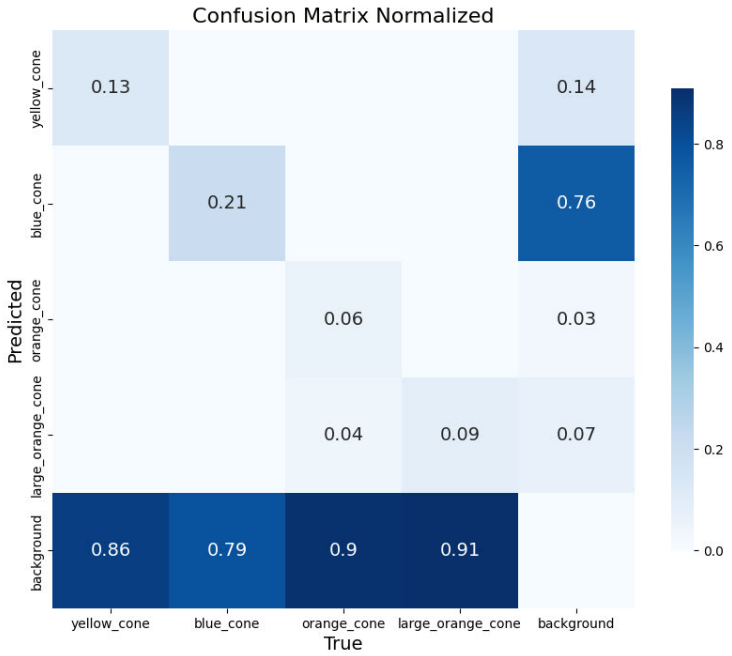


Figure 5.2: Confusion Matrix for RGB-D-M Model Validation

5.2.3 RGB-D-M Model With Mixed Training Dataset

To address the domain gap between synthetic and real images and enhance the diversity of the training dataset, several improvements were made. Specifically, 30% of the synthetic images were modified to have a depth value of zero across all pixels, mirroring the real images used during validation. Additionally, 15% of the dataset was composed of real images, resulting in a mixed dataset with a total of 6098 training images. Out of these, 896 images are real, with depth and mask values set to zero, while the remaining 85% are synthetic. Within this synthetic subset, 70% of the images contain depth information, while the remaining 30% have both depth and mask values set to zero. The training parameters are shown in the Table 5.3.

Parameter	Value
Training Images	6098
Background Images (%)	10%
Epochs	50
Batch Size	16
Image Size	640
Optimizer	AdamW
Model Architecture	Nano
Learning Rate	0.01
Momentum	0.937
Weight Decay	0.0005
Validation Images	1500
Input Channels	5 (RGB-D-M)

Table 5.3: Training Parameters for the RGB-D-M Model With Mixed Dataset

Model Evaluation with 640 Pixel Image

The confusion matrix shown in Fig. 5.3 provides an overview of the model's performance on the validation set. By incorporating both synthetic RGB images and real images with depth values defaulting to 0, the model's performance has notably improved, showing a twofold increase compared to the previous RGB-D-M model. Specifically, the true positive rates for various cone classes have increased, with the model now correctly identifying yellow cones with a 36% accuracy rate, blue cones with 36%, orange cones with 29%, and large orange cones with 35%. This marks a significant improvement over earlier models.

However, the model still struggles with generalization. The matrix indicates substantial confusion between different types of cones, particularly between orange and large orange cones, where misclassifications are common. The background classification has improved, yet a considerable proportion of cones are still being misclassified as background, with true positive rates for background ranging from 61% to 69% depending on the cone type.

This difficulty arises because the cones in the images occupy only small portions of the overall image. Additionally, resizing the images from their original dimensions of (1920,1080) or more to (640,640) poses a challenge for the model, as the smaller size limits the amount of detail available for learning and detecting the cones effectively. This loss of detail likely contributes to the observed misclassification rates, particularly for small or similarly colored cones, where the model might struggle to differentiate between subtle features.

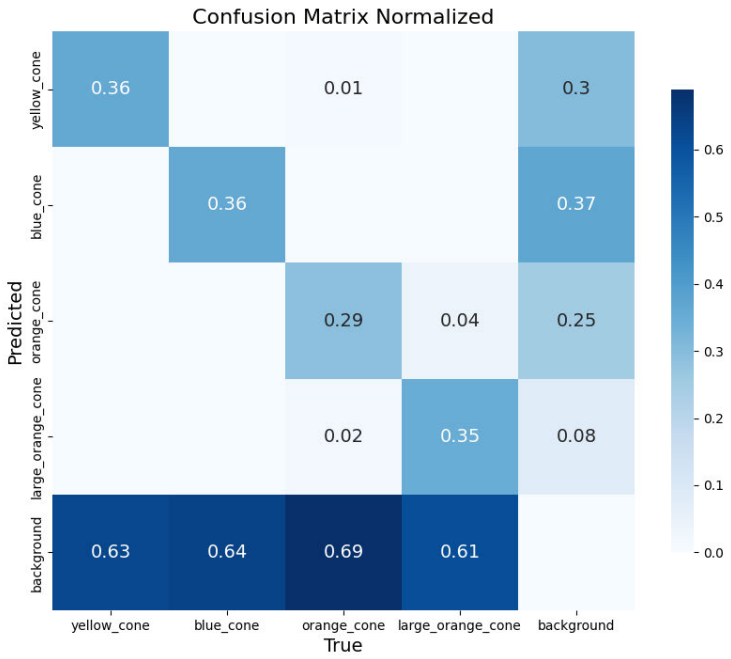


Figure 5.3: Confusion Matrix for RGB-D-M Model Validation at 640px Resolution

Model Evaluation with 1440 Pixel Image

The confusion matrix displayed in Fig.5.4 summarizes the model's performance on the validation set, where the images were evaluated at a resolution of 1440px. Since the cones occupy only a small portion of the overall image, detecting them in lower-resolution images (640px) is challenging. To address this, the image resolution was increased to 1440px during validation. This adjustment allows the model to better capture the fine details and improve detection accuracy. As a result, there is a significant improvement in performance, with a noticeable increase of at least 35% in the detection of true positives.

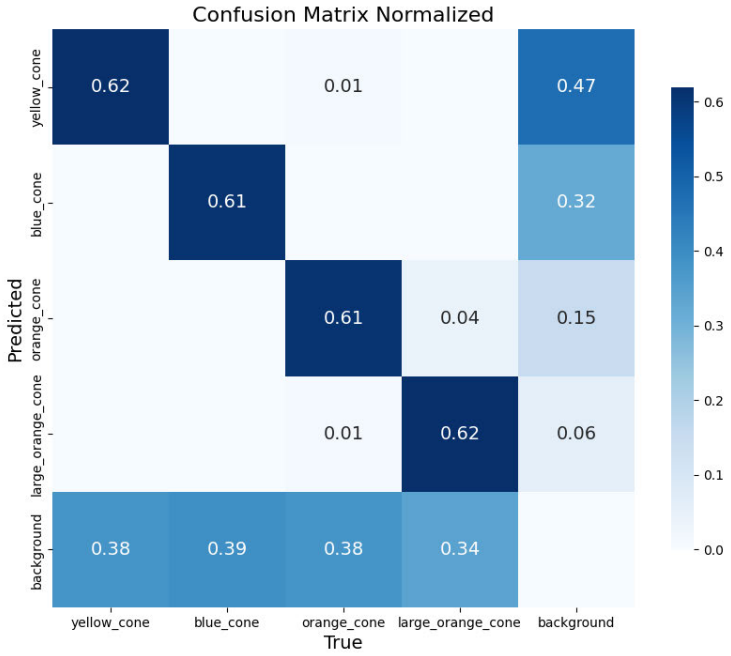


Figure 5.4: Confusion Matrix for RGB-D-M Model Validation at 1440px Resolution

5.2.4 RGB-D-M Model With Mixed Training Dataset Using YOLOv8s

Until now, all the models discussed were using the YOLOv8 nano architecture. To further increase the true positive detections, we experimented with the YOLOv8 small architecture, using the parameters shown in Table 5.4. In this experiment, 6014 real RGB images were used for validation to verify if the model is generalizing across the different images available in the dataset.

Model Evaluation with 1440 Pixel Image

Fig.5.5 illustrates an additional improvement in detection performance, with an increase of 6%-7%. The instances where true labels were mistakenly identified as background require further investigation. In the background, 30% of the objects are classified as false negatives.

Parameter	Value
Training Images	6098
Background Images(%)	10%
Epochs	50
Batch Size	16
Image Size	640
Optimizer	AdamW
Model Architecture	Small
Learning Rate	0.01
Momentum	0.937
Weight Decay	0.0005
Validation Images	6014
Input Channels	5 (RGB-D-M)

Table 5.4: Training Parameters RGB-D-M Model Using Yolov8s

Most of the cones contributing to this value are background cones that are far away from the vehicle and occupy a very small part of the images, further analysis is discussed in the following section 5.3.

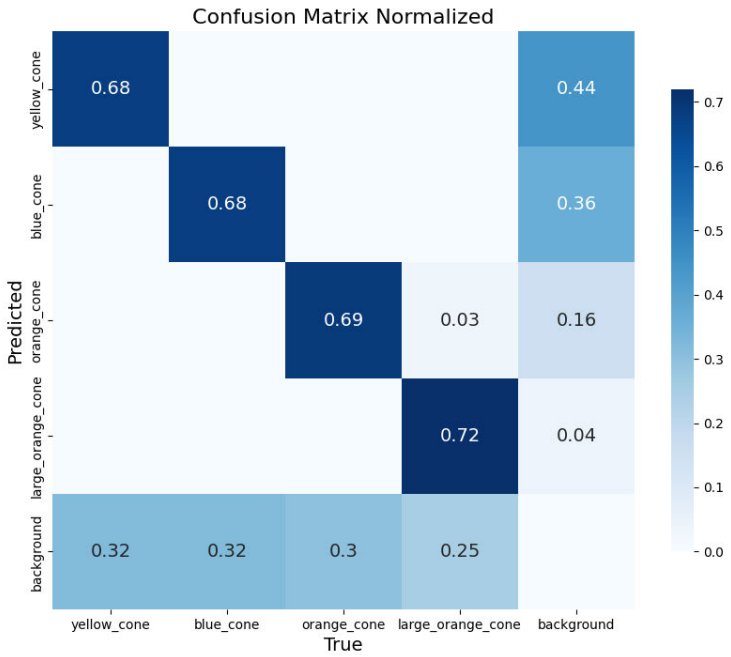


Figure 5.5: Confusion Matrix for RGB-D-M YOLOv8s Model Validation at 1440px Resolution

5.3 Background and Foreground Cone Metrics

In this thesis, we noticed that cones located closer to the vehicle occupy a larger number of pixels, making them more easily detectable. In contrast, cones that are farther away appear smaller and are thus more difficult for the model to detect and classify accurately, as shown

in Fig.5.6. To tackle this challenge, we chose to categorize the metrics into foreground and background cones. Given that this thesis is concerned with detecting cones in AD systems, our primary focus is on accurately identifying cones that are near the vehicle, as these are of greater importance compared to those situated at a greater distance.

Moreover, we only considered images captured from the car's camera, excluding all other images taken indoors or by hand, as they do not provide meaningful insights for studying the detection of foreground and background objects. The dataset used consisted of approximately 3619 images, where inference was performed using an IOU threshold of 0.6 for NMS and a confidence threshold of 0.6. A classification logic was then applied to segregate the cones into foreground and background categories.

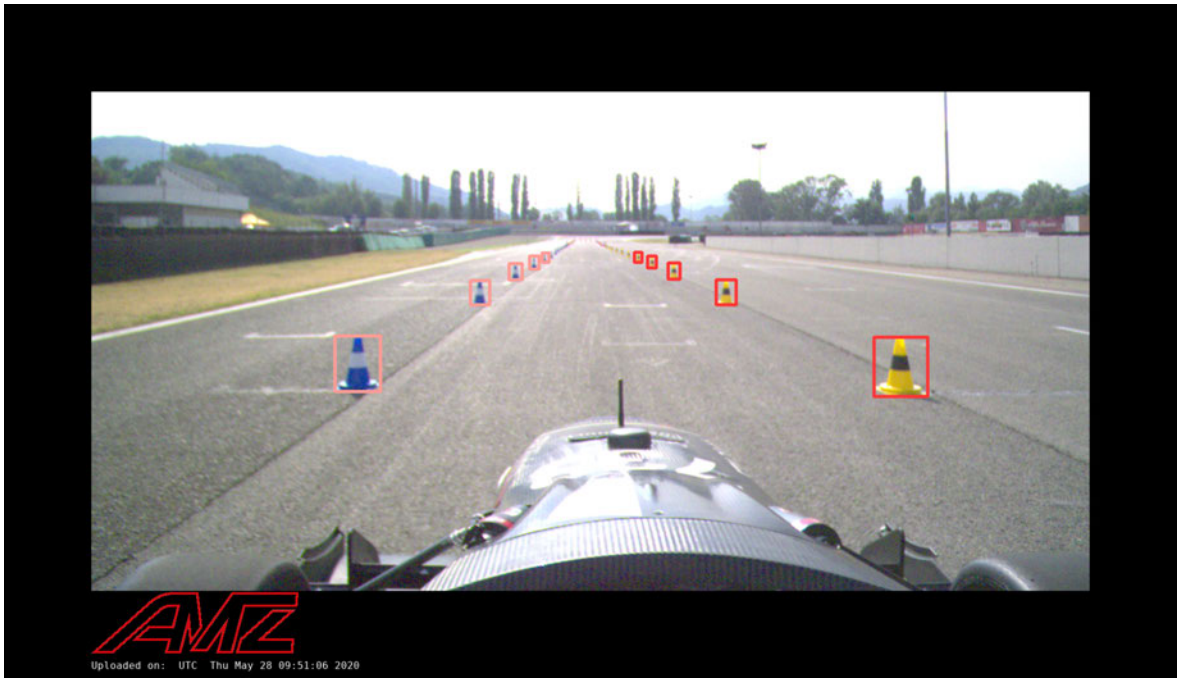


Figure 5.6: Illustration of detection between foreground and background cones

5.3.1 Classification of Cones Based on Size Metrics

To determine whether a cone belongs to the foreground or background, we first calculate the cone's area relative to the image dimensions and then set a threshold to classify it accordingly. Since the images have varying sizes, a consistent method that works across different image dimensions is needed to define the area occupied by the cone and the threshold area. The following subsections outline the concepts used in this thesis for classifying cones as either foreground or background.

Normalized Area Calculation

First, we calculate the normalized area of the cone. The normalized area is determined from the bounding box dimensions normalized with respect to the image size. Specifically:

$$N_{area} = N_{width} \times N_{height}, \quad N_{area}, N_{width}, N_{height} \in \mathbb{R}$$

where:

- N_{width} is the width of the bounding box relative to the image width.
- N_{height} is the height of the bounding box relative to the image height.

The normalized area is a fraction of the image area that the cone occupies and varies based on the size of the image.

Threshold-Based Classification of Cones

To classify the cones, we employ a threshold-based approach using the normalized area of the cones. This classification method compares the normalized area of each cone to a predefined normalized threshold area. Specifically:

- If $N_{area} > N_{threshold\_area}$, the cone is classified as a foreground object.
- If $N_{area} \leq N_{threshold\_area}$, the cone is categorized as a background object.

where:

- $N_{threshold\_area} \in \mathbb{R}$ is the predefined threshold area in normalized units, representing the minimum fraction of the image area required for a cone to be considered as foreground.

This approach ensures a consistent classification of cones based on their size relative to the image, allowing for effective differentiation between foreground and background objects.

Conversion of Normalized Area to Pixel Dimensions

To work with pixel dimensions, the normalized area needs to be converted into pixel units. This is done by scaling the normalized area by the total number of pixels in the image:

$$P_{area} = N_{area} \times Image_{width} \times Image_{height}$$

where:

- $P_{area} \in \mathbb{R}$ is the area in pixels, representing the pixel dimensions in the image.
- $Image_{width} \in \mathbb{R}$ is the width of the image in pixels.
- $Image_{height} \in \mathbb{R}$ is the height of the image in pixels.

Threshold Area Calculation in Pixel Unit

Similarly, the normalized threshold area needs to be converted to pixel dimensions. This conversion is done by:

$$P_{threshold\_area} = N_{threshold\_area} \times Image_{width} \times Image_{height} \tag{5.1}$$

- $P_{threshold\_area} \in \mathbb{R}$ is the threshold area in pixels, representing the pixel dimensions in the image.

By following these steps, we can effectively classify cones based on their size in both normalized and pixel dimensions, distinguishing between foreground and background objects in the image.

5.3.2 Explanation of Threshold Area Interpretation

In our analysis, we use a normalized threshold area to classify cones as either foreground or background. The normalized threshold area is set to 0.0005 to 0.0002, which represents the fraction of the total image area that the cone must occupy to be considered a foreground object. However, due to the varying sizes of images in our dataset, this normalized threshold translates into different pixel thresholds depending on the image dimensions.

Conversion from Normalized to Pixel Area

The normalized threshold area defined in equation 5.1 is applied as follows:

Given:

- $N_{threshold\_area}$ is 0.0005.
- $Image_{width}$ and $Image_{height}$ are the dimensions of the image in pixels.

For clarity, let's look at how this normalization works for different image sizes:

- **Image Size:** 632×920

$$P_{threshold\_area} = 0.0005 \times (632 \times 920) = 290.72 \text{ pixels}$$

- **Image Size:** 1000×1560

$$P_{threshold\_area} = 0.0005 \times (1000 \times 1560) = 780.00 \text{ pixels}$$

- **Image Size:** 1360×1720

$$P_{threshold\_area} = 0.0005 \times (1360 \times 1720) = 1169.60 \text{ pixels}$$

- **Image Size:** 3268×5592

$$P_{threshold\_area} = 0.0005 \times (3268 \times 5592) = 9137.33 \text{ pixels}$$

- **Image Size:** 3256×4248

$$P_{threshold\_area} = 0.0005 \times (3256 \times 4248) = 6915.74 \text{ pixels}$$

Interpretation of Threshold Area in Pixel Units

The normalized threshold area 0.0005 itself is dimensionless and represents the proportion of the total image area. As such, for each image size, it results in different pixel thresholds:

- For smaller images, the pixel threshold is smaller (e.g., 290.72 pixels).
- For larger images, the pixel threshold is larger (e.g., 9137.33 pixels).

By converting the normalized threshold area to pixels using the formula provided, we ensure that the classification of cones as foreground or background is consistent across images of different sizes. This approach maintains a proportionate measure of the cone's size relative to the image, allowing for accurate and meaningful classification regardless of the image dimensions.

In summary, while the normalized threshold area 0.0005 might seem abstract on its own, it is crucial for maintaining consistency in classification across various image sizes. Converting this normalized area into pixel dimensions helps in practical application, adapting the threshold to the specific scale of each image.

5.3.3 Results

In this section, we discuss the results obtained after classifiying the foreground and background metrics based on different thresholds.

Remark 5.5
In all the confusion matrices and bar plots presented in the Results, "Threshold" refers to the Normalized Threshold Area.

RGB-D-M Model,Yolov8s,Normalized Threshold Area=0.0002

Using the RGB-D-M Model which was discussed in the training section, with YOLOv8 small architechture, the results of the foreground and background with absolute numbers are displayed in the Fig. 5.7 and Fig. 5.8.

RGB-D-M Model,Yolov8s,Normalized Threshold Area=0.0005

Using the RGB-D-M Model which was discussed in the training section, with YOLOv8 small architechture, the results of the foreground and background with absolute numbers are displayed in the Fig. 5.9 and Fig. 5.10.

As emphasized earlier, our primary focus is on accurately detecting cones that are close to the vehicle. By comparing the foreground confusion matrices for two different thresholds, shown in Fig. 5.7 and Fig. 5.9, we observe that reducing the threshold—i.e., considering cones closer to the vehicle—leads to a proportional decrease in both false negatives and false positives. Also from the bar graphs shown in the Fig.5.11 and Fig.5.12 provide a detailed breakdown of the model's predictions for foreground cones across all cone colors, showing the number of cones that were detected correctly and those that were not detected. From these plots, we observe that the number of cones not detected is minimal, amounting to less than one-third of the total number of validation images used in the model. This suggests that false detections primarily occur among background cones, which are smaller and more challenging for the model to detect.

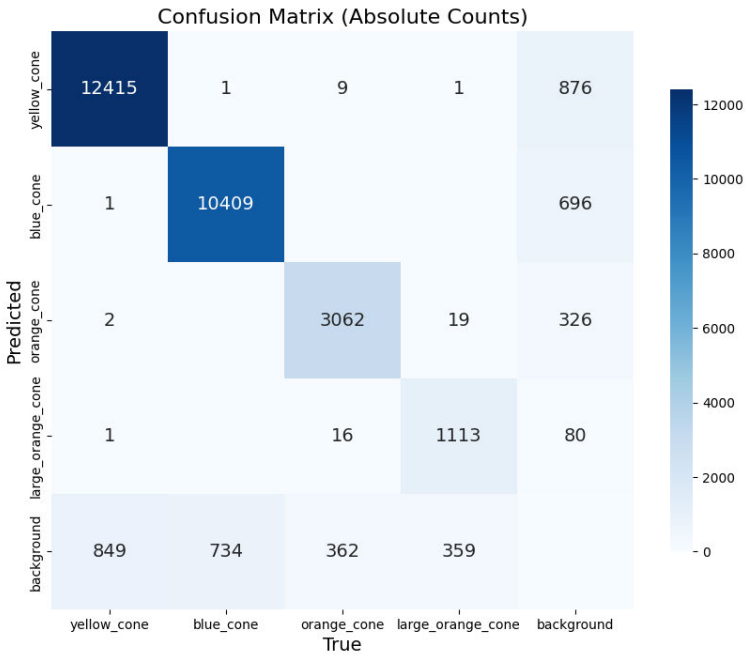


Figure 5.7: Confusion Matrix for RGB-D-M YOLOv8s Foreground Cones (Threshold = 0.0002)

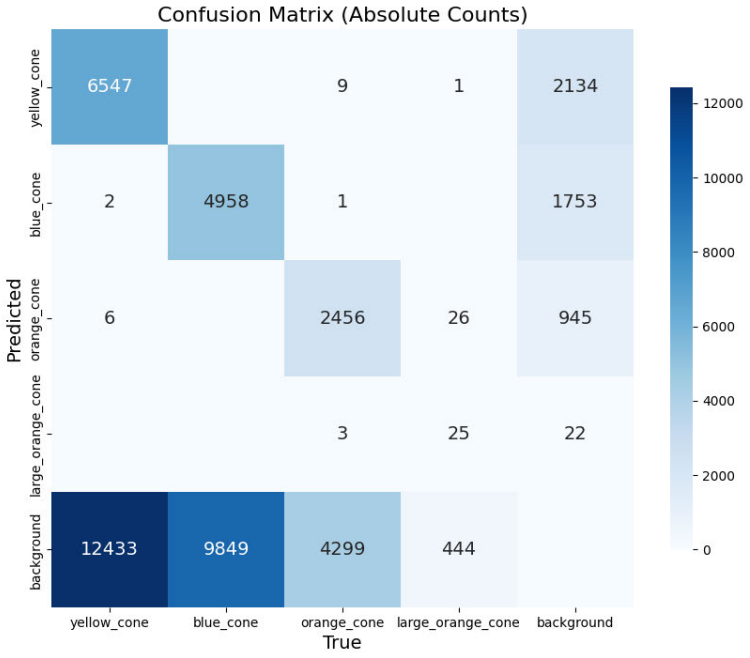


Figure 5.8: Confusion Matrix for RGB-D-M YOLOv8s Background Cones (Threshold = 0.0002)

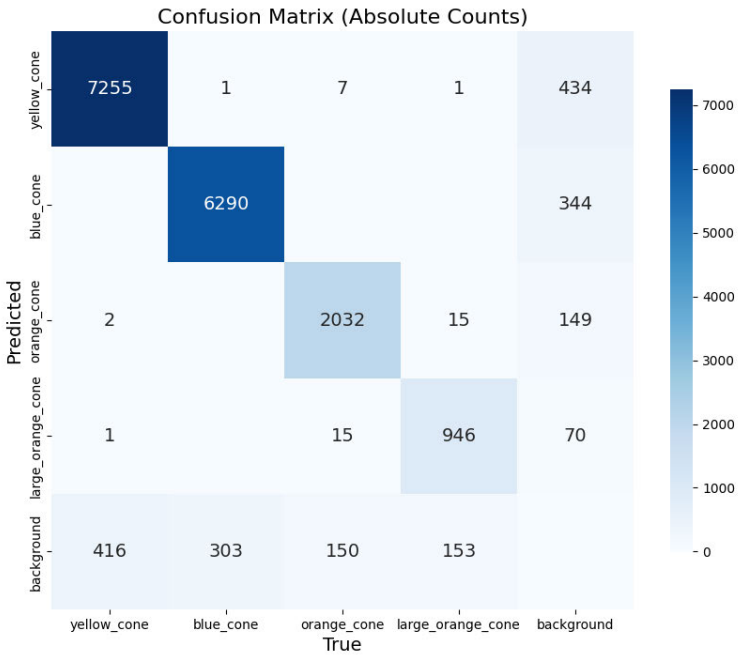


Figure 5.9: Confusion Matrix for RGB-D-M YOLOv8s Foreground Cones (Threshold = 0.0005)

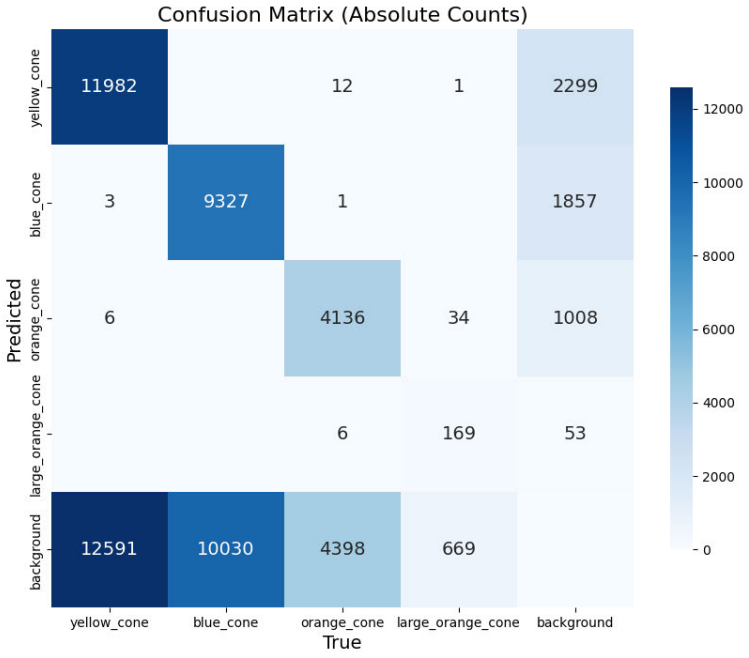


Figure 5.10: Confusion Matrix for RGB-D-M YOLOv8s Background Cones (Threshold = 0.0005)

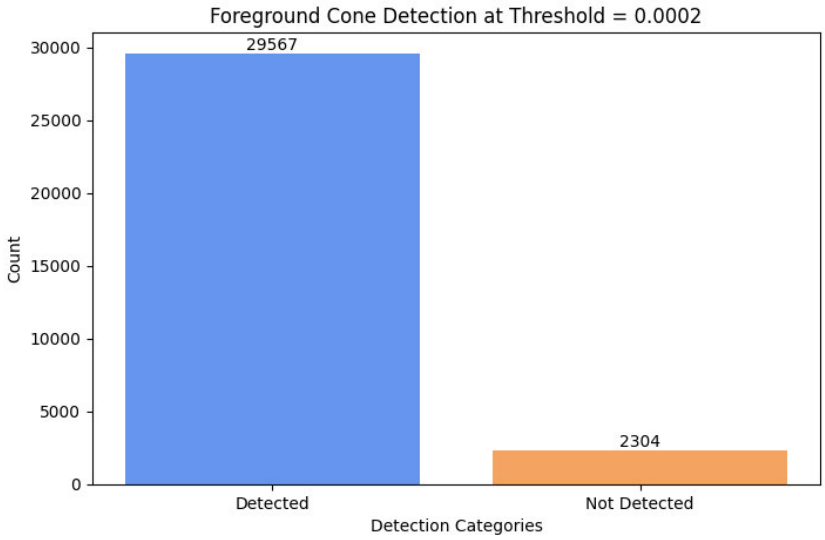


Figure 5.11: Foreground Cone Detection at Threshold = 0.0002

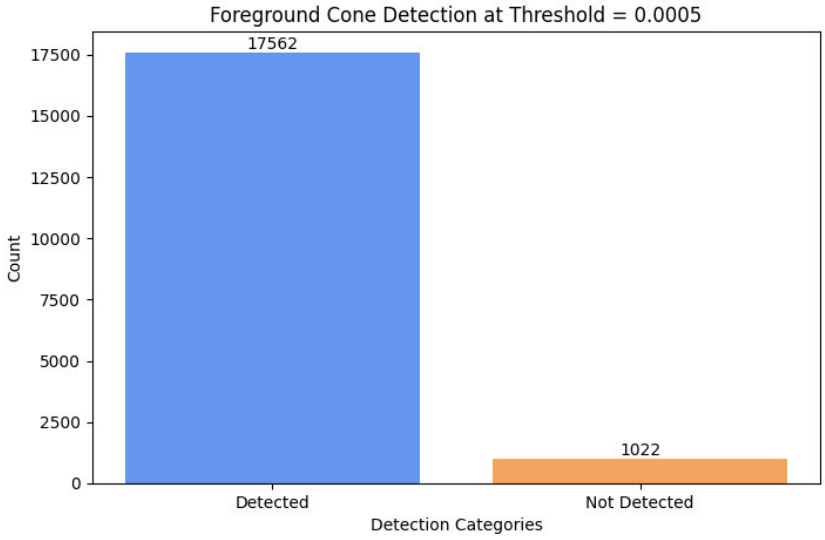


Figure 5.12: Foreground Cone Detection at Threshold = 0.0005

6 Late Fusion Model Implementation

6.1 Objective and Scope

As detailed in Chapter 2, Late Fusion Implementation involves developing two distinct models: the RGB model and the Depth model using the YOLOv8 architechture. Both models are trained and evaluated using synthetic data, followed by a result fusion process during inference to produce the final predictions, as illustrated in Fig. 2.5. This approach aims to assess the effectiveness of this technique for cone detection in AD and to use wieghted boxes fusion [47] for getting the final prediction.

6.2 Training

In this section, we will discuss the training details for both the RGB and Depth models. The subsequent section will cover the concept of WBF and present the final results achieved using this technique.

6.2.1 RGB Model

Initially, the RGB model was trained on a dataset of 2,569 synthetic RGB images, obtained by driving the car through the world and capturing the RGB images and generating the labels to these RGB images using the method detailed in Chapter 4. The training employed the YOLOv8 architecture, specifically the Nano variant, with the following parameters shown in Table.6.1:

Parameter	Value
Training Images	2569
Epochs	25
Batch Size	16
Image Size	640
Optimizer	AadamW
Model Architecture	Nano
Learning Rate	0.01
Momentum	0.937
Weight Decay	0.0005
Validation Images	642
Input Channels	3 (RGB)

Table 6.1: Training Parameters RGB Model

6.2.2 Depth Model

The Depth model was trained on a dataset comprising 2,569 synthetic LiDAR images, generated by mapping point clouds onto the image plane using camera information and perspective projection methods as described in Chapter 4. In these labels, the color of the cones is kept constant since no color information is available for the cones in the LiDAR images. The training employed the YOLOv8 architecture, specifically the Nano variant, with the following parameters shown in Table.6.2:

Parameter	Value
Training Images	2569
Epochs	25
Batch Size	16
Image Size	640
Optimizer	SGD
Model Architecture	Nano
Learning Rate	0.01
Momentum	0.937
Weight Decay	0.0005
Validation Images	642
Input Channels	3 (RGB)

Table 6.2: Training Parameters Depth Model

6.3 Weighted Boxes Fusion

WBF [47] is a post-processing technique that improves object detection results by aggregating overlapping bounding boxes that likely represent the same object. It combines all overlapping detections into a single bounding box. It uses the confidence scores of each box to compute a weighted average of the bounding box coordinates, resulting in a more accurate final prediction. This method addresses the limitations of NMS by preserving valuable information from all detections and reducing false negatives.

6.3.1 WBF vs NMS

WBF and NMS are techniques used to handle overlapping bounding boxes in object detection. NMS retains only the bounding box with the highest confidence score within overlapping regions and discards others based on a fixed IOU threshold. This can lead to suboptimal results, especially if valuable information from less confident detections is discarded. In contrast, WBF aggregates all overlapping bounding boxes into a single prediction by calculating a weighted average of the coordinates, where the weights are proportional to the confidence scores. This approach ensures that all detection data is utilized, reducing false negatives and enhancing overall detection accuracy. WBF is particularly beneficial in scenarios with multiple models or dense overlaps, offering a more reliable and accurate final result than traditional NMS. The comparison between WBF and NMS is illustrated in Figure 6.1.

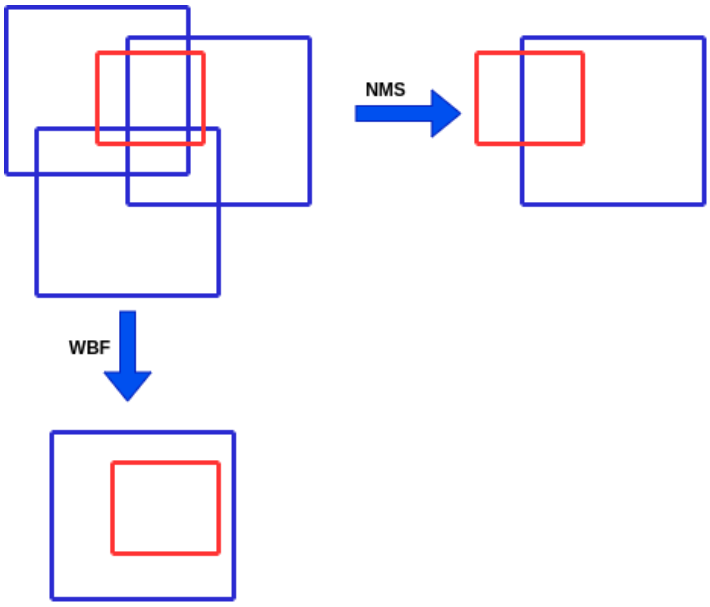


Figure 6.1: Comparison of WBF and NMS

6.3.2 RGB-LiDAR Fusion and Validation Process

In the validation set, which includes synchronized RGB and LiDAR images, the following process is used:

1. **Inference on Image Pairs:** Inference is performed on each synchronized RGB and LiDAR image pair, resulting in predictions from both modalities.
2. **Prediction Fusion:** Predictions from RGB and LiDAR are combined using WBF, creating a unified set of predictions that leverages both modalities.
3. **Comparison with Ground Truth:** The fused predictions are compared against RGB ground truth labels, which provide both spatial and semantic information. Although LiDAR offers spatial data, it lacks color and texture for full classification.
4. **Evaluation and Visualization:** Predictions are evaluated using an IOU threshold to match with ground truth labels, generating a confusion matrix shown in Fig.6.2. This matrix highlights performance across six classes, with the inclusion of a "lidar_fp" class representing false positives from the LiDAR data. These false positives typically arise from background objects misclassified by the Depth model. Notably, 322 objects in the background class were detected as false positives by the Depth model, reflecting the challenges of distinguishing between objects and background in the LiDAR images.

This process allows for a comprehensive evaluation of minimal models created for both RGB and depth data, which are trained and evaluated on synthetic images. The final predictions are obtained through WBF, combining results from both modalities. While the fusion model demonstrates promising performance on synthetic images, it reveals certain limitations, particularly in the Depth model's behavior. Notably, the Depth model occasionally misclassifies background objects with cone-like shapes as actual cones. This highlights a potential drawback of the current approach, stemming from the limited diversity in the training data.

To address this issue, future iterations could benefit from an expanded training dataset that includes a wider variety of background objects with cone-like shapes, but which are not actually cones. This would help the model better distinguish between true cones and cone-shaped background elements. Despite this challenge, the process effectively leverages the complementary strengths of RGB and LiDAR data, with the RGB model providing rich semantic information and the LiDAR model contributing structural information from the point cloud. This structural information is particularly useful when clear visual details are absent or obstructed in the RGB images.

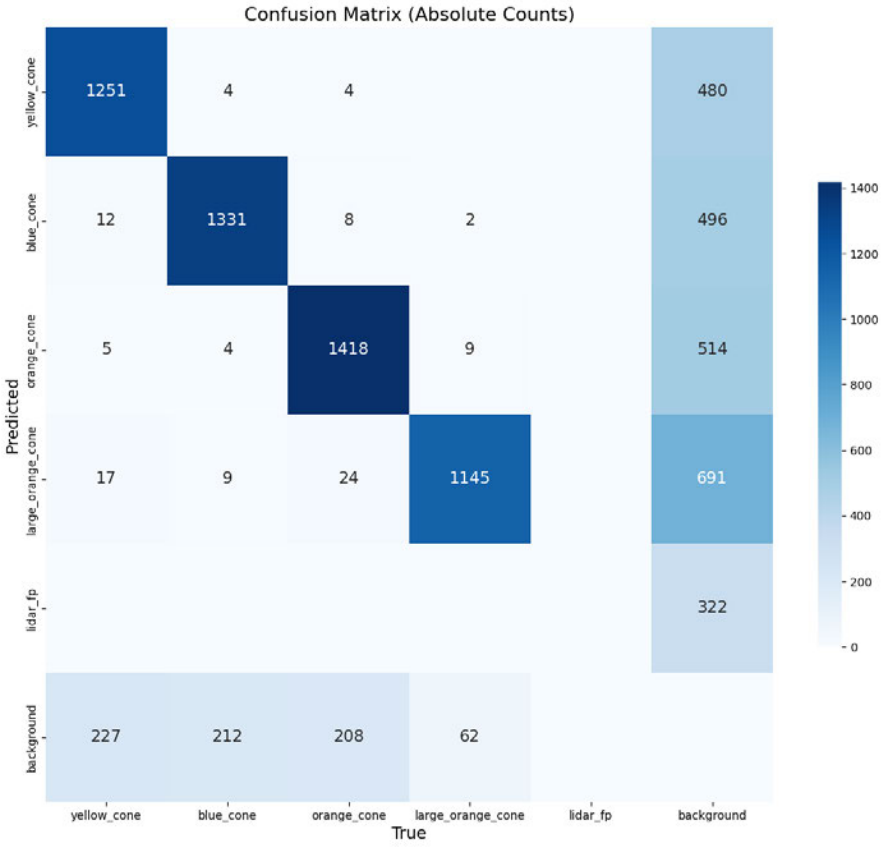


Figure 6.2: Confusion Matrix of Late Fusion Model

7 Conclusion

The primary objective of this thesis was to explore methods for synthesizing high-quality training data using simulation software and to automate the annotation process, aiming to optimize neural networks through early and late fusion techniques for enhanced model generalization from synthetic to real-world scenarios, specifically for AVs. The research began with evaluating various simulation software for generating synthetic data, ultimately selecting the Gazebo simulator for its lightweight design. We then set up a suitable environment, collected camera, LiDAR, and vehicle position data using ROS, and generated RGB-D and LiDAR images for fusion models. Automated labeling was performed using transformation, rotation, and perspective projection techniques.

Initially, we implemented an early fusion technique with RGB-D models. These models performed poorly on real-world images due to their heavy reliance on depth information, leading to high false negative rates and unreliable predictions. Adding a mask channel to the synthetic images improved true positive rates slightly but did not fully address the domain gap and depth reliance issues. Introducing approximately 15% real images to the dataset significantly improved performance, with a twofold increase in true positive rates for cone detection. Despite this, the model still struggled with generalization and misclassification, particularly between similar cone types and backgrounds, partly due to image resizing.

Increasing the image resolution to 1440px during validation notably enhanced detection accuracy, improving true positive rates by at least 35%. Using a smaller variant of Yolov8 further improved detection by 6-7%. We observed that cones closer to the vehicle, occupying more pixels, were classified as foreground, while those farther away were considered background. A threshold-based approach with normalized and pixel area calculations ensured consistent classification across image sizes. Lowering the threshold reduced both false negatives and false positives. Analysis of confusion matrices and bar graphs indicated minimal undetected cones, with false detections mostly among smaller, distant background cones.

In the final step, we implemented a late fusion model with WBF to combine RGB and depth data for final predictions. This approach effectively leverages the strengths of both modalities, though the Depth model sometimes misclassifies cone-like background objects.

This thesis provides a comprehensive approach to synthesizing and annotating training data for AVs, highlighting the importance of combining synthetic and real data to bridge domain gaps. These domain gaps between synthetic and real images can be better addressed by using game engines, which can more accurately mimic real-world scenarios than Gazebo. Further improvements can be made in fusion techniques, and additional methods should be investigated to enhance the detection of smaller and distant objects. Additionally, optimizing inference speed for smaller image sizes could improve detection performance compared to using larger images.

Bibliography

[1] "Safety in higher level automated vehicles: Investigating edge cases in crashes of vehicles equipped with automated driving systems," *Accident Analysis & Prevention*, vol. 203, p. 107 607, 2024, ISSN: 0001-4575. DOI: https://doi.org/10.1016/j.aap.2024.107607. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0001457524001520.

[2] F. Klaver, "The economic and social impacts of fully autonomous vehicles," *Compact*, 2023, Accessed: 2024-08-16. [Online]. Available: https://www.compact.nl/articles/the-economic-and-social-impacts-of-fully-autonomous-vehicles/.

[3] K. Korosec, "Intel: Self-driving cars will rack up 7 trillion miles by 2050 and create a 7 trillion industry," *The Verge*, 2017, Accessed: 2024-08-16. [Online]. Available: https://www.theverge.com/2017/6/1/15725516/intel-7-trillion-dollar-self-driving-autonomous-cars.

[4] U.S. Chamber of Commerce, *Unlocking the social and economic benefits of autonomous vehicles*, https://www.uschamber.com/technology/unlocking-the-social-and-economic-benefits-of-autonomous-vehicles, Accessed: 2024-08-16, 2023.

[5] N. Vödisch, D. Dodel, and M. Schötz, "Fsoco: The formula student objects in context dataset," *SAE International Journal of Connected and Automated Vehicles*, vol. 5, no. 12-05-01-0003, 2022.

[6] S. D. Pendleton *et al.*, "Perception, planning, control, and coordination for autonomous vehicles," *Machines*, vol. 5, no. 1, 2017, ISSN: 2075-1702. DOI: 10.3390/machines5010006. [Online]. Available: https://www.mdpi.com/2075-1702/5/1/6.

[7] A. Faisal, M. Kamruzzaman, T. Yigitcanlar, and G. Currie, "Understanding autonomous vehicles," *Journal of transport and land use*, vol. 12, no. 1, pp. 45–72, 2019.

[8] D. Christie, A. Koymans, T. Chanard, J.-M. Lasgouttes, and V. Kaufmann, "Pioneering driverless electric vehicles in europe: The city automated transport system (cats)," *Transportation Research Procedia*, vol. 13, pp. 30–39, Dec. 2016. DOI: 10.1016/j.trpro.2016.05.004.

[9] R. Krueger, T. H. Rashidi, and J. M. Rose, "Preferences for shared autonomous vehicles," *Transportation Research Part C: Emerging Technologies*, vol. 69, pp. 343–355, 2016, ISSN: 0968-090X. DOI: https://doi.org/10.1016/j.trc.2016.06.015. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0968090X16300870.

[10] Z. Wadud and G. Mattioli, "Fully automated vehicles: A cost-based analysis of the share of ownership and mobility services, and its socio-economic determinants," *Transportation Research Part A: Policy and Practice*, vol. 151, pp. 228–244, 2021, ISSN: 0965-8564. DOI: https://doi.org/10.1016/j.tra.2021.06.024. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0965856421001737.

[11] A. Nerds, *Trends and the future of autonomous vehicles*, Accessed: 2024-09-12, 2024. [Online]. Available: https://angrynerds.co/blog/trends-and-the-future-of-autonomous-vehicles/#:~:text=Autonomous%20Driving%20Software,driving%20force%20behind%20this%20trend.

[12] McKinsey & Company, "Autonomous driving's future: Convenient and connected," 2021, Accessed: 2024-07-25. [Online]. Available: https://www.mckinsey.com/industries/automotive-and-assembly/our-insights/autonomous-drivings-future-convenient-and-connected.

[13] IIoT World, "Challenges in training algorithms for autonomous cars," 2024, Accessed: 2024-07-25. [Online]. Available: https://www.iiot-world.com/artificial-intelligence-ml/autonomous-vehicles/challenges-in-training-algorithms-for-autonomous-cars/.

[14] K. Muhammad, A. Ullah, J. Lloret, J. D. Ser, and V. H. C. de Albuquerque, "Deep learning for safe autonomous driving: Current challenges and future directions," *IEEE Transactions on Intelligent Transportation Systems*, vol. 22, no. 7, pp. 4316–4336, 2021. DOI: 10.1109/TITS.2020.3032227.

[15] Basic.ai. "Data annotation for autonomous driving." Accessed: 2024-07-25. (2021), [Online]. Available: https://www.basic.ai/blog-post/data-annotation-for-autonomous-driving.

[16] Admon W, *Data annotation for autonomous driving: Key techniques and challenges*, https://www.basic.ai/blog-post/data-annotation-for-autonomous-driving, Accessed: 2024-08-16, 2023.

[17] A. Geiger, P. Lenz, C. Stiller, and R. Urtasun, "Vision meets robotics: The kitti dataset," *International Journal of Robotics Research (IJRR)*, 2013.

[18] F. Yu *et al.*, "Bdd100k: A diverse driving dataset for heterogeneous multitask learning," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun. 2020.

[19] M. Cordts *et al.*, "The cityscapes dataset," in *CVPR Workshop on The Future of Datasets in Vision*, 2015.

[20] P. Sun *et al.*, *Scalability in perception for autonomous driving: Waymo open dataset*, 2020. arXiv: 1912.04838 [cs.CV]. [Online]. Available: https://arxiv.org/abs/1912.04838.

[21] H. Caesar *et al.*, "Nuscenes: A multimodal dataset for autonomous driving," in *CVPR*, 2020.

[22] P. Kaur, S. Taghavi, Z. Tian, and W. Shi, *A survey on simulators for testing self-driving cars*, 2021. arXiv: 2101.05337 [cs.RO]. [Online]. Available: https://arxiv.org/abs/2101.05337.

[23] Gazebo Simulator, *Gazebo classic*, Accessed: 2024-07-26, 2024. [Online]. Available: https://classic.gazebosim.org/.

[24] Gazebo Simulator, *About gazebo*, Accessed: 2024-07-26, 2024. [Online]. Available: https://gazebosim.org/about.

[25] Wikipedia contributors, *Simulink*, Accessed: 2024-07-26, 2024. [Online]. Available: https://en.wikipedia.org/wiki/Simulink.

[26] HERE Technologies, *Here hd live map*, Accessed: 2024-07-26, 2024. [Online]. Available: https://www.here.com/platform/HD-live-map.

[27] ASAM, *Opendrive*, Accessed: 2024-07-26, 2024. [Online]. Available: https://www.asam.net/standards/detail/opendrive/.

[28] MathWorks, *Ground truth labeler app*, Accessed: 2024-07-26, 2024. [Online]. Available: https://de.mathworks.com/help/driving/ref/groundtruthlabeler-app.html.

[29] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun, "CARLA: An open urban driving simulator," in *Proceedings of the 1st Annual Conference on Robot Learning*, 2017, pp. 1–16.

[30] G. Rong *et al.*, "Lgsvl simulator: A high fidelity simulator for autonomous driving," *arXiv preprint arXiv:2005.03778*, 2020.

[31] F. S. Germany. "Driving an autonomous vehicle in a 3d simulation environment with recorded synthetic data." Accessed: 2024-09-12. (2021), [Online]. Available: https://www.formulastudent.de/pr/news/details/article/driving-an-autonomous-vehicle-in-a-3d-simulation-environment-with-recorded-synthetic-data/.

[32] Open Source Robotics Foundation, *Robot operating system (ros)*, Accessed: 2024-07-26, 2024. [Online]. Available: https://www.ros.org/.

[33] Blender Foundation, *Blender*, Accessed: 2024-07-26, 2024. [Online]. Available: https://www.blender.org/.

[34] T. Zhou, D.-P. Fan, M.-M. Cheng, J. Shen, and L. Shao, "Rgb-d salient object detection: A survey," *Computational Visual Media*, vol. 7, no. 1, pp. 37–69, Jan. 2021, ISSN: 2096-0662. DOI: 10.1007/s41095-020-0199-z. [Online]. Available: http://dx.doi.org/10.1007/s41095-020-0199-z.

[35] K. Gadzicki, R. Khamsehashari, and C. Zetzsche, "Early vs late fusion in multimodal convolutional neural networks," in *2020 IEEE 23rd international conference on information fusion (FUSION)*, IEEE, 2020, pp. 1–6.

[36] Z. Zou, K. Chen, Z. Shi, Y. Guo, and J. Ye, "Object detection in 20 years: A survey," *Proceedings of the IEEE*, vol. 111, no. 3, pp. 257–276, 2023.

[37] S. Ruder, *An overview of gradient descent optimization algorithms*, 2017. arXiv: 1609.04747 [cs.LG]. [Online]. Available: https://arxiv.org/abs/1609.04747.

[38] I. Loshchilov and F. Hutter, *Decoupled weight decay regularization*, 2019. arXiv: 1711.05101 [cs.LG]. [Online]. Available: https://arxiv.org/abs/1711.05101.

[39] Y. Saito, *Vehicle_sim*, Accessed: 2024-07-31, 2021. [Online]. Available: https://github.com/yukkysaito/vehicle_sim.

[40] MathWorks, *Coordinate transformations in robotics*, Accessed: 2024-07-03, 2024. [Online]. Available: https://de.mathworks.com/help/robotics/ug/coordinate-transformations-in-robotics.html.

[41] OpenCV, *Opencv documentation: Camera calibration and 3d reconstruction*, Accessed: 2024-07-31, 2024. [Online]. Available: https://docs.opencv.org/4.x/d9/d0c/group__calib3d.html.

[42] Amro, *Cv.projectpoints*, Accessed: 2024-09-07, 2024. [Online]. Available: https://amroamroamro.github.io/mexopencv/matlab/cv.projectPoints.html.

[43] G. Jocher, A. Chaurasia, and J. Qiu, *Ultralytics YOLO*, version 8.0.0, Jan. 2023. [Online]. Available: https://github.com/ultralytics/ultralytics.

[44] M. Sohan, T. Sai Ram, and C. V. Rami Reddy, "A review on yolov8 and its advancements," in *Data Intelligence and Cognitive Informatics*, I. J. Jacob, S. Piramuthu, and P. Falkowski-Gilski, Eds., Singapore: Springer Nature Singapore, 2024, pp. 529–545, ISBN: 978-981-99-7962-2.

[45] Roboflow, *What's new in yolov8?* Accessed: 2024-07-25, 2023. [Online]. Available: https://blog.roboflow.com/whats-new-in-yolov8/.

[46] Ultralytics, *Issue 3432: Yolov8 rgbd*, https://github.com/ultralytics/ultralytics/issues/3432, Accessed: 2024-07-25, 2024.

[47] R. Solovyev, W. Wang, and T. Gabruseva, "Weighted boxes fusion: Ensembling boxes from different object detection models," *Image and Vision Computing*, pp. 1–6, 2021.

Declaration of Authorship

I do hereby declare that I have composed the presented work independently on my own and without any other resources than the ones given.

All thoughts taken directly or indirectly from external sources are correctly acknowledged.

This work has neither been previously submitted to another authority nor has it been published yet.

Mittweida, 14. September 2024

Location, Date

Kedharnath Kuruba Basvaraj