DE-43 (DC & SE)

**NUST COLLEGE OF**
**ELECTRICAL AND MECHANICAL ENGINEERING**

# LiDAR based Object Detection and Tracking for NUSTAG Electrical Vehicle

A PROJECT REPORT

DE-43 (DC & SE)

***Submitted by***


NS SYED MUHAMMAD IRTAZA HYDER

NS MUHAMMAD ZAKRIA MEHMOOD

NS MUHAMMAD ABDULLAH


**BACHELORS**

**IN**

**COMPUTER ENGINEERING**

**YEAR**

**2025**

**Project Supervisor**

PROF. USMAN AKRAM

PROF. FAHAD MUMTAZ MALIK


DEPARTMENT OF COMPUTER & SOFTWARE ENGINEERING
COLLEGE OF ELECTRICAL & MECHANICAL ENGINEERING
NATIONAL UNIVERSITY OF SCIENCES AND TECHNOLOGY,
ISLAMABAD, PAKISTAN


YEAR 2025

# Certification

This is to certify that Syed Muhammad Irtaza Hyder [378514], Muhammad Zakria Mehmood [391449] and Muhammad Abdullah [372567] have successfully completed the final project LiDAR based Object Detection and Tracking for NUSTAG Electrical Vehicle, at the NUST College of Electrical and Mechanical Engineering, to fulfill the partial requirement of the degree Bachelors in Computer Engineering.

Signature of Project Supervisor
Prof. Usman Akram
Head of Department

# Sustainable Development Goals (SDGs)

| SDG No | Description of SDG | SDG No | Description of SDG |
|---|---|---|---|
| SDG 1 | No Poverty | **SDG 9** | **Industry, Innovation, and Infrastructure** |
| SDG 2 | Zero Hunger | SDG 10 | Reduced Inequalities |
| SDG 3 | Good Health and Well Being | **SDG 11** | **Sustainable Cities and Communities** |
| SDG 4 | Quality Education | SDG 12 | Responsible Consumption and Production |
| SDG 5 | Gender Equality | SDG 13 | Climate Change |
| SDG 6 | Clean Water and Sanitation | SDG 14 | Life Below Water |
| SDG 7 | Affordable and Clean Energy | SDG 15 | Life on Land |
| SDG 8 | Decent Work and Economic Growth | SDG 16 | Peace, Justice and Strong Institutions |
| | | SDG 17 | Partnerships for the Goals |


Sustainable Development Goals

# Complex Engineering Problem

**Range of Complex Problem Solving**

| | **Attribute** | **Complex Problem** | |
|---|---|---|---|
| 1 | Range of conflicting requirements | Involve wide-ranging or conflicting technical, engineering and other issues. | ✓ |
| 2 | Depth of analysis required | Have no obvious solution and require abstract thinking, originality in analysis to formulate suitable models. | |
| 3 | Depth of knowledge required | Requires research-based knowledge much of which is at, or informed by, the forefront of the professional discipline and which allows a fundamentals-based, first principles analytical approach. | ✓ |
| 4 | Familiarity of issues | Involve infrequently encountered issues | ✓ |
| 5 | Extent of applicable codes | Are outside problems encompassed by standards and codes of practice for professional engineering. | |
| 6 | Extent of stakeholder involvement and level of conflicting requirements | Involve diverse groups of stakeholders with widely varying needs. | |
| 7 | Consequences | Have significant consequences in a range of contexts. | |
| 8 | Interdependence | Are high level problems including many component parts or sub-problems | ✓ |

**Range of Complex Problem Activities**

| | **Attribute** | **Complex Activities** | |
|---|---|---|---|
| 1 | Range of resources | Involve the use of diverse resources (and for this purpose, resources include people, money, equipment, materials, information and technologies). | ✓ |
| 2 | Level of interaction | Require resolution of significant problems arising from interactions between wide ranging and conflicting technical, engineering or other issues. | ✓ |
| 3 | Innovation | Involve creative use of engineering principles and research-based knowledge in novel ways. | ✓ |
| 4 | Consequences to society and the environment | Have significant consequences in a range of contexts, characterized by difficulty of prediction and mitigation. | |
| 5 | Familiarity | Can extend beyond previous experiences by applying principles-based approaches. | |

*Dedicated to the family, friends, and professors who support us throughout our university journey.*

# Acknowledgment

First, we express our humblest gratitude to the Divine, Allah Almighty, for giving us the strength and fortitude to carry out this endeavor. All our efforts were guided towards a positive outcome, and help prevailed over us in times of uncertainty.

We thank our project supervisor, Dr. Usman Akram, for his unwavering help and support throughout the project. His mentorship and guidance provided us with the motivation to give our best and strive for excellence.

We also acknowledge our co-supervisor, Dr. Fahad Mumtaz Malik, for his crucial assistance and support during the project. He provided all equipment, including Ouster LiDAR, the NUSTAG EV, Jetson Xavier AGX, and UAV lab access. Furthermore, we thank Dr. Usman Akbar, who provided us with resources to train our Deep Learning models. We also extend our thanks to Muneeb Ahmad from MTS-43 (C) for his help in designing the LiDAR mount and aiding us in attaching it with the Autonomous Vehicle.

Finally, we want to thank our family and friends for their encouragement and support throughout this effort.

# Abstract

LiDAR (Light Detection and Ranging) is a remote sensing technique that estimates distance via laser pulses to obtain a RANGE image. From the RANGE image derived, a point cloud is obtained from which pseudo-BEV images are created. LiDAR is a crucial sensor in the perception task stack for autonomous driving, as it provides vehicles with a very detailed view of their environment, regardless of light, with minimal interference from environmental weather factors. Major players in the industry, Waymo and Zoox, make extensive use of LiDAR for navigation that is safe and reliable.

The final year project seeks to build on this foundation in improving real-time autonomous perception for the NUSTAG Electric Vehicle through LiDAR-based object detection and tracking. Fast inference on an energy-efficient platform, Nvidia Jetson Xavier AGX, was achieved through the development of an efficient process pipeline-an object detector using the YOLO model, acceleration of YOLO on tensor cores by TensorRT, and ByteTrack object tracking.

This system has been installed on the NUSTAG EV-an electric vehicle assembled by the NUSTAG team-and has been tested using simulated environments. Test results show accuracy and real-time performance, making this approach ideal for autonomous vehicle applications. This work is part of an extensive effort to make fully automated electric vehicles in later phases of development.

**Keywords:** Autonomous Systems, Complex YOLO, Embedded AI, Embedded Systems, LiDAR, NVIDIA Xavier, Object Detection, Object Tracking, Perception pipeline, Real-time Processing, TensorRT Acceleration, YOLO

# Contents

# List of Figures

# List of Tables

# Chapter 1

# Introduction

## 1.1 Introduction

Recent progressive steps in autonomous systems and edge computing have fostered a plethora of intelligent technologies like self-driving cars, robotics, and smart cities. Among these technologies, LiDAR-based sensing has become a key player in reliable object detection due to its ability to provide accurate 3D spatial representation of complex environments [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15]. LiDAR poses numerous challenges for realtime deployment on edge devices and very much-used embedded platforms due to its heavy computational requirements [16]. So far, the scene has relied on powerful GPUs hosted in the cloud or upon lightweight 2D camera systems [17]; these, however, do not allow for the robust real-time on-device 360-degree perception needed for those autonomous vehicles acting in dynamic environments.

The present work describes an optimized pipeline for LiDAR-based object detection and tracking based on a YOLO-based architecture [18], leveraging TensorRT for efficient deployment on edge devices like the NVIDIA Jetson series [19]. We have implemented the proposed system into the NUSTAG EV, an electric vehicle developed at the National University of Sciences and Technology, and examined it through real-time testing within a campus environment. The results showed that the system exhibited high accuracy and low latency performance, establishing it as a solution for real-world applications. While basic

automation functionalities such as braking and throttle control are supported in the current setup, the objectives of future versions will be to push for higher levels of autonomy at Level 3 and Level 4 in autonomous driving.

## 1.2 Motivation

Camera-based perception systems have long dominated the autonomous driving market due to their low cost and high-resolution imaging capabilities. However, in recent year other perception sensors, such as the LiDAR, have significantly matured. Utilizing light rays to generate a 3D map of the environment, it has become a popular choice for environmental awareness and depth perception.

With this improvement, two main school of thoughts have emerged. On one hand we have Waymo's sensor-fusion model, which heavily integrates LiDAR, and on the other Tesla's vision-only strategy, which depends solely on cameras. Despite camera's affordability and availability, camera systems have trouble with low-light conditions and depth estimation. In contrast, LiDAR-centric strategy improves detection and navigation accuracy by providing precise 3D spatial data.

In light of this, the project aims to incorporate LiDAR-based perception to replace/enchance the NUSTAG autonomous vehicle's perception stack, which previously only used a front view camera. The end goal is to develop a dependable and safe system for efficient detection and tracking of pedestrians, cyclists and cars.

## 1.3 Problem Statement

Our project, LiDAR based Object Detection and Tracking for NUSTAG Autonomous Vehicle tackles the challenge of perception. Initially the vehicle used a single mono-camera for road segmentation and path planning. Our goal was to introduce the LiDAR sensor to enhance the perception of the vehicle. LiDAR provided the vehicle with a 360

degree FoV alongside being tolerant and robust even in more tougher weather conditions (such as rain and fog) in comparison to the camera.

## 1.4 Scope

The scope of this project was to utilize the LiDAR as the primary sensor in the perception for the NUSTAG autonomous vehicle. It tackles the computational bottlenecks faced by LiDAR-based perception for edge devices by means of a hardware-accelerated architecture for TensorRT optimized YOLO [16, 20, 18, 21].

The high-performance pipeline that we have deployed on the NUSTAG EV performed well on campus roads, proving to be scalable and efficient. On top of that, the system's low-latency and high-accuracy performance render it applicable in other areas like robotics, security, and agriculture, especially where edge devices operate in remote or resource-constrained settings. Running complex models like YOLO on low-power platforms is, in a broader sense, contributing to sustainable AI.

## 1.5 Aims and Objectives

### 1.5.1 Object Detection

The first objective of this project was to detect and classify objects using the point cloud data generated by LiDAR input

### 1.5.2 Object Tracking

The second objective of this project was to track the objects being detected. So that we can estimate their trajectory and avoid any collisions.

### 1.5.3 Deployment Goal

The aim of this project is to deploy the perception system on low powered edge device. For that, the system have to have a high accuracy, low latency and low resource consumption as it is meant to be deployed on an edge device. As the edge device would also be running many other processes, such as navigation and communication with other sensors, the developed system must be as small and efficient as possible.

## 1.6 Outcomes

### 1.6.1 Object Detector

We processed the 3D point cloud into 2D RGB-BEV images. Then we used `yolo11s-obb` as the 2D backbone of our Object Detector.

### 1.6.2 Object Tracker

After detection comes tracking. For this purpose, we used the ByteTrack multi-object tracking algorithm. We extended its functionality to perform what we call a *Multi-step Look ahead*.

### 1.6.3 Simulations

Testing and validation of the Deep Learning Model in the Gazebo simulation environment. Test environments were developed where there were pedestrians, cars and a mix of both to test the models accuracy in detecting and tracking their trajectories.

## 1.7 Report Organization

The organization of the thesis is as follows:

### 1.7.1 Chapter 2

This chapter lists and references the literature reviewed before starting the project. It is further divided into 4 parts: LiDAR sensor, Edge Computing, Detection, and Dataset.

### 1.7.2 Chapter 3

This chapter includes the requirements to develop deep learning neural network models, the edge devices, and dataset used.

### 1.7.3 Chapter 4

This chapter discusses the object detection model used, how it was trained and optimized and its performance on the KITTI dataset.

### 1.7.4 Chapter 5

This chapter discusses the object tracking algorithm, its foundations and its implementation.

### 1.7.5 Chapter 6

This chapter discusses the simulator Gazebo, the development of the digital twins, the simulation environments and the testing of the models in the simulated environment.

### 1.7.6 Chapter 7

This chapter discusses the ROS2 workflow used to develop the visualizer. It also introduces features such as viewing live, recording and replaying sensor data obtained from the simulator and visualizer.

### 1.7.7 Chapter 8

This chapter includes the detail of deploying our model on the Jetson Xavier and discusses the future work for working on the Jetson.

# Chapter 2

# Background & Related Work

This chapter deals with the essential conception and previous research on what can be called LiDAR-based object detection and tracking, particularly concerning the kinds of solutions that would be intelligent enough to use edge computing platforms. It also goes on to discuss how LiDAR can be an answer to understanding the reading of point clouds, the advances made in the field during research into multiple and unprecedented target detection algorithms, model efficiency for real-time performance, and the role of benchmark datasets. It provides the environment for the contributions made here in this thesis, specifically on an optimized pipeline for LiDAR object detection and tracking for an edge device such as the NVIDIA Jetson family. The challenges of deploying fast algorithms for perception at this level, and how they are addressed, form a major component of the comparative study related to this work. The generation of dense, accurate 3D point clouds of the environment provides an abundance of spatial information so crucial for tasks ranging from object detection to localization and mapping [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15]. Processing LiDAR data is very challenging, requiring the presence of high computing power due to the very high points output per second, as well as the complexity associated with the algorithms used to extract meaning from that point cloud. This poses real challenges, especially in real-time applications,

using the low resource configuration [16]. Conventional approaches have used high-end GPU processor power in the cloud or relatively less costly systems based on 2D cameras. Cloud processing is very effective because it provides huge computational capacity [17].

## 2.1 LiDAR Technology for Autonomous Perception

The capability of robotic building blocks combined with actuator control is the generation of self-directing vehicles. It will be possible by LiDAR based systems that detect and differentiate obstacles from free space and respond they make real-time autonomous navigation possible even in dynamic environments.

LiDAR went further into being the crucial and real sensing technology towards autonomous vehicles and robotics, with detailed advantages against other perceptions modalities such as cameras and radar. It generates images capturing different perspectives: that view from above and down and around an object—providing a much broader comprehension of the space in all directions, making it essential for the whole understanding of a large environment.

LiDAR technology examination encompasses this subject, given its operational principles, advantages and limitations in real-world applications. These limitations include issues such as global practicality, covariance with nonautonomous platform systems, and the requirement to process substantial amounts of LiDAR data fast and efficiently.

The perception of environment by LiDAR systems with high-performance computing and real-time data processing equips an autonomous platform for smart decision-making and autonomous mobility in any situation with actuator control.

### 2.1.1 Challenges of LiDAR Processing

Despite its advantages, LiDAR technology also presents several challenges:

- **Cost of Computation:** A massive amount of data is continuously generated by the

LiDAR sensors, which contains several processing works like filtering, segmentation, object detection, and tracking [16] for real-time applications. This becomes a significant obstacle to realizing deployment in embedded or edge devices, which typically have limited processing power and memory.

- **Sensitive to Harsh Weather:** In many ways, LiDAR performs better than cameras, but the performance can degrade when it comes to severe weather conditions like heavy rains, snowstorms, or fog, all of which scatter or absorb the laser beams, resulting in very noisy or sparse data.
- **Sparsity over Long Distances:** The density of point clouds reduces with distance, and that makes it increasingly difficult to detect and classify objects at long ranges.
- **Cost and Mechanical Complexity:** The high-performance LiDAR sensors have, in the past, been costly ones with moving mechanical parts, although solid-state LiDAR is becoming common and addressing some of these issues.
- **Absence of Color/Texture Information:** The LiDAR point cloud is sparse in terms of color and texture information available from cameras, which capture close-up, intimate views beneficial for classifying objects. This often drives sensor fusion ideas.

The work presented in this thesis directly addresses the computational cost challenge by developing an efficient pipeline optimized for edge devices.

## 2.2 Edge Computing for Autonomous Systems

Although edge computing generally refers to the processing of data near its capture source rather than relaying it to a centralized cloud or data center, for autonomous vehicles, the term edge usually refers to computing platforms present onboard the respective vehicles.

### 2.2.1 Edge Computing for Autonomous Vehicles: Motivation and Platform

Edge computing is of crucial value for the scaling/designing of intelligent, real-time operations to autonomous vehicles. The following motivations make processing data locally at the vehicle level important:

- **Low Latency:** It is essentially real-time decision making that relates to the safety requirements. For example, emergency braking or when an object must be avoided in a very critical situation. All the processing occurs on the vehicle device, which is very much lower in latency than what would be all the computation for the scenario in the cloud that would later contribute to the system delay.
- **Less Bandwidth Consumption:** A number of sensors such as LiDAR or cameras are incorporated in the system, which generates a huge amount of data. High-bandwidth requirements exist for continuous real-time traffic of raw sensor data transfer to the cloud, which is not always available or reliable in various driving conditions.
- **Reliability and Availability:** The perception and control systems of the vehicle remain active in the case of network outages or poor connectivity, thanks to the edge processing.
- **Increased Privacy and Security:** Local processing results in the continued exposure of sensitive sensor data to external networks and therefore enhances data privacy while minimizing the cyberattack surface.

The design and implementation of the study of the NVIDIA Jetson Xavier AGX have for the purpose of being used as an edge computing module specifically for robot and autonomous systems [16, 19, 20]. The critical architectural and software elements providing vital assistance for efficient AI inference on edge devices encompass:

- **GPU Accelerated:** The platform provides an integration of NVIDIA Volta architecture GPU with Tensor Cores that allow for extreme parallel processing power suited for deep learning and its applications.
- **Multi-core CPU:** This supports all kinds of general-purpose computations by its multiple ARM CPU cores.
- **Deep Learning Accelerators (DLAs):** These purpose-built accelerators for neural network inference function with low power consumption.
- **A Large Memory:** It equips 32GB RAM for efficiently loading large models and high-resolution sensor data.
- **Strong Software Stack:** The Jetson Xavier AGX is well supported by CUDA, cuDNN, and TensorRT, which is an inference optimization library essential to deploying models like YOLO with low latency and high throughput.

TensorRT optimization techniques such as layer fusion, precision calibration (FP16/INT8), kernel auto-tuning, and dynamic memory management will enable deep learning models to work very well on resource-constrained edge devices [19, 20, 21].

## 2.3 Detection of Objects in 3D Point Clouds

Detecting objects present in 3D point clouds is a core algorithm pertaining to autonomous navigation, enabling vehicles to recognize and react to cars, pedestrians, or cyclists as well as other obstacles. This section presents an analysis of some prominent approaches found in the literature, many of which are already cited in the original paper.

### 2.3.1 Voxel-Based Methods

Voxel-based methods for discretizing 3D point cloud space into a regular grid of volumetric elements, so-called voxels.

- **VoxelNet [6]:** VoxelNet, being one of the early work in this area, divides the point cloud into 3D voxels, encodes the points within each non-empty voxel into a feature vector by a mini-PointNet structure, and then applies 3D convolutions to these voxel features. A Region Proposal Network (RPN) is used to generate bounding box detections for the 3D instances. VoxelNet demonstrated the feasibility of end-to-end learning for 3D object detection but was computationally expensive.

- **SECOND (Sparsely Embedded Convolutional Detection) [14]:** SECOND is an improvement to VoxelNet by introducing sparse convolution operations. These sparse convolutions preferentially execute computation only on the non-empty voxels, thus negating the cost of computation and memory which standard dense 3D convolutions require on the empty voxels. Consequently, they avoid surplus compute and memory at unprecedented speeds while maintaining similar accuracy.

These methods leverage the well-established convolutional architectures from 2D image processing by extending them to 3D. However, the voxelization step can lead to information loss and quantization errors, and the computational cost of 3D convolutions, even sparse ones, can still be high.

### 2.3.2 Point-Based Methods

Point-based methods operate direct on the raw data of irregular points, avoiding the information loss that is usually associated with voxelization.

- **PointNet [5]:** This was the pioneering work that proposed an architecture of neural network designed to directly operate on unordered point sets. It learns point-wise features using the shared Multi-Layer Perceptrons (MLPs), later aggregated into a global shape feature using a symmetric function e.g. max pooling to guarantee permutation invariance. It is originally designed for classification and segmentation but inspired many further complex tasks like detection.

- **PointNet++:** It further extends PointNet but hierarchically evaluates local geomet-

ric structures by processing at different input point levels. PointNet run recursively on nested partitions of input point set instead of directly learning features from the input point set at a single level.

- **PointRCNN [11]:** It is a two-stage approach: First, it gets 3D region proposals directly from the point cloud through a segmentation network based on PointNet++. Then, each proposal is refined and classified. PointRCNN is accurate but a little slower because it's two-stage processing and real point-wise experimentation.

- **Part-A^2 Net [12]:** A newer model directed, in particular, towards marking intra-object parts and thus improving an object's detection accuracy. It has two main stages, one for the generation of part-aware proposals and the second for part aggregation in RoI feature learning, thus showcasing the finer feature representation benefits.

Point-based methods act as good conveyors of precise geometric information but, many times, have trouble extracting contextual information efficiently, being computationally expensive for large point clouds.

### 2.3.3 Pillar-Based Methods

Pillar-based algorithms provide an in-between speed and accuracy, thus filling the gap left by voxel-based and point-based approaches.

- **PointPillars [1]:** An important paper referenced here, PointPillars discretizes the point cloud into vertical columns or pillars, not 3D voxels. Each pillar's points are encoded into a feature vector using a simplified PointNet-like architecture. These pillar features are scattered into a 2D pseudo-image so that efficient 2D convolutional networks can be used for detection. The PointPillars model strikes a nice balance between speed (reported at 62 Hz) and accuracy on benchmarks like KITTI [22], hence suited for real-time applications. The paper mentions the use of a PointPillar voxelization scheme for the NUSTAG EV's pipeline, mapping points to pixels

during the BEV generation process.

### 2.3.4 Projection-Based (Bird's Eye View - BEV) Methods

The method is that it projects the 3D point cloud onto the corresponding 2D plane, which mainly would be the Bird's Eye View (BEV), and applies further techniques of 2D object detection. Here programming complexity is highly reduced.

- **MV3D (Multi-View 3D Object Detection Network) [7]:** One of the earlier works that fused features from multiple views (BEV, front view, and camera images) to perform 3D object detection. The paper mentions that its RGB-BEV feature map generation follows ideas proposed by Chen et al. [MV3D].
- **Pixor [13]:** This method performs 3D object detection directly from a BEV representation of the point cloud. It uses a fully convolutional network to predict objectness and bounding box parameters for each cell in the BEV grid, enabling fast, single-stage detection.
- **Complex-YOLO [2]:** This is a key reference in the paper. Complex-YOLO adapts the YOLO (You Only Look Once) architecture for 3D object detection from BEV images derived from point clouds. It encodes height, intensity, and density information into the RGB channels of the BEV image. A significant contribution is the Euler-Region Proposal Network (E-RPN), which uses complex regression for orientation estimation, avoiding singularities associated with traditional angle computation. Complex-YOLO demonstrated very high speed (over 5x faster than some contenders) and accuracy.
- **BirdNet+ [10]:** Another BEV-based approach that focuses on end-to-end 3D object detection directly in bird's eye view and aims for efficient and accurate detection.

They are fast because they use mature two-dimensional CNNs. However, projection often results in a loss of information, especially about how objects extend vertically. Conse-

quently, the code for encoding the BEV images as presented in the paper (Equation **??**) is important for preserving salient features.

### 2.3.5 Treating Sparsity and Uncertainty

The point cloud of LiDAR is sparse, and the ground truth annotations may be uncertain, since they could be occluded or contain human errors.

- **Sparse Convolutional Networks (SCN) [4]:** As said about SECOND, SCNs (for example, Sub manifold Sparse Convolutional Networks) are meant for the efficient processing of spatially sparse data. In this, convolution is done only on active sites, that is, non-empty voxels/points and their neighbours, thereby saving large computation as compared to a dense convolution. They work very well for tasks such as semantic segmentation, but deploying them on edge boxes, which are constrained in their resources, could be very challenging without a particular library support created for the target architecture as was mentioned (aarch64) in the future work of the paper.

- **GLENet [3]:** This paper addresses the issue of uncertain labels in 3D object detection; it presents a generative model through conditional variational autoencoders (CVAEs) and proposes such a model to incorporate uncertainty into the detection pipeline with the potential to improve robustness with which detections can be made under ambiguous conditions or even when occlusions occur. The paper notes, however, that while powerful, these advanced models still confront challenges in terms of deployment on edge devices due to limitations in frameworks.

### 2.3.6 Model Efficiency and Real-Time Performance

This necessity has resided in close proximity to all associated past literature, and more prominently among them, as also base focus of the paper- model efficiency for real-time performance of edge devices.

- The paper highlights **PointPillars [1]** and **Complex-YOLO [2]** for their emphasis on improving process efficiency. Their specific approaches include pillar encoding and a 2D CNN backbone by PointPillars and BEV projection with a lightweight YOLO architecture by Complex-YOLO.
- Another example that demonstrates this pursuit is the choice of the paper among different YOLO variants (**YOLOv3 tiny, YOLOv4tiny, YOLOv11 Small OBB, YOLOv12 Nano/Small OBB, RT-DETR**) [18, 23, 24]. Since YOLO is single-stage processing and integrates bounding boxes and class probabilities for all images together, it impacts inference speeds.
- Furthermore, concerning the field and the speed trade-off for any deployment, optimization by TensorRT [19, 20, 21] proves vital for covering the distance between trained models and deployable real-time systems on NVIDIA hardware, as well demonstrated throughout the paper.

However, the accuracy-speed trade-off is a constant consideration. Such models as SA-SSD [15] achieve maximum accuracy using structure-aware single-stage detection but tend to be heavier computationally compared to the lightweight YOLO variants chosen for deployment at the edges of this paper.

## 2.4 Benchmark Datasets for 3D Object Detection

Standardized datasets are crucial for training models and evaluating their performance objectively. The KITTI dataset is prominently featured in the paper and is a cornerstone in autonomous driving research, we have mentioned about it in the chapter 3.

### 2.4.1 Other Notable Datasets

While KITTI is central to our paper, there also appeared other large-scale datasets that continued to promote the advancement of research in 3D perception.

- **Waymo Open Motion Dataset [25]:** In the context of GLENet, the Waymo dataset is much larger than KITTI and offers data collected from a much wider range of sensors in various scenarios with high-resolution LiDAR data and camera images together with detailed annotation for 3D object detection, tracking, and motion forecasting.
- **nuScenes:** Another large autonomous-driving dataset, nuScenes features a complete sensor suite that includes LiDAR, cameras, and radar, together with comprehensive annotations. It provides a more diverse range of weather and lighting conditions than KITTI.
- **Argoverse:** This dataset goes on to provide extensive sensor data and detailed maps, focusing primarily on motion forecasting and 3D tracking.

With larger-scale and more complex datasets, researcher communities are being challenged in the development of perception algorithms that are robust and generalized across different environments. Nevertheless, with respect to model optimizations under various considerations as proposed by the paper for edge deployment research, KITTI can be considered an equally relevant and controllable benchmark.

## 2.5 Conclusion

This chapter discussed the relevant research papers and models studied for undertaking the project. There were three main types of research papers referred, namely about the LiDAR, detection using point cloud data and processing point cloud data on edge devices. Additionally the datasets used in these papers are also discussed.

# Chapter 3

# Components & Materials

## 3.1 Technologies

1. **Python3**: Primary programming language utilized to develop the models.
2. **Pytorch**: Used for Training Deep Learning Models
3. **Ultralytics:** Provided the models and functions to train them on custom data.
4. **TensorRT**: Used to deploy trained models on the edge device
5. **ROS2**: Used for developing Simulations and Visualizer

## 3.2 Sensor

In this work, the Ouster OS-1-64 LiDAR sensor served as the primary and sole source of environmental perception data. The general specifications of this LiDAR model are detailed in Table 3.1:

The Ouster OS-1-64 is a high-performance digital solid-state LiDAR known for its robust design, high resolution, and reliability, making it well-suited for demanding applications such as autonomous driving and robotics.

Table 3.1: LiDAR Specifications

| Parameter | Value |
|---|---|
| Range (80% Lambertian Reflectivity) | 110 m @ >90% detection probability, 100 klx sunlight |
| Minimum Range | 0.8 m for point cloud data |
| Range Accuracy | ±5 cm for lambertian targets, ±10 cm for retroreflectors |
| Range Resolution | 0.3 cm |
| Vertical Resolution | 64 channels |
| Horizontal Resolution | 512, 1024, or 2048 (configurable) |
| Vertical Field of View | +16.6° to -16.6° (33.2°) |
| Horizontal Field of View | 360° |

## 3.3 Edge Device

The NVIDIA Jetson AGX Xavier served as the central processing unit and "brain" of the NUSTAG autonomous vehicle in this project.

The key specifications of the Jetson AGX Xavier (32GB version) utilized in this project are detailed in Table 3.2:

Table 3.2: Jetson AGX Xavier (32GB) Specifications

| Parameter | Value |
|---|---|
| AI Performance | 32 TOPS (Trillion Operations Per Second) |
| GPU | 512-core NVIDIA Volta architecture GPU with 64 Tensor Cores |
| GPU Max Frequency | 1377 MHz |
| CPU | 8-core NVIDIA Carmel Arm®v8.2 64-bit CPU 8MB L2 + 4MB L3 |
| CPU Max Frequency | 2.2 GHz |
| Power | 10W to 30W |

An edge device like the Jetson AGX Xavier is crucial for autonomous systems as it provides the necessary computational power to process sensor data, run complex algorithms (such as perception, planning, and control), and make real-time decisions directly on the vehicle, minimizing reliance on external cloud infrastructure and reducing latency.

## 3.4 Dataset

### 3.4.1 KITTI Dataset

The KITTI Vision Benchmark Suite [22, 26] is one of the most popularly used datasets in mobile robotics and autonomous driving research. This was the primary dataset used for training the models.

- **Sensor Suite:** Data was collected from a standard station wagon with a complete sensor setup of high-resolution stereo cameras (one color, one grayscale), a Velodyne HDL-64E LiDAR scanner and a high-precision GPS/IMU localization unit.
- **Data Variety:** It contains hours of driving data collected across different environments: structurally urban streets, rural roads, and highways at different weather and lighting conditions.
- **Annotations:** KITTI provides 3D bounding boxes for several object classes (cars, pedestrians, cyclists) in both the LiDAR points cloud and camera images, which are very carefully annotated for object detection. These annotations contain object class, 3D location, dimensions, and orientation.
- **Tasks:** This covers different types of benchmark activities in stereo vision, optical flow, visual odometry, 3D object detection, and 3D tracking.
- **Effect**: KITTI has had a great deal of contribution to the advancement of 3D perception. Almost all state-of-the-art models discussed in the paper, PointPillars, Complex-YOLO, etc., are evaluated and benchmarked on KITTI. In fact, this paper trains its YOLO models on the KITTI dataset.

### 3.4.2 Preprocessing

The raw LiDAR point clouds are dense with 3D information; however, they are often described as sparse, unordered, and high-dimensional, which is contrary to how standard CNNs work. Therefore, a very important first step consists in changing the point clouds into a more efficient structure that can be processed by much deeper learning models. We outline a second approach in the paper: this involves producing a 2D Bird's Eye View (BEV) representation.

Pre-processing is fundamental to dense input data and to improve the inference speed of the object detector. The 3D point cloud of a single frame, acquired by the LiDAR, is converted into a single RGB-BEV, covering an area of 50m x 50m, in front of the sensor. Inspired by Chen et al. (MV3D) [7], the RGB-map is encoded by height, intensity, and density.

$\mathcal{P}_\Omega = \{\mathcal{P} = [x, y, z]^T \mid x \in [0, 50m], y \in [-25m, 25m], z \in [-1.75m, 2m]\}$

We define a mapping function $S_j = f_{PS}(P_{\Omega i}, g)$ with $S \in R_{mxn}$ mapping each point with index $i$ into a specific grid cell $S_j$ of our RGB-BEV. A set describes all points mapped into a specific grid cell:

$P_{\Omega i \to j} = \{P_{\Omega i} = [x, y, z]^T | S_j = f_{PS}(P_{\Omega i}, g)\}$

Hence, we can calculate the channel of each pixel, considering the *SIGNAL* strength as $I(P_\Omega)$:

$$z_g(S_j) = \max(P_{\Omega i \to j} \cdot [0, 0, 1]^T)$$

$$z_b(S_j) = \max(I(P_{\Omega i \to j}))$$

$$z_r(S_j) = \min(1.0, \log_{64}(N + 1)) \quad \text{where } N = |P_{\Omega i \to j}|$$

Here, N describes the number of points mapped from $P_{\Omega i}$ to $S_j$. Hence, $z_g$ encodes the maximum height, $z_b$ the maximum intensity, and $z_r$ the normalized density of all points mapped into $S_j$.

### 3.4.3 Dataset Preparation and Label Conversion

The RGB-BEV images were used for training the models from the KITTI dataset. The KITTI dataset has 3D bounding box annotations for objects in 3D in point clouds. Conversion of these 3D labels into 2D BBs for training purposes of the OBB YOLO models on the BEV images is then done as follows:

- Project the corners of the 3D bounding box onto the 2D BEV plane.
- Parameterize this OBB into the form required by the YOLO model that is $(x, y, w, h, r)$ representation.

Consequently, training images were kept at a resolution of 608x608 pixels, which is normally input in terms of various versions of YOLO. During training, the augmentation was also employed as provided by the `ultralytics` framework.

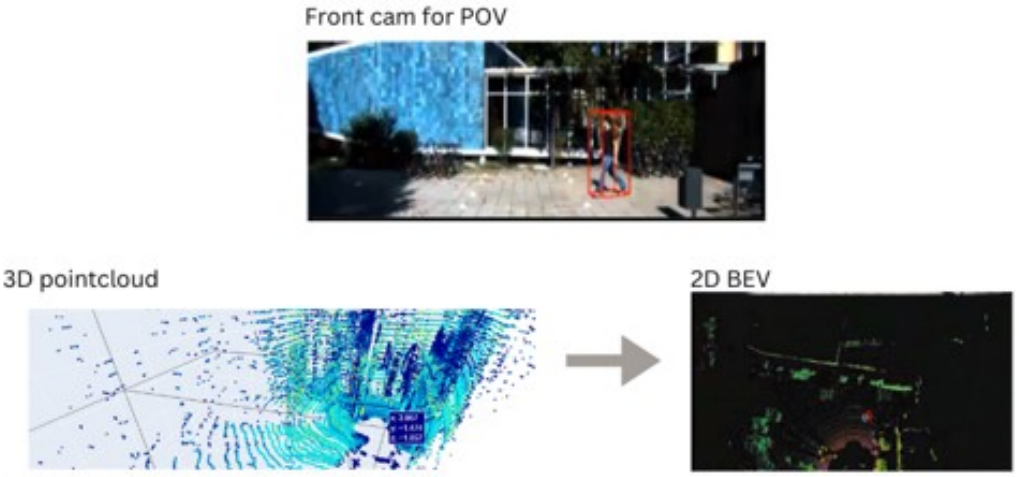



Figure 3.1: Point Cloud to 2D Bird Eye View Conversion.

## 3.5 Conclusion

This chapter discussed the technologies used and the detailed specifications of the Ouster Gen1 OS-1 LiDAR, the Nvidia Jetson Xavier and the KITTI dataset.

# Chapter 4

# Object Detection

Autonomous systems employed for environmental perception, especially autonomous vehicles, depend on object detection. It involves figuring out where the object of interest, according to sensor data, is located, how big it is, and what all classes it would belong to. For instance, one is interested in identifying cars, pedestrians, and cyclists among others. In particular LiDAR systems, this activity is practically defined by the task of real-time processing of large chunks of 3D point cloud data for extracting this vital information. Therefore, the performance of the object detection module will directly affect the safety and reliability of all downstream operations, including path planning, collision avoidance, and higher-level decision-making.

This chapter describes the object detection pipeline developed by this work within the overall perception system. The chapter begins by detailing the preprocessing of raw LiDAR data into suitable representations for deep learning-based detectors-most notably Bird's Eye View (BEV) projections.

A thorough account is then provided on the YOLO (You Only Look Once) family of object detection models. Their performance brings speed and accuracy in poised balance, which is why they form the backbone detection strategy of our work. The chapter also discusses the training procedure, including dataset preparation, data augmentation techniques, and

Oriented Bounding Boxes (OBB) to improve object localization accuracy in the space.

Special attention is also given to the optimization methods employed to efficiently deploy these models on edge computing hardware, particularly the NVIDIA Jetson Xavier AGX platform. Optimization with TensorRT, which includes techniques like layer fusion, precision calibration (FP16/INT8), and memory tuning, is highlighted as essential for achieving real-time inference on constrained hardware.

Lastly, the chapter outlines the experimental setup used for evaluating different YOLO configurations. Performance is analyzed on the KITTI dataset, and results are used to justify the final model selection for integration into the perception system of the NUSTAG EV.

## 4.1 YOLO (You Only Look Once) for Object Detection Architecture

The network takes a bird's-eye-view RGB-map as input. It uses a YOLO11-obb CNN architecture to detect accurate multi-class oriented objects while operating in real-time.

The YOLO (You Only Look Once) [18] family of object detectors has been selected in this project because it has proven in literature to be a very good compromise between speed and accuracy and so well suited for real-time applications on embedded systems.

### 4.1.1 Core Principles of YOLO

YOLO views object detection as one end-to-end regression problem: from image pixels to bounding box coordinates and class probabilities. This monolithic structure stands in contrast to two-stage detectors, for example, Faster R-CNN, which propose regions of interest and then classify them. Core features of YOLO are:

- **Single Pass:** Entire image processed in a single pass through neural network.

- **Grid-Based Detection:** Divide the input image into $S \times S$ grid. If an object's center falls within a grid cell, that grid cell is responsible for detecting that object.
- **Bounding Box Prediction:** Every grid cell predicts a fixed number of bounding boxes (B) and confidence scores for those boxes. The confidence score reflects how confident the model is that the box contains an object and also how accurate it thinks the box is.
- **Class Probabilities:** Each grid cell also predicts conditional class probabilities for C classes, $P(Class_i|Object)$.
- **Unified Prediction:** Predictions usually include the bounding box coordinates $(x, y, w, h)$ (the center coordinates: width, height), a confidence score, and class probabilities for each predicted box.

This one-stage approach offers a drastically improved inference speed over a two-stage detection mechanism.

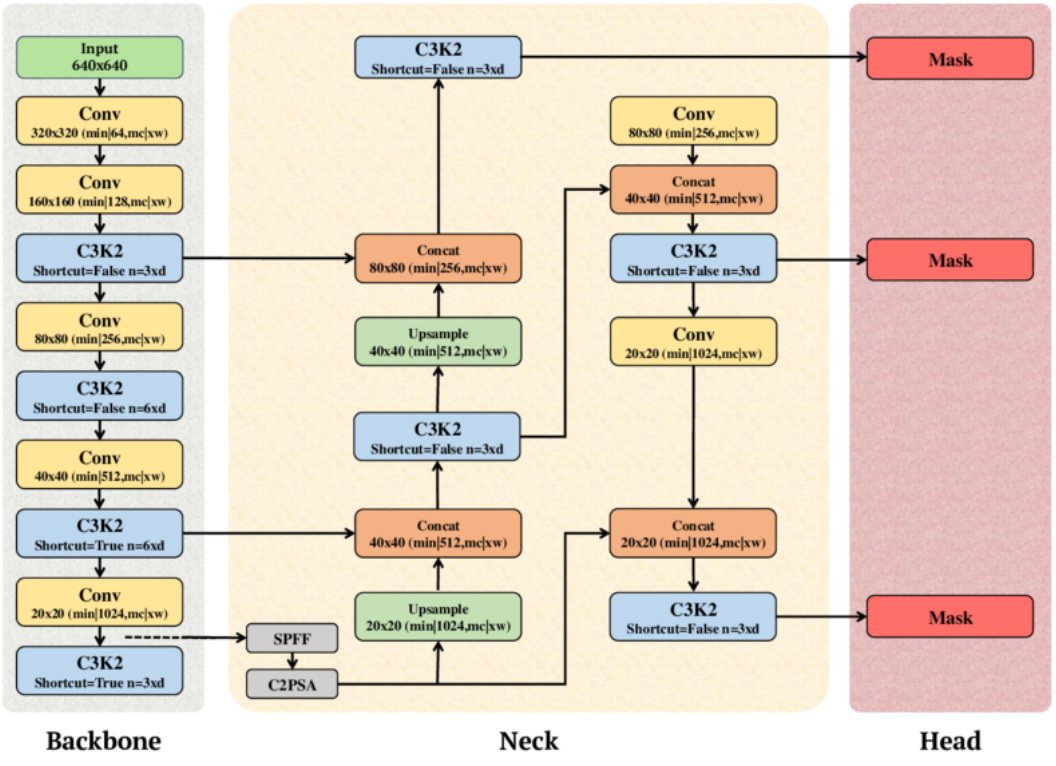



Figure 4.1: YOLOv11 base architecture.

### 4.1.2 Evolution and Variants Used

From its inception, YOLO has been through many iterations, advances, or improvements, obtaining more accuracy, being faster, and more adept at detecting objects with different scales. In this paper, we analyze some of the variants:

- **Complex-YOLOv3 Tiny & Complex-YOLOv4 Tiny [2]:** These are adaptations of the lightweight "tiny" versions of YOLOv3 and YOLOv4 explicitly tailored for 3D object detection on BEV images as proposed in the Complex-YOLO paper. They have very small network backbones in order to achieve maximum speed of inference for edge devices.
- **YOLOv11 Small OBB [27]:** This is not a commonly acknowledged official release in the main YOLO lineage (up to YOLOv9/YOLO-NAS/YOLO-World as of early 2024), so it is most probably newly developed or project-specific. "Small" indicates a mini architecture, "OBB" indicates its capability to predict Oriented Bounding Boxes, and it was the strongest model in the paper's experiments.
- **YOLOv12 Nano OBB & YOLOv12 Small OBB [23]:** Once again, just like YOLOv11, "YOLOv12" appears to be a name for a very new or project-specific nomenclature. "Nano" and "Small" convey two levels of complexity, with "OBB" denoting oriented boxes.
- **RT-DETR (Real-Time DEtection TRansformer) [24]:** It is other architecture using transformers for object detection. Though DETR-based models have relatively good performance, they are still under the active research banner in terms of real-time processing on edge devices. The inclusion adds a comparison against modern detectors not based on convolutional architecture.

Such models include the small, nano, and tiny versions, which highlight the performance goals of the project towards resource-constrained edge deployment.

### 4.1.3 Oriented Bounding Boxes (OBB) for BEV

Unfortunately, traditional object detectors predict axis-aligned bounding boxes (AABBs). For example, vehicles in a bird-eye view (BEV) image can be oriented at different angles, so AABBs generally produce extremely poor fittings: they contain large values when matched against background and miss some portion of the object, especially for elongated objects aligned with an angle.

Oriented Bounding Boxes (OBBs) fit a tighter bounding box, by allowing rotation of the bounding box. Typically defined by center $(x, y)$, width $(w)$, height $(h)$, and an angle of rotation $(\theta)$, an OBB is capable of mathematically representing a rotated bounding box. The paper states that OBB models have been trained, which means that the YOLO variants used herein have been modified as well as chosen to predict these five instead of the four for AABBs.

There are three advantages of employing OBBs in BEV object detection:

- **Improved Localization:** Tighter boxes provide better IoU with reality and increase accuracy in localizing objects.
- **Less Overlap Ambiguity:** Overlap that narrows the possible interpretations of the bounding box is maintained to a minimum at the boundaries of nearby objects.
- **Improved Size and Shape Estimation:** This will improve the representation of geometric footprint and orientation of the object, both highly important for tracking and path planning.

The E-RPN from the Complex-YOLO paper, for example, targeted robust angle regression relevant to 3D object detection [2]. Modifying the output layer of the YOLO model for 2D BEV OBB detection allows for the prediction of an additional angle parameter.

## 4.2 Model Training

To train robust, accurate object detection models, careful preparation of data is needed, appropriate augmentation strategies have to be considered, and a well-configured training pipeline is in place.

### 4.2.1 Training Setup and Pipeline

The training takes place on a very powerful workstation:

- **CPU:** 32 Intel i9 processors
- **GPU:** NVIDIA GTX 4090 24GB
- **RAM:** 128GB RAM

Such hardware can ensure reasonably fast iterations in training. The paper mentions using the Ultralytics training function. The Ultralytics framework provides implementations and training scripts for almost all YOLO versions (e.g., YOLOv5, YOLOv8). All models used here were trained with standard training arguments, but for 325 epochs and an image size of 608 pixels. Specifics of hyperparameter tuning are not given but can be deduced as being part of standard training procedures for purposes of maximizing detection mean Average Precision (mAP) and generalization.

Typically, the core of a training pipeline consists of the following:

- **Loss Function:** Most YOLO models employ a composite loss function that incorporates bounding box coordinate regression (like CIoU loss), object confidence (binary cross-entropy), and class prediction (either binary cross-entropy or cross-entropy) components.
- **Optimizer:** Adam or SGD with momentum would commonly be seen as optimizers.

- **Learning Rate Schedule:** Learning rate decay and cyclic learning rates are common strategies.
- **Batch Size:** This is picked from the limitations imposed by the GPU memory.

Inferencing on edge devices such as NVIDIA Jetson Xavier does involve great optimizations for real-time performance while training mostly takes place on very high-performance workstations. Would-be-one-of-its-key-enabling technology is NVIDIA TensorRT.

### 4.2.2 The Need for Optimization on Edge Devices

Edge devices indeed have much less computation architecture, memory, and energy budget compared to their server counterparts or workstations. With almost no optimization, simply sending a trained PyTorch or TensorFlow model onto an edge device would suffer from unacceptably high latency and low throughput. Hence optimization becomes necessary to:

- **Reduce Latency:** Ensure predictions are available fast enough for live decision-making.
- **Increase Throughput:** Process more frames per second.
- **Minimize Resource Usage:** Reduce CPU, GPU, and memory footprint.

## 4.3 Experimental Evaluation of Detection Models

The paper went on to do an experimental comparison of the performance of different variants of YOLO over TensorRT optimization, implemented on the NVIDIA Jetson Xavier.

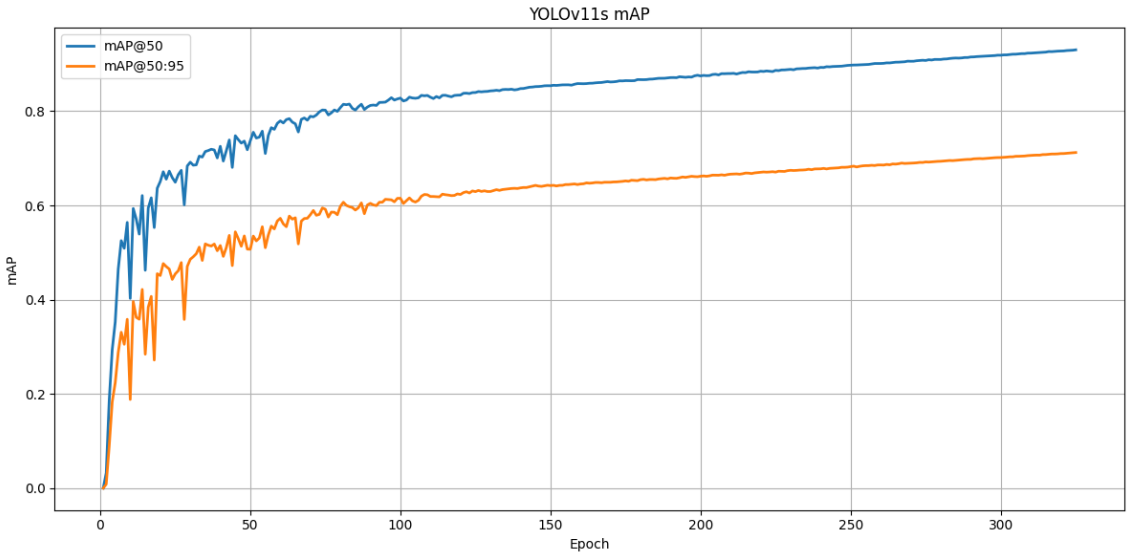



Figure 4.2: YOLOv11 small training curve.

### 4.3.1 Models Evaluated

The following models were trained on KITTI BEV dataset and later tested on TensorRT for Jetson Xavier performance evaluation:

1. Complex YOLOv3 Tiny
2. Complex YOLOv4 Tiny
3. **YOLOv11 Small OBB** (best performing)
4. YOLOv12 Nano OBB
5. YOLOv12 Small OBB
6. RT-DETR

The model configurations were chosen such that their computational requirements (GFLOPS) are within performance specifications of the NVIDIA Jetson Xavier (mentioned in the paper as under 32 GFLOPS, although this seems low for the peak of the Xavier AGX; perhaps a sustained target was meant).

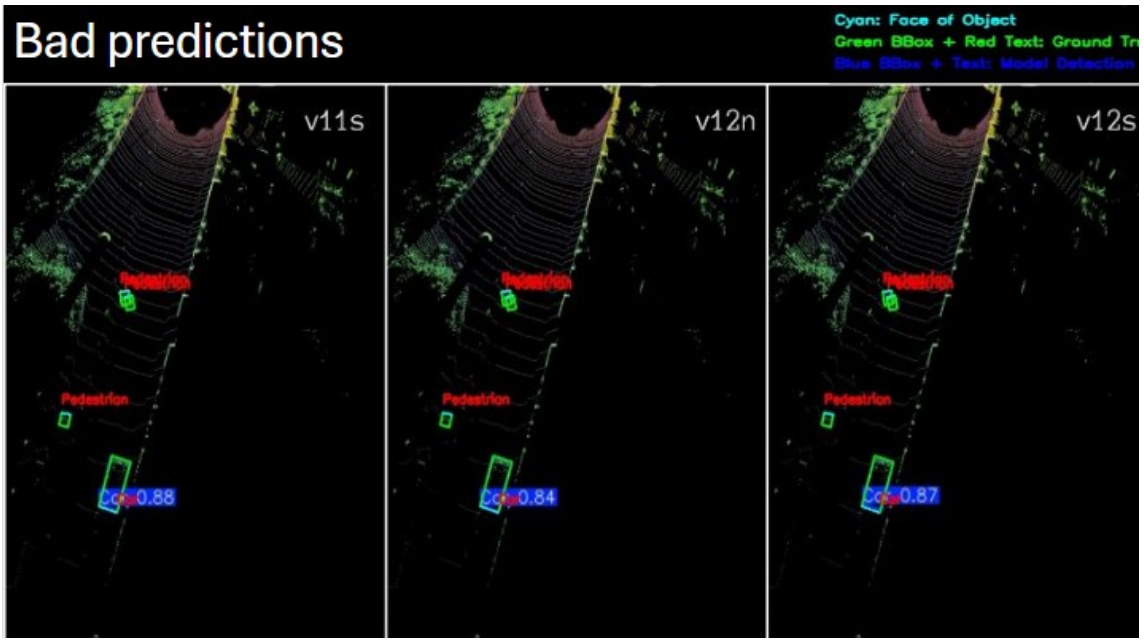



Figure 4.3: Models bad performance.


Figure 4.4: Model's good performance.

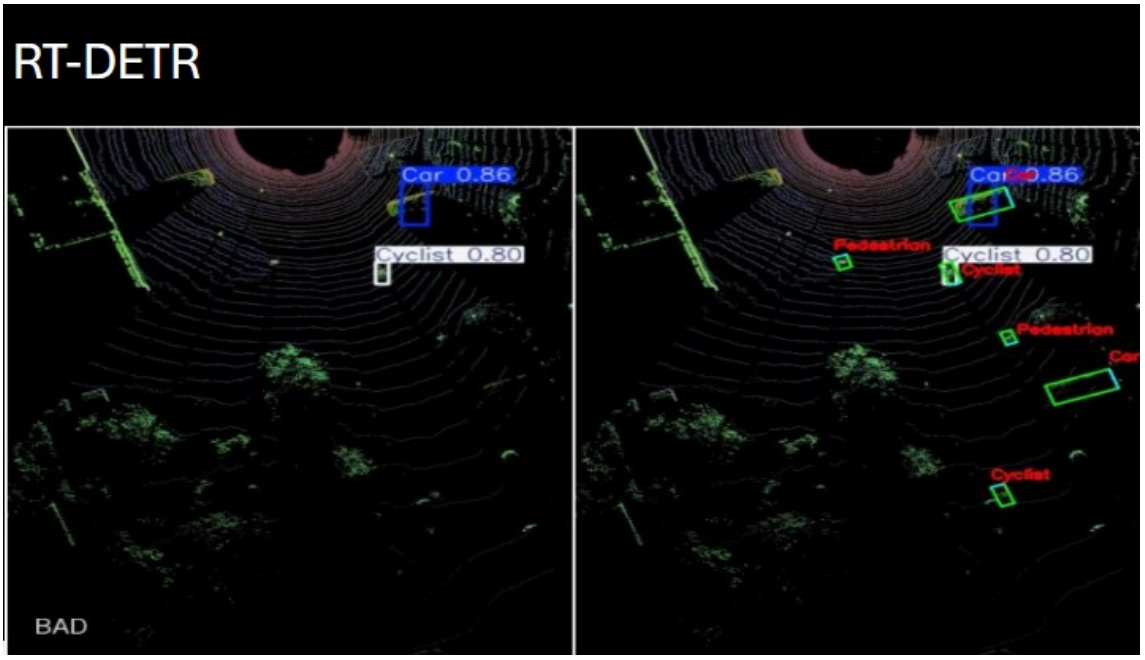



Figure 4.5: RTDETR models performance.

### 4.3.2 Performance Measures

Some main measures of evaluation were:

- **TensorRT Inference Time (ms):** The time to perform inference using an optimized TensorRT engine on a single BEV image by the Jetson Xavier. This is a direct measure of speed.
- **mean Average Precision (mAP):** The common metric for measuring the accuracy of object detection.
  - **mAP @ 0.5 IoU:** mAP computed using a 0.5 threshold for Intersection over Union (IoU). Common baseline usually used for PASCAL VOC-style evaluation.
  - **mAP @ 0.5:0.95 IoU:** mAP averaged over multiple IoU thresholds-from 0.5 up to .95 with step 0.05. This is a stricter metric and is often used for evaluation in the COCO style and favors a more precise localization.

IoU is the measure of overlap between the predicted bounding box and the ground truth bounding box, as defined:

$$IoU = \frac{Area(PredictedBox \cap GroundTruthBox)}{Area(PredictedBox \cup GroundTruthBox)} \tag{4.1}$$

### 4.3.3 Result Analysis

The summary of the evaluated models is presented in Table **??** (Table I in the paper).

Table 4.1: Reiteration of Table I: Comparison of Object Detection Models Performance on Xavier (Trained at 325 Epochs)

| # | Model | TRT Inference Time (ms) | mAP (%) at 50 IoU | mAP (%) at 50-95 IoU |
|---|---|---|---|---|
| 1 | Complex YOLOv3 Tiny | 51 | 56.87 | 48.6 |
| 2 | Complex YOLOv4 Tiny | 45 | 64.11 | 59.1 |
| 3 | **YOLOv11 Small OBB** | **10** | **93.0** | **71.2** |
| 4 | YOLOv12 Nano OBB | 15 | 87.5 | 66.7 |
| 5 | YOLOv12 Small OBB | 32 | 86.8 | 68.7 |
| 6 | RT-DETR | 50 | 65.4 | 58.9 |

Key observations from the results:

- `yolo11s-obb` showed prime value in all major parameters:
  - **Inference Time:** This is 15ms.
  - **Highest mAP @ 0.5 IoU:** That is 93.0%
  - **Highest mAP @ 0.5:0.95 IoU:** This is 71.2%
- The Complex-YOLO tiny variants were meant to be fast, but unfortunately turned out to be much slower (45-51 ms). This only proves how much the latest YOLO architectures have advanced.
- `yolo12s` was slightly slower, with inference times of 18ms, and gave slightly lower accuracy compared with `yolo11s`.
- `RT-DETR` gave high inference time (50 ms), and in mAP it falls below that of the best YOLO variants on this specific task.

Consequently, YOLOv11 Small OBB was taken up as the flagship detection model in the NUSTAG EV perception pipeline since it offered an optimal trade-off between speed and accuracy on the edge device. Apart from the experiments that were carried out using the above-mentioned detection model with improved performance from the literature in comparison with other existing 3D object detection models (Table **??** in the paper), the true performance of YOLOv11 Small OBB is even more contextualized.

Table 4.2: 3D Object Detection Models Performance Comparison on Nvidia Xavier on KITTI Dataset

| **Model** | **Inference (ms)** | **mAP (%) @** | |
|---|---|---|---|
| | | **0.5 IoU** | **0.5-0.95 IoU** |
| BirdNet+ [10] | – | – | 65.14 |
| PointRCNN [11] | 747 | 89.58 | 85.66 |
| Part-A2 [12] | 540 | 89.04 | 88.19 |
| Pixor [13] | 372 | 82.59 | 79.46 |
| Second [14] | 165 | 89.90 | 83.71 |
| SA-SSD [15] | – | – | 89.00 |
| PointPillars [1] | 101 | 89.89 | 83.23 |
| **YOLOv11 small** | **10** | **93.0** | **71.2** |


Figure 4.6: Performance of Object detection models on Xavier.

## 4.4 Conclusion

This chapter discussed the YOLO OBB model used for object detection, its training and performance evaluation. It was further compared to the other models found in literature such as BirdNet, PointRCNN, Pixor, PointPillars, etc. It is observe that the YOLOv11s model gave the least inference time of 10ms and a high mAP of 93.0.

# Chapter 5

# Object Tracking

## 5.1 Tracker

Tracking is performed by ByteTrack algorithm. It is a multi-object tracking (MOT) algorithm that effectively uses low-confidence detection boxes by associating them with tracklets, leading to improved performance, especially in crowded scenes and with occlusions.

## 5.2 ByteTrack MOT

ByteTrack utilizes a Kalman filter at its core for state estimation and prediction of the tracked objects. It is a mulit-object tracker (MOT) and therefore maintains a history of the objects tracked in the form of data records called tracklets. These tracklets maintain positional data of the past 30 frames for each object. We extended the prediction of tracked objects to estimate their trajectory. The Kalman filter has an 8 dimensional state vector to estimate the position of the objects being tracked.

In the Kalman filter, the predicted state based on object dynamics (e.g., velocity and direction) $\hat{x}_{k|k-1}$ is fused with the probabilistic detection from YOLO, whose inaccuracy may vary from time to time with the noisy state $z_k$, as given in [Eq: 5.1].

The following equation is at the core of Kalman filters,

$$\hat{x}_k = \hat{x}_{k|k-1} + K_k(z_k - H\hat{x}_{k|k-1}) \tag{5.1}$$

*here,*

$\hat{x}_k$: Updated state estimate at time $k$

$\hat{x}_{k|k-1}$: Predicted state estimate at time $k$ given information up to $k-1$

$K_k$: Kalman gain at time $k$

$z_k$: Measurement vector at time $k$

$H$: Observation matrix

$(z_k - H\hat{x}_{k|k-1})$: Innovation or measurement residual

### 5.2.1 Tracklet Interpolation

Tracklet is a sequence of tracked object positions. Tracklet interpolation is useful for cases where the object detector misses detections in between frames.

Tracklet interpolation estimates the bounding boxes of these missed objects. If a tracklet T loses its detection at time step $t$ such that $t_1 < t < t_2$ and $t_2 - t_1 < \sigma$ then the bounding box $B_t$ of the tracklet $T$ is linearly interpolated between $B_{t_1}$ and the first redetected box $B_{t_2}$. The hyperparameter $\sigma$ defines the acceptable time of occlusion. The following is the formula for linear interpolation [Eq: 5.2].

$$B_t = B_{t_1} + (B_{t_2} - B_{t_1})\frac{t - t_1}{t_2 - t_1} \tag{5.2}$$

Tracklet interpolation is useful to maintain the structure of the tracklets since they require thirty positional data points for adjacent frames.

## 5.3 Multi-step Look ahead

The Kalman filter inherently performs a forward prediction of the state for a single time step. We used [Eq: 5.3] with $n = 20$ to predict the state two seconds into the future.

$$\hat{x}_{k+n} = \hat{x}_k \times F^n \tag{5.3}$$

In the above equation [Eq: 5.3],

$\hat{x}_{k+n}$ is the future state at time step n

$\hat{x}_k$ is the currently predicted state

$F^n$ is the state space matrix of shape $8 \times 8$

$$\hat{x} = \begin{bmatrix} x & y & a & h & \dot{x} & \dot{y} & \dot{a} & \dot{h} \end{bmatrix}$$

**here,**

$x$ and $y$ are the abscissa and ordinate of the center of the bounding box

$a$ is the aspect ratio between width $w$ and height $h$ of the bounding box such that $w = a \times h$

$\dot{x}$ and $\dot{y}$ are velocities in $x$-direction and $y$-direction respectively while $\dot{a}$ and $\dot{h}$ are rate of change in $a$ and $h$ respectively.

$$F = \begin{bmatrix} 1 & 0 & 0 & 0 & \Delta t & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & \Delta t & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & \Delta t & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & \Delta t \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

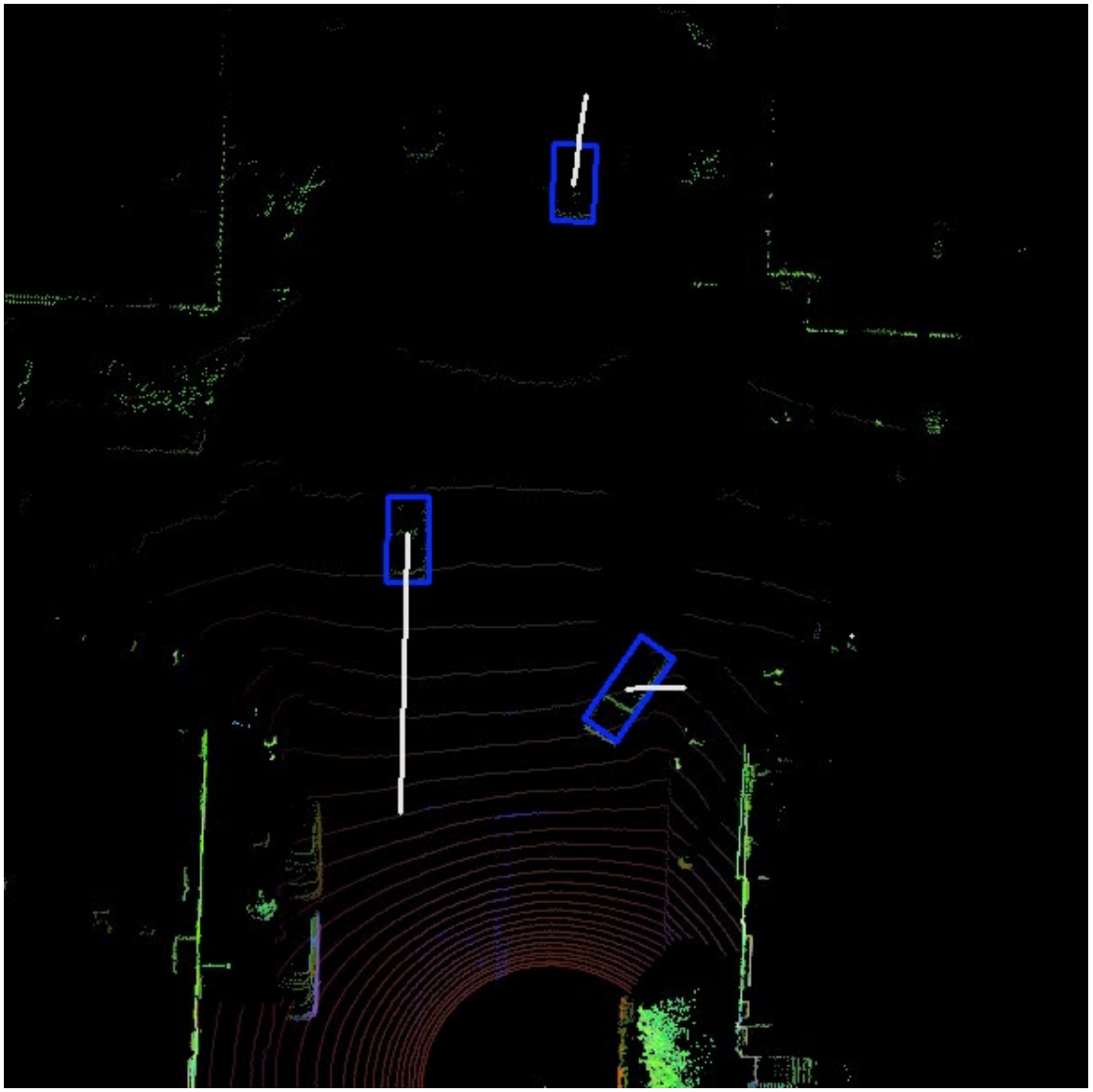


Figure 5.1: Tracking test LiDaR data of the San fransisco drive.

## 5.4 Conclusion

This chapter discusses the ByteTracker algorithm and tracklet interpolation. It discusses how the Kalman Filter is used for forward predictions and how the tracker utilizes it for predicting the trajectory of detected objects after 2 seconds.

# Chapter 6

# Hardware in Loop Simulations

In this chapter we will discuss the Gazebo Simulator, why it was choosen and how the simulations that were made using Gazebo Ignition and ROS2. Furthermore, the outputs of the Deep Learning models on the simulated environments are also shown and discussed.

## 6.1 Gazebo Simulator

Gazebo is a collection of open-source software libraries designed to simplify the development of simulations for robotics applications. It is widely used by robot developers, designers, and educators. Each library within Gazebo has minimal dependencies and are made to be as modular as possible, making them suitable for tasks such as solving mathematical transforms for kinematics and inverse kinematics, sensor imitation, and managing physics simulation [28]

Simulations provided a safe and efficient way to test the deep learning model's performance under different conditions. Gazebo was chosen as the simulation platform for this project for testing and evaluation of the NUSTAG electric vehicle in a digital environment.

## 6.2 Digital Twin Development

### 6.2.1 Measurements

All the measurements are either manually calculated or taken from the Tecknofest Documentation[29].

Table 6.1: Car Specifications

| **Parameter** | **Value** |
|---|---|
| Weight | 80 kg |
| Height | 169 cm |
| Width | 90 cm |
| Length | 258 cm |
| Ground Clearance | 13 cm |
| **Wheel Positions** | |
| Front Wheels Offset | $46 \pm 12.5$ cm from arc center |
| Back Wheel Offset | $61 \pm 12.5$ cm from arc center |
| **Front Area** | |
| Thickness | 5 cm |
| Width | 63 cm |
| Length | 190 cm |
| **Back Area** | |
| Length | 35 cm |
| Thickness | 5 cm |
| Width | 70 cm |
| **Wheels** | |
| Thickness | 7 cm |
| Wheelbase | 170 cm |
| Front Wheel Opening | 101 cm |
| Back Wheel Opening | 82 cm |
| Diameter | 60 cm |

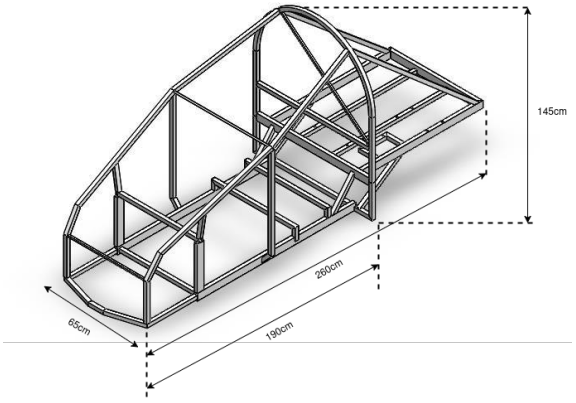



Figure 6.1: Chassis Dimensions

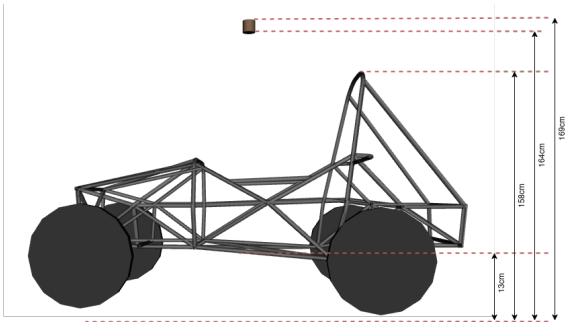



Figure 6.2: Side View of Digital Twin

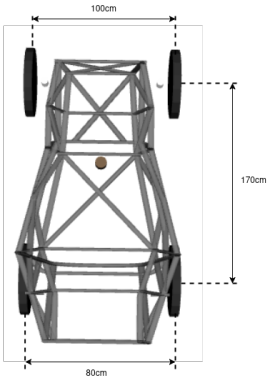



Figure 6.3: Top View of Digital Twin

### 6.2.2 URDF Development

URDF (Unified Robot Description Format) is an XML-based file format used to represent robot properties such as its physical configuration, joints and link structures. It is used by ROS2 ecosystem to describe the kinematic and dynamic properties of a robot like its links, joints, inertial data, visual geometry, and collision models.

The URDF model of the digital twin of the car was designed to mimic the physical and structural specifications of the real vehicle. The key features of the URDF model are as follows:

1. **Accurate Physical Dimensions:**

   The URDF contains the electric vehicles' geometry, including exact measurements of length, width, height, wheelbase, and ground clearance. This ensures that the simulated robot behaves similar to its physical counterpart in terms of spatial interaction and kinematics.

2. **Four Continuous Wheel Joints:**

   The vehicle is equipped with four continuous joints representing its four rotating wheels. They essentially simulate realistic rolling behavior of the wheels during forward and reverse motion.

3. **Front Wheel Steering via Revolute Joints:**

   The two front wheels are connected via revolute joints, enabling them to rotate about their vertical axes for steering. These joints are configured according to the Ackermann steering geometry [30], allowing for realistic driving dynamics and vehicle mobility.

4. **LiDAR Mount:**

   A specific link is defined in the URDF to represent the mounting point for the simulated LiDAR. The link is fixed to the vehicle's chassis at a height of $164$cm.

5. **IMU Mount:**

   Similar to the LiDAR, a fixed link is assigned for the Inertial Measurement Unit (IMU) sensor. The simulated IMU gives orientation, acceleration, and angular velocity readings in the virtual world. These readings are required for state estimation and control algorithms.

### 6.2.3 Ackermann Controller

Ackermann steering geometry is a configuration which reduces tire slippage and enhances turning efficiency, especially at low speeds, by ensuring all wheels follow concentric turning paths. Ackermann geometry ensures that the inner front wheel turns at a sharper angle than the outer front wheel. This setup enables both front wheels to align with the instantaneous center of rotation, improving maneuverability and minimizing lateral tire slip.

The Ackermann Steering is implemented in the NUSTAG Electric Vehicle, hence to mimic the steering and turning behavior of the vehicle an Ackermann Controller was designed for the digital twin.

#### Ackermann Geometry Calculations

The following parameters and formulas were used to calculate inner and outer wheel steering angles and the turning radius based on Ackermann principles. All measurements were taken manually after turning the car wheels to their maximum turning angle.

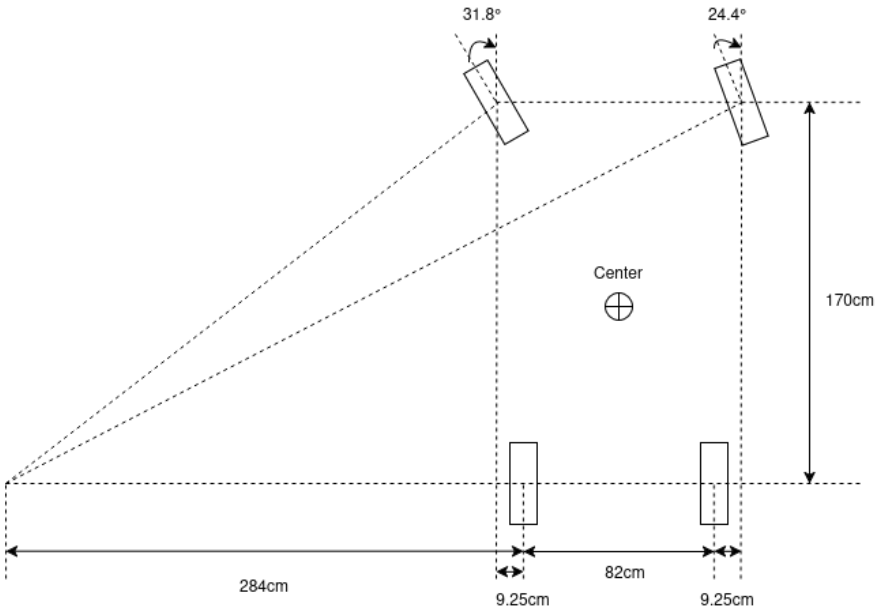



Figure 6.4: Ackermann Diagram

**Given Data:**

- Wheelbase: $170\,\text{cm}$
- Rear wheel track width: $82\,\text{cm}$
- Front wheel track width: $100\,\text{cm}$
- Distance from Rear Left wheel to Centre of Turning Radius: $D = 284\,\text{cm}$
- Distance between Front and Rear wheel = $9.25\,\text{cm}$

**Inner Wheel Steering Angle ($\theta_{\text{in}}$):** Using Pythagorean Theorem:

$$\theta_{\text{in}} = \tan^{-1}\left(\frac{\text{Wheelbase}}{D - \text{Offset}}\right)$$

Substituting the values:

$$\theta_{\text{in}} = \tan^{-1}\left(\frac{170}{284.05 - 9.25}\right) = \tan^{-1}\left(\frac{170}{274.8}\right) \approx 31.8^\circ$$

**Outer Wheel Steering Angle ($\theta_{\text{out}}$):** Using Pythagorean Theorem:

$$\theta_{\text{out}} = \tan^{-1}\left(\frac{\text{Wheelbase}}{D + \text{Rear wheel track width} + 9.25}\right)$$

Substituting the values:

$$\theta_{\text{out}} = \tan^{-1}\left(\frac{170}{284.05 + 82 + 9.25}\right) = \tan^{-1}\left(\frac{170}{458.55}\right) \approx 24.4^\circ$$

**Turning Radius Calculation**

The turning radius $R$ is computed using the Pythagorean theorem by taking the distance from the center of the car:

$$R = \sqrt{(\frac{170}{2})^2 + (284 + (\frac{82}{2}))^2}$$
$$R = \sqrt{(85)^2 + (284 + 41)^2}$$
$$R = \sqrt{7225 + 105625} \approx 335.93\,\text{cm}$$

### 6.2.4 Sensor Integration

The gz_sensor library was used to create dummy sensors to imitate sensor data. The Electric Vehicle only utilizes the Ouster Gen 1 OS-1 LiDAR, which consists of:

1. **LiDAR:**

   Generates a 3D map of the sensor using point clouds. The LiDAR simulated shares similar specifications to the Ouster Gen 1 OS-1 LiDAR, operating at 10 Hz and having 1024 horizontal beams x 64 vertical beams at an angle of $\pm 16.6^\circ$.

2. **IMU:**

   Ouster LiDARs consists of an integrated and calibrated IMU, providing motion compensation and Simulatanous Localization and Mapping (SLAM). The IMU is

simulated for the sake of completeness.

## 6.3 Simulation Environments

To evaluate the accuracy of the trained models, several basic simulated environments were created in Gazebo. Due to the significant RAM and GPU requirements of the simulator, the environments were kept simple, using static objects with lightweight meshes.

### 6.3.1 Car-Only Environment

This environment simulates a street with a total of seven parked cars—six on the left side and one on the right—all facing downward. The autonomous vehicle is programmed to drive down the road and perform a U-turn to the left upon reaching the end.

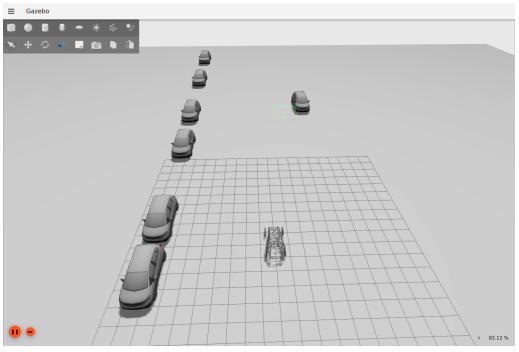


Figure 6.5: Simulated car-only environment in Gazebo

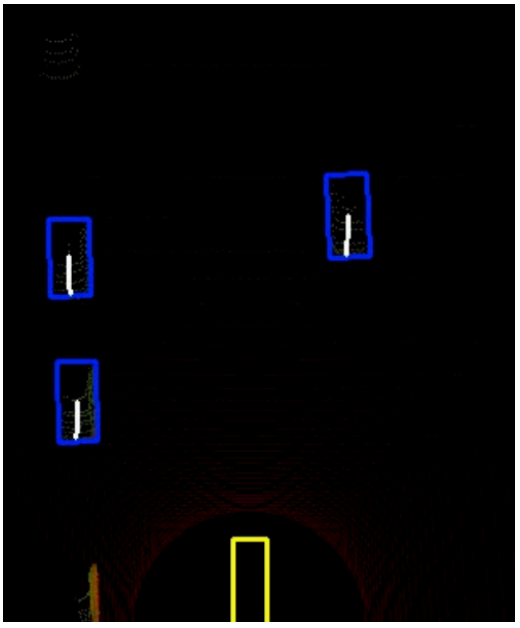


Figure 6.6: Model detection results in the car-only environment

### 6.3.2 Pedestrian-Only Environment

This environment simulates a market area populated with fifteen static pedestrian models. The pedestrians are evenly spaced across the grid, with four facing right, four facing

downward, and seven facing left. The vehicle navigates through the pedestrians and returns to its starting position.

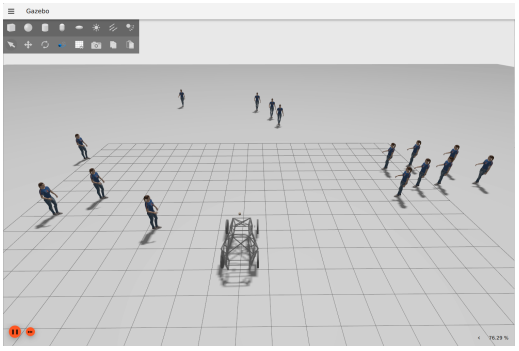


Figure 6.7: Simulated pedestrian-only environment in Gazebo

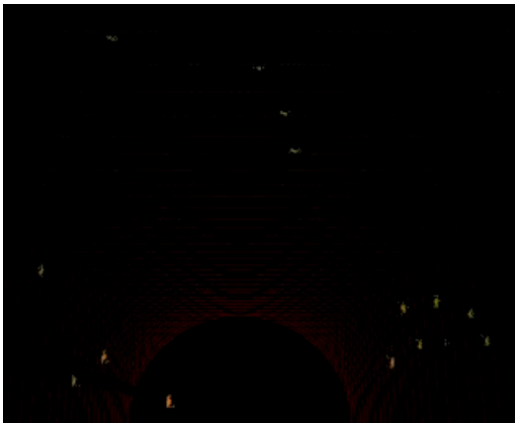


Figure 6.8: Model detection results in the pedestrian-only environment

### 6.3.3 Mixed Environment: Cars and Pedestrians

This environment features a simulated intersection populated with twelve cars and seven pedestrians. A turn is located at the edge of the map. The autonomous vehicle is tasked with navigating around all static obstacles to reach the end of the scene.

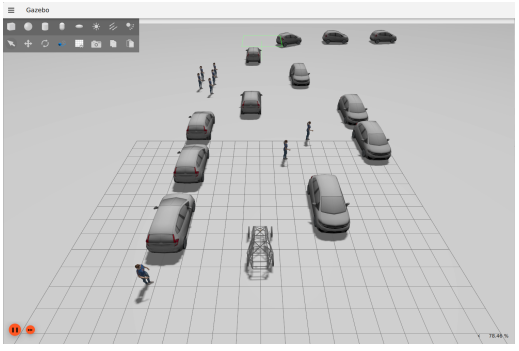


Figure 6.9: Simulated environment with both cars and pedestrians

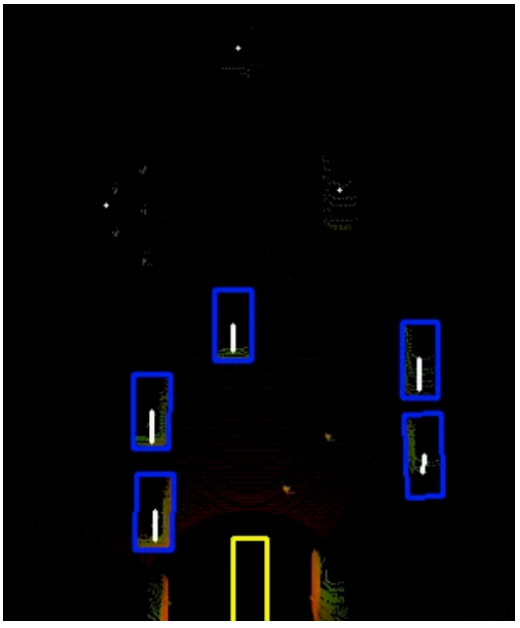


Figure 6.10: Model detection results in the mixed environment

## 6.4 Conclusion

This chapter discusses the details about the Gazebo Simulator and the background of Gazebo Fortress, the simulator used for this final year project. Furthermore, details of the digital twin developed for the simulations is discussed, which included measurements and calculation to compute the ackermann steering angles ($\theta_{\text{in}} = 31.4^{\circ}$ and $\theta_{\text{out}} = 24.4^{\circ}$) and the turning radius of $3.35m$ for the car.

The ROS2 pipeline is also discussed, which dictates how sensor data from the Gazebo simulator is stored and processed to be sent to the Jetson Xavier for processing. Finally, detailed simulation environments are created. Three types of cases (Car ONLY, Pedestrian ONLY, and Car and Pedestrian mixed) are made, and the output of the model is observed in all three environments.

# Chapter 7

# ROS2 Architecture and Visualization Tools

## 7.1 ROS2 Framework

This section discusses the Robotic Operating System (ROS2), its use in developing the visualizations for the sensor data using Rviz2 and the features programmed for easier user interaction with the system.

### 7.1.1 ROS2

Robot Operating System 2 (ROS2) is an open-source framework of libraries and tools for building robot applications. It provides a modular publishing/subscribe and server/client architecture for inter-node communication, drivers for hardware, and integration with simulation tools like Gazebo. ROS2 was chosen for its support of distributed computing and real-time control in robotics.

### 7.1.2 Visualizations with Rviz2

Rviz2 is a ROS2 package which allows the user to display different types of data. It provides support for viewing images, point cloud data, laser scanner data, robot states (link and joints positions and orientations) and much more [31].

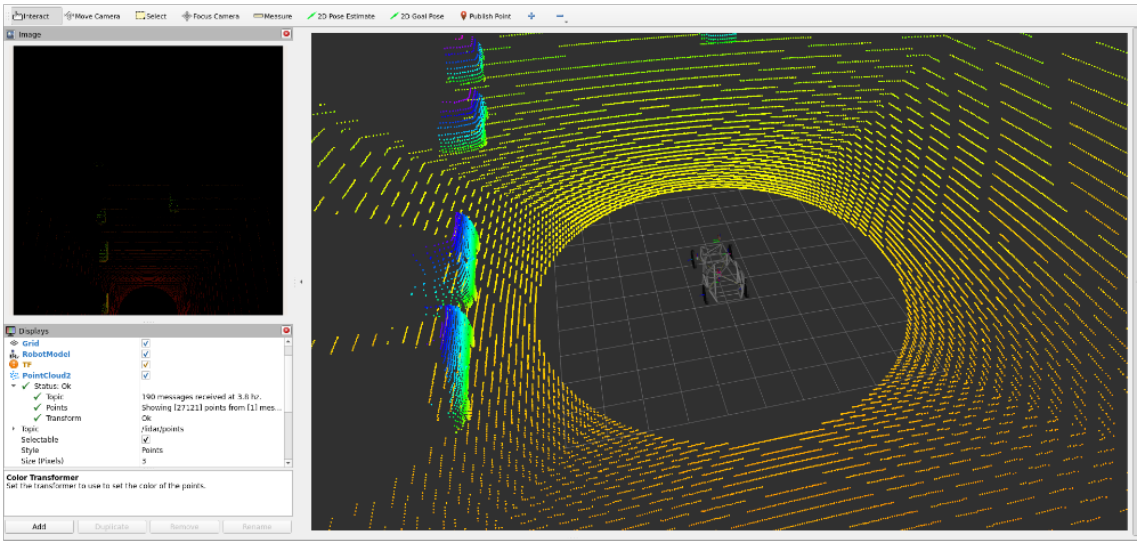



Figure 7.1: Rviz2 Output for Simulation

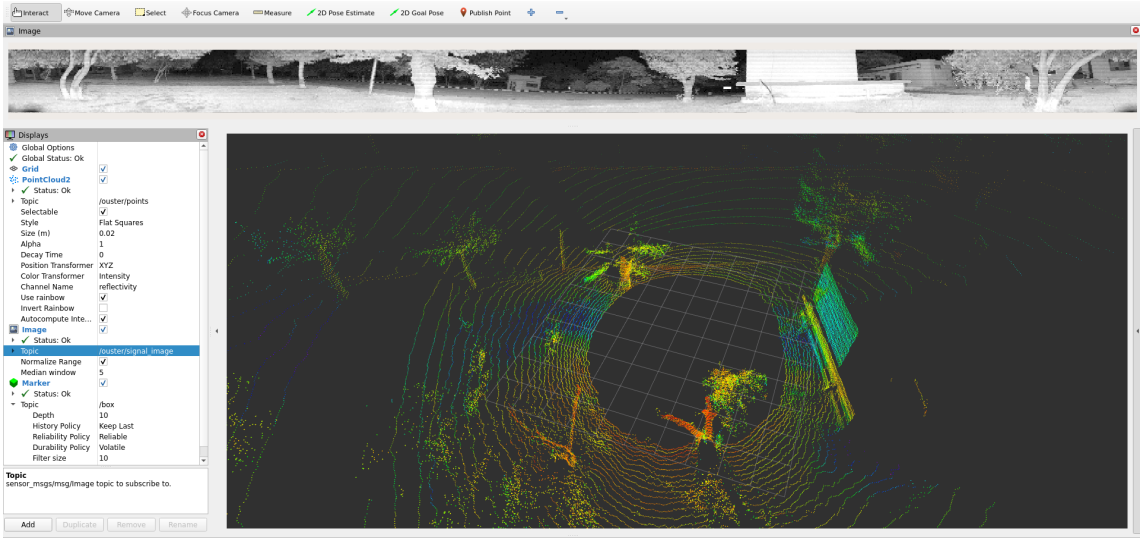



Figure 7.2: Rviz2 Output for LiDAR sensor

#### 7.1.2.1 Topics

Topics in ROS2 refers to the data that follows the publisher/subscriber architecture. A publisher pushes data to the topic, and all the subscribers listening on the topic receive the data simultaneously. Each topic is strongly typed, meaning that they use static datatypes with well-defined semantics (i.e. all topics using angles are defined to use radian rather than degrees).

All the sensors (LiDAR and IMU) are publishers and Rviz2 is the subscriber (receiving and displaying the sensor data).

The sensor datatype are referred to interfaces. Interfaces define the structure of the message.

1. **PointCloud2:**

   The interface consists of a header (which contains the timestamp and id of the message), height and width of the 2D structure of the point cloud, the fields (i.e range, reflectivity, signal and NIR for Ouster LiDAR), length of each field in bytes, and the actual data.

   Rviz2 Output for Simulation shows the type of *PointCloud2* interface utilized by the Simulation and Ouster LiDAR. Note that the Ouster LiDAR offers more fields in comparison to the simulated LiDAR. This does not affect our BEV image as we are utilizing the xyz coordinates and intensity values retrieved from the LiDAR.

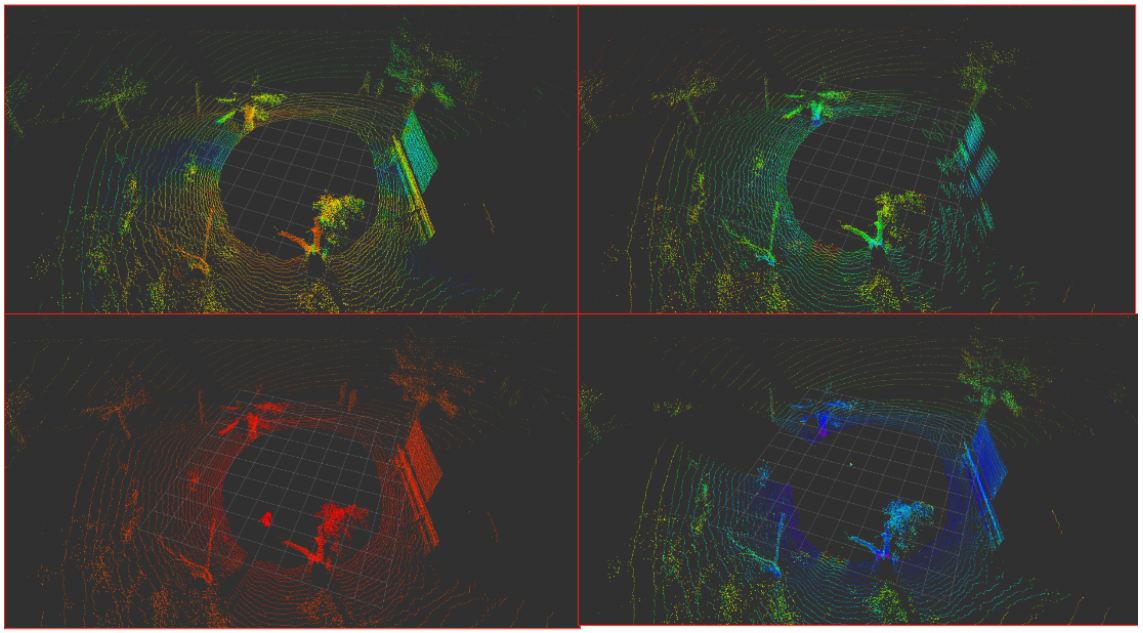


Figure 7.3: Point Cloud Data visualized on Rviz2.
*reflectivity* (top left), *intensity* (top right),
*range* (bottom left), *ambient* (bottom right)

2. **IMU:**

   The interface consists of a header, orientation (*quaternion*), orientation covariance matrix, angular velocity (*rad/s*), angular velocity covariance matrix, linear acceleration and the linear acceleration covariance matrix. The matrices show the uncertainty in the sensor measurements and are a compact way to represent error in 3D space.

3. **Image:**

   The interface consists of a header, height and width of the image, encoding (RGB, HSV etc), the size of the image (*bytes*), and the pixel-wise data. The topic is being used by the LiDAR to publish the NIR, Range, Signal and Reflectivity Images shown in figure below. The BEV node also utilizes this interface to publish the BEV image.

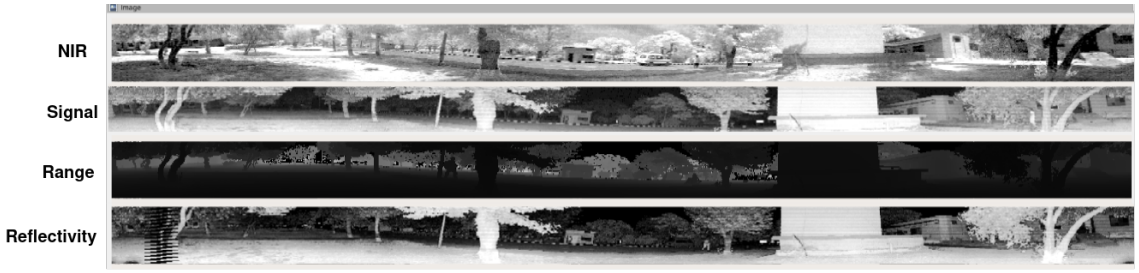
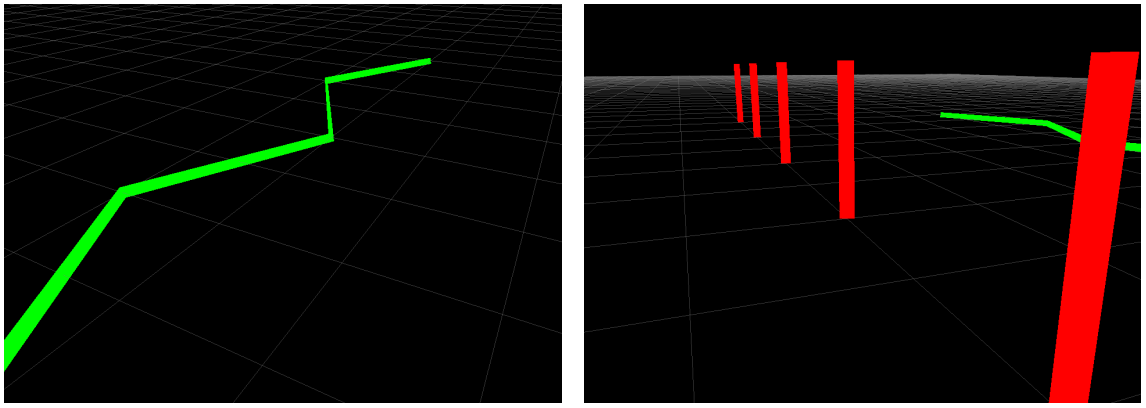



Figure 7.4: Bounding Boxes using Line Strip and Line List Markers

4. **Markers:**

   Markers are special type of Rviz2 interfaces which allow for programmatic addition of different shapes to 3D view. [32]. Two main type of markers were utilized for generating the bounding box on Rviz2: LINE_STRIP (for generating a connected 2D square) and LINE_LIST (for generating vertical lines to represent height).



(a) Line strip Marker (b) Line list Marker

Figure 7.5: Rviz2 Display Markers

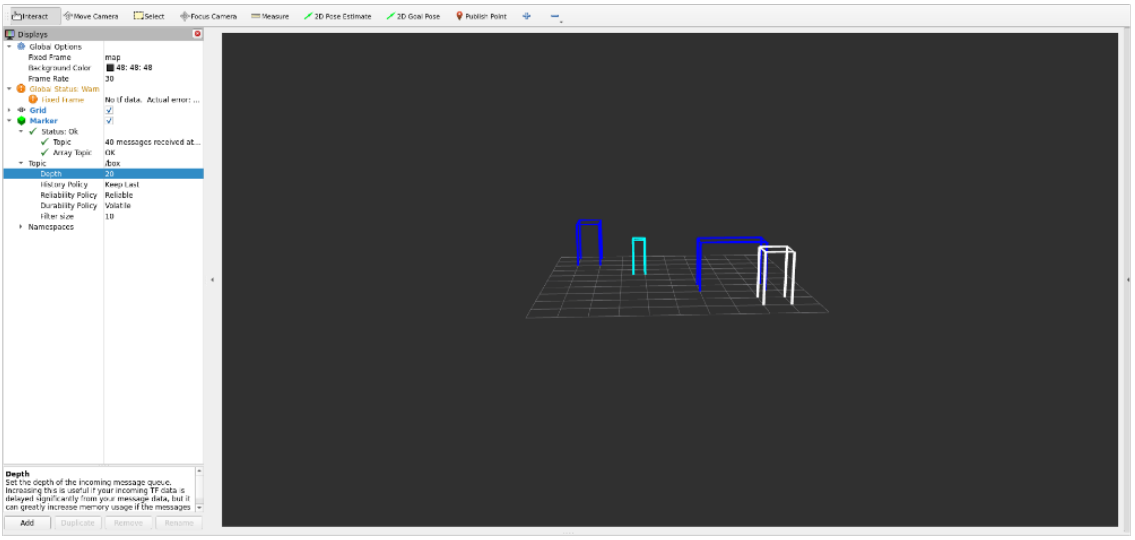


Figure 7.6: Bounding Boxes using Line Strip and Line List Markers

## 7.2 Nodes and Topics

The RQT Graph shows the interaction between the different nodes and topics in our system. The Gazebo Simulation and LiDAR RQT graphs are shown below. In the figures the rectangle represents the topics and the oval shows the nodes.

### Gazebo RQT

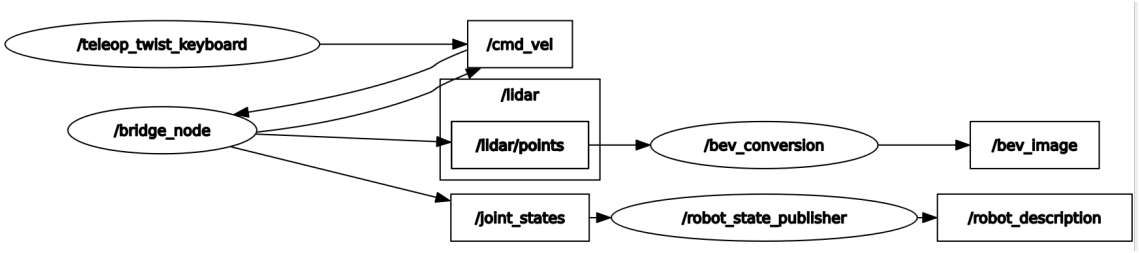



Figure 7.7: Node and Topics interaction in Gazebo Simulation

In the graph we observe that there are two main topics, the **joint_state** and **lidar/points**. In the simulation as we are directly receiving the point cloud, we directly convert the x, y, z and intensity values to the BEV image. The joint states update the car's joints (mainly

wheels and front wheel steering joints) to update the link and joints of the robot shown on Rviz2.

**LiDAR RQT**

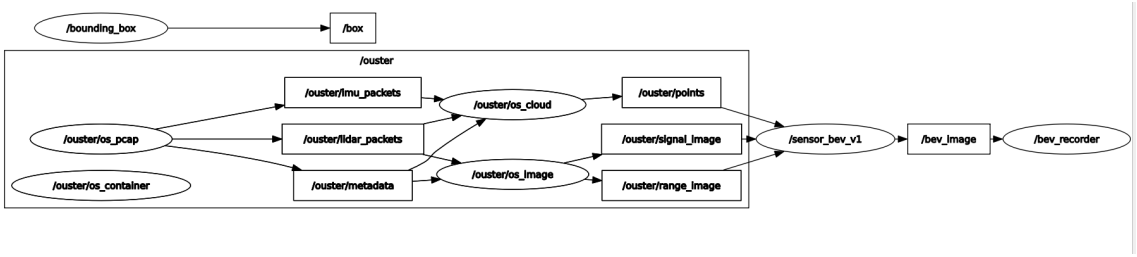



Figure 7.8: Node and Topics interaction in the LiDAR

In the above graph we observe that the ouster lidar is uses the **signal_image**, **range_image** and **ouster_points** to generate the BEV Image. There are two main methods to generate the BEV from the LiDAR output:

1. Convert Range Image to xyz points using XYZLut provided by the ouster-sdk which requires the LiDAR metadata file. Then concatenate it with the normalized signal image.

2. Directly take the ouster/points message and filter the relevant fields (x, y, z and intensity). The conversion are done implicitly.

The first approach is used to stay consistent with preprocessing mentioned in Preprocessing.

## 7.3 Features

The section defines the different nodes and processes programmed in ROS2 to provide users with a method to visualize and interact with the data. The figure below shows an abstract workflow of how the ROS2 pipeline generates its features.

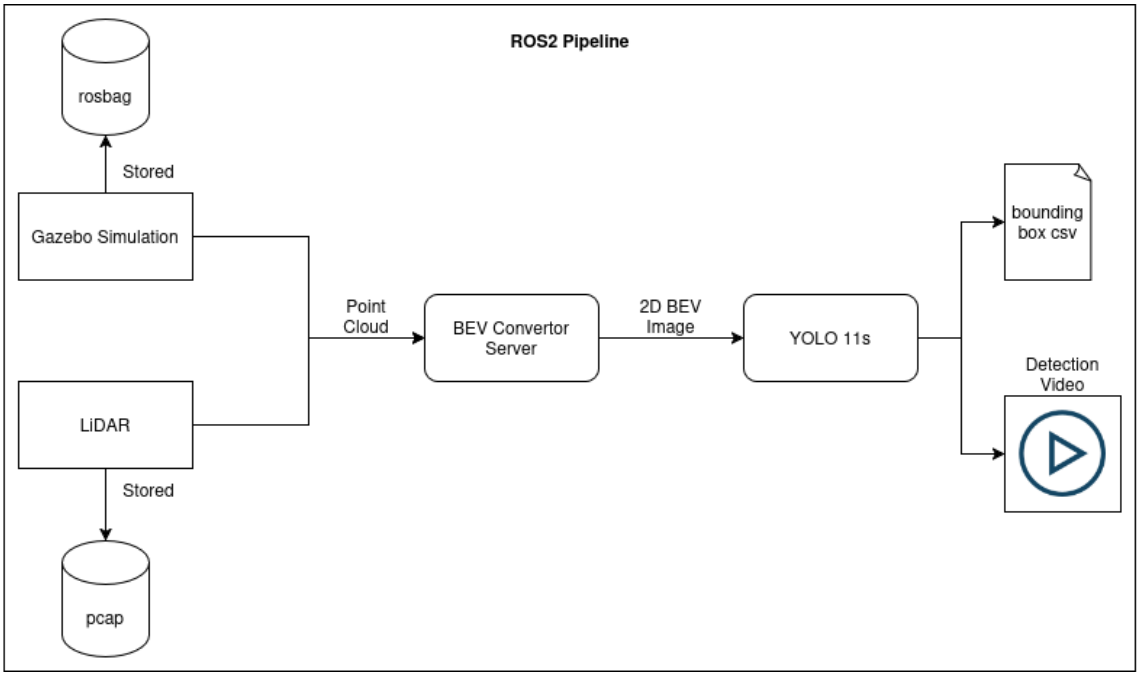



Figure 7.9: Overview of ROS2 Workflow

### 7.3.1 BEV Conversion

To convert the point cloud data to BEV image, we first extract the relevant LiDAR features then filter and finally convert them to the RGB BEV format (Preprocessing).

#### 7.3.1.1 Extraction

The Simulation and Hardware feature extraction is a bit different. After the features are extracted the rest of the process is similar for both workflows.

1. **Simulation**

   From the simulations we extract the *x*, *y*, *z* and *intensity* fields in the **lidar/points** topic (using *PointCloud2* interface). The obtained numpy matrix has the dimensions *1x262144*. We first convert all the data to *float32* datatype, then reshape the array to *1024x64x3* and then transpose it to make it matrix *64x1024x3*.

In the above matrix the dimensions refers to the $64$ vertical lidar beams, $1024$ horizontal beams and $4$ channels. These are the configurations used in the Ouster LiDAR Sensor.

2. **Hardware**

   The Ouster LiDAR does not directly send point cloud data. As it uses the Ethernet protocol, the data received by the hardware is in the form of packets. These packets are converted into *RANGE* and *SIGNAL* images, which are then converted to point cloud points (x, y, z) using the *XYZ Look Up Table* generated from the metadata file generated by the LiDAR upon its initialization.

#### 7.3.1.2 Filtering

The following boundary conditions have been used to filter point cloud points. These configuration values are taken from .

| | |
|---|---|
| minX | 0 |
| maxX | 50 |
| minY | -25 |
| maxY | 25 |
| minZ | -1.73 |
| maxZ | 2.27 |

The above values show the distance (in meters) captured by the LiDAR. It looks $50$m ahead and $25$m left and right. As the height of our car is $1.73$m, we define the minZ as the car's height and the maxZ of $2.27$m.

A mask is generated to filter the points that satisfy these conditions, then the z channel is subtracted with the minZ value.

#### 7.3.1.3 Generating Features

The filtered point cloud data is discretized into a fixed grid $608$x$608$ pixels (for BEV image). The X and Y coordinates of each point are converted into discrete image coordinates, and points are sorted to prioritize those with the highest elevation. From this discretized

and sorted point cloud, three feature maps are generated: a height map representing the normalized vertical position (Z-axis) of the topmost point in each cell, an intensity map capturing the reflectance value of the first point in each cell, and a density map that encodes the number of points per grid cell using a logarithmic normalization. These three maps are stacked as separate channels in a single RGB image, where red corresponds to density, green to height, and blue to intensity.

### 7.3.2 Visualize Live Sensor Data

The script written first initializes the LiDAR's UDP ports to start transmission of data. It then converts the packets send from the LiDAR into *PointCloud2* format and publishes it on the **ouster/points** topic. It also publishes on the **range_image**, **reflectivity_image**, **NIR_image**, and **signal_image** topics.

The following command's can be used to for this feature:

```
ros2 launch ouster_ros sensor.launch.xml \
sensor_hostname:=<sensor host name>
```

Just need to specify the IPv4 address of the LiDAR

```
ros2 launch ouster_ros driver.launch.py \
params_file:=<path to params yaml file>
```

requires path to configuration file which specifies LiDAR properties such as its IPv4 address, its beam configuration, frequency etc.

```
ros2 launch av_car sensor_launch
```

Wrapper on the above command to automatically generate and link the configuration files for launching sensor.

```
ros2 launch av_car sensor_with_bev_launch.py sensor:=true
```

Launch the sensor visualizer with BEV image.

### 7.3.3 Record Data

In Overview of ROS2 Workflow we observe that the Simulation data is stored in rosbag whereas the LiDAR data is stored in as pcap.

#### 7.3.3.1 Simulation

ROS2 provides support for storing data recorded in simulations. These include messages, topics they are generated from and the timestamps. It stores this data as a compressed .sqlite3 file.

To record in rosbag format the command

```
ros2 bag record -ao <filename>
```

is used. The command listens and stores data published on all topics.

To filter the topics that are recorded,

```
ros2 bag record <topic_name> -o <filename>
```

can be used where the list of desired topics can be provided.

#### 7.3.3.2 Hardware

Upon establishing connection with the LiDAR, there are two main methods to record the LiDAR.

1. **rosbag:** Rosbag saves specific topics published by the LiDAR. The recorded data saved is in the .sqlite3 format.

   ```
   ros2 launch ouster_ros record.launch.xml \
   sensor_hostname:=<sensor host name> \
   bag_file:=<optional bag file name> \
   metadata:=<json file name>
   ```

2. **pcap:** Raw LiDAR packets are stored directly.

```
ouster-cli source <sensor_hostname> save <filename>.pcap
```

### 7.3.4 Replay Data

#### 7.3.4.1 rosbag:

This commands replays all the data stored on saved topics.

```
ros2 bag replay <bag_file_dir>
```

#### 7.3.4.2 pcap:

Reads the pcap file and converts the data into the topics ouster/points, range_image, reflectivity_image, NIR_image, and signal_image to visualize it on Rviz2.

```
ros2 launch ouster_ros replay_pcap.launch.xml \
pcap_file:=<path to ouster pcap file> \
metadata:=<json file name>
```

## 7.4 Conclusion

This chapter discussed the ROS2 framework, the Rviz2 visualizer to show the sensor data (point clouds and images), the ros2 pipeline for converting point cloud data to RGB BEV image, and finally the features implemented using ROS2 (recording, replaying and live viewing of the LiDAR data).

# Chapter 8

# Deployment & Validation

## 8.1 Deployment on Nvidia Jetson

TensorRT (an inference optimizer and runtime developed by Nvidia) was used to get efficient real-time performance on the Nvidia Jetson. TensorRT builds a hardware-optimized inference graph of our trained object detection model. Various optimizations (specific to the target Nvidia GPU architecture) such as layer fusion and quantization are applied to the input model to make the model run more smoothly on the hardware.

The deployment pipeline first converts the trained PyTorch model into the ONNX (Open Neural Network Exchange) format. ONNX serves as an intermediary representation that allows interoperability between different deep learning frameworks. At the end, this ONNX graph is built into a TensorRT engine runtime.

### 8.1.1 Limitations of TensorRT

The latest TensorRT version supported for Jetson Xavier (running Jetpack 5.1) is 8.5.2. This is why ONNX Opset version 16 was used. This version was chosen because it represents one of the latest ONNX operator sets with comprehensive compatibility with the version of TensorRT available.

### 8.1.2 Benefits of TensorRT

Employing TensorRT to significantly speed up inference, as we observed a 330% increase in our FPS (Frames Per Second). This metric jumped from 18 FPS to 62 FPS, for `yolo11s-obb` model for *PyTorch* to *TensorRT*.

## 8.2 Live Detections on Jetson

Utilizing ouster-sdk, a Python script was written to capture the LiDAR frames, convert them to RGB BEV images and finally feed them into the model. The model then generated bounding boxes on each frame. An average inference time of 20ms was observed upon detection of a car, cyclist or pedestrian.

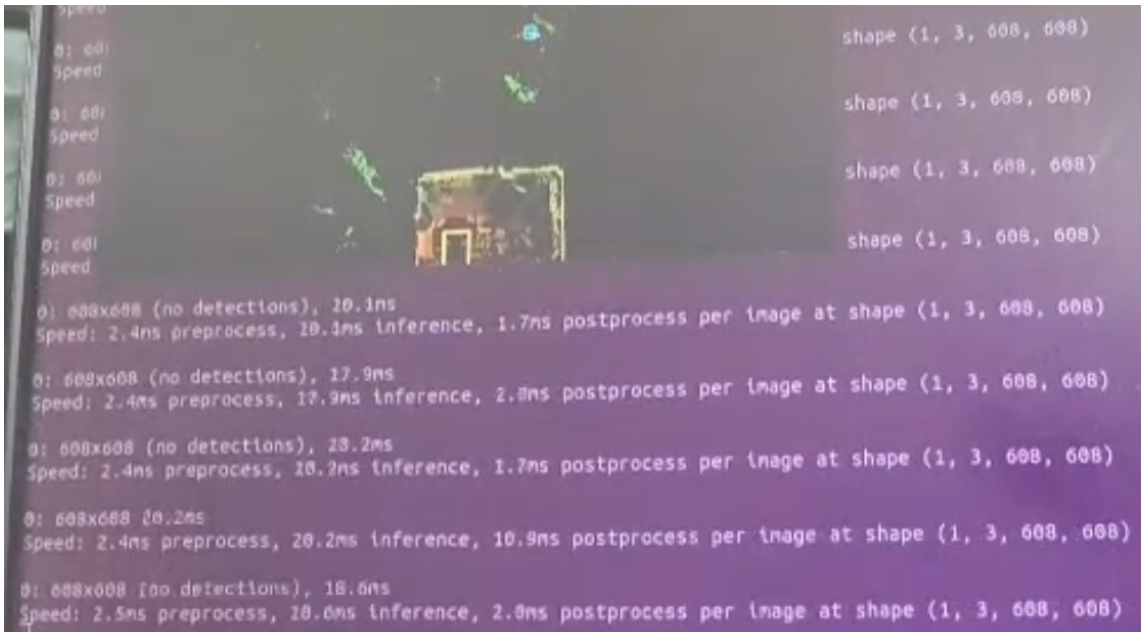



Figure 8.1: Object Detection and Tracking Models running on Xavier

## 8.3 Integration with NUSTAG Autonomous Vehicle

The Jetson and LiDAR were to be mounted on the autonomous vehicle for complete integration. As the mount for Jetson was already present, only a mount for the LiDAR

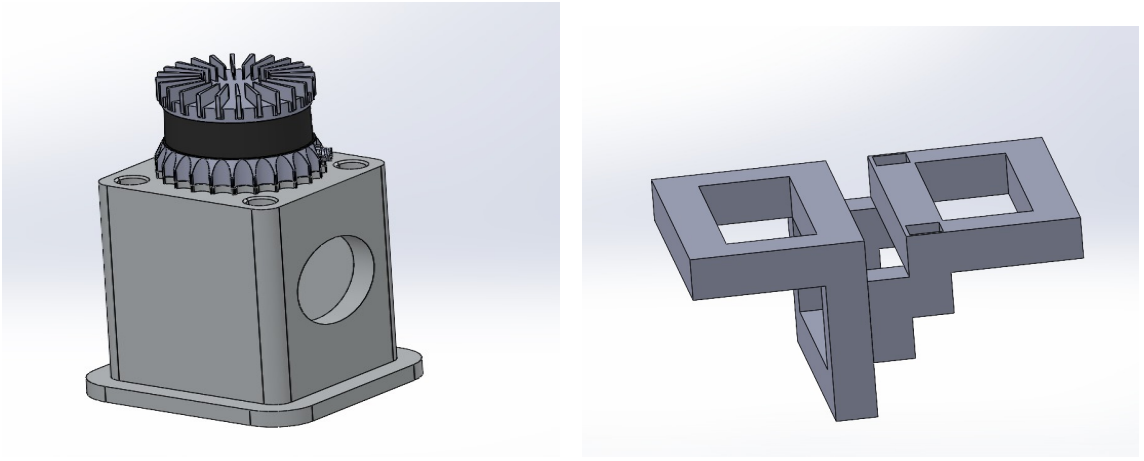
was required.

### 8.3.1 Mount Design

The CAD model for the mount was design keeping in mind the vehicle's limitations. The mount was empty from the inside and had two openings on the front and back to accommodate the mono-camera already mounted on the car.

The mount had two parts, the top part acted as a holder for the LiDAR and the bottom part was bolted to the vehicle to provide a strong base for holding the structure together.

All credit for the designs go to Muneeb Ahmad from 43-MTS.



(a) Top Part of Mount (b) Bottom Part of Mount

Figure 8.2: CAD Models of the LiDAR Mount

### 8.3.2 Mounting LiDAR on Vehicle

The top and bottom parts were drilled together, making the complete mount. The lower portion of the mount was further drilled and then bolted to attach it to the vehicle's chassis.

The top part was made using acrylic on account of it being cheap, strong and resistant to thermal deformation (unlike the 3D printed materials such as PPG). The bottom part was made with hollow scrap metal bars welded together.

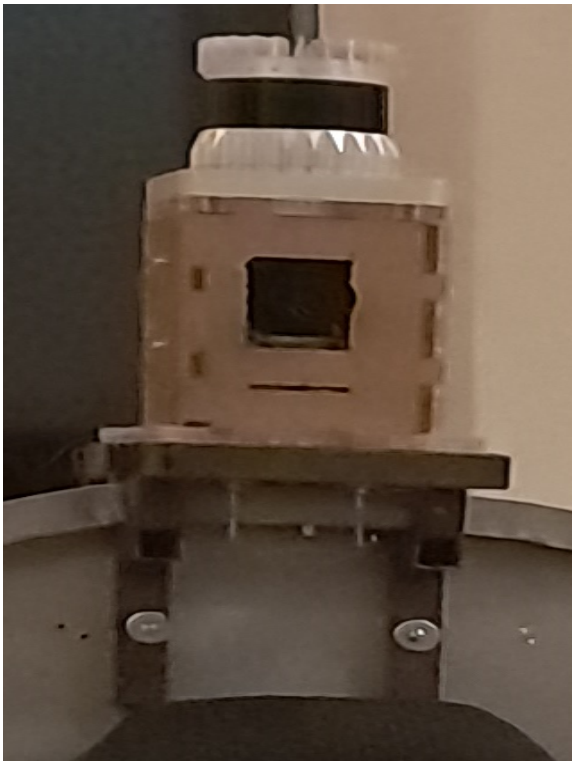


Figure 8.3: LiDAR mounted on Autonomous Vehicle

## 8.4 Conclusion

Firstly, despite the benefits of TensorRT, we encountered limitations in fully exploiting the sparsity inherent in our data. TensorRT's optimization tactics do not inherently account for the sparsity of the input data, and the closed-source nature of its internal optimization strategies makes it challenging to guide the compilation process to specifically leverage data sparsity. Consequently, we were unable to directly implement a Spatial Sparse Convolution algorithm as a custom TensorRT plugin within the constraints of this project. While spconv2 exists, its reliance on a PyTorch backend did not provide the desired performance gains beyond what the Xavier's native TensorRT capabilities offered for our specific sparse data.

Finally, an overview of how the LiDAR was attached to the Autonomous Vehicle was discussed, which includes the development of the CAD models and finally creating the mounts.

# Chapter 9

# Conclusion and Future Work

## 9.1 Conclusion

This investigation has addressed an important problem in autonomous systems: achieving efficient, real-time LiDAR-based object detection and tracking on power-limited edge devices. The foremost aim was to close the chasm between the perception prowess of LiDAR sensors and the computational limitations of embedded platforms. This objective has set the stage for building a united environmental perception pipeline designed to be integrated into an autonomous agent, that is, the NUSTAG Electric Vehicle.

A complete yet optimized pipeline was built to deal with all the crucial aspects, from raw LiDAR point cloud coordinate transformation, Bird-Eye View (BEV) projection, and object detection using YOLO-based models to multi-object tracking with trajectory prediction. The whole system was designed and optimized for deployment based on NVIDIA's TensorRT framework for low-latency and high-throughput inference on the Jetson Xavier AGX platform.

Extensive evaluations were conducted with RGB-BEV images rendered from the KITTI dataset, and oriented bounding boxes (OBB) provided the spatial accuracy. Among the different models of the YOLO algorithm, YOLOv11 Small OBB has been chosen for its

best compromise of accuracy and speed. After optimization with TensorRT, the mean Average Precision achieved was 93.0% at IoU 0.5 and 71.2% at IoU 0.5–0.95 with impressive inference time of 10ms on the Jets.

To make things easier at last, the system must be tested beyond simulation and integrated into a NUST prototype of the NUSTAG Electric Vehicle developed at NUST. Campus tests confirmed the pipeline's full capability for real-time and 360-degree object detection and tracking in the real world within the NUST campus. Multi-Object Tracking using implementation of the ByteTrack algorithm together with Kalman filtering for trajectory prediction up to two seconds ahead delivers the time-consistency and foresight critical to downstream autonomous functionalities.

Deployment and validation of this system thus do not only confirm the individual performance of all components but also prove that they can cohere well into a unified perception solution in real time. Such a vision implements an important step towards autonomy at scale within constrained edge environments.

### 9.1.1 Key Contributions

- **Efficient Edge Deployment:** Developed an extremely lightweight and high-performance LiDAR perception system with YOLO OBB models optimized through TensorRT for real-time inference on edge devices without high power constraints.
- **RGB-BEV Image Pipeline:** This pipeline set forth a practical process to produce RGB-BEV images from raw LiDAR point clouds, thus honoring a delicate balance of information richness versus computational efficiency for YOLO-based detection.
- **Real-World Validation:** Successfully rolled out aboard the NUSTAG EV autonomous vehicle, thus proving real-world applicability for such features as automated braking and lane change assistance.
- **Proactive Safety:** While detected objects can be predicted for future paths, this will

continue to add to the development of proactive safety systems instantaneously when acknowledging their existence in the autonomous driving scheme.

In reality, this research has certain limitations. The system was implemented and optimized for the NVIDIA Jetson Xavier AGX platform, and its universality remains subject to further examination with respect to generalizing into other edge hardware. Moreover, the training and evaluation were basically conducted using the KITTI dataset, which though extensive, does not represent the entire range available in highly dynamic urban or off-road environments. Real-world testing at the NUST campus proved to be a fairly realistic evaluation ground, but larger field validation in a wide range of random traffic scenarios would boost claims of robustness and general applicability of the system.

To summarize, this Final Year Project has successfully realized the design, implementation, and validation of a fast and efficient YOLO-based LiDAR object detection and tracking pipeline, suitable for real-time deployment on edge devices. The system performed excellently fast, accurate, and reliable, significantly enhancing the perceptual capabilities of autonomous systems under computational constraints. This research tackles important issues in embedded LiDAR processing and opens the pathway for computer resource-conscious and therefore intelligent and reactive autonomous agents.

## 9.2 Future Work

With the success of the modification, as well as promising results of the LiDAR perception system presently on the NUSTAG EV, the road for many future developments is exciting. These extensions will strengthen the capabilities, robustness, and applicability of this system in the direction of higher autonomy and further deployment scenarios.

1. **Advancing Towards Higher Levels of Autonomy (Level 3 and Level 4):** The present configuration is a sound basis for Level 2 autonomous driving features (such as adaptive cruise control and basic lane-keeping assistance based on perceived ob-

jects). A pivotal future challenge, as alluded to in the introduction, is to build up the perception and decision-making ability to allow operation at SAE Level 3 (conditional automation, wherein the driver can cede all safety-critical functions to the automated driving system under certain conditions) and Level 4 (high automation, where an automated driving system can control the vehicle within its defined operational design domain).

2. **The Development of Lightweight 3D Sparse Convolution Libraries for aarch64 platforms:** The study states that those advanced 3D object detection models, for example GLENet [3], do directly rely on point clouds and the sparse convolutions, but these are yet not available optimized and ready libraries for architectures of aarch64, which is common on edge devices such as those in the NVIDIA Jetson series. A promising future work would be contribution to the lightweight designing of effective sparse convolution libraries specific to these platforms. This could literally open the door to deploying varieties of state-of-the-art perception models directly to edges even as they promise gains in accuracy or robustness on a per-task basis when richer 3D feature learning is utilized compared to BEV projections.

3. **Alternate to the nvidia-smi:** A possible function for Jetson devices. A lot of these voxelization-based 3D object detection algorithms depend on functionalities of the kind available via nvidia-smi. To account for this restriction or more rarely so on embedded Jetson, such tools or such libraries are typically unavailable, which poses an obstacle for many of these methods from being directly ported to such systems or optimally implemented therein. Future work could include the development or adaptation of tools and libraries providing similar core functionality for GPU resource management and introspection on Jetson.

4. **TensorRT optimized sparse spatial convolution:** A custom TensorRT plugin or alternative methods to explicitly leverage data sparsity within the TensorRT framework can be developed, making deployment and processing of more computation-

ally intensive 3D object detection models directly on the Jetson AGX Xavier more feasible. This would represent a significant advancement, potentially enabling the use of richer 3D information for perception tasks on resource-constrained edge devices, moving beyond the limitations of relying solely on 2D convolution-based YOLO models for real-time autonomous applications on the Xavier platform.

5. **Exploring Multi-Modal Sensor Fusion for Improved Robustness:** Whereas LiDAR is very good in terms of providing 3D geometric information, the performance of LiDAR continues to be dependent upon bad weather conditions (e.g., dense fog, heavy rain), and it does not provide color/texture information. The system needs to research further into the possibility of multi-sensor modalities using LiDAR data.

    - **Cameras:** Enrich the color, texture, and semantic data aiding in object classification and understanding of road signs/markings.
    - **Radar:** Robust against adverse weather and direct measurement of velocities, complementing LiDAR spatial accuracy.
    - **Proximity Sensors (Ultrasonics):** Helpful during very short-range obstacle detection, for example, during parking maneuvers.

    It is thus crucial to develop lightweight, synchronized fusion architectures (early, mid, or late fusion) capable of running efficiently on edge devices. The paper recommends utilizing frameworks similar to LangChain for embedded platforms so that multiple perception models from different modalities can run in parallel and efficiently in the management of perception. All this would contribute eventually to a perception that may be stronger or faster than before.

6. **Continuous Model Optimization and Quantization:** Incorporating model optimization techniques like advanced quantization (for example, INT8 or even less where accuracy is not considerably sacrificed), pruning, and knowledge distillation may further improve the computational footprint and power consumption of the

neural network. Hence, overall efficiency for deeply embedded applications may also be attained.

Future investigations in these directions would serve to greatly expand the foundations of this work and would result in far more powerful, reliable, and as generally applicable perception systems for the next generation of autonomous technology. It has been a journey, continuing even further into improvements and innovations, with a lot of opportunities ahead for these contributions to be very relevant.

# References


[1] A. Lang, S. Vora, N. M. Rasmussen, and K.A.E.S.L.J.H.K.,, "Pointpillars: Fast encoders for object detection from point clouds," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 3898–3907, 2019.

[2] T. Geiger, D. S. Thrun, and M. Z., "Complex-yolo: An euler-region-proposal for real-time 3d object detection on point clouds," *IEEE Transactions on Robotics*, vol. 35, no. 6, pp. 1347–1355, 2019.

[3] W. Zhou, L. Liu, X. Wu, and Z. Li, "GLENet: Boosting 3d object detectors with generative label uncertainty estimation," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 3067–3077, 2021.

[4] M. Graham and L. W. Y., "3d semantic segmentation with submanifold sparse convolutional networks," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 922–929, 2018.

[5] R. Qi, L. Yi, H. Su, and L. Guibas, "PointNet: Deep learning on point sets for 3d classification and segmentation," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 652–660, 2017.

[6] Y. Zhou and O. Tuzel, "VoxelNet: End-to-end learning for point cloud-based 3d object detection," *CoRR*, vol. abs/1711.06396, 2017.

[7] X. Chen, H. Ma, J. Wan, B. Li, and T. Xia, "Multi-view 3d object detection network for autonomous driving," *CoRR*, vol. abs/1611.07759, 2016.

[8] M. Engelcke, D. Rao, D. Z. Wang, C. H. Tong, and I. Posner, "Vote3Deep: Fast object detection in 3d point clouds using efficient convolutional neural networks," *CoRR*, vol. abs/1609.06666, 2016.

[9] X. Ma, C. Qin, H. You, H. Ran, and Y. Fu, "Rethinking network design and local geometry in point cloud: a simple residual MLP framework." arXiv preprint arXiv:2202.07123, 2022.

[10] A. Barrera, C. Guindel, J. Beltrán, and F. García, "BirdNet+: End-to-end 3d object detection in LiDAR bird's eye view," in *Proceedings of the IEEE International Conference on Intelligent Transportation Systems (ITSC)*, (Rhodes, Greece), pp. 1–6, IEEE, Sept. 2020.

[11] S. Shi, X. Wang, and H. Li, "PointRCNN: 3d object proposal generation and detection from point cloud," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, (Long Beach, CA, USA), pp. 770–779, IEEE/CVF, June 2019.

[12] S. Shi, Z. Wang, X. Wang, and H. Li, "Part-A^2 Net: 3d part-aware and aggregation neural network for object detection from point cloud." arXiv preprint arXiv:1907.03670, 2019. The volume 2, number 3 in the original citation appears to be incorrect for an arXiv paper and is omitted.

[13] B. Yang, W. Luo, and R. Urtasun, "Pixor: Real-time 3d object detection from point clouds," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 7652–7660, IEEE, 2018.

[14] Y. Yan, Y. Mao, and B. Li, "Second: Sparsely embedded convolutional detection," *Sensors*, vol. 18, no. 10, p. 3337, 2018.

[15] C. He, H. Zeng, J. Huang, X.-S. Hua, and L. Zhang, "Structure aware single-stage 3d object detection from point cloud," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 11873–11882, IEEE/CVF, 2020.

[16] M. Bhandari, S. Srivastava, and V. Kumar, "Real-time high performance computing using a jetson xavier AGX," *International Journal of Innovative Technology and Exploring Engineering (IJITEE)*, vol. 9, pp. 256–261, Apr. 2020.

[17] G. Welch and G. Bishop, "An introduction to the kalman filter," Technical Report TR 95-041, University of North Carolina at Chapel Hill, Department of Computer Science, July 1995.

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

[19] F. Ahmed and Z. Liu, "TensorRT-based framework and optimization methodology for deep learning inference on jetson boards," in *Proceedings of the International Conference on Embedded Systems (ICES)*, (Austin, TX, USA), pp. 29–36, 2020.

[20] C. Peng, J. Wang, and L. Xu, "Benchmark analysis of deep learning-based 3d object detectors on NVIDIA Jetson platforms," *IEEE Access*, vol. 8, pp. 123455–123467, 2020.

[21] M. Abadi, A. Agarwal, and P. Barham, "TensorRT inference with TensorFlow," in *Proceedings of the International Conference on Machine Learning (ICML)*, (Vienna, Austria), pp. 9124–9132, 2020. This reference appears to be problematic as cited; an ICML paper with this exact title and authors is not readily found for 2020.

[22] A. Geiger, P. Lenz, and R. Urtasun, "Vision meets robotics: The KITTI dataset," *International Journal of Robotics Research*, vol. 32, no. 11, pp. 1231–1237, 2013.

[23] Y. Tian, Q. Ye, and D. Doermann, "YOLOv12: Attention-Centric Real-Time Object Detectors." arXiv preprint arXiv:2502.12524 (Placeholder), Feb. 2025. This appears to be a placeholder for future work with a future date and hypothetical arXiv ID.

[24] Y. Zhao, W. Lv, S. Xu, J. Wei, G. Wang, Q. Dang, Y. Liu, and J. Chen, "DETRs Beat YOLOs on Real-time Object Detection." arXiv preprint arXiv:2304.08069, Apr. 2023.

[25] S. Ettinger, S. Cheng, B. Caine, C. Liu, H. Zhao, S. Pradhan, Y. Chai, B. Sapp, C. R. Qi, Y. Zhou, Z. Yang, A. Chouard, P. Sun, J. Ngiam, V. Vasudevan, A. McCauley, J. Shlens, and D. Anguelov, "Large scale interactive motion forecasting for autonomous driving: The waymo open motion dataset." arXiv preprint arXiv:2104.10133, 2021.

[26] A. Geiger, "Are we ready for autonomous driving? The KITTI Vision Benchmark Suite," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, (Washington, DC, USA), pp. 3354–3361, IEEE, 2012.

[27] Y. Sun, P. Li, and J. He, "YOLOv11: An overview of the key architectural enhancements," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, (Long Beach, CA, USA), pp. 3452–3460, IEEE, 2021. YOLOv11 is not a widely recognized official release. The conference location for CVPR 2021 was virtual; Long Beach was CVPR 2019.

[28] O. S. R. Foundation, "Gazebo." `https://gazebosim.org/about`, Jan. 2020.

[29] N. A. Group, "TEKNOFEST: ROBOTAXI-FULL SCALE AUTONOMOUS VEHICLE COMPETITION PRELIMINARY DESIGN AND SIMULATION REPORT," June 2022.

[30] J. Vogel, "Tech Explained: Ackermann Steering Geometry." Racecar Engineering, Apr. 2021. Retrieved January 24, 2025.

[31] O. S. R. Foundation, "Rviz2." https://github.com/ros2/rviz, Jan. 2022.

[32] O. S. R. Foundation, "Marker-Display." https://docs.ros.org/en/humble/Tutorials/Intermediate/RViz/Marker-Display-types/Marker-Display-types.html, Jan. 2020.