UNIVERSITY OF THESSALY

SCHOOL OF ENGINEERING

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

Sensor Fusion for Autonomous Vehicles in a Simulated Smart City

Diploma Thesis

Eirini Eleftheria Tsatrafili

Supervisor: Nikolaos Bellas

September 2025

UNIVERSITY OF THESSALY

SCHOOL OF ENGINEERING

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

Sensor Fusion for Autonomous Vehicles in a Simulated Smart City

Diploma Thesis

Eirini Eleftheria Tsatrafili

Supervisor: Nikolaos Bellas

September 2025

ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ

ΠΟΛΥΤΕΧΝΙΚΗ ΣΧΟΛΗ

ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

Συγχώνευση Αισθητήρων για Αυτόνομα Οχήματα σε μια Προσομοιωμένη Έξυπνη Πόλη

Διπλωματική Εργασία

Ειρήνη Ελευθερία Τσατραφίλη

Επιβλέπων: Νικόλαος Μπέλλας

Σεπτέμβριος 2025

Approved by the Examination Committee:

Supervisor **Nikolaos Bellas**
Professor, Department of Electrical and Computer Engineering, University of Thessaly

Member **Spyros Lalis**
Professor, Department of Electrical and Computer Engineering, University of Thessaly

Member **Christos Antonopoulos**
Professor, Department of Electrical and Computer Engineering, University of Thessaly

Acknowledgements

Reaching the completion of my Diploma Thesis, I feel deeply grateful to those who stood by me along the way.

First and foremost, I want to thank my supervisor, Dr. Nikolaos Bellas, for consistently offering clear guidance, honest feedback, and practical advice throughout the process.

I'd also like to thank my co-supervisor, Alexandros Patras, whose support, availability, and enthusiasm for this thesis made the entire process possible.

To my friends, thank you for being the best part of this journey. You made the stressful days lighter and the good ones even better. A special thank you to Elena, for always being there for me, in every shared project and in life. One day, we'll finally make TseleTsatra happen, and it'll be worth the wait.

Lastly and above all, I'm beyond grateful to my family. A most special thank you to my parents, Dimitris and Panagiota, and my brother, Aggelos. Your patience, love, immense support and belief in me are what keep me motivated and give me the strength to pursue and achieve everything I set my mind to.

DISCLAIMER ON ACADEMIC ETHICS AND INTELLECTUAL PROPERTY RIGHTS

«Being fully aware of the implications of copyright laws, I expressly state that this diploma thesis, as well as the electronic files and source codes developed or modified in the course of this thesis, are solely the product of my personal work and do not infringe any rights of intellectual property, personality and personal data of third parties, do not contain work / contributions of third parties for which the permission of the authors / beneficiaries is required and are not a product of partial or complete plagiarism, while the sources used are limited to the bibliographic references only and meet the rules of scientific citing. The points where I have used ideas, text, files and / or sources of other authors are clearly mentioned in the text with the appropriate citation and the relevant complete reference is included in the bibliographic references section. I also declare that the results of the work have not been used to obtain another degree. I fully, individually and personally undertake all legal and administrative consequences that may arise in the event that it is proven, in the course of time, that this thesis or part of it does not belong to me because it is a product of plagiarism».

The declarant

Eirini Eleftheria Tsatrafili

Diploma Thesis

Sensor Fusion for Autonomous Vehicles in a Simulated Smart City

Eirini Eleftheria Tsatrafili

Abstract

Autonomous driving depends on the ability of a vehicle to sense and understand its surroundings in real time. To drive safely, the system must detect and identify objects and track their motion. Relying on a single type of sensor is not enough, as they have weaknesses. Cameras depend on lighting and clarity, LiDAR offers precise depth but limited semantics, and radar is weather-resistant but low in resolution. Combining these sensors leads to a more reliable perception of the environment.

This thesis presents the design and implementation of a multi-sensor fusion system for object detection and tracking in simulated driving scenarios. Two complementary pipelines are developed: in 2D, LiDAR and radar data are projected onto the image plane and fused with camera detections, while in 3D, stereo camera reconstruction is combined with LiDAR and radar information in a unified coordinate system. Both early fusion, where raw data are combined before detection, and late fusion, where independent detections are merged at decision level, are examined. Tracking is achieved through Kalman filters, which maintain object trajectories and improve stability even under occlusions or fast motion.

The system is implemented in the CARLA simulator, which provides a realistic traffic environment. A strong focus is placed on real-time performance, including GPU acceleration and frame synchronization, to ensure that the system can handle high sensor data rates. Experiments show that sensor fusion improves both detection accuracy and tracking robustness, while meeting the timing requirements of autonomous driving.

Keywords:

autonomous driving, multi-sensor fusion, object detection, object tracking, CARLA simulator, real-time systems

Διπλωματική Εργασία

Συγχώνευση Αισθητήρων για Αυτόνομα Οχήματα σε μια Προσομοιωμένη Έξυπνη Πόλη

Ειρήνη Ελευθερία Τσατραφίλη

Περίληψη

Η αυτόνομη οδήγηση βασίζεται στην ικανότητα του οχήματος να αντιλαμβάνεται και να κατανοεί το περιβάλλον σε πραγματικό χρόνο. Για ασφαλή πλοήγηση απαιτείται ανίχνευση, αναγνώριση και παρακολούθηση αντικειμένων. Η χρήση ενός μόνο αισθητήρα δεν επαρκεί: οι κάμερες εξαρτώνται από φωτισμό και την διαύγεια, το LiDAR προσφέρει ακρίβεια βάθους αλλά λίγη σημασιολογική πληροφορία, ενώ το ραντάρ είναι ανθεκτικό στις καιρικές συνθήκες αλλά με χαμηλή ανάλυση. Ο συνδυασμός τους οδηγεί σε πιο αξιόπιστη αντίληψη.

Η εργασία αυτή παρουσιάζει τον σχεδιασμό και την υλοποίηση ενός συστήματος συγχώνευσης αισθητήρων για ανίχνευση και παρακολούθηση αντικειμένων σε προσομοιωμένα σενάρια οδήγησης. Υλοποιούνται δύο pipelines: στον δισδιάστατο χώρο, δεδομένα LiDAR και ραντάρ προβάλλονται στην εικόνα και συγχωνεύονται με ανιχνεύσεις κάμερας, ενώ στον τρισδιάστατο χώρο, η στερεοσκοπική ανακατασκευή απο την κάμερα συνδυάζεται με LiDAR και ραντάρ σε κοινό σύστημα συντεταγμένων. Μελετώνται δύο είδη συγχώνευσης: πρώιμη, όπου τα δεδομένα συνδυάζονται πριν την ανίχνευση, και όψιμη, όπου οι ανιχνεύσεις ενώνονται σε επίπεδο απόφασης. Η παρακολούθηση υλοποιείται με φίλτρα Kalman για σταθερότητα ακόμη και σε χάσιμο δεδομένων από τους αισθητήρες ή σε ταχεία κίνηση.

Το σύστημα υλοποιήθηκε στον προσομοιωτή CARLA, που προσφέρει ρεαλιστικό περιβάλλον κυκλοφορίας οχημάτων. Δόθηκε έμφαση στη λειτουργία σε πραγματικό χρόνο, με χρήση GPU και συγχρονισμού καρέ, ώστε να επεξεργάζεται πολλά δεδομένα από τους αισθητήρων. Τα πειράματα έδειξαν ότι η συγχώνευση βελτιώνει την ακρίβεια ανίχνευσης και τη σταθερότητα παρακολούθησης, ικανοποιώντας τις απαιτήσεις της αυτόνομης οδήγησης.

Λέξεις-κλειδιά:

αυτόνομη οδήγηση, πολυαισθητηριακή συγχώνευση, ανίχνευση αντικειμένων, παρακολούθηση αντικειμένων, προσομοιωτής CARLA, συστήματα πραγματικού χρόνου

Table of contents

List of figures

List of tables

Abbreviations

AI	Artificial Intelligence
API	Application Programming Interface
CCD	Charge-Coupled Device
CMOS	Complementary Metal-Oxide Semiconductor
CNN	Convolutional Neural Network
CPU	Central Processing Unit
CUDA	Compute Unified Device Architecture
DBSCAN	Density-Based Spatial Clustering of Applications with Noise
ect	et cetera
FoV	Field of View
FPS	Frames Per Second
FPN	Feature Pyramid Network
GHz	GigaHertz
GPU	Graphics Processing Unit
IDF1	Identification F1
IoU	Intersection over Union
KF	Kalman Filter
LiDAR	Light Detection and Ranging
mAP	mean Average Precision
MEMS	Micro-Electromechanical Systems
MOT	Multiple Object Tracking
MOTA	Multiple Object Tracking Accuracy
Radar	Radio Detection and Ranging
R-CNN	Region-based Convolutional Neural Network
ResNet	Residual Network

RGB	Red Green Blue
SOT	Single Object Tracking
SSD	Single Shot Detector
YOLO	You Only Look Once

Chapter 1

Introduction

Autonomous driving is one of the most rapidly growing research areas, combining computer vision, robotics, and artificial intelligence. Self-driving technology has the potential to completely change transportation by increasing road safety, decreasing traffic congestion and making mobility services more efficient. A fundamental problem of autonomous driving is perception. A vehicle must be able to sense its environment, identify and classify nearby objects, and track their motion over time in order to make safe driving decisions. In practice, this means that these systems must operate reliably in highly dynamic and unpredictable environments, where conditions change from one moment to the next.

Although recent progress has been remarkable, perception continues to pose significant difficulties. Systems that rely on a single type of sensor, such as cameras alone, are vulnerable to poor lighting, bad weather, or occlusions. LiDAR and Radar, while more robust in some areas, have some limitations in resolution and semantic detail. These limitations highlight the importance of combining data from different sensors, a process known as *sensor fusion*. This creates a perception system that is both accurate and reliable. By merging complementary information, such systems can make up for the weaknesses of individual sensors and provide a more complete understanding of the driving scene. Another challenge is making sure that the system performs in real time, since autonomous vehicles must process large volumes of data quickly without compromising safety.

This thesis is situated within this context. It addresses these issues by exploring how multiple sensors can be integrated to improve object detection and tracking, with experiments being performed in a simulated environment.

1.1 Scope of the Thesis

The scope of the thesis is to design, implement, and test a multi-sensor fusion system for detecting and tracking objects in a simulated autonomous driving environment. The aim is to improve accuracy, make the system more reliable under challenging conditions, and ensure it works in real time. Combining different sensors helps overcome the weaknesses of using a single sensor, such as cameras failing in low light or LiDAR providing limited semantic detail.

The work addresses four main challenges:

1. Single-sensor systems often lack robustness. Cameras may struggle with fog, rain or darkness, while LiDAR and radar have lower resolution and less semantic information.

2. Whether it is better to perform the early or the late fusion strategy.

3. Tracking objects consistently over time is difficult, especially when they are partially hidden or in dense traffic.

4. The system must process large amounts of data quickly, without delay.

A perception pipeline is used to address these issues. Camera images are analyzed with deep learning models like YOLOv8, while LiDAR and radar add 3D information. In 2D, visual detections are combined with LiDAR and radar points that are projected onto the camera plane. In 3D, LiDAR and radar measurements are combined with the reconstruction and fusion of stereo camera data.

When objects are partially hidden or sensors momentarily fail, Kalman filters are used to track them over time, maintaining constant positions and velocities. The CARLA simulator, which offers realistic traffic scenarios, is used to run the system. Real-time performance is achieved through GPU acceleration and careful synchronization of sensor data.

Finally, the system is assessed based on runtime performance, tracking stability, and detection accuracy, demonstrating that combining several sensors enhances perception while still being feasible for autonomous driving.

1.1.1 Contributions

The contributions of this thesis are summarized as follows:

1. The configuration of a simulated autonomous driving environment in CARLA, including camera, LiDAR, and radar sensors.

2. The development of a 2D and a 3D perception pipelines with Kalman filter–based tracking for consistent object detection and motion estimation

3. The implementation of real-time techniques, including GPU acceleration and frame synchronization, to achieve efficient system performance.

4. The evaluation of the system in terms of detection accuracy, tracking stability, and runtime efficiency under diverse simulation scenarios.

1.2 Thesis Structure

The remainder of this thesis is structured as follows. Chapter 2 reviews related work in autonomous driving perception. Chapter 3 provides the necessary background, introducing the main sensors (RGB cameras, LiDAR, and radar), object detection and tracking methods, fusion strategies, and the CARLA simulator. Chapter 4 presents the overall system architecture. Chapter 5 details the virtual sensor setup in CARLA, data preprocessing, synchronization, and calibration procedures. Chapter 6 discusses object detection, covering both 2D approaches (from camera, LiDAR, and radar) and 3D methods. Chapter 7 analyzes multi-sensor fusion, describing early and late fusion in both 2D and 3D spaces. Chapter 8 focuses on tracking, including Kalman filtering in 2D and 3D as well as multi-object tracking. Chapter 9 presents the experimental design and evaluation. This includes the simulation scenarios, performance analysis (profiling, GPU acceleration, and real-time feasibility), evaluation metrics, and both quantitative and visual results. Finally, Chapter 10 concludes the thesis with a discussion of the findings and directions for future work.

Chapter 2

Related Work

Research in autonomous perception has grown rapidly in recent years, with many studies focusing on improving object detection, tracking, and sensor fusion. A key focus has been the combination of LiDAR and camera data, since each sensor provides different and complementary information. The following studies provide important background for this thesis and guide its approach.

In object detection, Liu et al. [1] presented a real-time LiDAR–Camera fusion framework, demonstrating that using two sensors together, significantly improves detection and is more reliable than using only one. Zhao et al. [2] proposed a dual transformer architecture for 3D detection, which helps the system handle cases where objects are partly hidden by others. These works show how fusion at the feature level and decision level can make detection more accurate. Wang et al. [3] extended this work by studying collaborative 3D detection between vehicles, emphasizing the importance of communication and cooperation across agents. Together, these studies emphasize the challenge of finding a good balance between accuracy and computation, which is also one of the aims of this thesis.

Work on object tracking has followed a similar path. Zhang et al. [4] introduced ByteTrack, a method that improves multi-object tracking by linking all detection boxes, instead of discarding those with low confidence. This approach is especially effective in crowded scenes. Later, Zhang et al. [5] proposed ACNTrack, which uses information across different sensors to keep tracking consistent in dynamic settings. Boukamchahamdi [6] explained how Kalman filters can be applied to 3D tracking, while Satya [7] demonstrated their usefulness for combining uncertain measurements in sensor fusion. These studies highlight that filtering and association methods are strongest when used together, and this idea is also applied in this

thesis to maintain stable and reliable tracking.

Sensor fusion itself has been studied from several perspectives. Tayyebi [8] compared early and late fusion for camera–LiDAR systems, concluding that early fusion often gives higher accuracy, but is also more sensitive to calibration errors. Yang et al. [9] developed pipelines where camera–LiDAR fusion can adjust dynamically, making the system more flexible in changing environments. Going further, Farag [10] showed the benefits of combining LiDAR and radar for tracking in poor weather and other difficult conditions, while Liu et al. [11] combined roadside camera, LiDAR, and radar for infrastructure-supported perception. More advanced studies have looked at neural-based fusion: Ravindran et al. [12] designed a Bayesian neural network for multi-sensor fusion, while Prakash et al. [13] used transformers for end-to-end driving systems. These works demonstrate that fusion is not just the simple joining of data, but a careful process of designing methods that take full advantage of each sensor's strengths.

Several review papers also add important context. Chaklader [14, 15] organized fusion approaches into three categories: competitive, complementary, and coordinative strategies. Think Autonomous [16] discussed practical challenges of LiDAR–camera fusion, such as synchronization and calibration issues. These reviews highlight the common agreement in the field that no single sensor can provide the level of reliability needed for autonomous systems.

Taken together, these studies lead to three main lessons. First, multi-model detection is required to overcome the weaknesses of individual sensors. Second, tracking performs best when classical filtering methods are combined with strong strategies for linking detections, which makes results more stable. Third, sensor fusion strategies are still developing, with flexible and adaptive methods offering strong potential for future systems. Building on these findings, this thesis brings detection, tracking, and fusion together into a single framework, with a focus on making it efficient, reliable, and practical for real-world use.

Chapter 3

Background

3.1 Sensors

Autonomous vehicles rely on different types of sensors to perceive their surroundings. In this section, we provide an overview of the main sensors used in autonomous systems. These include RGB cameras, LiDAR, and radar.

3.1.1 RGB Cameras

RGB cameras [17] capture images in the visible light range (about 400–700 nm). They are the main source of semantic information for perception. Cameras provide high-resolution images, several megapixels, and usually run at 30–60 frames per second. Most RGB cameras use CMOS or CCD sensors with a color filter array (CFA), often in a Bayer pattern. In this setup, each pixel records only one color (red, green, or blue). The missing colors are then reconstructed through a process called *demosaicing* to create a full-color image.

Cameras are suitable for tasks such as object detection, traffic sign recognition, and object classification. Their performance depends heavily on lighting and weather conditions. Cameras cannot measure depth directly. Stereo setups or structure-from-motion methods can provide approximate 3D information.

3.1.2 LiDAR

LiDAR [18] measures distances by emitting laser pulses and recording the time of flight (ToF) of the returned signal.The distance is computed as:

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

where c is the speed of light and t is the round-trip travel time. By steering the laser across the field of view, by using mechanical rotation or MEMS, LiDAR produces a dense 3D point cloud. The point cloud contains spatial coordinates, intensity values, and sometimes reflectivity. LiDAR provides very precise distance measurements, often accurate to a few centimeters. It also achieves high angular resolution.

Modern LiDARs operate at wavelengths of 905 or 1550 nanometers. Its typical frame rate ranges from 10 to 20 fps. High-end sensors generate over two million points per second at 10 hertz. However, LiDAR has limitations. Its update rate is lower than that of cameras. LiDAR are also relatively expensive.

3.1.3 Radar

Radar [19] uses radio waves to detect objects and determine their position and motion. It works by sending out radio signals that reflect off objects in the environment. Radars commonly operate in the 24 or 77 GHz frequency bands.

The distance is calculated either by measuring the time it takes for the signal to return or by analyzing changes in the signal frequency. Radar can also measure the velocity of an object using the Doppler effect. The Doppler effect occurs when the relative motion between the radar and the object causes a change in the frequency of the reflected waves. If the object moves toward the radar, the frequency increases while if it moves away, the frequency decreases. Radar sensors can also estimate the azimuth angle, which is the horizontal direction of an object relative to the vehicle. This is done by using multiple antennas that direct the radar signal toward specific directions.

Automotive radars are categorized by range. Long range radars, that operate at 10 to 20 fps, can detect objects up to 250 meters, while short range radar, that operate at 50 to 100 fps, are used for parking assistance. Radar is pretty robust in any weather conditions. It can directly measure object velocity. However, angular resolution is relatively poor, meaning it is difficult to separate closely spaced objects using radar alone.

Table 3.1: Comparison of the performance of the sensors.

Category	Camera	LiDAR	Radar	Fusion
Object Detection	○	✓	✓	✓
Object Classification	✓	○	✗	✓
Pedestrian Detection	✓	✗	✗	✓
Distance Accuracy	✗	✓	✓	✓
Range of Visibility	✗	✓	○	✓
Weather Conditions	✗	○	✓	✓
Lighting Conditions	✗	✓	✓	✓
Velocity	✗	✗	✓	✓
Resolution	✓	○	✗	✓
Semantics	✓	✗	✗	✓

3.2 Object Detection

Object detection is the task of identifying and classifying dynamic and static elements within the driving environment. [20] [21] Early approaches primarily relied on hand-crafted features such as Histogram of Oriented Gradients (HOG) or Haar-like descriptors, combined with sliding-window classifiers. While these methods provided the first breakthroughs, they struggled with scalability and robustness in complex, real-world scenes [22]. The adoption of deep learning significantly improved performance by enabling models to learn hierarchical feature representations directly from data [23].

Deep neural detectors are broadly divided into two-stage and one-stage architectures. Two-stage detectors, such as Faster R-CNN, operate by first generating a set of candidate regions that are likely to contain objects, and then performing classification and bounding-box regression on each candidate. This approach tends to provide higher accuracy due to its explicit proposal stage, but the additional computation often leads to slower performance. By contrast, one-stage detectors such as SSD and YOLO predict object classes and bounding boxes directly from dense sampling of the feature map in a single pass, without an intermediate proposal step. These models prioritize real-time inference and strike a balance between speed and accuracy. [24]

While 2D detection remains essential for recognizing semantic objects such as traffic

lights, signs, vehicles, and pedestrians, it is inherently limited by the lack of depth information. To address this, 3D object detection methods have been developed. Stereo-based methods exploit the disparity between two calibrated cameras to reconstruct depth information more reliably. More robust solutions are achieved with LiDAR, which generates point clouds representing the 3D environment with high geometric fidelity. Clustering methods such as DBSCAN or Euclidean clustering are often applied to segment foreground points into object candidates, thereby reducing background noise and improving downstream detection performance.[25]

Multi-modal methods combine camera and LiDAR or radar data to further enhance detection, providing both semantic and geometric information.[26] Detection performance is evaluated using metrics such as mean Average Precision (mAP) and 3D Intersection-over-Union (IoU). These methods are critical in autonomous driving, traffic monitoring, and safety-critical applications, where reliable identification of moving and static objects in complex environments is essential.

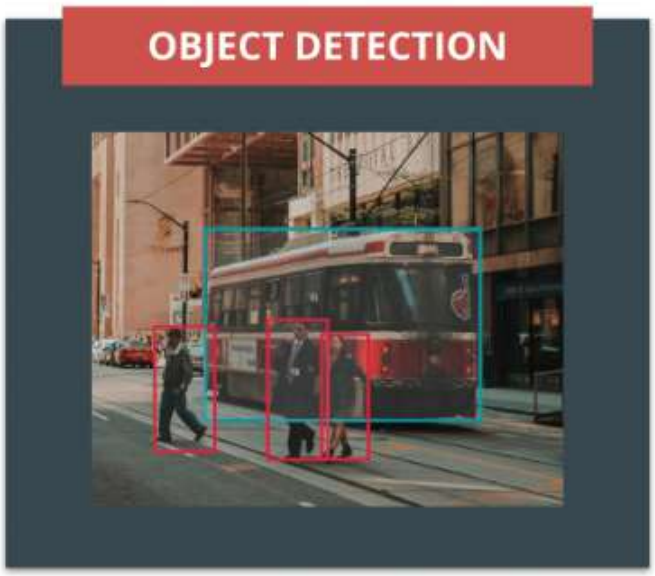


Figure 3.1: Detection Example in the 2D space

3.3 Object Tracking

Object tracking refers to the task of following objects as they move through time. [27] While object detection identifies items in a single frame, tracking ensures that their identity and trajectory are preserved across successive frames. This ability is fundamental in domains

such as autonomous driving, video surveillance, sports analysis, and augmented reality, where understanding motion is as important as recognizing the objects themselves.

The process generally involves three stages. First, objects are detected in each frame. Second, they are assigned unique identifiers to distinguish them from one another. Finally, their positions are updated as they move. Tracking systems can operate in real time and are often divided into two categories. Single Object Tracking (SOT) focuses on following one target, initialized in the first frame. Multiple Object Tracking (MOT), by contrast, extends this principle to many targets simultaneously, which greatly increases the complexity of the task. [28]

In the context of MOT, one of the most difficult challenges is data association. Each detection must be matched with the correct trajectory, and mistakes can quickly lead to identity switches. Approaches range from simple nearest-neighbor assignment to more sophisticated techniques such as probabilistic data association or Multi-Hypothesis Tracking, which consider multiple possible matches before selecting the most likely one.

The effectiveness of tracking systems is usually assessed with established benchmarks. Metrics such as Multiple Object Tracking Accuracy (MOTA) and Identification F1 score (IDF1) are widely used, as they measure not only how well objects are tracked but also whether their identities are preserved over time.

Figure 3.2: Tracking Example in the 2D space

3.4 Fusion Techniques

Sensor fusion [29] [30] refers to the integration of data from multiple sensors in order to achieve a more reliable understanding of the surrounding environment. The motivation for fusion comes from the fact that no single sensor can provide complete information under all operational conditions. Fusion allows systems to benefit from the strengths of each sensor while compensating for their weaknesses

Fusion strategies can be distinguished by the level at which integration occurs. Early fusion [31], sometimes called low-level fusion, integrates raw sensor measurements before feature extraction. A common approach is to project LiDAR point clouds into the image plane to enrich visual features with depth information. While this strategy provides dense input, it is highly dependent on precise extrinsic calibration and temporal synchronization, which may be challenging in practice. Late fusion [32], on the other hand, integrates outputs from modality-specific detectors, merging predictions or trajectories at the decision stage. This approach is modular and more tolerant of imperfect calibration, but it risks losing fine-grained correspondence between modalities.

From a system architecture perspective, fusion can be centralized, where raw sensor data is transmitted to a single processing unit, decentralized, where each sensor processes its data independently and only shares outputs, or distributed, which combines local preprocessing with a central integration layer. Moreover, fusion can serve different functional roles: it may be competitive, where the system selects the most reliable sensor depending on the situation; complementary, where different modalities provide unique, non-overlapping information; or coordinative, where multiple sensors interact to refine a shared understanding of the environment. [33]

3.5 CARLA Simulator

CARLA [34] is an open-source simulator designed for autonomous driving research. It provides a virtual environment where perception, planning, and control algorithms can be tested. The system is built on Unreal Engine, which makes it ideal for realistic graphics and realistic physics.

The simulator supports many types of sensors. These include RGB cameras, LiDAR, radar, GPS, and IMU. Sensor setups can be configured for resolution, field of view, update

rate, etc. Cameras usually operate between 30 and 60 frames per second. This means a frame time of about 16 to 33 ms in real time. LiDAR sensors often run at 10 to 20 Hz. Radar can reach up to 100 Hz.

CARLA also gives control over environmental conditions. Weather settings include rain, fog, wet surfaces, puddles and wind. Lighting can be set to day, sunset, or full night. Cloudiness and sun position can also be adjusted. This makes it possible to test sensors under many different weather and lighting conditions. These features are important because real sensors often fail or degrade under heavy rain, low visibility or poor light.

The simulator also provides dynamic agents such as pedestrians, cyclists, and vehicles. These can be controlled by AI traffic managers. Vehicle physics are modeled with acceleration, braking, and tire-road interactions.

CARLA provides both a Python and a C++ API. It can be connected with deep learning frameworks such as PyTorch and TensorFlow. A major advantage of CARLA is reproducibility. Experiments can be repeated under identical conditions. This is not possible in real-world testing. It also allows stress testing of perception systems by changing frame rates, weather, or sensor setups while keeping the scenario fixed.

Figure 3.3: Carla Simulator

Chapter 4

System Architecture

This chapter provides an overview of the complete pipeline implemented in this thesis. The purpose is to give the reader a high-level understanding of how the different modules, sensor synchronization and calibration, object detection, sensor fusion, and tracking, work together to build a reliable perception system for autonomous driving. This chapter presents the design and the overall flow of information. Subsequent chapters (Chapters 5–8) describe each module in detail.

Overview of the Pipeline

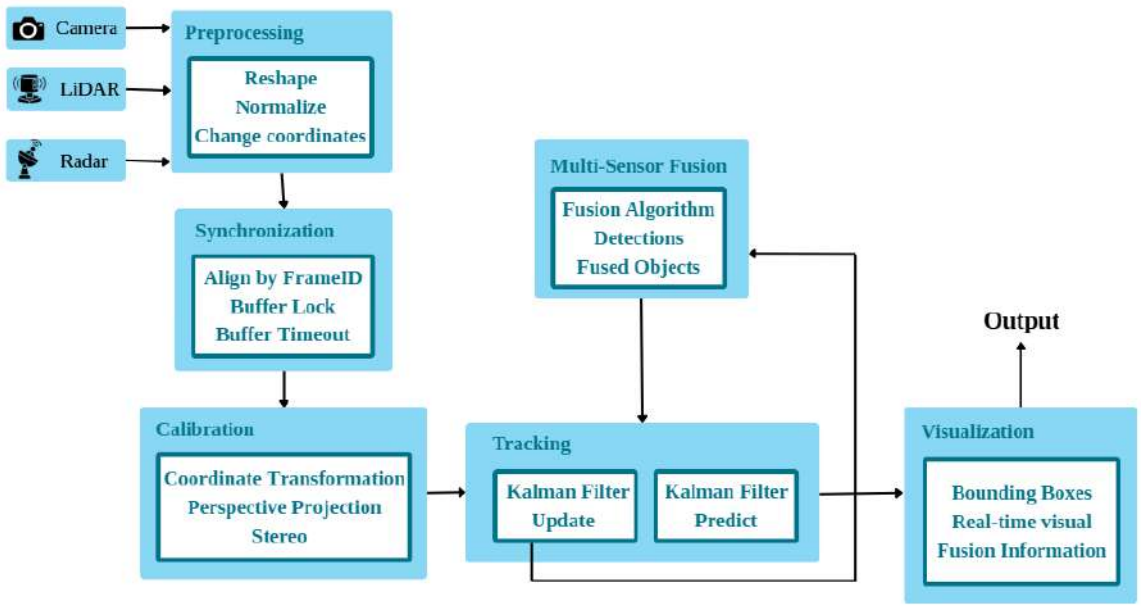


Figure 4.1: High-level architecture of the perception system, showing the flow of information from raw sensor data to multi-object tracking.

The perception pipeline can be seen as a step-by-step flow, where raw sensor data is gradually turned into a reliable model. It starts with the sensors, cameras, LiDAR, radar, and stereo cameras, where each is capturing different information about the surroundings. The raw data from these sensors first goes through preprocessing. For example, camera images are reshaped and normalized.

After preprocessing, the data is placed in a buffer, where frames from all sensors are aligned in time. A timeout mechanism ensures that the system does not freeze if one sensor delivers late or fails temporarily. Once synchronized, calibration is applied to bring all sensors into the same vehicle centered coordinate frame.

At this stage, the pipeline branches depending on the fusion method. In *early fusion*, raw data from different sensors are merged before running detection. In *late fusion*, each sensor runs its own detection first, and the results are then combined. Fusion can be done in 2D (on the image plane) or in 3D (in world coordinates). Either way, the goal is to create one unified set of detections rich with information from all sensors.

These fused detections are then sent to the tracking module. Tracking links objects across frames using Kalman filtering. This step reduces noise, predicts object motion, and keeps track of objects even if some sensor data is missing or unreliable. By combining new detections with predictions from the previous frames, the system produces a stable and continuous world model.

This cycle repeats for every new frame. In this way, the perception system runs in real-time, while remaining robust against noise, delays, or partial sensor failures.

Figure 4.1 illustrates this flow. Each block processes its input and passes structured information to the next stage, ultimately producing a robust world model suitable for autonomous decision-making.

Chapter 5

Sensor Setup and Processing

In autonomous systems, sensors are the primary means of perceiving the environment. Accurate perception depends not only on high-quality sensor data but also on precise calibration. Calibration [35] is the process of estimating the parameters that map each sensor's measurements to a shared reference in space and time. For example, a LiDAR may be mounted on the roof while a camera is placed in front of the car. Without extrinsic calibration, the LiDAR points would not correctly project onto the camera image, causing misalignment in fused data. Calibration ensures that these data from different sensors can be correctly fused in space and time. Without it, small pose or timing errors produce wrong depths, wrongly projected points, duplicated objects, and unstable tracks, which reduces system reliability.

5.1 Virtual Sensors in CARLA

The CARLA simulator provides realistic virtual sensors, allowing testing and calibration in a controlled environment. CARLA supports sensors used in autonomous driving, including RGB cameras, LiDAR, radar, etc. Each sensor exposes resolution, field of view, range, and update rate, and can run in synchronous mode with fixed simulation steps or in asynchronous mode to mimic real pipelines. Sensors are installed on the vehicle using positions and angles that the user sets. This setup allows testing whether the code that combines sensor data works correctly and whether the calibration is accurate.

5.1.1 Autonomous Agent

The autonomous agent in this thesis is represented by a Tesla Model 3 within the CARLA simulator [34]. The vehicle serves as the ego vehicle, equipped with multiple sensors to perceive the environment. Its coordinate system defines the origin for all sensor placements, with the X-axis pointing forward, the Y-axis pointing to the left, and the Z-axis pointing upward. Accurate placement of sensors relative to this coordinate frame is essential for correct spatial fusion and calibration.

The ego vehicle can be controlled either manually through programmed commands or autonomously using CARLA's autopilot mode. The autopilot module enables the vehicle to navigate the environment while obeying traffic rules, lane constraints, and dynamic obstacles, allowing realistic testing of perception, detection, and tracking pipelines under autonomous driving scenarios.

All sensors described in subsequent sections (RGB cameras, LiDAR, radar) are mounted on this vehicle, with their positions and orientations defined relative to the vehicle's coordinate frame. This setup ensures consistency across simulation runs and enables precise evaluation of calibration and fusion algorithms.

Figure 5.1: Tesla Model 3 within the CARLA simulator

5.1.2 RGB Camera Setup

Monocular Camera

A monocular camera captures 2D images from a single viewpoint. Important parameters include image resolution, frame rate, and field of view (FOV). The resolution determines the spatial detail of the captured image, while frame rate defines how frequently images are recorded and the FoV specifies the angle of the scene. In this thesis, monocular cameras are primarily used for 2D perception in detecting and classifying objects in the image plane. The 2D output serves as a basis for fusion with other sensors, providing semantic information that complements geometric measurements.

Figure 5.2: View of the Monocular Camera mounted on the Agent under rainy conditions.

Stereo Camera

Stereo cameras consist of two lenses that capture slightly different viewpoints, allowing depth information to be inferred from the disparity between left and right images. Important parameters include the same factors as for monocular cameras, resolution, frame rate, and FoV, along with the additional parameter of baseline, the distance between the two lenses. A larger baseline improves depth estimation for distant objects but reduces accuracy for nearby objects, whereas a smaller baseline enhances precision in closer perception. Stereo cameras

provide explicit depth information, enabling 3D reconstruction and precise localization of objects. Stereo calibration and disparity matching are essential to generate accurate depth maps. In this thesis, stereo cameras provide 3D spatial information that can be fused with LiDAR and radar measurements, enhancing object detection and tracking accuracy in three-dimensional space. Table 5.1 summarizes the main camera parameters.

(α′) Left view of the Stereo RGB Cameras (β′) Right view of the Stereo RGB Cameras

Figure 5.3: Views of the Stereo RGB Cameras mounted on the Agent

Table 5.1: RGB Camera Parameters in CARLA

Parameter	**Mono**	**Stereo (Left / Right)**
Resolution	800 × 600 px	800 × 600 px
Field of View (FOV)	90°	90°
Frame rate	User-defined	User-defined
Upward offset	1.0 m	1.0 m
Forward offset	2.0 m	2.0 m
Sideward offset	0.0 m	–0.2 m / +0.2 m
Baseline	–	0.4 m

5.1.3 LiDAR Setup

LiDAR provides dense three-dimensional point clouds by emitting laser pulses and measuring their return times. The most important parameters of a LiDAR sensor are the number of vertical channels, the rotation frequency, the measurement range, and the vertical and horizontal fields of view. A higher vertical channel count produces denser point clouds, which improves object detection and mapping accuracy but increases computational cost. Similarly, the rotation frequency affects the temporal resolution of the point cloud. Higher values allow faster moving objects to be tracked more reliably, while lower values reduce data rates. The measurement range defines how far the sensor can detect surfaces, and is often chosen to balance long-distance awareness with near-field precision. The vertical FoV must be wide enough to capture both the road surface and elevated objects such as trucks or overpasses.

Mounting is also a key consideration in LiDAR deployment. In most autonomous driving setups, the LiDAR is placed on the vehicle roof or another elevated position to minimize occlusion from the vehicle body and to maximize environmental coverage. This central, elevated placement ensures that the LiDAR provides a comprehensive view of the environment, which is critical for mapping, obstacle detection, and sensor fusion.

Figure 5.4: View of the LiDAR mounted on the Agent

Table 5.2: LiDAR Sensor Parameters in CARLA

Parameter	Value
Vertical Channels	128
Points per second	1,000,000
Rotation frequency	20 Hz
Range	100 m
Frame rate	User-defined
Upper FoV	30°
Lower FoV	-20°
Noise standard deviation	0.01 m
Upward offset	2.0 m

5.1.4 Radar Setup

Radar sensors operate by transmitting electromagnetic waves and measuring their reflections, providing direct information about the range and relative velocity of objects. Key parameters include the horizontal and vertical fields of view, maximum detection range, angular resolution, and measurement rate. A wider FoV allows the radar to monitor a larger portion of the environment, but this reduces the angular resolution. The maximum detection range determines whether the sensor is more suited to highway scenarios, where long-distance detection is crucial, or urban environments, where short-range awareness is needed. The update rate is also important, as it affects how quickly objects can be tracked.

In practice, radar units are typically mounted at the front and side of the vehicle, with pitch adjustments to reduce ground reflections. Radars that are mounted on the front, often cover long ranges for highway driving, while radars that are mounted on the side are used for blind spot detection. Mounting height and in angle affect the quality of the radar's measurements, as incorrect placement can increase clutter or reduce detection results. When calibrated, radar provides velocity and distance information that significantly enhances the robustness of sensor fusion.

Figure 5.5 shows the radar point cloud, where the points are color coded according to the relative velocity of each object. Red points indicate objects moving away from the sensor, white points indicate objects moving at the same speed as the ego vehicle, and blue points

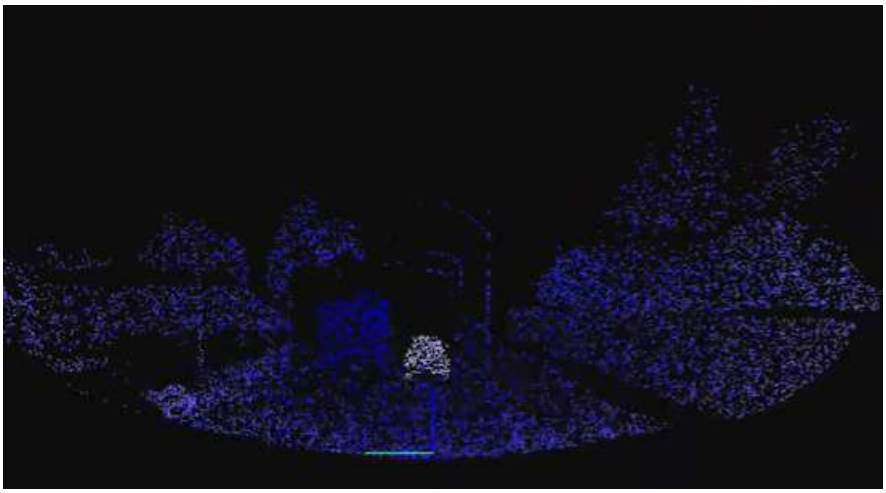


Figure 5.5: View of the Radar mounted on the Agent

indicate objects approaching. In the image, we can see a white car directly ahead moving at roughly the same speed as the ego vehicle, while a dark blue truck in the adjacent lane is approaching, indicating it is moving faste towards the agent.

Table 5.3: Radar Sensor Parameters in CARLA

Parameter	**Value**
Horizontal FOV	100°
Vertical FOV	50°
Points per second	250,000
Range	100 m
Frame rate	User-defined
Pitch	5°
Upward offset	1.0 m
Forward offset	2.0 m

5.2 Sensor Data Preprocessing

Each sensor provides raw measurements that require preprocessing before they can be processed into a consistent perception pipeline. Proper preprocessing ensures that the data is aligned, cleaned, and represented in a form suitable for tasks.

RGB Camera

Monocular cameras capture 2D RGB images, which are initially provided as raw byte arrays. These arrays must be reshaped into $(H \times W \times 3)$ arrays corresponding to the image height, width, and color channels. Pixel values are typically normalized, and images are resized to match the input of neural networks.

LiDAR

LiDAR sensors produce dense 3D point clouds, where each point contains position and intensity information (x, y, z, i). The raw data is typically provided as a flat float buffer, which must be reshaped into $(N, 4)$ arrays for N points.

Radar

Radar sensors provide measurements in polar coordinates, specifically azimuth θ, altitude ϕ, range r, and relative velocity v. These measurements are converted into Cartesian coordinates using:

$$x = r\cos\phi\cos\theta, \quad y = r\cos\phi\sin\theta, \quad z = r\sin\phi.$$

The final output is a structured array (x, y, z, v).

Table 5.4: Sensor Data and Preprocessing Summary

Sensor	**Raw Data**	**Preprocessing**
Camera	RGB image	Reshape, normalize, resize
LiDAR	$(x, y, z, intensity)$	Reshape, filter
Radar	(θ, ϕ, r, v)	Convert to (x, y, z, v), stack

5.3 Synchronization

In a multi-sensor perception pipeline, accurate synchronization is just as critical as spatial calibration. This is essential because autonomous driving environments are highly dynamic: even small temporal misalignments can result in significant errors, such as placing objects at incorrect positions.

5.3.1 Different Sensor Rates

Each sensor operates at a different frequency. For example, cameras provide faster than LiDAR and Radar. Moreover, radar operates at a much slower rate due to the extensive preprocessing it requires. This means that, for a given moment in time, the sensors do not automatically provide a consistent set of measurements. Figure 5.6 illustrates this challenge. It shows in the passage of time how frames from different sensors are processed at completely different rates.

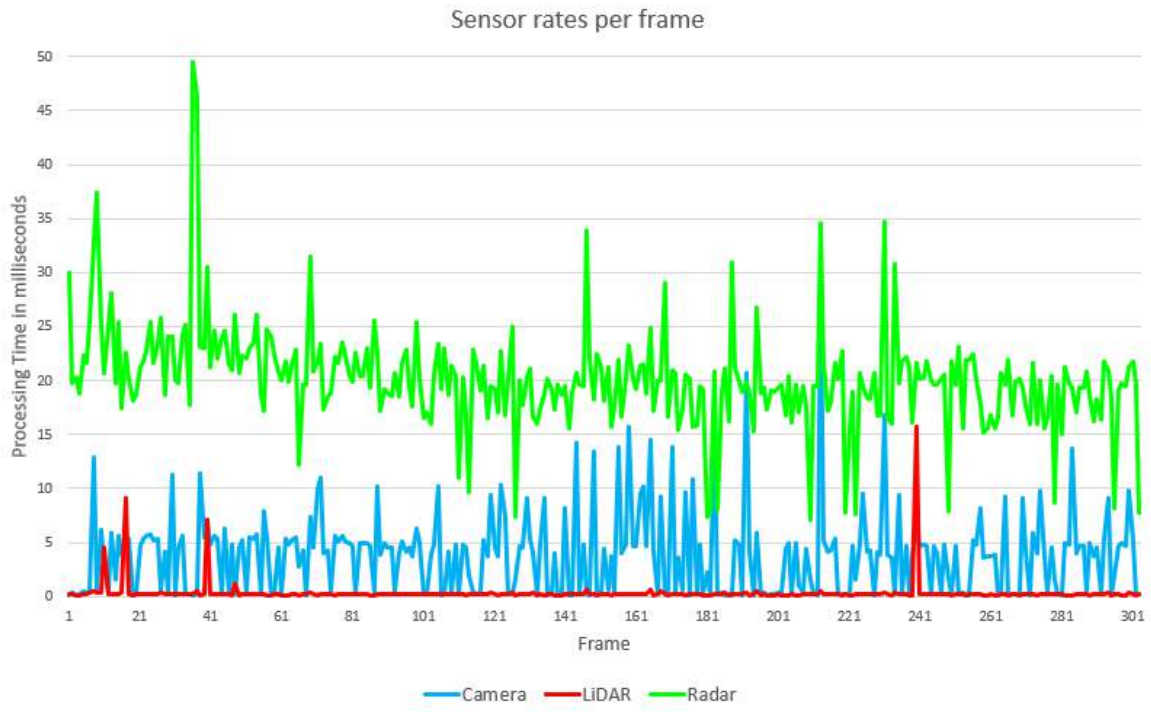


Figure 5.6: Sensor preprocessing rates per frame.

5.3.2 Frame Buffer Synchronization

Synchronization ensures that data from all sensors corresponds to the same frame ID. To address this, a frame buffer mechanism was implemented. The buffer stores data from each sensor, indexed by a unique *frame ID*. When a measurement from a sensor arrives, it is temporarily blocked until matching data from all other sensors with the same frame ID is available. Only when all expected inputs are received does the buffer release the synchronized frame to the perception pipeline. This ensures that multi-sensor fusion operates on temporally aligned data, avoiding inconsistencies caused by rate mismatches.

Handling Missing Data

In real world and simulated environments, it is not guaranteed that all sensors deliver data reliably at every frame. To maintain real time operation, a timeout mechanism is applied. If a specified waiting period passes after the first sensor delivers data for a given frame ID, and one or more other sensors have not yet provided their data, the missing measurements are marked as unavailable. The buffer then releases the partially complete set of sensor data to the pipeline, ensuring that perception continues without blocking.

This limitation shows one of the most important reasons for relying on multiple different sensors rather than a single one. Individual sensors can fail temporarily, produce noisy measurements, or return unreliable information in challenging conditions, e.g. fog for cameras.

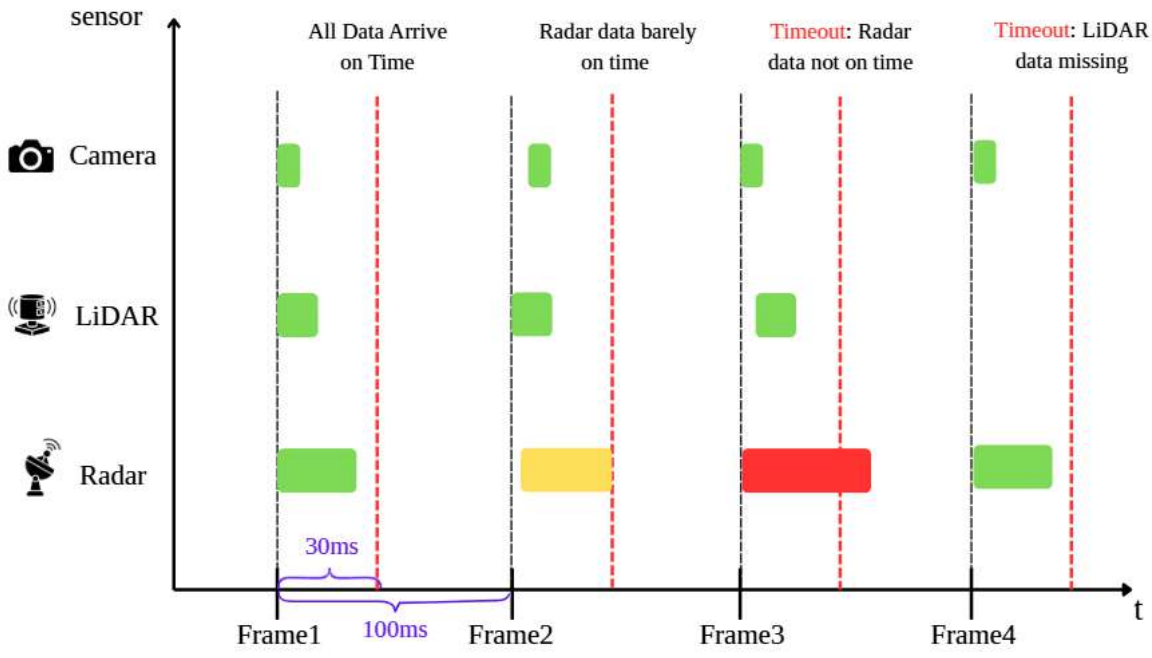


Figure 5.7: Example of Sensor Stream Alignment with Timeout

Figure 5.7 presents a full example of how the buffer works on real-time weather the data have arrived before the timeout or not. It is established that the frames arrive every 100 ms and that the timeout margin is 30 ms (see Figure 5.6 for an understanding of how many sensor data points fall outside this margin). Frame 1 shows all data arriving on time. Frame 2 shows that data might start arriving at different times but all data should arrive before the timeout even if they bareky reach it. Frame 3 displays a timeout due to late arrival of data, while Frame 4 shows a timeout due to missing data.

This design has two key advantages:

- **Robustness:** The pipeline continues to run in real-time, even if one sensor temporarily fails or delivers data late.
- **Consistency:** When all sensors succeed in providing data on time, their measurements are guaranteed to be aligned by frame ID, producing synchronized fusion results.

In summary, synchronization ensures that multi-sensor data can be correctly combined, while redundancy protects the perception system against delays, failures, or degraded measurements from any single sensor. This forms the foundation of reliable multi-sensor fusion, enabling robust operation even under imperfect real world conditions. Figure **??** shows the complete flow of the synchronization process.

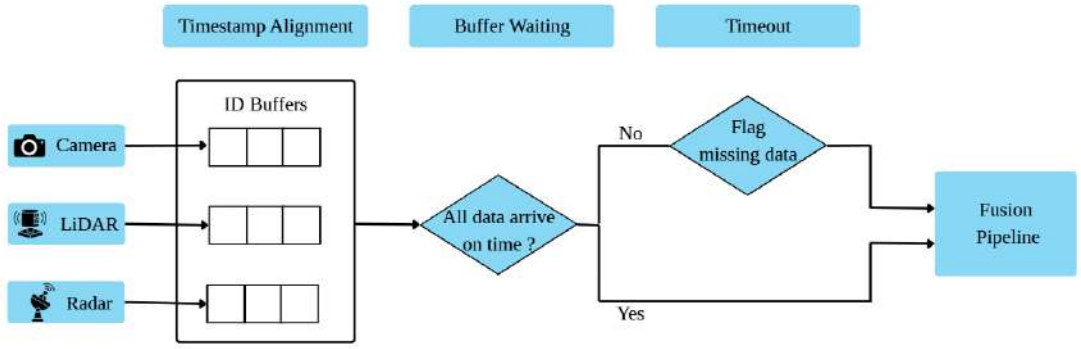


Figure 5.8: General synchronization process for multi-sensor alignment. Streams are aligned by timestamp, buffered, checked for timeout, and either released as a complete set or flagged with missing data.

5.4 Sensor Calibration

Sensor calibration is a crucial step in any perception system, as it ensures that all sensors provide geometrically consistent and accurate measurements. Calibration involves determining both the intrinsic parameters of each sensor, which describe the internal geometry and imaging properties, and the extrinsic parameters, which define the position and orientation of sensors relative to a reference frame. Accurate calibration enables multi-sensor fusion, 3D reconstruction, object detection, and depth estimation in autonomous systems.

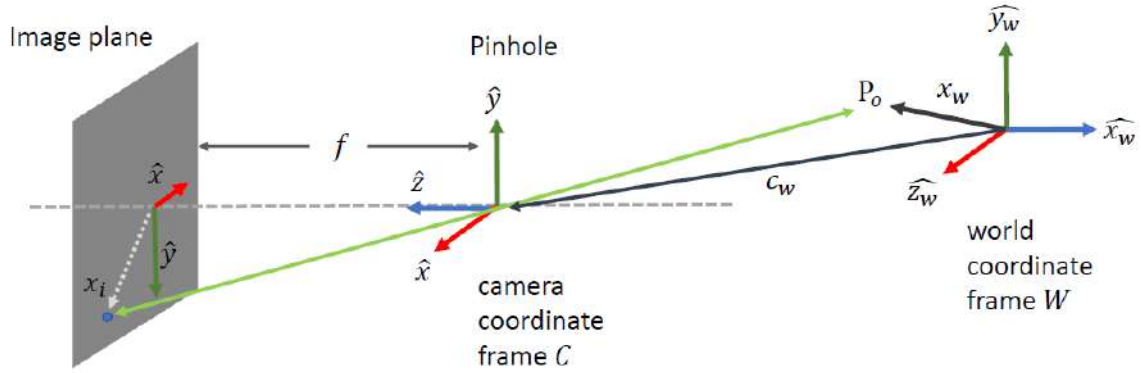


Figure 5.9: Forward Imaging Model: 3D to 2D

5.4.1 Perspective Projection (Intrinsic Calibration)

Perspective projection is the fundamental imaging model in which all rays from the 3D scene pass through a single projection center before intersecting the image plane. This projection center, called the *pinhole*, serves as the viewpoint of the camera. In this central projection, every 3D point is mapped to a 2D image point along a straight line through the pinhole. Calibration ensures that the internal characteristics of the camera are accurately represented, including the focal length, pixel geometry, and sensor alignment, which are critical for multi-sensor fusion.

The intrinsic parameters encode the internal geometry of the camera sensor and lens system. They determine how rays in the camera coordinate frame are mapped to pixel coordinates in the image plane. In homogeneous coordinates, the intrinsic calibration matrix is defined as :

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

where each term has a distinct physical meaning:

- Focal lengths (f_x, f_y). These represent the focal length expressed in pixels along the horizontal and vertical axes:

$$f_x = \frac{f}{m_x}, \qquad f_y = \frac{f}{m_y},$$

where f is the physical focal length of the lens and m_x, m_y are the physical pixel dimensions (mm/pixel). If the sensor has square pixels ($m_x = m_y$), then $f_x = f_y$. If

the pixels are rectangular ($m_x \neq m_y$), the scales differ and geometric distortions occur, circles are projected as ellipses, unless corrected.

In the CARLA simulation, the effective focal length is computed directly from the image resolution and field of view as :

$$f_x = f_y = \frac{\text{width}}{2\tan(\text{FoV}/2)},$$

assuming square pixels. This ensures that the projection of 3D points onto the 2D image plane respects the simulated camera's FOV and resolution.

- Principal point (c_x, c_y). The principal point represents the intersection of the camera's optical axis with the image plane. In theory, it is the geometric center of the sensor, but in practice, there are small shifts due to lens assembly or sensor misalignment. Mathematically, it is expressed as :

$$c_x = \frac{\text{width} - 1}{2}, \quad c_y = \frac{\text{height} - 1}{2}.$$

Including the principal point in the intrinsic matrix ensures that the projection of 3D points onto the image plane is correctly referenced relative to the sensor origin.

Thus, given a 3D point in the camera coordinate frame $P_C = (X_C, Y_C, Z_C)$, the corresponding image pixel (u, v) is computed using the intrinsic parameters as:

$$u = f_x \frac{X_C}{Z_C} + c_x, \qquad v = f_y \frac{Y_C}{Z_C} + c_y.$$

In homogeneous coordinates, the projection can be compactly represented as:

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \mathbf{K} \begin{bmatrix} X_C \\ Y_C \\ Z_C \\ 1 \end{bmatrix},$$

This form is essential in computer vision pipelines, as it provides a direct mapping from 3D camera-centric coordinates to 2D image coordinates, enabling precise alignment between sensor measurements and image data.

5.4.2 Coordinate Transformation (Extrinsic Calibration)

While intrinsic calibration describes the internal properties of a camera, extrinsic calibration defines its position and orientation relative to the world or other sensors. The extrinsic parameters consist of a rotation matrix $\mathbf{R} \in \mathbb{R}^{3\times3}$ and a translation vector $\mathbf{t} \in \mathbb{R}^3$, which together locate the camera in the world coordinate frame $\mathcal{W}$.

The rotation matrix $\mathbf{R}$ is an orthogonal 3×3 matrix that aligns the world coordinate axes with the camera's axes. It can be built from rotations about the standard X, Y, and Z axes:

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

A general rotation matrix $\mathbf{R}$ is obtained by multiplying these elementary matrices in a specific order depending on the rotation sequence (e.g., ZYX, XYZ). For example, for yaw (ψ), pitch (θ), and roll (ϕ), the combined rotation is:

$$\mathbf{R} = \mathbf{R}_Z(\psi)\mathbf{R}_Y(\theta)\mathbf{R}_X(\phi).$$

This matrix transforms coordinates from the world frame to the camera frame, aligning the camera's axes with the world axes.

The translation vector $\mathbf{t} = [t_x, t_y, t_z]^T$ represents the position of the camera's origin in the world frame. It shifts the origin of the world frame to the camera's location. Together, rotation and translation allow transforming a point $\mathbf{P}_W = [X_W, Y_W, Z_W]^T$ from world coordinates to camera coordinates:

$$\mathbf{P}_C = \mathbf{R}(\mathbf{P}_W - \mathbf{t}).$$

Here, the world point $\mathbf{P}_W$ is first translated by $-\mathbf{t}$ to the camera origin, then rotated by $\mathbf{R}$ to align with the camera's axes. The complete extrinsic transformation can also be expressed as a 4×4 matrix in homogeneous coordinates:

$$\mathbf{T}_{WC} = \begin{bmatrix} \mathbf{R} & \mathbf{t} \\ \mathbf{0}^T & 1 \end{bmatrix},$$

so that :

$$\mathbf{P}_C = \mathbf{T}_{WC}\,\mathbf{P}_W, \quad \mathbf{P}_C = \begin{bmatrix} X_C \\ Y_C \\ Z_C \\ 1 \end{bmatrix}, \quad \mathbf{P}_W = \begin{bmatrix} X_W \\ Y_W \\ Z_W \\ 1 \end{bmatrix}.$$

This form provides a consistent and precise way to map points from the world frame into the camera frame, which is essential for 3D reconstruction.

5.4.3 Multi-Sensor Coordinate Transformations

Building on the extrinsic calibration framework, multi-sensor systems require expressing measurements from different sensors in a common reference frame. Typical transformations include mapping LiDAR or radar points into the camera frame.

Let $\mathbf{T}_{C\to W}$, $\mathbf{T}_{L\to W}$, and $\mathbf{T}_{R\to W}$ be the extrinsic transformation matrices of the camera, LiDAR, and radar with respect to the world frame W respectively. The transformation from a sensor frame S (LiDAR or radar) to a target frame T (camera, radar, or LiDAR) is given by:

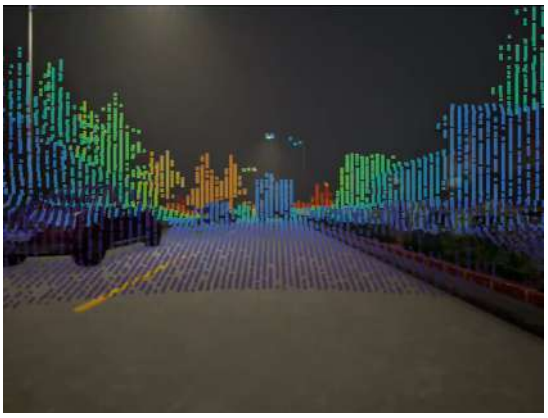
$$\mathbf{T}_{S\to T} = \mathbf{T}_{W\to T}\,\mathbf{T}_{S\to W},$$

where $\mathbf{T}_{W\to T} = \mathbf{T}_{T\to W}^{-1}$ is the inverse of the target-to-world transformation.

In simple words, firstly the point is moved from the sensor's local frame into the world frame ($\mathbf{T}_{S\to W}$). Then it is mapped from the world frame into the target sensor's frame ($\mathbf{T}_{W\to T}$). The result is the same point, now expressed in the coordinates of the target sensor. This guarantees that all measurements, whether they come from LiDAR, radar, or a camera, can be directly compared or combined.

3D point to 2D point

A common case is projecting 3D points onto a 2D image. To do this, the 3D points are first transformed into the world frame, then into the camera frame using the above formula. Once expressed in camera coordinates, they can be projected onto the image plane using the intrinsic matrix. This enables tasks like overlaying depth information onto camera images.



(α′) LiDAR points mapped and projected into the camera frame, colored according to the points depth.

(β′) Radar points mapped and projected into the camera frame, colored according to their relative velocity.

(γ′) View of the RGB Camera for reference.

3D Mapping

Another useful case is mapping Radar points into the LiDAR frame. Both sensors are mounted on the same platform but usually at different positions. The LiDAR points are transformed into the world frame and then into the radar frame. This process is important because it allows radar detections to be directly compared with LiDAR point clouds. By aligning both sources of data in the same coordinate frame, the object locations can be verified.

By chaining transformations through the world frame, any sensor's measurements can be expressed in another sensor's frame. This unified view is the backbone of multi-sensor fusion pipelines, enabling cameras, LiDARs, and radars to work together.

(α′) Radar points mapped and projected into the LiDAR frame.

(β′) View of the RGB Camera for reference.

5.4.4 Stereo Calibration

Stereo calibration is the process of determining the geometric relationship between two cameras observing the same scene. Unlike intrinsic calibration, which describes the internal geometry of a single camera, stereo calibration focuses on the relative pose between a left camera and a right camera. Let the left camera be considered as the reference coordinate frame C_L, and the right camera as C_R. The transformation between them is defined as:

$$\mathbf{P}_{C_R} = \mathbf{R}_{L\to R}\,\mathbf{P}_{C_L} + \mathbf{t}_{L\to R},$$

where $\mathbf{P}_{C_L}$ and $\mathbf{P}_{C_R}$ are the coordinates of the same 3D point expressed in the left and right camera frames, $\mathbf{R}_{L\to R}$ is the rotation matrix, and $\mathbf{t}_{L\to R}$ is the translation vector. In homogeneous coordinates, this relation is expressed as :

$$\mathbf{T}_{L\to R} = \begin{bmatrix} \mathbf{R}_{L\to R} & \mathbf{t}_{L\to R} \\ \mathbf{0}^T & 1 \end{bmatrix}.$$

If $\mathbf{T}_{C_L\to W}$ and $\mathbf{T}_{C_R\to W}$ are the extrinsic matrices of the left and right cameras with respect to the world frame, then the relative transformation from the left to the right camera can be written as :

$$\mathbf{T}_{L\to R} = \mathbf{T}_{W\to C_R}\,\mathbf{T}_{C_L\to W} = \left(\mathbf{T}_{C_R\to W}\right)^{-1}\mathbf{T}_{C_L\to W}.$$

The main outcome of stereo calibration is the ability to compute depth using the principle of triangulation. A point in the scene is projected onto the left image at pixel coordinates (u_L, v_L) and onto the right image at (u_R, v_R). After stereo rectification, the two image planes are aligned so that the vertical coordinates are equal, $v_L = v_R$. The horizontal shift

$$d = u_L - u_R$$

is called the *disparity*. The disparity is directly related to the depth of the observed point.

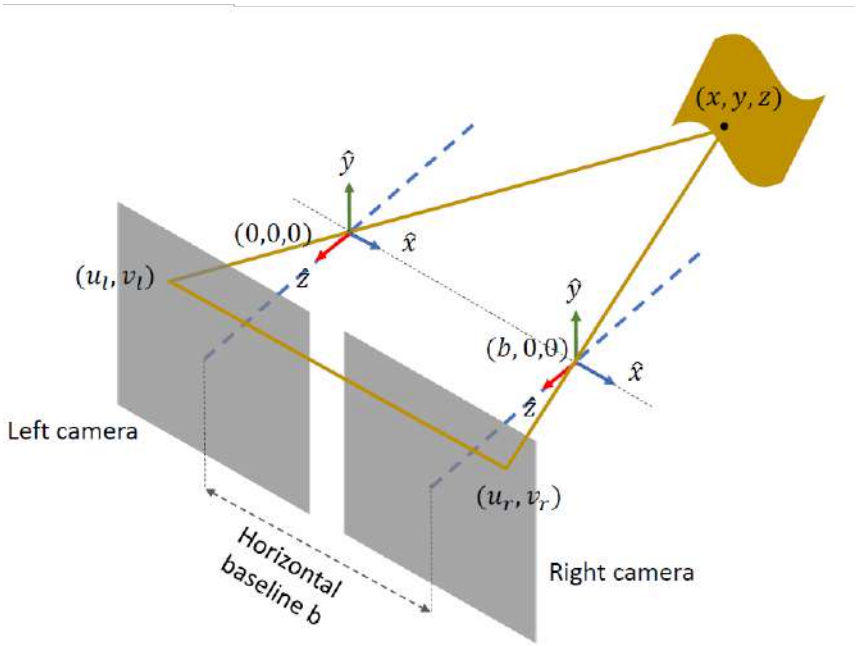


Figure 5.12: Stereo geometry with baseline B and 3D point triangulation.

Let f_x, f_y be the focal lengths along the image axes, and let c_x, c_y be the principal point. The baseline $B = \|\mathbf{t}_{L\to R}\|$ is the distance between the optical centers of the two cameras. With these parameters, the depth Z of the 3D point can be computed as :

$$Z = \frac{f_x \cdot B}{d}.$$

This shows that large disparities correspond to points that are close to the cameras, while small disparities correspond to points further away.

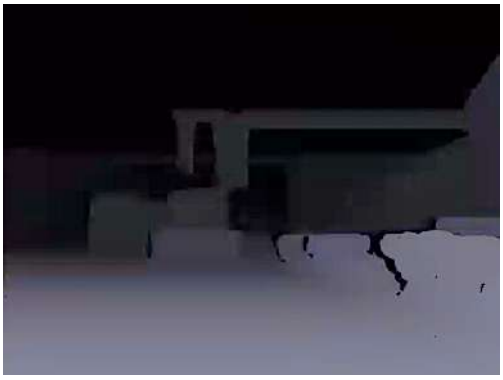
Once Z is known, the full 3D coordinates of the point in the left camera frame can be reconstructed:

$$X = \frac{(u_L - c_x)\, Z}{f_x} = \frac{(u_L - c_x)\, B}{d}, \quad Y = \frac{(v_L - c_y)\, Z}{f_y} = \frac{(v_L - c_y)\, f_x B}{f_y\, d}, \quad Z = \frac{f_x B}{d}$$

These equations show analytically how disparity, baseline, and camera intrinsics work together to recover the metric 3D coordinates (X, Y, Z). Importantly, the accuracy of the reconstruction strongly depends on the baseline length B and the precision of disparity. A larger baseline improves depth perception.

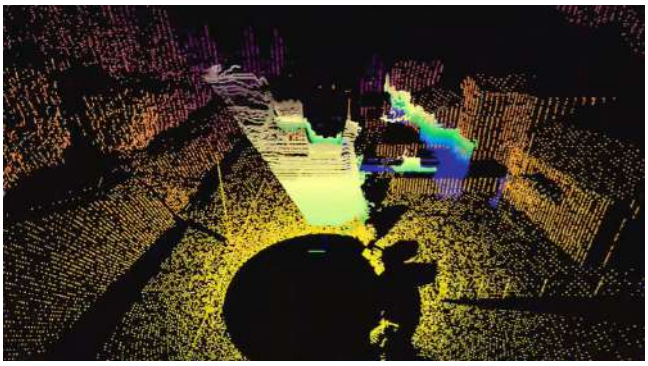
The disparity map is a visual representation of depth information derived from stereo image pairs. Each pixel in the map encodes the horizontal shift, or disparity, between its projections in the left and right images. The disparity map provides a depth estimate of the

scene, which can be directly used for 3D reconstruction, environment understanding, and obstacle detection.



(α′) Disparity map.

(β′) View of the RGB Camera for reference.



(γ′) Stereo 3D points mapped and projected into the LiDAR frame.

Figure 5.13: Comparison of disparity, stereo projection and the RGB camera view.

Stereo 3D points can be transformed into the LiDAR frame using the multi-sensor transformation framework as seen in Figure 5.13γ′. By projecting 3D points reconstructed from stereo disparity into the world frame and then transforming them into the LiDAR coordinate system, one can directly compare the stereo point cloud with LiDAR measurements. It should be noted that, the quality of disparity is highly dependent on environmental conditions: in bad weather such as heavy rain, fog, or low light, the disparity map becomes noisy and unreliable, whereas in clear and good weather conditions it produces accurate and consistent results as seen in Figure 5.13α′.

Chapter 6

Object Detection

Object detection is the process of turning raw sensor data into useful information about the surrounding environment. In practice, this means finding where objects are and what they are. Cameras, LiDAR, radar, and stereo vision contain valuable information, but without further processing, the data remain unstructured. Object detection methods extract patterns, classify them into semantic categories and provide geometric descriptions of their position in space. This step is very important because it forms the basis for later tasks such as tracking, motion prediction, and decision-making in autonomous driving.

It is important to note that the detection methods presented in this chapter can be applied in both early and late fusion schemes, which will be discussed in detail in the following chapter. Early fusion benefits from raw or intermediate sensor representations, where detections from different modalities are aligned at the feature or data level. Late fusion, on the other hand, combines independent detection results. Depending on the application and system constraints, either approach can take advantage of the 2D and 3D detection pipelines described here.

6.1 2D Detection

In 2D detection, objects are localized inside an image. The output is usually a rectangular bounding box with a class label and a confidence score. This information allows a system to understand what objects are present and where they are located in the image plane.

6.1.1 Camera

Detection architectures are generally divided into two-stage detectors and one-stage detectors, as explained in the *Background*. In previous work of mine, several object detection models were evaluated in terms of speed, accuracy, and hardware requirements. The models tested were:

- **YOLOv8**, a fast one-stage detector designed for real time. It predicts bounding boxes and class labels in a single pass, offering a strong speed/accuracy trade-off.

- **FCOS-ResNet50-FPN**, an anchor-free, one-stage model with a ResNet-50 backbone and an FPN. Its design is simple, but in experiments it was less accurate and less robust than YOLOv8 and Faster R-CNN, especially for challenging cases.

- **Faster R-CNN**, a two-stage model divided in two categories:

  - *MobileNet v3 Large FPN*, optimized for speed and efficiency using depthwise separable convolutions and an FPN for multi-scale features. It is suitable for limited hardware.

  - *ResNet50 FPN v2*, higher accuracy for complex scenes due to a stronger backbone and FPN, but it increases the computation.

- **SSD-Lite 320 (MobileNet v3 Large)**, a lightweight SSD variant for fast inference on embedded devices. It is very efficient but generally less accurate than the larger models above.

Testing showed that Faster R-CNN ResNet50 FPN v2 achieved the highest detection accuracy on high-performance GPUs due to its strong feature extraction capabilities, making it the preferred choice for complex tasks. However, its computational cost is higher, making it a non real-time option. On the other hand, YOLOv8 offered the best balance of speed and accuracy, enabling efficient real-time detection.

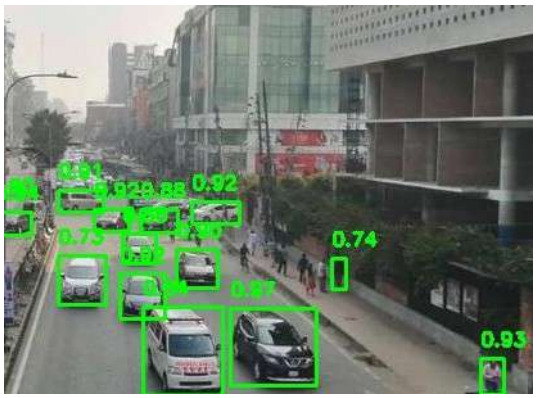


Figure 6.1: Excellent detection example: Faster R-CNN ResNet50 FPN v2.

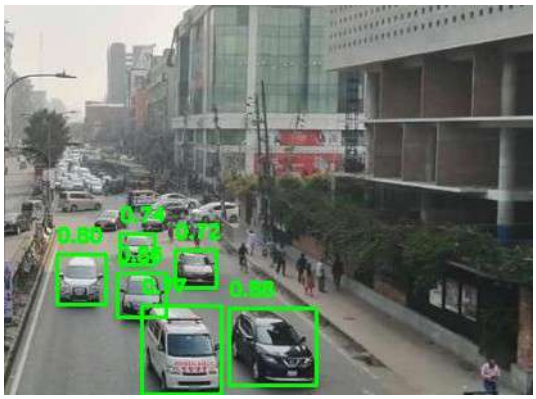


Figure 6.2: Good detection example: YOLOv8s.

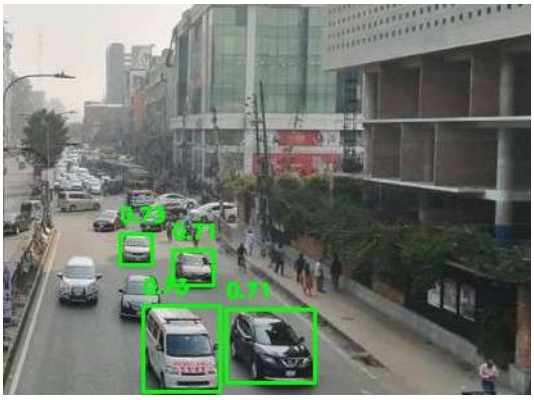


Figure 6.3: Bad detection example: FCOS-ResNet50-FPN.

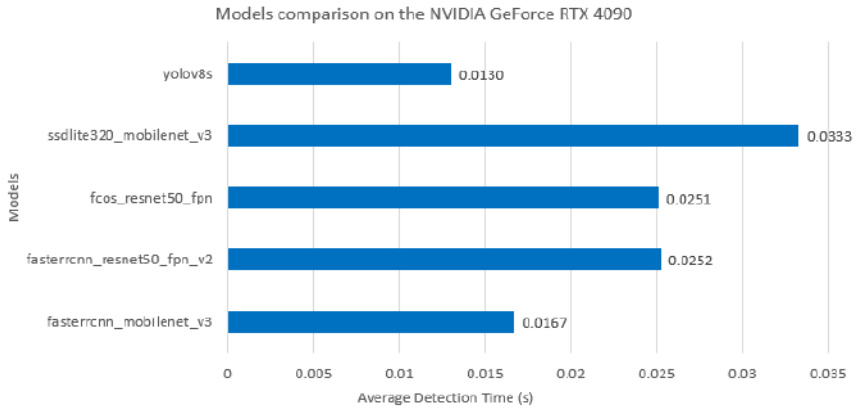


Figure 6.4: Models comparison on the NVIDIA GeForce RTX 4090 on average detection time per frame.

Based on these results, YOLOv8 was selected for 2D detection in this thesis, providing an optimal trade-off between computational efficiency and detection performance in real time.

YOLOv8 Architecture

YOLOv8 is a one-stage object detector designed for real-time applications. For each detected object, the network outputs three pieces of information: the bounding box coordinates, which show where the object is in the image, the class label, which identifies what the object is, and the confidence score, which indicates how certain the model is about the detection.

Before being passed to the network, images go through a preprocessing stage to ensure consistent input and improve detection quality. Each image is resized to match the model's input dimensions, and pixel values are normalized to make training and inference more stable. During training, data augmentation techniques such as flips, rotations, and small color changes are also applied. These steps make the model more robust to variations in lighting, scale, and viewpoint, helping it perform better on new, unseen data.

The YOLOv8 network is organized into three main components. The **Backbone** is a convolutional network that extracts features from the image, capturing both simple patterns like edges and more complex ones like object parts. The **Neck** combines features from different scales using feature fusion structures, which helps detect both large and small objects. Finally, the **Head** produces the final detections by predicting bounding boxes, class labels, and confidence scores in a single step.

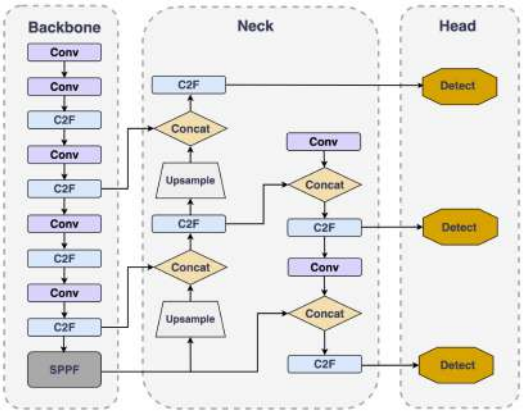


Figure 6.5: YOLOv8 Architecture

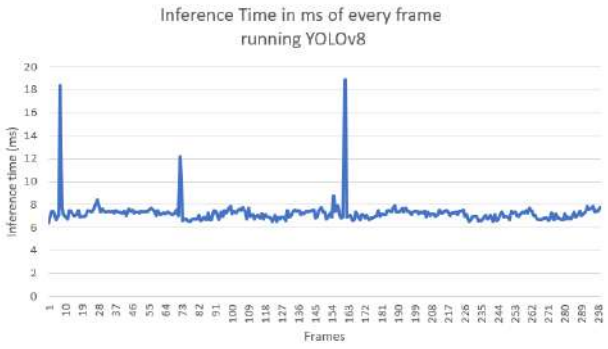
This design allows YOLOv8 to achieve high accuracy while keeping inference fast enough for real-time use. Compared to earlier YOLO versions, YOLOv8 introduces a cleaner architecture and improved training strategies, making it more flexible and accurate without losing the speed.

Figures 6.6α′, 6.6β′, 6.6γ′ shows two aspects of the YOLOv8 detector. The first images

illustrates the model's output on a sample traffic scene, with bounding boxes and class labels over detected objects. The last image presents the execution time for each frame, highlighting the model's real-time performance and consistency across multiple frames.

(α′) Example of YOLOv8 detection on a traffic scene in CARLA in bad weather.

(β′) Example of YOLOv8 detection on a traffic scene in CARLA in nice weather.



(γ′) Execution time per frame for YOLOv8 inference.

6.1.2 LiDAR/Radar

For 2D object detection using LiDAR or Radar, the 3D points captured by the sensors are first projected onto the image plane. Each point is given a color that represents its distance from the sensor. This creates a 2D image where the depth information is preserved, allowing the data to be combined with camera images. This way, the 3D problem is simplified to 2D while keeping important distance information.

The detection process uses the DBSCAN algorithm to group points into object clusters. DBSCAN finds dense regions of points as clusters and labels sparse points as noise. In this method, the algorithm considers each point's (x, y) coordinates and depth color. This helps detect objects even in cluttered environments or when they are partially hidden. The process

Figure 6.7: Non depth-encoded 2D representation of LiDAR points.

Figure 6.8: Depth-encoded 2D representation of LiDAR points.

has three main stages. First, the *projection stage* converts the 3D points into a depth-colored 2D image. Then, the *clustering stage* applies DBSCAN to form object clusters. Lastly, the *post-processing stage* converts each cluster into a 2D bounding box by determining its size and position in the image.

LiDAR and radar have different characteristics that affect clustering. LiDAR produces dense, accurate points, while radar points are sparser and noisier. Because of this, DBSCAN needs different settings for each sensor: smaller radii and more points for LiDAR, larger radii and fewer points for radar. Proper tuning ensures both sensors' points are effectively grouped into object clusters in the depth-encoded 2D representation.

A key advantage of this method is its speed and simplicity. Unlike complex 3D learning-based networks, this approach can run in real time on any hardware. It is robust to changes in point density and sensor noise. By using depth information in a 2D framework, this approach achieves a practical balance between detection accuracy and computational efficiency.

Figures 6.9, 6.10, 6.11 shows two aspects of the 2D DBSCAN detector. The first two images illustrate the model's output on a sample traffic scene, with bounding boxes. The last image presents the execution time for each frame, showing the model's real-time performance.

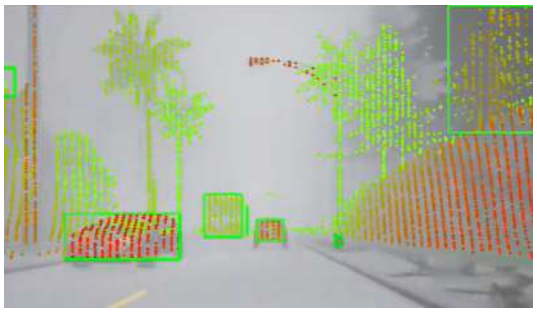


Figure 6.9: 2D Clustering of LiDAR points.

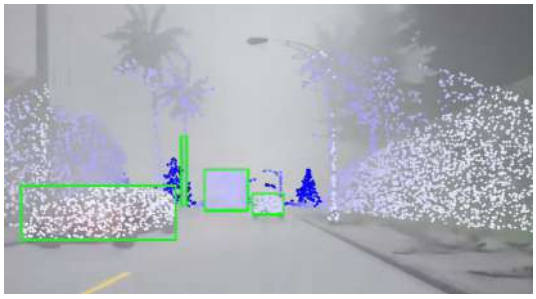


Figure 6.10: 2D Clustering of Radar points.

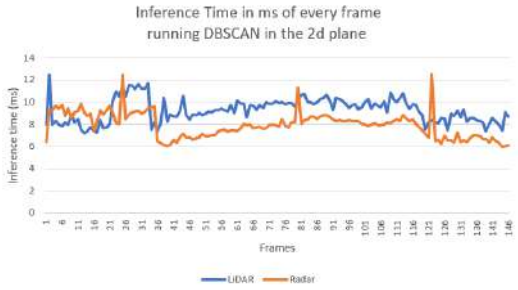


Figure 6.11: Execution time per frame for DBSCAN inference in a 2d plane.

6.2 3D Detection

Three-dimensional detection extends object detection beyond the image plane into real-world space. Instead of only finding objects in 2D, the system now estimates their true position, size, and orientation in three dimensions. This allows more accurate tracking and a better understanding of how objects move in the environment. 3D detection is especially important for applications like autonomous driving, where depth and distance are critical for safe decision making.

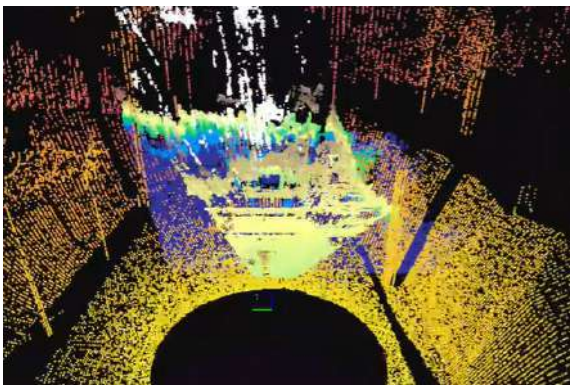
LiDAR and Radar sensors provide direct measurements of the 3D environment. Unlike in the 2D case, where the point cloud is projected onto an image plane, the detection here is performed directly on the raw 3D points. As for the stereo, its data are generally less accurate and more sensitive to lighting conditions than LiDAR and Radar. It is a low-cost alternative and can serve as a useful complement to camera-based perception when LiDAR or Radar are not available. The DBSCAN clustering algorithm is applied in three dimensions, grouping nearby points into clusters that represent objects. By using the full spatial coordinates (x, y, z), DBSCAN can separate objects more precisely and preserve their shape in space.

Once clusters are identified, each one is enclosed in a 3D bounding box. A 3D bounding box provides a compact way to describe an object in space, capturing the following:

- **Position:** the center coordinates (x, y, z) of the object in the sensor's reference frame.
- **Dimensions:** the width, length, and height of the object, defining its physical size.
- **Orientation:** the rotation angle (often yaw) that indicates how the object is aligned in the scene.

(α′) 3D Clustering of Lidar points

(β′) 3D Clustering of Radar points



(γ′) 3D Clustering of Stereo points

(δ′) View of the RGB Camera for reference.

Figure 6.12: Comparison of 3D clustering results across different sensors.

The effectiveness of this step depends on careful tuning of DBSCAN parameters, since LiDAR and radar produce point clouds with very different densities and characteristics. Proper parameter choice ensures that objects are segmented cleanly without merging or splitting errors.

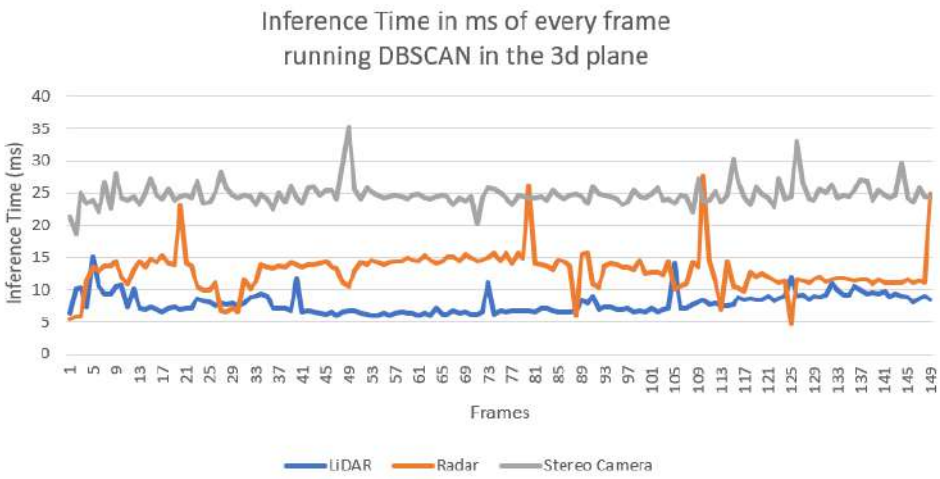


Figure 6.13: Execution time per frame for DBSCAN clustering for LiDAR and Radar and stereo points

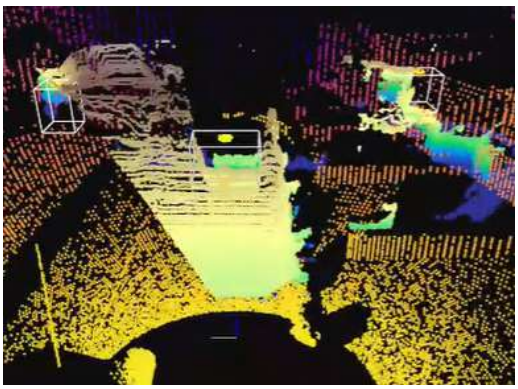


Figure 6.14: Non depth-encoded 2D representation of LiDAR points.

Figure 6.15: Depth-encoded 2D representation of LiDAR points.

The stereo camera can produce accurate 3D points in good weather, but it performs poorly in bad conditions. Rain, fog, or low light make disparity estimation unreliable, creating noisy or incorrect 3D reconstructions, as shown in Figure 6.12γ′. In contrast, in clear weather, the stereo camera provides accurate depth maps and reliable 3D detections, complementing LiDAR and radar, as seen in Figure 6.14. To ensure robust multi-sensor fusion, the stereo camera should not be used for detections in bad weather, as it only adds noise.

Chapter 7

Multi-Sensor Fusion

Sensor fusion is an essential technology for improving perception in autonomous systems. Each sensor provides partial information about the environment, as discussed in previous chapters. By combining data from multiple sensors, we can create a more accurate, robust, and reliable representation than any single sensor alone. This chapter presents the main fusion strategies, their theory, and how they were applied in this work.

Sensor fusion is essential for autonomous driving. Its goal is to combine different sensor types to get a more complete understanding of the surroundings. No sensor is perfect: cameras can classify objects but struggle with depth, LiDAR provides precise 3D geometry but limited semantic information, and radar works well in rain or fog but has low angular resolution.By combining them, we can exploit their strengths and reduce their weaknesses.

7.1 Fusion Strategies Overview

Fusion can be applied at different levels, depending on when in the processing pipeline it happens:

- **Early Fusion (Data-Level):** Raw sensor data are merged before any detection. This gives the most complete spatial information but requires careful alignment and calibration.

- **Late Fusion (Decision-Level):** Outputs from independent detectors are merged to produce final predictions. This method is robust to single-sensor failures but may lose some advantages of combining features earlier.

Each strategy has strengths and weaknesses in terms of accuracy, robustness, and computational cost.

7.2 Early Fusion (Data-Level)

Early fusion combines raw measurements from different sensors before any detection or classification. This allows geometric and semantic information from multiple sensors to be preserved. The calibration and synchronization methods required for this step were discussed in Chapter 4. Here, we focus on 2D and 3D strategies for merging raw data.

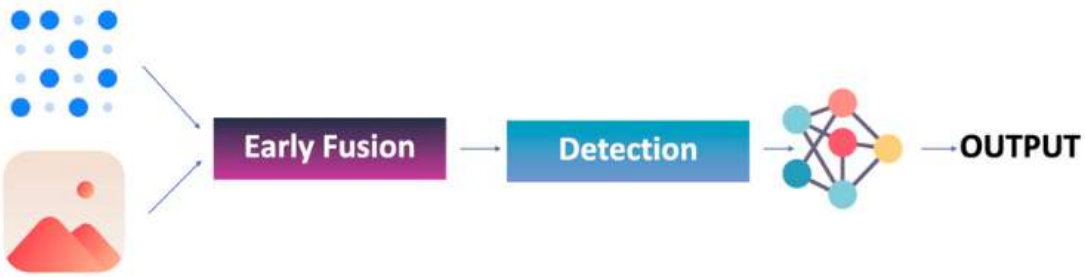


Figure 7.1: Early Fusion Pipeline

In practice, early fusion is challenging because sensors differ in dimensionality, density, and noise. Cameras produce dense 2D pixel grids, LiDAR gives sparse 3D points, and radar returns sparse points with velocity. Therefore, synchronization, calibration, and alignment are very critical to avoid inconsistencies across modalities.

2D Projection Fusion

In the 2D plane, firstly, LiDAR and radar points are projected into the image space. There, camera, LiDAR and Radar generates bounding boxes for detected objects. Each sensor then generates detections (bounding boxes). The detections are combined, producing a fused output where each sensor contributes equally: LiDAR provides geometry and depth, the radar adds distance and velocity, and the camera provides labels and outlines of objects.

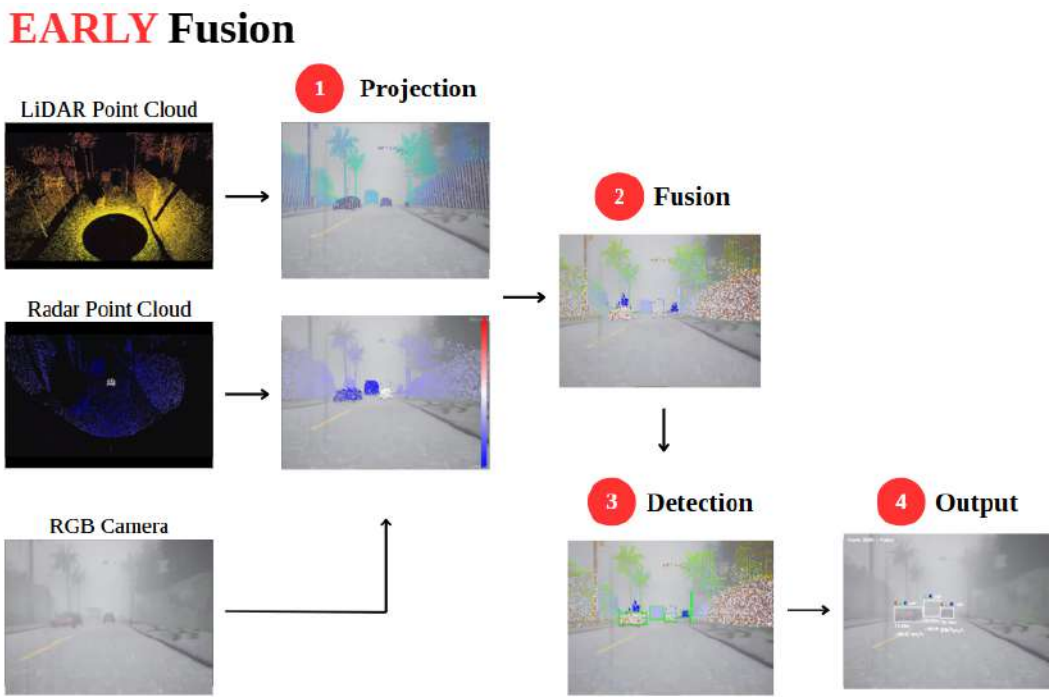


Figure 7.2: Illustration of an early fusion pipeline in 2D Projection.

A limitation of 2D projection fusion is that projected LiDAR or radar points may fall into the wrong bounding box, or even into the background, leading to incorrect associations. This ambiguity is common when objects overlap in the image or when detections are imprecise.

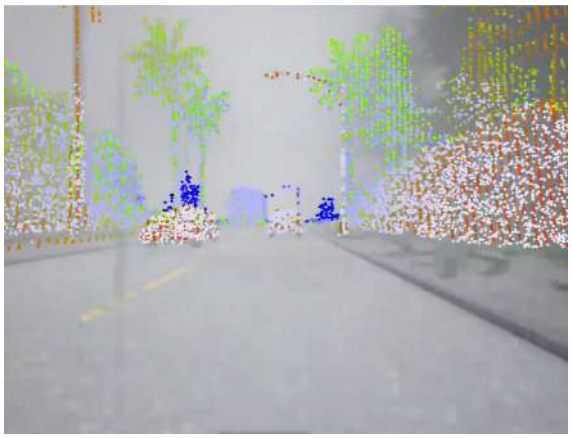


Figure 7.3: LiDAR, radar points and camera merged into the image space

Figure 7.4: Result of early fusion: fused detection combining 2D detection.

Figures 7.3 and 7.4 show the process. The first one shows the LiDAR and radar points projected into the camera image. The second one displays the final fused output, where all sensor detections are combined into a single, consolidated representation, preserving both geometric and semantic information.

The second one presents the final fused output, where detections from all sensors are combined into a unified representation. This fusion not only preserves the geometric and semantic information of the scene but also enriches it with additional information. Specifically, each detected object is displayed with its distance and speed, allowing for a better situational awareness. Furthermore, the visualization includes the sensor sources that contributed to each detection, C in red is for the Camera, L in green is for the LiDAR and R in blue is for the Radar.

3D Projection Fusion

While projection of 3D point clouds into 2D image space is the most common approach, the inverse process is also possible, lifting 2D pixels or visual features into 3D space, as described in Chapter 4. Here, image pixels are projected using depth information to match 3D geometry. This approach is widely used in visual SLAM and 3D reconstruction. Compared to projecting 3D LiDAR points into the 2D plane, lifting 2D features into 3D has several advantages. First, it avoids overlapping bounding boxes in image space, since the fusion occurs directly in 3D where objects are better separated. Second, semantic labels from the image can be tied to precise 3D positions. Finally, this method enables rich reconstructions because every pixel can contribute to the 3D representation.

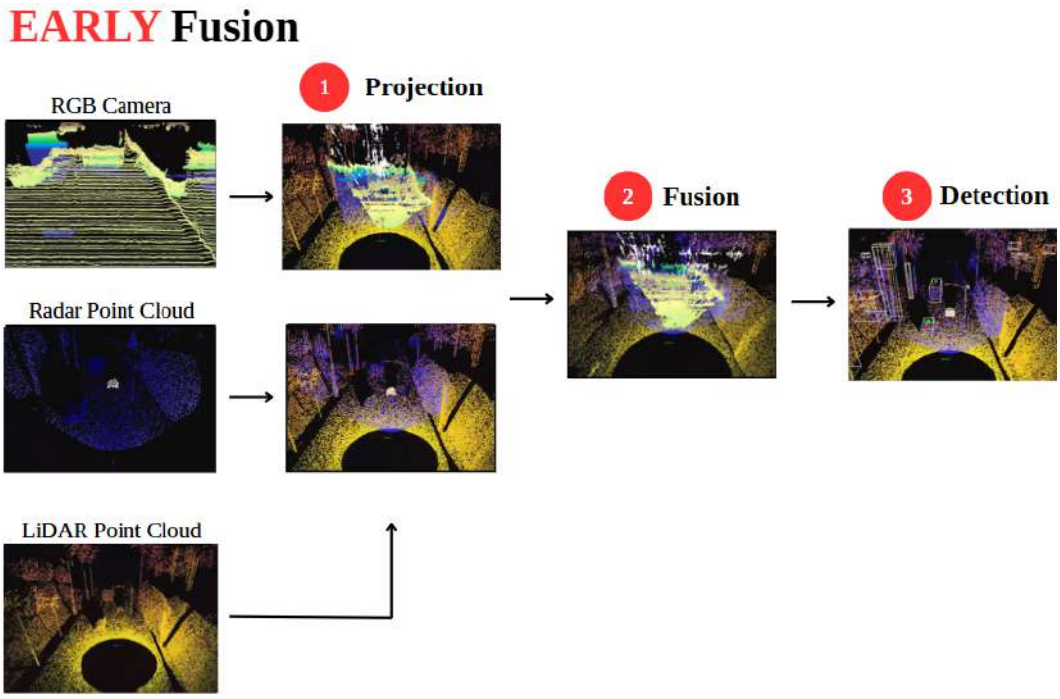


Figure 7.5: Illustration of an early fusion pipeline in 3D Projection.

However, this strategy comes with challenges. Accurate 3D lifting requires high-quality depth estimates, which may introduce errors under textureless regions, reflective surfaces, or different lighting. It is also more computationally demanding than projecting LiDAR and Radar points into 2D, since it often involves per-pixel operations. Despite these drawbacks, pixel-to-3D lifting provides a more geometrically consistent basis for sensor fusion, especially in applications that require detailed spatial reasoning such as mapping, planning, and augmented perception.

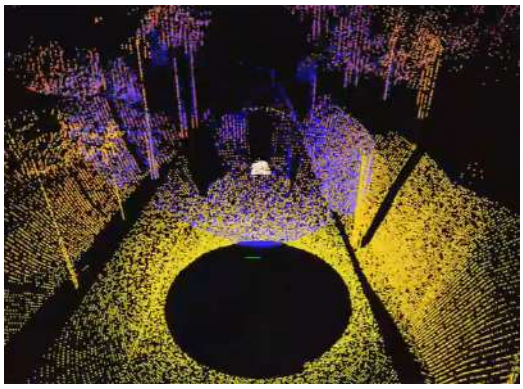


Figure 7.6: LiDAR and radar points merged into a common 3D point cloud.

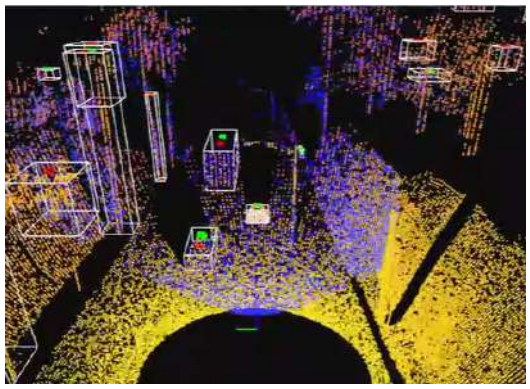


Figure 7.7: Visual result of early fusion detection.

The left image shows the raw integration of LiDAR, radar, and stereo camera measurements into a common 3D point cloud. This unified representation is the input to the early fusion pipeline. The right image shows the outcome of applying early fusion detection on this data. Objects are clustered and detected directly from the combined multi-sensor point cloud.

7.3 Late Fusion (Decision-Level)

Late fusion combines the outputs of independent detectors rather than the raw sensor data. In this work, LiDAR, radar, and camera detections were first generated separately and then merged into a unified output. This process can take place either in the 2D image space or directly in 3D space, depending on the task.

The procedure has three steps:

1. **Independent Object Detection:** Each sensor runs its own detection algorithm. For LiDAR, Radar, and Stereo camera clustering, as seen in Chapter 5, 3D bounding boxes

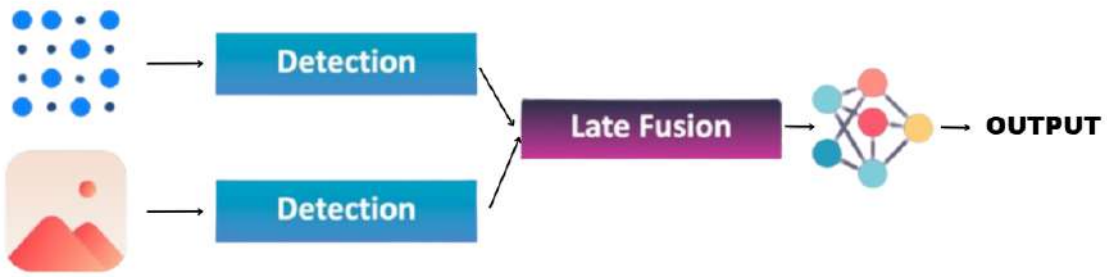


Figure 7.8: Late Fusion Pipeline

are generated. For the camera, YOLOv8 provides 2D bounding boxes.

2. **Bounding Box Association:** Detected objects are compared across sensors using an overlap metric such as Intersection-over-Union (IoU). If two bounding boxes from different sensors overlap sufficiently, they are considered to represent the same physical object.

3. **Temporal Matching:** To extend fusion over time, detections from consecutive frames can be linked using algorithms such as the Kalman Filter with Hungarian matching. This supports multi-object tracking and temporal fusion, discussed in Chapter 7.

2D Projection Fusion

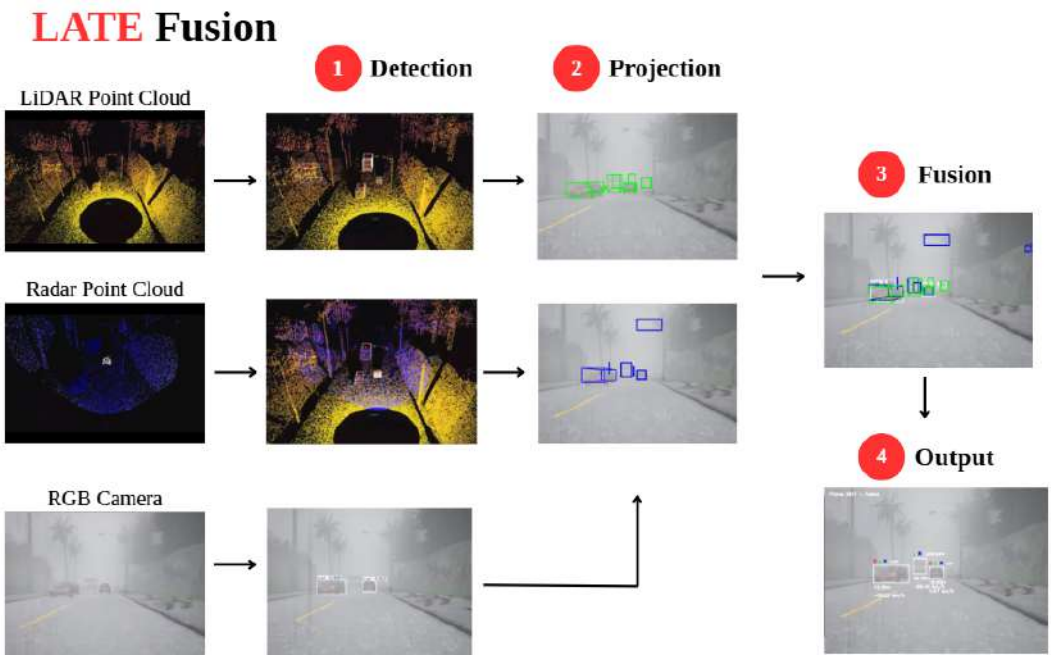


Figure 7.9: Illustration of a late fusion pipeline in 2D Projection.

In the 2D plane, all sensor detections are projected onto the camera image plane. LiDAR and radar points are clustered and the 3d bounding boxes that are extracted are projected into 2D. Camera detections provide bounding boxes directly. Overlapping 2D boxes are merged using IoU, producing a single, consolidated detection per object in the image. This representation is especially useful for visualization, 2D tracking, and tasks where 2D information suffices.

Figure 7.10: LiDAR, radar, and stereo detections into a common 2D image plane.

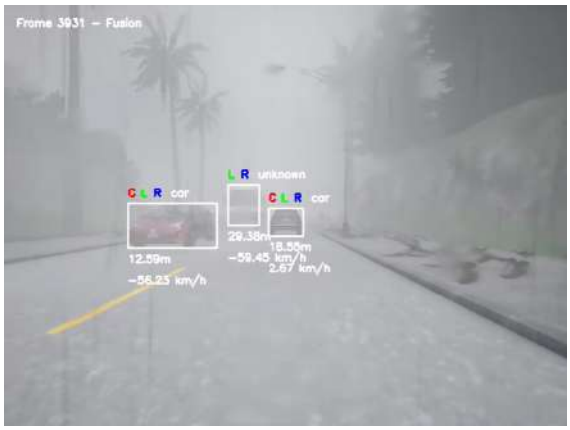


Figure 7.11: Visual result of late fusion detection in 2d plane.

Figures 7.10 and 7.11 illustrate the process. The first shows how raw detections from each sensor are projected onto the image plane, while the second shows the result after merging overlapping detections into a single bounding box per object.

3D Projection Fusion

In the 3D plane, detections stay in world coordinates. LiDAR clusters, radar points, and stereo camera 3D bounding boxes are compared in 3D space. Objects are merged based on spatial overlap, producing accurate 3D positions and sizes for each object. This is crucial for path planning, collision avoidance, and any task that needs 3D understanding.

Figure 7.12 shows how the late fusion pipeline works in the 3D projection. The fusion happens after the individual sensor processing is complete, rather than at the raw data level. The main advantage of this method is that each sensor can use its own best-suited detection algorithm, and the final fusion step combines the results into a more complete and reliable representation of the environment.

In the last fusion stage, the stereo camera was not included. This choice comes from

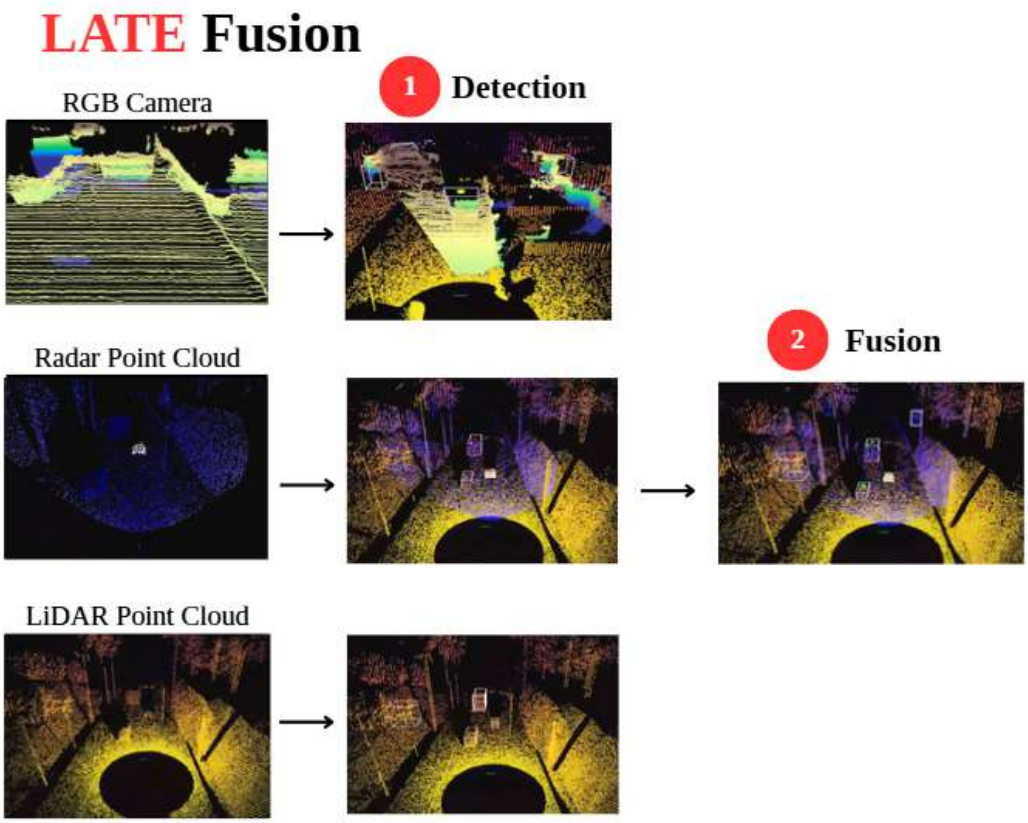


Figure 7.12: Illustration of an late fusion pipeline in 3D Projection.

the fact that the stereo system depends on matching images from the two lenses to estimate depth. When the weather is clear and bright, this process works well and the stereo camera can provide good 3D points. However, in poor conditions such as rain, fog, or low light, the image quality drops and the depth calculation becomes very unreliable. Instead of clear 3D points, the system produces a lot of random noise. If this noisy data is combined with LiDAR and radar, it can damage the quality of the final detection results and make the system less trustworthy. For this reason, the stereo camera was skipped in the final fusion. Relying only on LiDAR and radar, which are much more stable under different conditions, gives a cleaner and more consistent outcome. This shows that the stereo camera should only be used when the environment allows it, and it should not be part of the fusion in bad weather because it mainly produces noise.

Figures 7.13 and 7.14 illustrate this process. The first shows the independent 3D detections from each sensor in a common coordinate system, while the second shows the final fused 3D bounding boxes after merging overlapping detections.

Late fusion offers important advantages. Since sensors are processed independently, it is robust to individual sensor failures and it doesn't requires precise point-to-point calibration

at runtime. The fusion step is also relatively lightweight, making it computationally efficient. However, by fusing only at the decision stage, some complementary information between sensors is getting lost. This can reduce accuracy in detecting small or occluded objects.

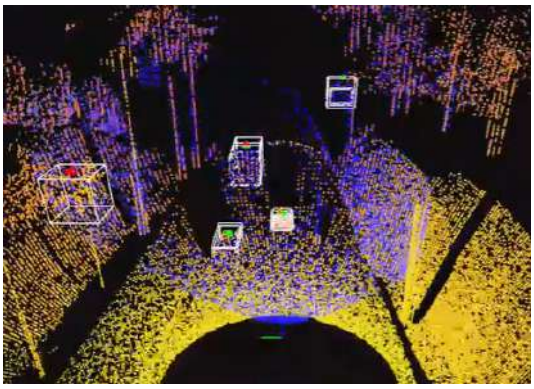


Figure 7.13: LiDAR and radar detections into a common 3D point cloud.

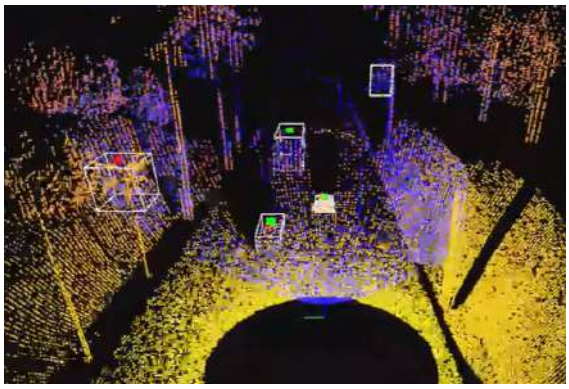


Figure 7.14: Visual result of late fusion detection in 3D plane.

Figure 7.15 compares the time each fusion method takes per frame. The results show that 2D fusion is faster than 3D fusion because it works with fewer points and simpler data, while 3D fusion needs to process dense point clouds and more complex computations.

Late fusion is also quicker than early fusion. In late fusion, each sensor is processed separately, and only the detection results are merged, so there is less work to do. Early fusion combines raw data from all sensors before processing, which increases the computational load.

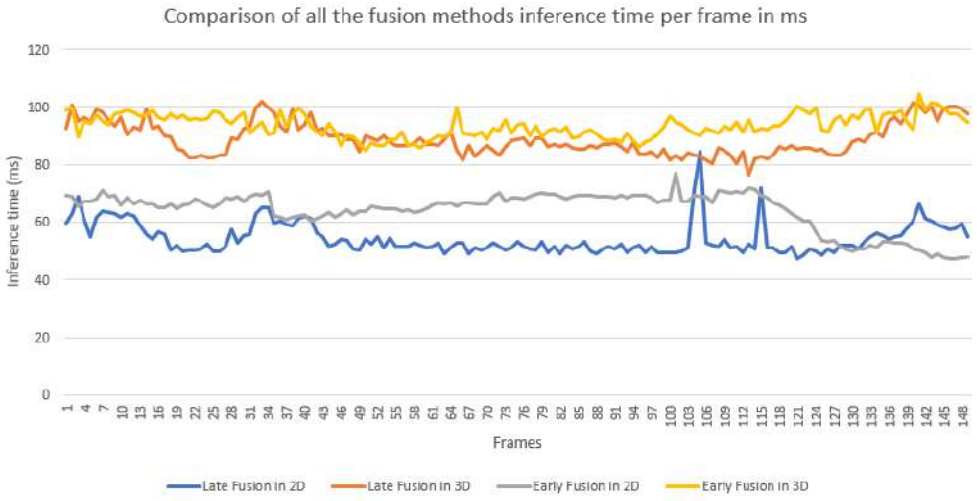


Figure 7.15: Comparison of inference time per frame (ms) for early and late fusion in both 2D and 3D space.

Chapter 8

Tracking

Tracking is an important part of perception systems in autonomous vehicles. While object detection gives the positions of nearby objects at a single moment, tracking connects these detections over time to create smooth and consistent paths. This continuity helps with predicting short-term motion, handling occlusions, and making planning more reliable. One of the most common tools for motion prediction in tracking is the Kalman filter.

Even though recent deep learning methods [5] can combine detection and tracking in one step, Kalman filters are still widely used. They are fast, reliable in real-time systems, and work well with multiple sensors. Unlike deep learning models, which often need large datasets and heavy computation, Kalman filters are lightweight, easy to understand, and suitable for embedded systems. For these reasons, they remain a preferred choice in many real-world multi-object tracking (MOT) applications, especially in autonomous driving, where speed, reliability, and low resource use are critical.

8.1 Kalman Filters

The Kalman filter is a mathematical method used to estimate the state of a system that changes over time, even when the available measurements are noisy or uncertain. It assumes that the system follows a linear model with random disturbances (Gaussian noise), and that the measurements are also affected by noise. By repeatedly combining predictions from the motion model with new sensor data, the filter produces an estimate of the system state that is optimal in the sense of minimizing the average squared error.

Each object being tracked is described by a state vector $\mathbf{x}$, which usually contains its

position, velocity, and sometimes additional properties such as size. The *covariance matrix* $\mathbf{P}$ represents the level of uncertainty in the estimate. The Kalman filter works in two alternating steps: the *prediction step* and the *update step*.

Prediction Step

In this step, the filter uses the previous state and a motion model to estimate the current state. Essentially, it asks, ”Where do I expect the object to be now, given what I knew before?”. This predicted state is given by:

$$\mathbf{x}_{k|k-1} = \mathbf{F}\mathbf{x}_{k-1|k-1},$$

where $\mathbf{F}$ is the state transition matrix describing the system dynamics.

The uncertainty associated with this prediction is also propagated forward:

$$\mathbf{P}_{k|k-1} = \mathbf{F}\mathbf{P}_{k-1|k-1}\mathbf{F}^{\top} + \mathbf{Q},$$

where $\mathbf{Q}$ represents the process noise covariance. This term accounts for model imperfections and unpredictable accelerations or disturbances. At this stage, the filter essentially produces a prior estimate, which will later be corrected using actual measurements.

Update Step

When a new measurement $\mathbf{z}_k$ arrives, the filter uses it to improve the prediction. The first step is to compare the actual measurement with what the filter expected. This difference is called the *innovation*:

$$\mathbf{y}_k = \mathbf{z}_k - \mathbf{H}\mathbf{x}_{k|k-1},$$

where $\mathbf{H}$ converts the predicted state into the same form as the measurement. Next, the filter estimates the uncertainty of this difference:

$$\mathbf{S}_k = \mathbf{H}\mathbf{P}_{k|k-1}\mathbf{H}^{\top} + \mathbf{R},$$

where $\mathbf{R}$ is the measurement noise covariance. The Kalman gain $\mathbf{K}_k$ then balances how much weight is given to the prediction versus the measurement:

$$\mathbf{K}_k = \mathbf{P}_{k|k-1}\mathbf{H}^{\top}\mathbf{S}_k^{-1}.$$

If the measurement is reliable (low noise), the filter trusts it more. If it is noisy, the filter relies more on the prediction. Finally, the prediction is corrected using the innovation and the Kalman gain:

$$\mathbf{x}_{k|k} = \mathbf{x}_{k|k-1} + \mathbf{K}_k \mathbf{y}_k, \qquad \mathbf{P}_{k|k} = (\mathbf{I} - \mathbf{K}_k \mathbf{H})\mathbf{P}_{k|k-1}.$$

This means the estimation is shifted towards the measurement, and the uncertainty is reduced. The update step adjusts the prediction with the new observation, finding the balance between trusting the model and trusting the data.

The Kalman filter is a compact and efficient method that can be applied to multi-dimensional tracking tasks, such as 2D and 3D object tracking. Because it can predict future states, the filter can smooth over missing or noisy measurements and allows reliable tracking over time, even in real-time applications.

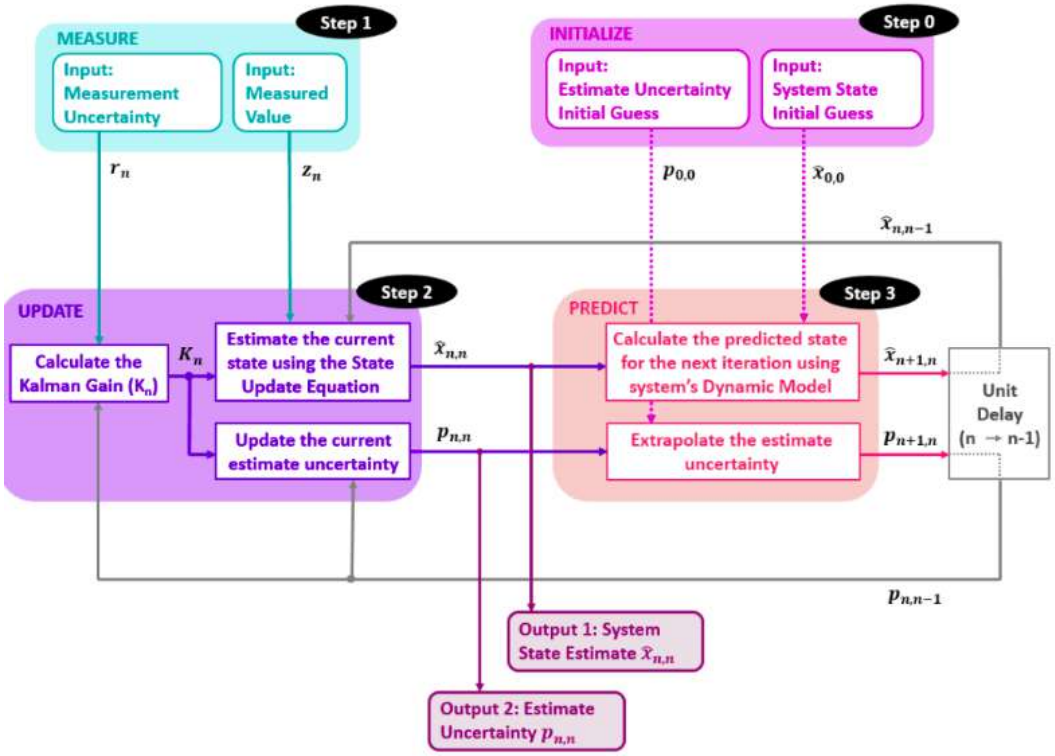


Figure 8.1: Linear Kalman Filter

8.1.1 2D Tracking

In two-dimensional tracking, the goal is to estimate how the bounding boxes of detected objects change over time in a sequence of images. The Kalman filter does this by combining predictions from a motion model with noisy measurements from the detector, producing a step-by-step estimate of the object's state. This method helps reduce flickering in detections, creates smooth object paths, and allows prediction of positions even when measurements are missing or the object is temporarily hidden.

The state of each tracked object is represented by an eight-dimensional vector :

$$\mathbf{x} = \begin{bmatrix} x & y & s & r & v_x & v_y & v_s & v_r \end{bmatrix}^\top,$$

where (x, y) are the center coordinates of the bounding box, s is its area, and r the aspect ratio (width over height). The terms (v_x, v_y, v_s, v_r) represent the corresponding velocities. Using (s, r) instead of width and height makes it easier to separate changes in size from changes in shape, avoids invalid cases such as negative dimensions, and provides a smoother description of objects moving closer or farther from the camera. The detector only provides the observable values (x, y, s, r), which form the measurement vector:

$$\mathbf{z} = \begin{bmatrix} x & y & s & r \end{bmatrix}^\top.$$

The evolution of the state is modeled under a velocity assumption,

$$\mathbf{x}_{k|k-1} = \mathbf{F}\,\mathbf{x}_{k-1|k-1},$$

with the transition matrix :

$$\mathbf{F} = \begin{bmatrix} \mathbf{I}_4 & \Delta t \cdot \mathbf{I}_4 \\ \mathbf{0}_4 & \mathbf{I}_4 \end{bmatrix} \in \mathbb{R}^{8\times 8},$$

where Δt is the time step between frames. This links positions and scales to their velocities. The measurements are related to the state by

$$\mathbf{z}_k = \mathbf{H}\,\mathbf{x}_{k|k-1} + \mathbf{v}_k,$$

where $\mathbf{v}_k \sim \mathcal{N}(0, \mathbf{R})$ represents detector noise, and the observation matrix :

$$\mathbf{H} = \begin{bmatrix} \mathbf{I}_4 & \mathbf{0}_4 \end{bmatrix} \in \mathbb{R}^{4\times 8},$$

shows that only (x, y, s, r) are directly observed, while velocities remain hidden and must be inferred over time.

As discussed in the *Prediction Step* and *Update Step*, the uncertainty of the estimates is represented by the covariance matrix $\mathbf{P} \in \mathbb{R}^{8\times8}$, which is propagated and corrected at each iteration. In this case, at initialization, large values are assigned to the velocity components of $\mathbf{P}_0$ to reflect limited knowledge of motion at first detection.

The process noise covariance $\mathbf{Q} \in \mathbb{R}^{8\times8}$ accounts for deviations from constant velocity, such as acceleration or changes in scale, while the measurement noise covariance $\mathbf{R} \in \mathbb{R}^{4\times4}$ models detector errors such as jitter or partial occlusion.

In practice, $\mathbf{P}$, $\mathbf{Q}$, and $\mathbf{R}$ are initialized as scaled identity matrices, which makes the initial uncertainties isotropic and easy to tune. Larger noise values in $\mathbf{R}$ make the filter rely more on measurements, while smaller values emphasize the predicted dynamics. By alternating prediction and update, the filter produces smooth and consistent trajectories even when detections are missing or noisy.

In addition to the state vector, the tracker can maintain an auxiliary estimate of the object's distance from the observer. If a previous distance and an estimated velocity are available, the distance is updated at each time step using a simple linear motion model: the signed speed is multiplied by the time step Δt and added to the previous distance.

Finally, for visualization, bounding boxes given by their corner coordinates $[x_1, y_1, x_2, y_2]$ are converted to the measurement format used by the filter by computing their width and height, the center coordinates, area, and aspect ratio:

$$w = x_2 - x_1, \quad h = y_2 - y_1, \quad x = x_1 + \tfrac{w}{2}, \quad y = y_1 + \tfrac{h}{2}, \quad s = w \cdot h, \quad r = \tfrac{w}{h}.$$

This conversion keeps the Kalman filter's internal state consistent with the bounding-box format needed for display and further processing.

The figure 8.2 shows the temporal evolution of 2D object tracking over 12 consecutive frames. In every third frame, sensor fusion occurs, meaning that the object's position, speed, and distance are updated using the actual sensor measurements from LiDAR, radar, and camera. In the frames in between, the tracker relies solely on its predictions, propagating the state forward. As a result, the bounding box smoothly moves along the predicted trajectory, and the displayed speed and distance values are estimated rather than directly measured.

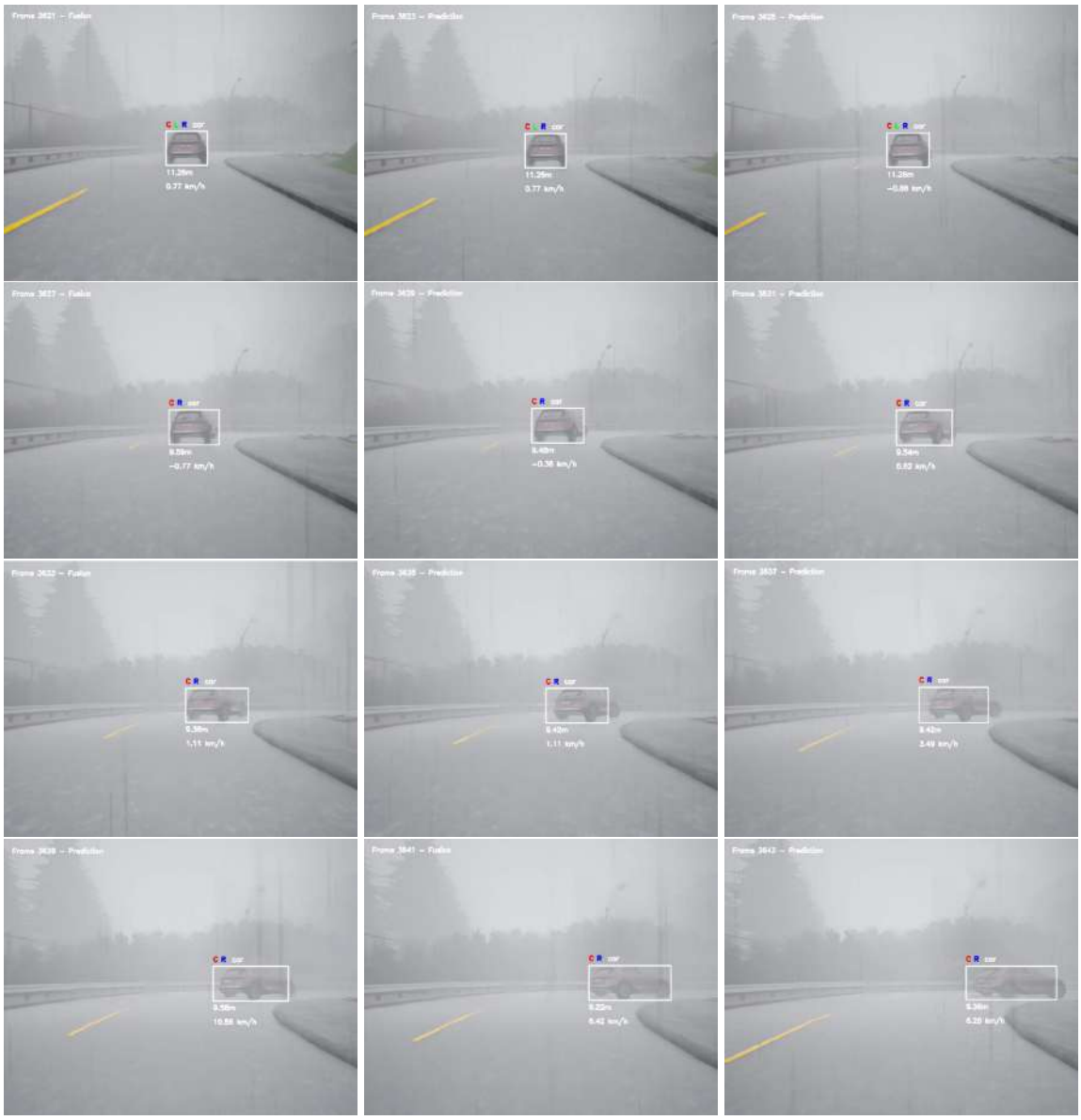


Figure 8.2: Example of 2D object tracking. The temporal evolution of object positions is shown in the 12 frames.

8.1.2 3D Tracking

Three-dimensional tracking extends the two-dimensional formulation to estimate the temporal evolution of objects in real-world 3D space. Each object is represented by a bounding box defined by its minimum and maximum coordinates along the x, y, and z axes. Unlike in 2D tracking, where the bounding-box size changes as an object moves closer or farther from the camera and must be estimated, in 3D tracking the object's physical size remains relatively constant, so only the position and velocity are modeled and tracked.

In 3D tracking, each object is represented by a six-dimensional state vector

$$\mathbf{x} = \begin{bmatrix} c_x & c_y & c_z & v_x & v_y & v_z \end{bmatrix}^\top \in \mathbb{R}^{6\times 1}.$$

The measurements contain only the positions

$$\mathbf{z} = \begin{bmatrix} c_x & c_y & c_z \end{bmatrix}^\top \in \mathbb{R}^{3\times 1}$$

The matrices $\mathbf{H} \in \mathbb{R}^{3\times 6}$ (observation matrix), $\mathbf{P} \in \mathbb{R}^{6\times 6}$ (covariance matrix), $\mathbf{Q} \in \mathbb{R}^{6\times 6}$ (process noise matrix), and $\mathbf{R} \in \mathbb{R}^{3\times 3}$ (measurement noise matrix) perform the same roles as in 2D tracking, but with dimensions adjusted for the 3D state. The rest of the Kalman filtering procedure, including prediction, update, deriving the speed and the distance, is implemented as in 2D tracking, producing smooth and consistent trajectories.

Bounding boxes, given in corner coordinates $[x_{\text{min}}, y_{\text{min}}, z_{\text{min}}, x_{\text{max}}, y_{\text{max}}, z_{\text{max}}]$, are converted to the state vector using the center coordinates

$$c_x = \frac{x_{\text{min}} + x_{\text{max}}}{2}, \quad c_y = \frac{y_{\text{min}} + y_{\text{max}}}{2}, \quad c_z = \frac{z_{\text{min}} + z_{\text{max}}}{2},$$

and conversely, predicted centers are mapped back to corner coordinates using the object dimensions along each axis.

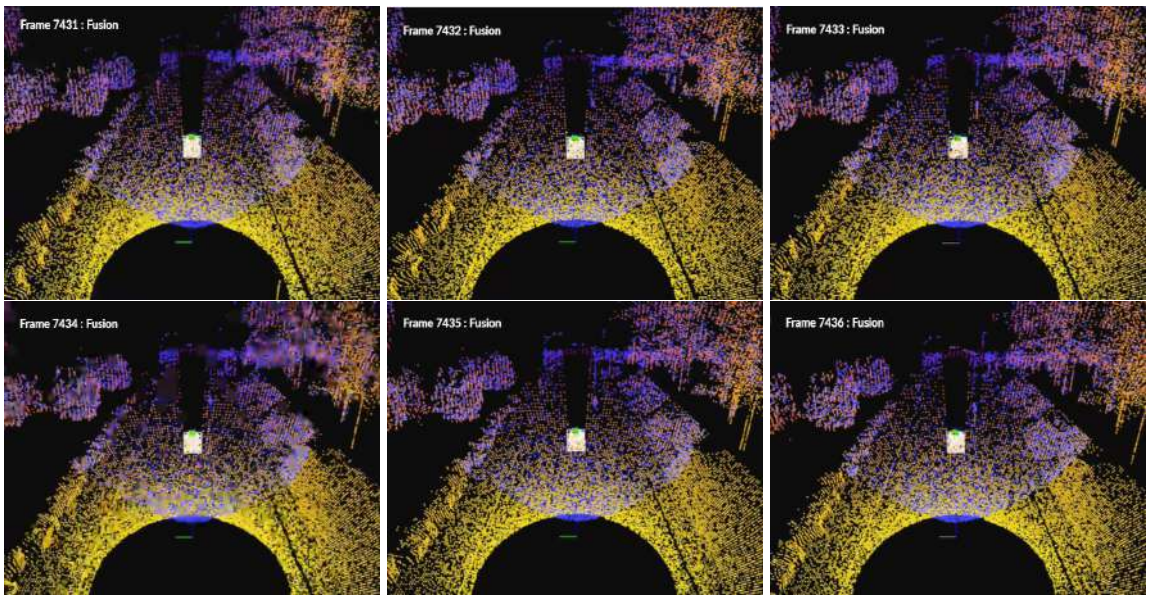


Figure 8.3: Example of 3D object tracking. The temporal evolution of object positions is illustrated in the 6 frames.

8.2 Multi-Object Tracking

Tracking multiple objects at the same time requires managing several Kalman trackers and matching new detections to the correct trackers. At each time step, all active trackers first predict the future positions of their objects using the motion model.

Next, the new detections are compared to the predicted tracker positions. The similarity between a detection and a tracker is measured using *Intersection over Union (IoU)*:

$$\mathrm{IoU}(\mathbf{b}_1, \mathbf{b}_2) = \frac{\mathrm{area}(\mathbf{b}_1 \cap \mathbf{b}_2)}{\mathrm{area}(\mathbf{b}_1 \cup \mathbf{b}_2)},$$

where $\mathbf{b}_1$ and $\mathbf{b}_2$ are the bounding boxes of the tracker and the detection. The Hungarian algorithm is used to assign detections to trackers in a way that maximizes the total IoU. Matches with low IoU are ignored, leaving some detections or trackers unmatched.

Matched detections are used to update the corresponding Kalman filters, improving the state estimates and keeping the object trajectories smooth. Unmatched detections start new trackers, while unmatched trackers that have not been updated for a few frames are removed. Each tracker can also keep additional information, like the object's speed, distance, label, or detection source, which are updated at every step.

Figure 8.4: Example of multi object tracking. The temporal evolution of object positions is shown in the 6 frames.

By combining Kalman filter predictions with assignment of detections to trackers, this method produces continuous and consistent trajectories for multiple objects, even when some measurements are missing or objects are partially hidden.

8.2.1 Role in the Pipeline

Kalman filters are computationally lightweight compared to the detection and fusion stages. While object detection and multi-sensor fusion can take tens of milliseconds per frame as seen in chapter 7, the Kalman filter's prediction and update steps typically require only a few milliseconds. This makes tracking nearly instantaneous in comparison, allowing smooth real-time performance without introducing significant latency.

In the pipeline, all detections from the fusion stage (2D, 3D, early, or late fusion) are passed to the Kalman filter. However, the filter can also operate independently for frames where no new detection is available, using only the prediction step to estimate object positions. This capability helps to handle missing or noisy data, maintaining continuous object trajectories even when sensors temporarily fail or detections are lost.

Using the Kalman filter without running fusion or detection in every frame significantly improves system performance: the per-frame processing time drops, the frame rate increases, and the buffer avoids congestion from heavy computations. This efficiency ensures that the pipeline can track objects smoothly while leaving resources available for other tasks.

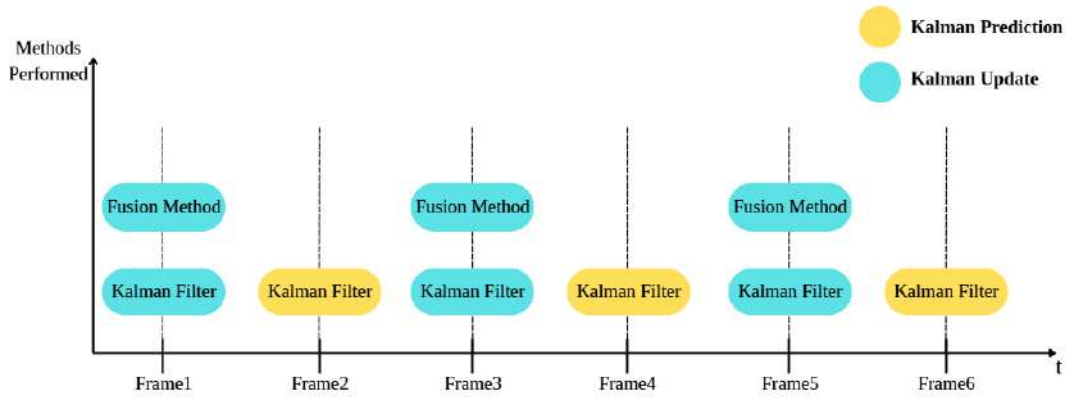


Figure 8.5: Timeline showing Kalman filter tracking with alternating detection and fusion. The filter maintains estimates for all frames, even when detection/fusion is not performed.

Figures 8.5 and 8.6 show how the Kalman filter works in the tracking pipeline and why it is efficient. Figure 8.5 demonstrates that the filter continues to estimate object positions for every frame, even when detection and fusion are skipped. This keeps the object trajectories smooth and avoids gaps caused by missing or noisy data, while also helping reduce computation. Figure 8.6 shows that the Kalman filter is much faster than the fusion and detection stages, taking only a few milliseconds per frame. Figure 8.7 makes clearer how faster is the Kalman Filter Method for both 2D and 3D tracking, since in Figure 8.6 their measurements

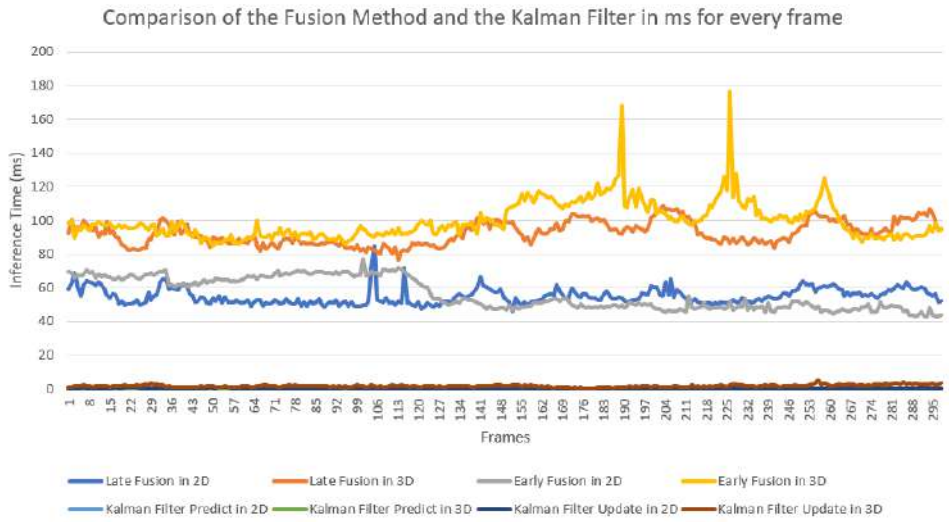


Figure 8.6: Per-frame runtime comparison of The fusion methods and kalman filter.

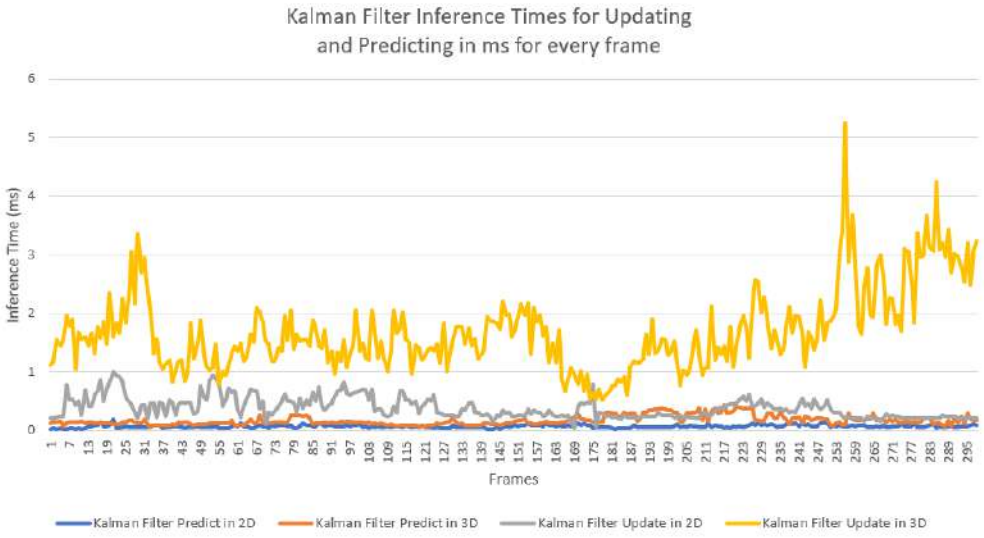


Figure 8.7: Kalman filter prediction and update are much faster than fusion methods.

seem negligible. This speed allows the system to track objects in real time while leaving resources available for other tasks.

By using the approach shown in Figure 8.5, the pipeline can effectively increase its processing frame rate, as the Kalman filter allows the system to predict object states. This means data can be fed into the tracking pipeline more frequently, improving temporal resolution and responsiveness. However, this improvement is limited by the sensors themselves: as noted in Chapter 5, high-resolution LiDARs are already operating near their maximum rate, with approximately 100 ms per frame, which constrains how much the effective frame rate can be increased.

Chapter 9

Experiment Design and Evaluation

This chapter presents the experimental evaluation of the proposed perception pipeline. The main objective is to compare the performance of individual sensors with that of multi-sensor fusion, and to examine how different fusion strategies perform under diverse driving scenarios in CARLA. In addition, the evaluation includes a detailed analysis of real-time performance and the optimizations applied to maintain efficiency. Overall, the experiments focus on four aspects: detection accuracy, tracking stability, robustness under challenging environmental conditions, and runtime efficiency to ensure real-time operation.

9.1 Simulation Scenarios

To ensure a realistic and comprehensive evaluation, the experiments were conducted under diverse driving conditions. The simulation scenarios designed in CARLA are the following:

- **Urban Traffic:** Complex inner-city environments with dense vehicle and pedestrian activity. This setup introduces frequent traffics, traffic flows, and sudden object appearances, which test the system's ability to handle crowded and dynamic scenes.
- **Highway Driving:** Long straight roads with vehicles moving at high speed. This scenario evaluates the system's capacity for long-range detection, high-velocity tracking, and handling of fast-changing relative positions.
- **Adverse Conditions:** Challenging conditions such as rain, fog, and nighttime lighting. These scenarios reduce sensor reliability and test the robustness of fusion strategies.

In practice, these conditions were not tested in isolation. Several experiments combined factors to better approximate real-world challenges, such as urban traffic under extreme fog, or highway driving during heavy rain. This mixing of scenarios ensures that the evaluation reflects the unpredictability and variability of real-world autonomous driving environments.

9.2 Performance Analysis

A critical aspect of the perception pipeline is its ability to operate in real time. In the CARLA simulator, new frames arrive at different rates depending on the sensor, as summarized in Table 9.1. Cameras typically operate at 30–60 Hz, producing frames every 16–33 ms, while LiDAR provides data more slowly at 10–20 Hz (50–100 ms). Radar is the most flexible, delivering measurements between 20 and 100 Hz, but in the worst case a single radar frame may take up to 100 ms to arrive.

Table 9.1: Typical sensor performance in CARLA simulator.

Sensor	**Frame Rate (Hz)**	**Frame Time (ms)**
RGB Camera	30–60	16–33
LiDAR	10–20	50–100
Radar	20–100	10–50

This means that the perception pipeline must be designed under the assumption that new raw data is available at most every 100 ms. In practice, this sets the real-time budget. The entire processing from raw sensor input through preprocessing, synchronization, calibration, detection, fusion, and tracking, must finish within 100 ms. If processing exceeds this limit, frames will accumulate, causing delays and reducing the system's responsiveness. Thus, the radar's maximum frame time becomes the effective constraint that defines the upper bound for real-time performance.

9.2.1 Profiling and Bottlenecks

To identify performance limitations, detailed profiling was performed across all stages of the pipeline. This analysis covered the entire flow, beginning with sensor data preprocessing. The detection stage was found to be one of the most computationally demanding,

involving YOLOv8 inference on images and DBSCAN clustering for LiDAR and radar point clouds. All the steps form the complete perception pipeline, each contributing differently to the overall runtime.

Profiling across all pipeline configurations revealed the main computational bottlenecks. The most demanding components were:

- **Radar Point Cloud Preprocessing:** Converting all the radar points from polar to Cartesian coordinates and stacking velocity information is computationally intensive, especially for large numbers of points.
- **Stereo Camera Calibration:** Rectifying left and right images and computing disparity maps requires matrix operations and optimization routines that are slow.
- **2D Object Detection (YOLOv8):** Running deep neural network inference on CPU is extremely time-consuming due to the large number of convolutional operations per frame.
- **3D Object Detection (DBSCAN):** Clustering point clouds in 3D space involves distance computations and neighborhood searches, which scale poorly with point count.
- **3D Visualization:** Rendering point clouds and bounding boxes in 3D for each frame requires significant resources and can introduce a noticeable delay in the pipeline.
- **Fusion and Tracking Process:** While individually fusion and tracking are not highly demanding, when executed sequentially they accumulate processing time, making this step a noticeable contributor to overall latency.

9.2.2 GPU Acceleration and Parallelism

Achieving real-time performance in the pipeline requires both hardware acceleration and efficient parallelization. Profiling revealed that the most computationally intensive stages and many of them are slow on CPU due to heavy matrix operations, convolutional neural network inference, point cloud clustering, and the need to execute multiple dependent tasks sequentially.

To address these bottlenecks, several optimizations were applied:

- **GPU inference:** Running YOLOv8 on CUDA/cuDNN reduced per-frame inference time from several hundred milliseconds on CPU to below 10 ms on GPU. The same applies to DBSCAN clustering for LiDAR and radar point clouds. On CPU it could take up to 200 ms, but with GPU acceleration it was reduced to under 30 ms by using CUDA kernels for fast distance calculations.

- **Parallel processing:** Camera, LiDAR, and radar callbacks were handled in separate threads, so no single sensor could block the pipeline. This allowed LiDAR and radar preprocessing to run at the same time as camera image handling, reducing waiting time and keeping the buffer filled smoothly.

- **Batch operations:** Heavy transformations like radar polar-to-Cartesian conversion and stereo rectification were implemented as vectorized GPU operations. For example, radar preprocessing, which was slow on CPU because each detection required trigonometric calculations, was sped up by processing all detections in parallel on the GPU. Stereo image rectification also ran faster on GPU, cutting the processing time by more than half.

- **Threads:** Even though fusion and tracking were not the slowest steps on their own, when run one after another on CPU they added noticeable delay. To fix this, they were placed in a separate thread from the main pipeline, while synchronization and buffering ran in another thread. This design reduced blocking between modules and allowed CPU resources to be used more efficiently.

Table 9.2 summarizes the runtime improvements achieved through these optimizations.

Table 9.2: Average Runtime comparison before and after GPU acceleration and parallelism.

Component	**Before Optimization (ms)**	**After Optimization (ms)**
YOLOv8 inference	23.4	7.32
DBSCAN inference	180-210	13-25
Radar preprocessing	203.15	20.42
Stereo rectification	129.38	16.33

9.2.3 Pipeline Runtime and Real-Time Feasibility

To evaluate real-time performance, the latency of each pipeline stage was measured across different fusion strategies. Table 9.3 summarizes the average per-frame runtimes obtained during profiling in over 300 frames. In each chapter, the runtimes were displayed analytically. They show clear differences between early and late fusion, and between 2D and 3D approaches.

Table 9.3: Average per-frame runtime of pipeline stages for different fusion strategies in milliseconds.

Stage	**Early Fusion 2D**	**Early Fusion 3D**	**Late Fusion 2D**	**Late Fusion 3D**
Preprocessing	21.37 ms			
Synchronization	21.37 (max: 30)			
Calibration	0.13 ms			
Fusion	67.11 ms	96.27 ms	51.73 ms	91.40 ms
Tracking (Updating)	0.58 ms	1.82 ms	0.58 ms	1.82 ms
Visualization	4.15 ms	32.51 ms	4.15 ms	32.51 ms
Total	93.34 ms	152.1 ms	77.96 ms	147.23 ms

In general, late fusion is faster than early fusion. For both 2D and 3D, the late fusion pipelines have lower total times. The difference is more visible in the 2D case, where late fusion reduced the runtime. This shows that using fusion later in the pipeline reduces the amount of computation needed at the fusion. The results also show that 2D pipeline are much faster than 3D pipeline. This is to be expected since 3D annotation and visualization is much more costly. This show the trade-off between 3D fusion that provides richer spatial information, but at the cost of higher latency and 2D fusion which achieves the best efficiency. Comparing all four strategies, late fusion in the 2D plane is the fastest option, while early fusion in the 3D plane is the slowest.

9.3 Evaluation Metrics

To evaluate the perception pipeline, several metrics are used. These metrics analyze detection quality, tracking consistency, and robustness under challenging conditions.

9.3.1 Detection Accuracy

Object detection quality is assessed using the **F1 score**, which uses precision and recall:

$$F1 = \frac{2 \cdot \text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$$

Precision and recall are defined as:

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

where:

- TP (True Positives): correctly detected objects.
- FP (False Positives): detections where no object exists.
- FN (False Negatives): missed objects that were present in the scene.

Thus, the high precision shows fewer false alarms, while the high recall indicates fewer missed objects. The F1 score captures the trade-off between the two metrics.

9.3.2 Tracking Accuracy

For tracking evaluation, the **Multi-Object Tracking Accuracy (MOTA)** metric is used:

$$MOTA = 1 - \frac{\sum_t (FN_t + IDSW_t)}{\sum_t GT_t}$$

where:

- FN_t: false negatives at time t.
- $IDSW_t$: identity (ID) switches at time t (when an object's ID changes during tracking).
- GT_t: number of ground truth objects at time t.

MOTA considers not only detection errors but also consistency. The pipeline needs a high MOTA value to have stability and accuracy of multi-object tracking.

9.3.3 Robustness

Robustness is evaluated under conditions that challenge the perception systems:

- **traffic:** when objects are partially or fully hidden by others.
- **Adverse Weather:** heavy rain or fog, which reduce visibility.
- **Lighting Variations:** low light, glare or night driving conditions.
- **Sensor Dropout:** temporary failure or delay in one modality.

The change in F1 metric between normal and under conditions that challenge perception system shows weather it is robust or not. These metrics were made possible because CARLA provides ground-truth bounding boxes for all vehicles, giving their exact positions relative to the ego vehicle at every frame. Using a specific region, the road and the sidewalk, the bounding boxes of the detections are matched to the bounding boxes of the CARLA. A specific region is used because LiDAR and Radar detect, numerous other objects in the environment, like building, trees and lights.

The evaluation results from different sensor configurations and fusion strategies are shown in the table below. The detection accuracy (F1) and tracking accuracy (MOTA) are reported.

Table 9.4: Detection and tracking performance across configurations under dynamic conditions.

Configuration	F1 Score (%)	MOTA (%)
Camera Only	68.24	61.43
LiDAR Only	71.51	67.86
Radar Only	67.66	49.14
Early Fusion 2D	82.32	74.65
Early Fusion 3D	80.71	78.27
Late Fusion 2D	85.16	79.42
Late Fusion 3D	83.58	82.66

Table 9.4 provides a comparison of single-sensor and fusion configurations.

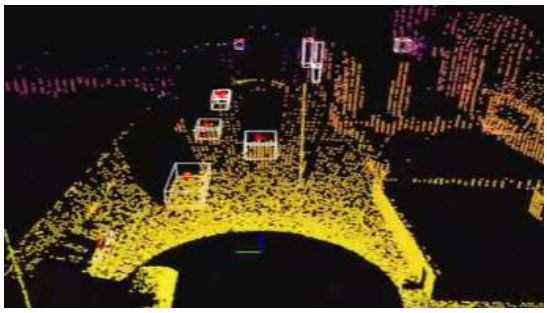
Table 9.5: Robustness under challenging conditions, calculated as relative F1 drop.

Configuration	Traffic	Rain	Fog	Night
Camera Only	-12.37%	-24.82%	-37.64%	-41.29%
LiDAR Only	-09.41%	-18.56%	-22.34%	-27.18%
Radar Only	-06.27%	-08.43%	-10.18%	-14.92%
Early Fusion 2D	-07.52%	-13.78%	-17.46%	-20.31%
Early Fusion 3D	-06.48%	-12.29%	-15.63%	-18.44%
Late Fusion 2D	-05.36%	-10.72%	-12.58%	-15.27%
Late Fusion 3D	-04.29%	-08.65%	-10.41%	-12.16%

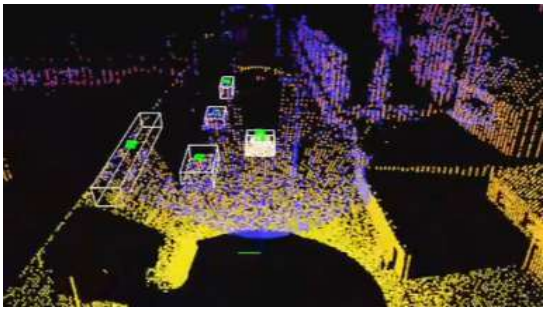
9.4 Visual Results

The visual results are presented after the metrics, to provide a better understanding and a visual of system behavior. While the tables represent detection accuracy, tracking stability, and robustness in quantity, the figures show to the human eye how each configuration actually perceives the scene under different conditions. By examining these visualizations, it becomes clear where certain methods succeed or fail, for instance, how the camera fails under extreme fog, or how LiDAR performs better in fog. The comparisons of early and late fusion show the practical differences between combining sensor data before or after detection.

(α′) Camera - Traffic

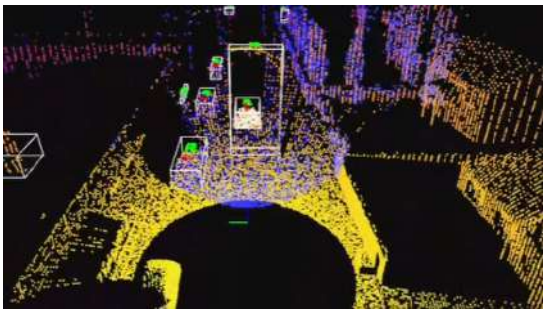


(β′) LiDAR - Traffic

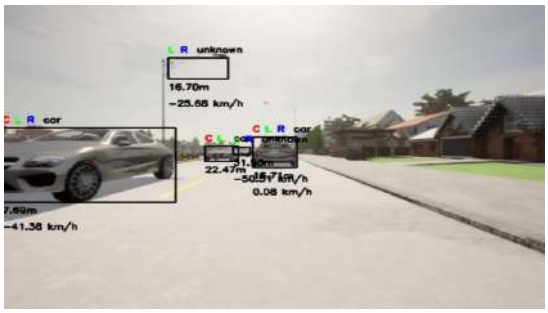


(γ′) Radar - Traffic

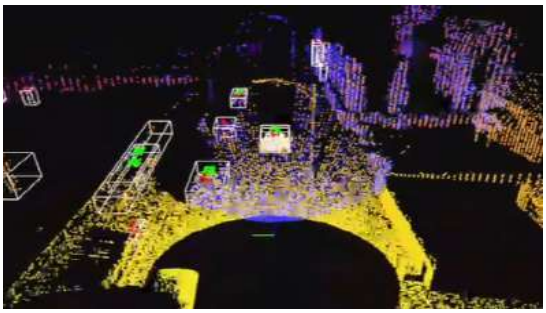
(δ′) Early Fusion 2D Projection - Traffic



(ε′) Early Fusion 3D Projection - Traffic



(στ′) Late Fusion 2D Projection - Traffic

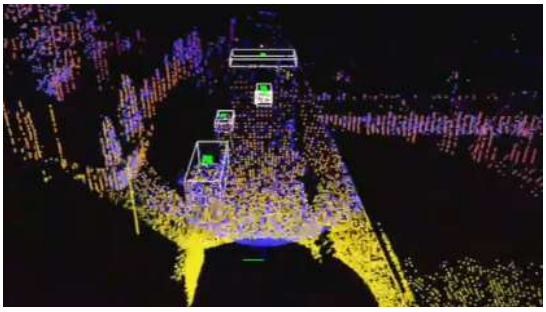


(ζ′) Late Fusion 3D Projection - Traffic

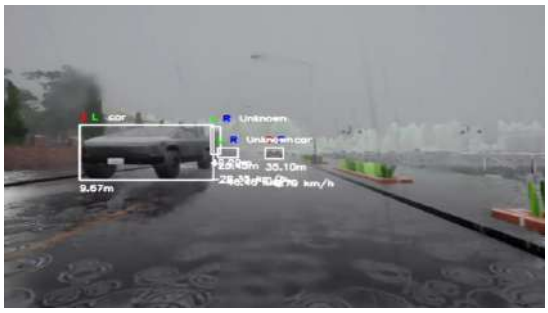
Figure 9.1: Comparison of all sensor configurations and fusion strategies under traffic. Top row: individual sensors (Camera, LiDAR, Radar). Middle row: Early Fusion (2D and 3D projections). Bottom row: Late Fusion (2D and 3D projections).

(α′) Camera - Rain

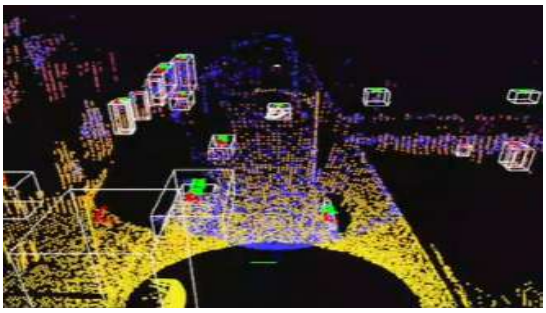
(β′) LiDAR - Rain



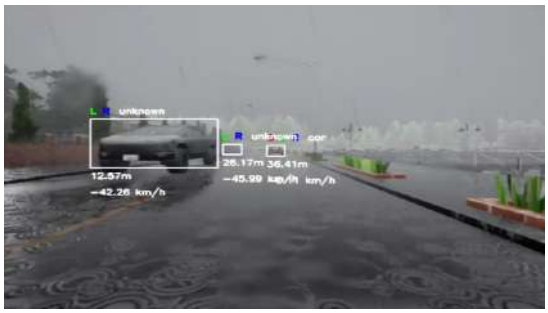
(γ′) Radar - Rain



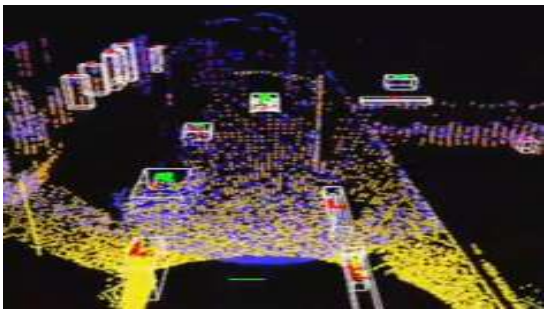
(δ′) Early Fusion 2D Projection - Rain



(ε′) Early Fusion 3D Projection - Rain



(στ′) Late Fusion 2D Projection - Rain

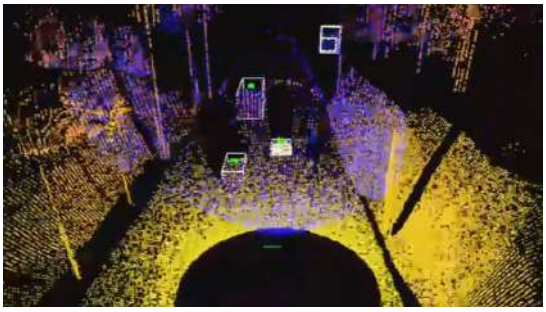


(ζ′) Late Fusion 3D Projection - Rain

Figure 9.2: Comparison of all sensor configurations and fusion strategies under rain. Top row: individual sensors (Camera, LiDAR, Radar). Middle row: Early Fusion (2D and 3D projections). Bottom row: Late Fusion (2D and 3D projections).

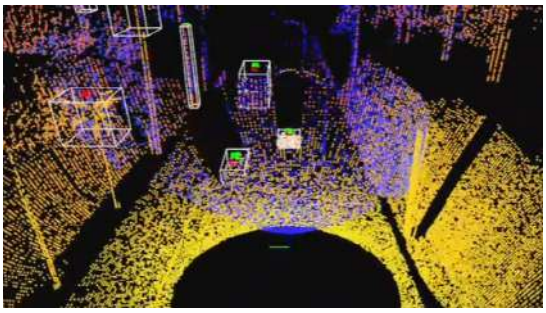
(α′) Camera - Fog

(β′) LiDAR - Fog

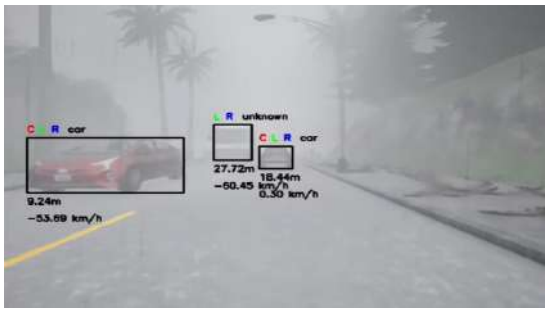


(γ′) Radar - Fog

(δ′) Early Fusion 2D Projection - Fog



(ε′) Early Fusion 3D Projection - Fog

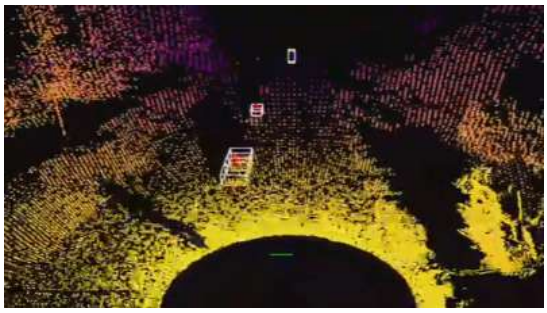


(στ′) Late Fusion 2D Projection - Fog

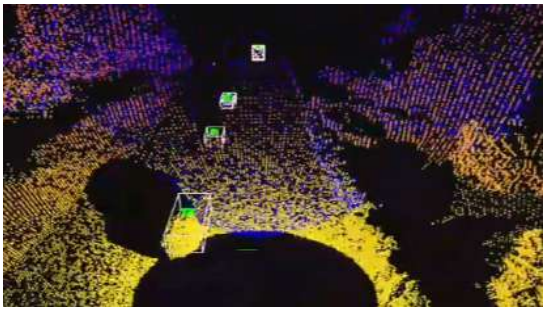
(ζ′) Late Fusion 3D Projection - Fog

Figure 9.3: Comparison of all sensor configurations and fusion strategies under fog. Top row: individual sensors (Camera, LiDAR, Radar). Middle row: Early Fusion (2D and 3D projections). Bottom row: Late Fusion (2D and 3D projections).

(α′) Camera - Night

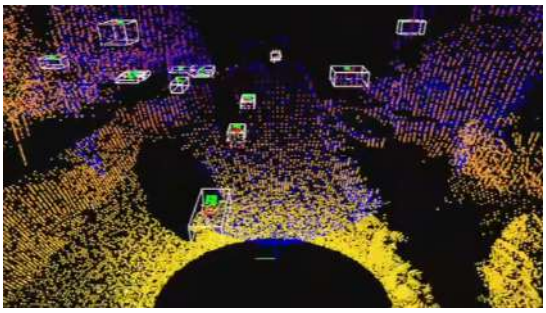


(β′) LiDAR - Night

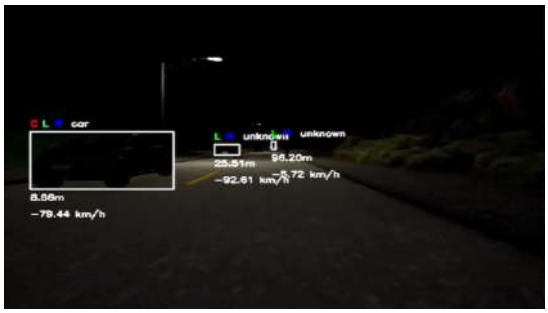


(γ′) Radar - Night

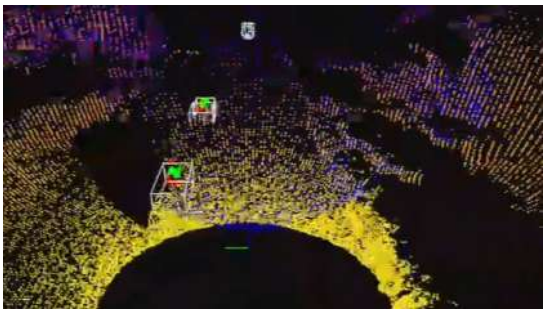
(δ′) Early Fusion 2D - Night



(ε′) Early Fusion 3D - Night



(στ′) Late Fusion 2D - Night



(ζ′) Late Fusion 3D - Night

Figure 9.4: Comparison of all sensor configurations and fusion strategies under night conditions. Top row: individual sensors (Camera, LiDAR, Radar). Middle row: Early Fusion (2D and 3D projections). Bottom row: Late Fusion (2D and 3D projections).

9.5 Observations

The final results showed big differences between the sensors only and fusion methods. The camera works well in normal conditions, but struggles in rain, fog, and at night because it needs good light. LiDAR performs good in bad weather better, especially fog, but can miss small or shiny objects. Radar is reliable in all conditions, but it is less accurate because it provides lower detail. Early fusion would be better than late fusion because combining raw data firstly should give more information about the environment. However, the results show late fusion is performing slightly better.

Furthermore, 3D fusion is slightly more accurate than 2D, for tracking objects in traffic or when they are partly hidden. 2D fusion is faster and more fisible and it can be useful when speed is more important than a bit more accuracy.

Finally, late fusion seems like the best option for fusion. It gives accurate detections, stable tracking, works well in bad conditions, and is fast enough for real-time scenarios.

Chapter 10

Conclusions

10.1 Discussion

This thesis explored multi-sensor fusion for real-time object detection and tracking in simulated autonomous driving environments using the CARLA simulator. The goal was to combine cameras, LiDAR, and radar to improve perception accuracy, tracking stability, and robustness under different conditions.

The results showed that single sensors have limitations. Cameras fail in low light or bad weather, LiDAR gives accurate 3D positions but limited semantic information, and radar is reliable in adverse conditions but has low resolution. Combining sensors improved performance overall.

Early fusion merged raw data before detection and worked well but did not always outperform late fusion due to calibration and timing problems. Late fusion, which combines independent detections, gave the best result of accuracy and real-time performance. 3D fusion provided better understanding of the environment and helped track objects, while 2D fusion was simpler but less precise. The CARLA simulator was very useful, as it gave accurate bounding boxes for all vehicles, allowing the evaluation of detection and tracking metrics.

In conclusion, multi-sensor fusion is nesessary for reliable autonomous driving perception.

10.2 Future Work

Although the system already shows real-time multi-sensor fusion with good accuracy and robustness, there are several ways it can be improved. The 3D visualization could be made clearer and more interactive, so that complex traffic situations are easier to understand. Also, information like the speed and the distance although they are registered, they are not being displayed. Future work would also look at machine learning methods for 3D object detection and fusion, which would allow the system to use more detailed features and adjust better when sensor conditions change. In addition, the multi-sensor pipeline created in this work could be used to train autonomous driving agents, where fused perception would provide reliable input for decision-making and control. These directions would help move the pipeline closer to a perception system for autonomous vehicles, combining accuracy, robustness, and flexibility in real-world environments.

Bibliography

[1] H. Liu, Y. Zhang, and X. Zhao. Real-time object detection using lidar and camera fusion. *Scientific Reports*, 2023.

[2] X. Zhao, H. Liu, and Y. Zhang. Dual transformer enhancement for 3d object detection. *ScienceDirect*, 2024.

[3] Y. Gong J. Wang, Y. Zeng. Collaborative 3d object detection for autonomous vehicles via learnable communications. IEEE Xplore, 2024.

[4] Yi Jiang Dongdong Yu Fucheng Weng Zehuan Yuan Ping Luo Wenyu Liu Xinggang Wang Yifu Zhang, Peize Sun. Bytetrack: Multi-object tracking by associating every detection box. `https://arxiv.org/abs/2110.06864`, 2021.

[5] L. Zhang, Z. Wang, and L. Chen. Acntrack: Agent cross-attention guided multimodal multi-object tracker. *ScienceDirect*, 2025.

[6] H. Boukamchahamdi. How to create a kalman filter for 3d object tracking. Medium, 2024.

[7] Satya. Sensor fusion with kalman filter, 2024.

[8] A. Tayyebi. Early vs. late camera-lidar fusion in 3d object detection: A performance study. Medium, 2025.

[9] Y. Yang, X. Gao, T. Wang, X. Hao, Y. Shi, X. Tan, X. Ye, and J. Wang. Explore the lidar-camera dynamic adjustment fusion for 3d object detection. *arXiv*, 2024.

[10] Wael Farag. Lidar and radar fusion for real-time road-objects detection and tracking. *Intelligent Decision Technologies*, 2021.

[11] Shijie Liu, Jianqing Wu, Bin Lv, Xinhao Pan, and Xiaorun Wang. Data fusion of roadside camera, lidar, and millimeter-wave radar. *IEEE Sensors Journal*, 2024.

[12] Mohsin M. Jamali Ratheesh Ravindran, Michael J. Santora. Camera, lidar, and radar sensor fusion based on bayesian neural network (clr-bnn). 2022.

[13] Aditya Prakash, Kashyap Chitta, and Andreas Geiger. Multi-modal fusion transformer for end-to-end autonomous driving. *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2021.

[14] C. A. Chaklader. Sensor fusion techniques in autonomous vehicle navigation: Delving into various methodologies and their effectiveness. Medium, 2024.

[15] C. A. Chaklader. Sensor fusion techniques in autonomous vehicle navigation: Delving into various methodologies and their effectiveness. Medium, 2024.

[16] Think Autonomous. Lidar and camera sensor fusion in self-driving cars. Think Autonomous, 2024.

[17] Technexion. RGB cameras: Definition, components, and integration. `https://www.technexion.com/resources/rgb-cameras/`, 2025.

[18] Synopsys. What is lidar and how does it work? `https://www.synopsys.com/glossary/what-is-lidar.html`, 2025.

[19] Radar sensor - an overview. `https://www.sciencedirect.com/topics/engineering/radar-sensor`, 2025.

[20] Zilliz. What is Object Detection? - Zilliz Learn. `https://zilliz.com/learn/what-is-object-detection`, 2025.

[21] Keylabs. YOLOv8: Object Detection Explained. `https://keylabs.ai/blog/under-the-hood-yolov8-architecture-explained/`, 2025.

[22] N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. In *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, 2005.

[23] Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton. Imagenet classification with deep convolutional neural networks. *Neural Information Processing Systems*, 2012.

[24] SharkYun. Computer Vision—Object Detection, One-Stage vs Two-Stage. https://sharkyun.medium.com/computer-vision-object-detection-one-stage-vs-two-stage-b05dbff88195, 2024.

[25] Shairoz Sohail. A Comprehensive Introduction to Clustering Methods. https://shairozsohail.medium.com/a-comprehensive-introduction-to-clustering-methods-1e1e4f95b501, 2018.

[26] Alex H. Lang Sourabh Vora Venice Erin Liong Qiang Xu Anush Krishnan Yu Pan Giancarlo Baldan Oscar Beijbom Holger Caesar, Varun Bankiti. nuScenes: A multimodal dataset for autonomous driving. https://arxiv.org/abs/1903.11027, 2020.

[27] Mrinal W. What is object tracking in computer vision? https://blog.roboflow.com/what-is-object-tracking-computer-vision/, 2022.

[28] ThinkAutonomous. A complete overview of object tracking algorithms in computer vision self-driving cars. https://www.thinkautonomous.ai/blog/object-tracking/, 2025.

[29] Gunay Ashouri Newsham Hobson, Lowcay. Sensor Fusion - an overview | ScienceDirect Topics. https://www.sciencedirect.com/topics/engineering/sensor-fusion, 2019.

[30] G. Velasco-Hernandez, L. Zhang, and L. Chen. Sensor and sensor fusion technology in autonomous vehicles: A review. 2021.

[31] ThinkAutonomous. 2 Ways to do Early Fusion in Self-Driving Cars (and when to use Mid Or Late Fusion). https://www.thinkautonomous.ai/blog/early-fusion/, 2024.

[32] Apxml. Late Fusion: Combining Model Decisions — apxml.com. https://apxml.com/courses/intro-to-multimodal-ai/chapter-3-techniques-integrating-modalities/late-fusion, 2025.

[33] ThinkAutonomous. 9 Types of Sensor Fusion Algorithms. https://www.thinkautonomous.ai/blog/9-types-of-sensor-fusion-algorithms/, 2021.

[34] CARLA Team. Carla simulator. https://carla.org/, 2025.

[35] Lefteris Kotsonis. What Is Sensor Calibration And Why It Matters — botasys.com. https://www.botasys.com/post/sensor-calibration, 2024.