

UNIVERSITY OF OKLAHOMA
GRADUATE COLLEGE

CAMERA-RADAR 3D MULTI OBJECT TRACKING ROBUST TO
SENSOR-FAILURES

A THESIS
SUBMITTED TO THE GRADUATE FACULTY
in partial fulfillment of the requirements for the
Degree of
MASTER OF SCIENCE

By
MATHIS MORALES
Norman, Oklahoma
2025

CAMERA-RADAR 3D MULTI OBJECT TRACKING ROBUST TO
SENSOR-FAILURES

A THESIS APPROVED FOR THE
SCHOOL OF ELECTRICAL AND COMPUTER ENGINEERING

BY THE COMMITTEE CONSISTING OF

Dr. Golnaz Habibi(Chair)

Dr. Samuel Cheng

Dr. Andrew H. Fagg

Acknowledgments

This thesis represents the accomplishment of a year and half of hard work, which simply could not have been possible without the unwavering support of my family and friends.

I would first like to thank my advisor, Dr. Golnaz Habibi, for her guidance throughout this project, along with the Gallogly College of Engineering of the University of Oklahoma and the school of Computer Science that funded this project.

I would also like to thank my committee members Dr. Andrew H. Fagg and Dr. Samuel Cheng, I started my coursework with both of your classes, and learned so much alongside you, opening my eye on the depth of AI fields and all their possible applications.

Furthermore, I would like to thank Dr. Nathan Goodman, ECE professor at the Advanced Radar Research Center and the Radar Innovations Lab, for his kind help on the radar-related sections of this project.

I also extend my gratitude to my lab mates. Daniel Vargas and Daniel Perez, I hope you will find success as you pursue your respective PhDs, thank you for those long discussions we had (and for letting me vent when I needed it). To all the other members I had the pleasure to work with, Matthew, Tyler, Phong, Ethan, Aksel, Maisha, Alexander, Keon, Anany, Vineel, Job, Evan, and Aaron: I was happy to work with all of you, share my (albeit still limited) knowledge of my field, and learn more about yours.

My gratitude also goes to the staff of the Computer Science School and the Electrical and Computer Engineering School for being so supportive and welcoming, and for providing me with work fuel (coffee).

Finally, to my close friends and to my family, thank you all for being there for me, both in times of joy and during hardships. Words do not describe how grateful I am.

Table of Contents

Acknowledgments	iv
List Of Tables	vii
List Of Figures	viii
Abstract	x
1 Introduction	1
2 Related Works	5
2.1 Data Degradation Synthesis and Recognition	5
2.2 3D Detection and Tracking	6
2.3 Sensor Failure Robustness	8
2.4 Dataset	9
3 SynthCRAV: Camera-Radar Noise Synthesizer for Autonomous Vehicles	12
3.1 Realistic Data Degradation Synthesizer	12
3.1.1 Camera Synthetic Degradation	12
3.1.2 Radar Synthetic Degradation	18
3.2 Noise Recognition	32
3.2.1 Dataset Generation	32
3.2.2 Noise Recognition Models	33
3.2.3 Noise Recognition Results	36
4 VelACaR-MOT: Velocity-Aware Camera-Radar Multi-Objects Tracking Pipeline	39
4.1 Core Concept	39
4.2 Detection Backbones	41
4.2.1 Radiant	41
4.2.2 CRN	41
4.3 Velocity Aware Tracking Pipeline	42
4.3.1 Prediction	43
4.3.2 Data Matching	45
4.3.3 Update	46

4.4	Initial Results	48
5	Extended VelACaR	53
5.1	Singer Model	55
5.2	Position and Velocity Matching	56
5.3	Double Round Matching	57
5.4	Additional Modules	58
5.5	Hyperparameters Tuning	59
5.6	Tracking Performance and Ablation Studies	62
5.6.1	Using a Constant Velocity Model	63
5.6.2	Using the Singer Model	68
6	SwitchTrack: Switching Sensor-Failure Robust Tracking Pipeline	72
6.1	Methodology	72
6.1.1	Approach	73
6.1.2	Camera-only Backbone	74
6.1.3	Radar-only Backbone	74
6.2	Improved Hyper-Parameter Tuning	75
6.3	Dataset Generation	78
6.4	Switching Thresholds	79
6.4.1	Using Detection Performance Across Noise Levels	79
6.4.2	Using Tracking Performance Across Noise Levels	84
7	Experimental Results	87
7.1	Static Pipelines	88
7.2	Adaptive Pipeline	93
8	Summary and Conclusions	103
	Reference List	108
	Appendix	113
1	Appendix A	113

List Of Tables

3.1	Overview of the radar output data structure	19
3.2	Noise recognition models results on <i>training</i> and <i>test</i> data splits	38
4.1	Tracking results of VelACaR on the nuScenes <i>test</i> dataset, using CRN’s detections, compared to other tracking methods	50
4.2	Ablation study on the effects of velocity awareness and tuned Noise Covariance Matrix (R) on the nuScenes <i>validation</i> dataset	51
5.1	Post-optimization baseline tracking results using constant velocity model and no additional matching modules, on the validation dataset	64
5.2	Post-optimization velocity matching tracking results using a constant velocity model, on the validation dataset	67
5.3	Post-optimization double-rounds matching tracking results using a constant velocity model, on the validation dataset	68
5.4	Post-optimization tracking results using the Singer model with no additional matching modules, on the validation dataset	69
5.5	Post-optimization tracking results using the Singer model and double-rounds matching , on the validation dataset	70
5.6	Post-optimization tracking results using the Singer model , double-rounds matching , and velocity matching , on the validation dataset	70
6.1	Preliminary tracking performance comparison between SwitchTrack and a static CRN-VelACaR tracker, on synthetically degraded data	83
7.1	Tracking performance comparison between static trackers and SwitchTrack with improved noise thresholds and ideal noise recognition , on the synthetic dataset	94
7.2	Tracking performance between static CRN-VelACaR and SwitchTrack using the noise recognition models , on the synthetic dataset	97
7.3	AMOTA performance on the camera -degraded scenes, with SwitchTrack using a real noise recognizer	99
7.4	AMOTA performance on the radar -degraded scenes, with SwitchTrack using a real noise recognizer	99
7.5	Distribution of data samples across backbones	102

List Of Figures

2.1	Sensor configuration of the nuScenes car from Caesar et al. (2020) . . .	10
3.1	Blur effect at 0% (original), 30%, 60%, and 100% (max) distortion levels	14
3.2	Overexposure (top) and underexposure (bottom) degradations at 0% (original), 30%, 60%, and 100% (max) distortion levels	16
3.3	Normally distributed additive noise degradation at 0% (original), 30%, 60%, and 100% (max) distortion levels	18
3.4	Effects of synthesized noise on a radar point cloud in a 3D scan at -0dB (original), -6dB, -12dB, and -20dB (max) noise levels	30
3.5	Effects of synthesized noise on the multi-radar point cloud scene reconstitution at -0dB (original), -6dB, -12dB, and -20dB (max) noise levels	31
3.6	Camera noise recognition model architecture	34
3.7	Radar noise recognition model architecture	36
3.8	Camera noise recognition model confusion matrix on the synthetic data <i>test</i> split	37
3.9	Radar noise recognition model confusion matrix on the synthetic data <i>test</i> split	38
4.1	Overview of the VelACaR multi-objects tracking pipeline	48
4.2	Illustration of the VelACaR tracking pipeline’s results with the CRN backbone	50
5.1	Diagram of the tracking pipeline before module improvements	54
5.2	Diagram of the tracking pipeline after module improvements	54
5.3	Illustration of the trailer category issue with the CRN detection method	65
6.1	Overview of the tracking pipeline and its switching mechanism	73
6.2	CRN’s performance evolution with increasing camera noise levels . . .	80
6.3	CRN’s performance evolution with increasing radar noise levels	81
6.4	Tracking performance evolution as the maximum camera noise threshold before switching is gradually increased	85
6.5	Tracking performance evolution as the maximum radar noise threshold before switching is gradually increased	86
7.1	Tracking performance drop induced by the synthetic data degradation with the CRN (camera-radar) backbone	90
7.2	Tracking performance drop induced by the synthetic data degradation with the Sparse4D (camera-only) backbone	91
7.3	Tracking performance drop induced by the synthetic data degradation with the RadarDistill (radar-only) backbone	92

7.4	Category-wise tracking accuracy comparison between all static models and an SwitchTrack, ideal noise recognition	95
7.5	Category-wise tracking precision comparison between all static models and an SwitchTrack, ideal noise recognition	96
7.6	Category-wise tracking accuracy comparison between the static CRN-VelACaR tracker and SwitchTrack, using noise recognition models .	97
7.7	Category-wise tracking precision comparison between the static CRN-VelACaR tracker and SwitchTrack, using noise recognition models .	98
7.8	Visual comparison between a static CRN-VelACaR tracker and SwitchTrack on a noisy camera data sample	101
A.1	The different range modes of the ARS-408-21	113
A.2	Continental TM 's ARS-408-21 Long Range Radar Sensor 77 GHz datasheet	116

Abstract

As autonomous vehicles navigate dense urban environments, three-dimensional detection and tracking of objects is a critical step to ensure passenger safety, making it a cornerstone of autonomous navigation. Furthermore, a robust tracking pipeline is of the utmost importance to ensure that the vehicle maintains awareness of its surroundings, even under high sensor data degradation. Early research focused on vision-based two dimensional (2D) object detection, but extending this to three-dimensional (3D) detection proved challenging due to difficulties in depth estimation with monocular cameras. To address this, camera data is now fused with range-measuring point cloud systems, enabling detection and tracking methods to achieve significantly higher accuracies. Given concerns over affordability and robustness, radars are particularly well-suited due to their availability and resilience to adverse weather conditions.

However, previous methods have not addressed the robustness of camera-radar fusion detection and tracking in scenarios involving sensor failure or data degradation from sources other than weather. In this project, I aim to lay a foundation for safer and more robust three-dimensional multi-object tracking.

I propose a versatile, plug-and-play pipeline that leverages filtering methods for their adaptability to new scenarios and detection backbones. My main contributions are the design of the first camera-radar realistic noise synthesizer, along with the first camera-radar noise recognition models, along with my backbone-switching approach. My noise synthesizer allows me to artificially recreate sensor failure and data degradation on an existing large-scale autonomous vehicle dataset, without the need for additional information. Furthermore, my noise recognition module is used to estimate sensor noise levels, enabling dynamic switching between off-the-shelf detection backbones to adapt to data degradation.

Average results across all the categories and scenes of the nuScenes validation dataset indicate that, in the event of sensor failure, my tracking method is on average 1.01%

more precise compared to using each detection backbone individually, with a small 0.1% tracking accuracy tradeoff. On the *car* category specifically, my method shows an increase in tracking accuracy of up to 2.6% along with an increase in precision of up to 1.05%. Accuracy and precision are represented by the AMOTA and AMOTP metrics, respectively, of the nuScenes dataset.

Chapter 1

Introduction

Autonomous navigation of air and ground vehicles relies first and foremost on the detection and tracking of surrounding objects. Moving in a complex and dynamic urban environment requires autonomous vehicles to continuously observe the three dimensional dynamics of nearby objects, to safely and correctly navigate to their destinations. The most commonly used sensors for this purpose are cameras, LiDAR, and radar (Shi et al. 2024).

Cameras are the most widely used sensor of the three, largely because they produce rich visual data that enables semantic understanding through downstream processing (Shi et al. 2024). They are also inexpensive and widely available. In fact, two dimensional detection of objects in images has had remarkable improvements over the past decades, with current State-of-The-Art (SOTA) methods, such as co-DETR, from Zong et al. (2023), reaching 66% accuracy on the 80-categories COCO dataset developed by Lin et al. (2015). However, the two dimensional pixel grid captured by monocular cameras makes camera-only 3D detection challenging, as it lacks direct depth information, making it difficult to accurately estimate an object’s distance, shape, and orientation.

To overcome this limitation, camera data is often fused with range-measuring point-cloud sensors, such as LiDAR or radar. Most current SOTA methods prefer LiDAR

systems for their highly dense point clouds, allowing for better 3D detection and tracking performance (Wang et al. 2024; Li et al. 2024, 2023). However LiDAR sensors are expensive and struggle in harsh weather conditions like rain, snow, or fog, limiting their real-world practicality (Dreissig et al. 2023).

In contrast, radars are significantly more affordable and highly robust to adverse weather, making them a strong candidate for reliable object detection and tracking in real-world applications, where environmental perturbations are inevitable. Furthermore, radars provide better range measurements accuracy than LiDARs, and have the unique capacity to measure object velocities using the Doppler-effect (Shi et al. 2024). However, automotive millimeter wave (mm-wave) radar point clouds are flat and much sparser, which makes extracting precise object features more challenging. As a result, despite being used for nearly a century in various applications, radar has only recently gained attention in the context of object detection and tracking for autonomous systems (Lang et al. 2020; William L. Melvin 2005; Patole et al. 2017).

Paradoxically, most SOTA camera-radar fusion methods focus solely on improving accuracy and overall performance, and few address their robustness to other real-world challenges such as sensor degradation or failure (Lee and Ang 2024; Wolters et al. 2024; Li et al. 2024; Wang et al. 2024). Since radars are chosen for their resilience to adverse weather conditions, I view this as a missed occasion to take in account additional failure origins (Shi et al. 2024).

This work aims at exploring this gap by creating a camera-radar 3D tracking pipeline capable of adapting to sensor failure, which I define by the degradation of the data, considering corrupted data more detrimental to detection than loss of input. To simulate the effect of noise, I create a noise synthesizer for images and radar point clouds, with the aim of generating realistic degradations on the existing data,

without needing additional information. The initial hypothesis behind my approach is that while camera-radar fusion methods yield better performance, their reliance on two sensors makes them more susceptible to sensor failure. Given this hypothesis, a radar-only detection method may outperform a camera-radar detector under degraded camera data. Likewise, a camera-only method may outperform a camera-radar method if the radar data is degraded enough.

My main approach is to create a camera-radar noise recognition dual neural network to estimate the level of degradation of each sensor. I then use a tracking-by-detection method with three detection backbones: the camera-radar CRN from Kim et al. (2023), the SOTA camera-only Sparse4D from Lin et al. (2023), and the SOTA radar-only RadarDistill from Bang et al. (2024). Using CRN as a default detection backbone, if the estimated degradation of either the cameras or the radars reaches a maximum threshold, I deactivate CRN and switch to either Sparse4D or RadarDistill.

My main contributions include:

- The design of the first Camera-Radar noise synthesizer. My method uses only the output data of the two sensors—images and point clouds—to generate realistic degradation at a controllable level. I increase the range of possible degradation origins of traditional camera noise synthesizers, and propose the first existing method for radar point clouds.
- The first camera noise recognition model capable of recognizing noise levels of more than one type. While existing image noise recognition methods focus only on additive noise, I also consider blur and exposure issues.
- The first mm-wave radar point cloud noise recognition.

- I propose a versatile tracking pipeline that combines noise recognition, off-the-shelf detection, and tracking-by-detection, to adaptively switch detection backbones if degradation levels get too high.
- The first mm-wave radar-only tracking pipeline of the nuScenes dataset.

This manuscript will present a detailed description of the different modules that compose the proposed pipeline, as well as the thought process behind their development.

To begin with, Chapter 2 presents an overview of previous works and of the dataset I used. Chapter 3 describes my sensor data degradation synthesizer and noise recognition models. Chapter 4 introduces the initial tracking method that served as the foundation for the final tracking pipeline, which is presented in Chapter 5. Chapter 6 outlines the integration of the tracking pipeline and noise estimation process, culminating to the design of an adaptive tracking pipeline. Finally, Chapter 7 presents the results of my work, comparing the performance of my adaptive tracking pipeline against non-adaptive baselines, under degraded sensor conditions.

Chapter 2

Related Works

2.1 Data Degradation Synthesis and Recognition

Data Degradation in Images:

In the existing literature on vision-based object detection, noise synthesis is often used to augment training data, improving the model’s generalization to new inputs. In more recent methods—such as those using GANs—data degradation is an integral part of the training process (Goodfellow et al. 2020). The most common form of degradation is additive noise, typically simulated by sampling from various random distributions (Zamir et al. 2020; Wei et al. 2020). While some methods, such as Kaneko and Harada (2021), also simulate blur effects in addition to additive noise, and others like Afifi et al. (2021) simulate over- and under-exposure, no existing method combines all four. Furthermore, I propose a novel approach using a single control dial to adjust the overall severity of image deformation, an option not offered by previous methods.

Noise Recognition in Images:

Most methods addressing data degradation aim to denoise the input image. However, only a few focus on estimating the level of degradation, even though such estimates could help denoising pipelines adjust their strength accordingly (Chuah et al. 2017; Vasuki et al. 2012). De-noising networks often assume the input is noisy, and a noise

estimation module could be used to bypass them when the input is already clean.

Data Degradation in Radar Point Clouds:

For mm-wave radar point cloud noise simulation, most methods rely on simple random deformations—either applied to individual points (e.g., noise on range, velocity, or azimuth values as in Svenningsson et al. (2021))—or to the entire scene (e.g., random rotations and translations, as in Thomas et al. (2024)). These approaches, usually used for generalization purposes, as in Bang et al. (2024) and Thomas et al. (2024), lack physical grounding and do not reflect the structured nature of radar noise.

Noise Recognition in Radar Point Clouds:

To the best of my knowledge, there are no existing methods to estimate the amount of noise in mm-wave radar point clouds.

2.2 3D Detection and Tracking

As mentioned in Chapter 1, most 3D detection and tracking methods rely on LiDAR systems for their precision. For instance PointNet, from Qi et al. (2017), and AB3DMOT, from Weng et al. (2020), are widely recognized as foundational methods for LiDAR-only detection and tracking, respectively. While vision-based approaches have historically performed well in 2D detection tasks, 3D detection has proven more challenging, as demonstrated by the adaptation attempts from Wang et al. (2021). Although recent camera-only methods, such as Sparse4D from Lin et al. (2023), remain competitive, many high-performing approaches rely on camera–LiDAR fusion to overcome the limitations of each sensor: the lack of depth information in images and the

sparse semantic content in point clouds. In 3D camera-LiDAR tracking, notable examples include those proposed by Wang et al. (2024), Li et al. (2024), and Li et al. (2023), all of which outperform SOTA camera methods by over 10% in tracking accuracy. It is worth noting that the work from Lin et al. (2023) remains the state-of-the-art among camera-only methods in both detection and tracking.

While camera-LiDAR fusion has historically dominated the field in terms of performance, there has been growing interest in camera-radar systems, driven by concerns about robustness in adverse weather conditions. Recent state-of-the-art camera-radar methods slightly outperform camera-only systems in 3D detection, exceeding their accuracy by approximately 1% (Lin et al. 2024). However, in the tracking domain, camera-radar approaches are still underdeveloped. The precision of the best-performing camera-radar tracking method to date trails behind the top camera-only trackers by around 10%, likely due to the limited research attention the area has received so far (Wolters et al. 2024).

Regarding tracking methodologies, I distinguish two standards: tracking-by-detection (TBD) and machine learning (ML)-based tracking. In tracking-by-detection, new detections are matched with the predicted positions of previously tracked objects, typically using mathematically defined models. The most common approach involves a filtering algorithm to estimate and update object states, with the Kalman filter from Kalman (1960) being the most widely used (Weng et al. 2020; Zhang et al. 2023; Shi et al. 2024). Most current SOTA tracking pipelines adopt the TBD standard, including high-performing methods such as Wang et al. (2024). ML-based tracking, by contrast, uses neural networks to replace or enhance components of the filtering pipeline. For example, Baumann et al. (2024) employs neural networks to extract visual features that improve data association. Other methods go further by combining detection and

tracking into a unified framework, often referred to as end-to-end tracking (Liang and Meyer 2022; Ding et al. 2023). However, ML-based tracking typically requires temporally ordered datasets and long training times, while offering limited interpretability and fewer avenues for manual refinement.

2.3 Sensor Failure Robustness

As mentioned in Shi et al. (2024), robust tracking remains a largely unexplored area of research. While a lot of them focus on denoising strategies, only few methods explicitly focus on robustness *to* sensor degradation. Bansal et al. (2022) demonstrate reliable tracking using only radar data when the camera fails under adverse weather conditions by independently extracting information from each sensor. However, it does not provide a solution for radar failure, nor for failure due to sensor degradation caused by electro-magnetic (EM) interference or thermal noise. Other methods such as Imm-Fusion, from Chen et al. (2023), explore the promising approach of using transformer networks to handle noise, either by selecting clean information from the sensors or by attempting to first regenerate the original data by removing the noise. However, the “black box” nature of these methods makes them less suitable for real-world autonomous vehicles. Once more, even if these methods were adopted, they could still benefit from an accurate estimation of the amount of noise, allowing systems to determine when noise should be removed, and how much to remove, in order to avoid losing valuable information.

In Kim et al. (2023), a robustness analysis was conducted, but it only considered the impact of input loss (e.g., losing input from one camera out of six). I argue that degraded or corrupted data can have a more detrimental effect on tracking accuracy and

precision than missing input, as it introduces erroneous information that may mislead the detection and tracking pipelines, resulting in incorrect predictions. For instance, degraded data may lead to false positive detections, which may lead to ghost tracks that merge with real tracks, causing fragmentation and identity switches, ultimately leading to overall reduced performance.

2.4 Dataset

Due to the challenging nature of 3D detection and tracking, there is a need for large 3D annotated datasets. In autonomous navigation, the nuScenes dataset from Caesar et al. (2020) is a reference, providing users with 1000 urban-focused scenes of 20 seconds, accumulating data from cameras, radars, and LiDAR. NuScenes provides benchmarks for 3D Detection across 10 categories, and for 3D tracking across 7 of them. As mentioned in Shi et al. (2024), it is considered to be one of the first large-scale datasets, with over 1.4 million images, 390 thousand LiDAR sweeps, 1.4 million radar sweeps, and 1.4 million object bounding boxes in 40 thousand keyframes.

The sensor set-up of the vehicle, illustrated in Figure 2.1, includes six monocular cameras with a frame rate of 12 FPS and a resolution of 1600x900, five long range Continental ARS 408-21 77GHz radars operating at 13Hz, and a 360° spinning Velodyne HDL-32E LiDAR sensor operating at 20 Hz with 32 beams and 32 channels (Caesar et al. 2020). The camera and LiDAR sensors are synchronized by triggering the camera exposure when the LiDAR sweep aligns with the center of the camera FOV. For radar data, synchronization is achieved by taking the closest radar acquisition for each camera frame. Additionally, the car is equipped with an Inertial Measurement Unit (IMU) and a Global Navigation Satellite System (GNSS) GPS for accurate localization.

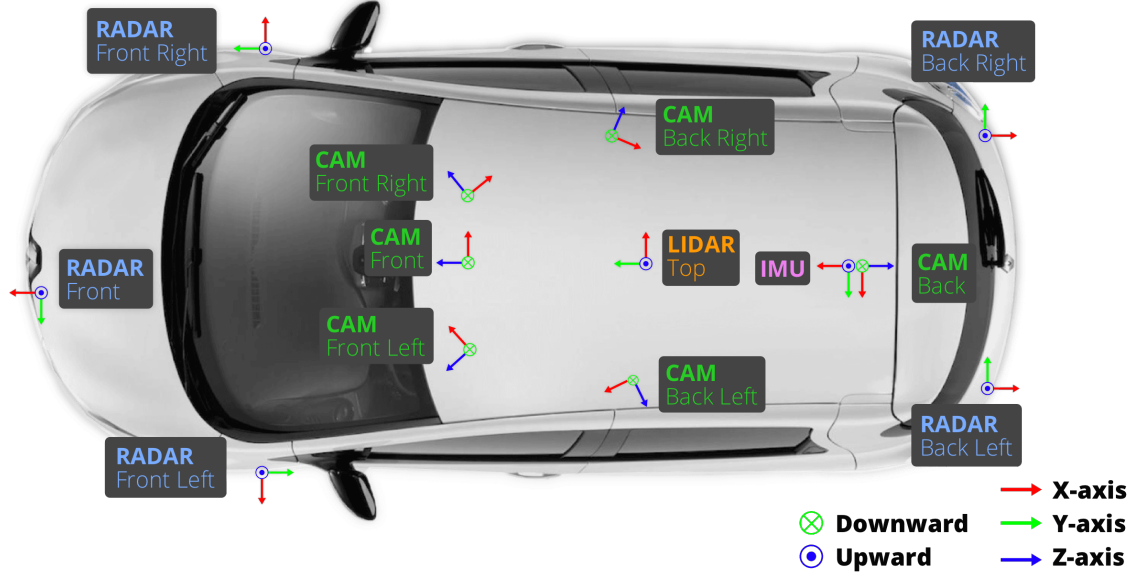


Figure 2.1: Sensor configuration of the nuScenes car from Caesar et al. (2020)

The dataset is organized hierarchically. Each scene is divided into **samples** and **sweeps**. Samples, also referred to as **keyframes**, are sampled at 2Hz, with manually annotated data. Sweeps are additional data points with interpolated annotations, leading to a total sampling frequency of 12Hz, with approximately six sweeps between each keyframes.

Information is accessed using unique tokens assigned to dictionaries named “*sample*”, “*sample_data*”, and “*sensor_data*”. The first sample token is provided in the scenes’ information. The *samples* dictionaries contain the tokens of each sensor’s *sample_data* (e.g., data from “CAM_FRONT”), while each *sample_data* dictionaries further contains the path to the data file and token to additional *sensor_data*, such as translation and orientation relative to the IMU.

For example, to access and parse the front camera data, I first extract the first *sample* dictionary, then the front camera’s *sample_data*, which contains the acquired image

and the *sensor_data* token. The *sensor_data* dictionary contains information to perform the projection of objects between the camera and IMU coordinate systems, as well as between the IMU and global coordinate systems. I then use the tokens contained in the *next* or *prev* keys of the *sample_data* dictionary to navigate forward or backward in the scene. These keys are available at both the *sample* and *sample_data* levels, allowing direct parsing of samples (using the *sample* keys) or of all the data points (using the *sample_data* keys). When there is no more *next* key, I move on to the next scene.

The dataset is split in the three common splits: training, validation, and testing, composed of 700 scenes, 150 scenes, and 150 scenes, respectively. A mini dataset is also available composed of 8 scenes from the training set and 2 scenes from the validation one.

Chapter 3

SynthCRAV: Camera-Radar Noise Synthesizer for Autonomous Vehicles

The first step of the proposed framework is creating a noise synthesizer, which serves three purposes:

1. Training and testing camera-radar noise recognizers using lightweight Machine Learning models.
2. Create a degraded version of the nuScenes validation dataset to analyze the effectiveness of my method.
3. Quantifying the maximum amount of noise the CRN-backed tracking pipeline can handle before a single-sensor backbone is preferable.

3.1 Realistic Data Degradation Synthesizer

3.1.1 Camera Synthetic Degradation

I propose to simulate four common types of image degradation: blurring, over/under-exposure, and additive noise. To control overall degradation intensity, I define a noise dial, denoted as N_{lvl} , ranging from 0 to 1, and with a step size of 0.1. This can be viewed as a noise level ranging from 0% to 100% with a step size of 10%.

Blurring: This distortion simulates improper light focusing of the camera lens from the scene into the image plane. Most of the time, this is due to a calibration issue, but blurring can also happen from auto-focus failure, poor light conditions, shock-induced misalignment, or condensation on the camera lens or casing. My blurring method can also be considered to simulate motion blur, even though accurate simulation would require a different model. A blurring effect is commonly re-created with a Gaussian kernel convolution. I simulate different noise levels by increasing both the size and strength of the Gaussian kernel, according to:

$$k = 2 \cdot 10N_{lvl} + 1, \quad (3.1)$$

where k is the size of the Gaussian kernel used for convolution, and N_{lvl} is the noise level. The values of the kernel are computed from the two-dimensional Gaussian distribution:

$$G_{x,y} = \frac{1}{2\pi\sigma} \exp\left(-\frac{x^2 + y^2}{2\sigma^2}\right), \quad (3.2)$$

where G is the Gaussian kernel matrix, (x, y) are the coordinates within the kernel (of size $k \times k$), and $\sigma = N_{lvl}$ the standard deviation, which controls the intensity of the blurring. Before performing the convolution, the Gaussian kernel is normalized so that the sum of all its elements equals one. Figure 3.1 illustrates the blurring applied to the same image, at different distortion levels.



Figure 3.1: **Blur** effect at 0% (original), 30%, 60%, and 100% (max) distortion levels

Overexposure/Underexposure: This distortion simulates issues with the auto-adjustment of the camera's exposure levels. When there is a sudden change in ambient light level, such as when entering (or exiting) a tunnel or passing under a bridge, there is a short instant before the camera's exposure settings adjust, leading to overexposure or underexposure of the image. Depending on the velocity of the car at that moment, the magnitude of the distortion can be different. I simulate over and under-exposure conditions by convolving the image with a small Gaussian kernel and adjusting pixel intensities. The overall brightness is adjusted by a multiplicative factor W , while

the convolution with a gaussian kernel helps smooth out the local distribution of the distortion:

$$K = W \cdot \frac{1}{16} \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}, \quad (3.3)$$

with K the Gaussian kernel, and W the intensity factor. To simulate overexposure, I use $W = 1 + 3N_{lvl}$, increasing image brightness up to four times at full distortion. Concurrently, to simulate underexposure, I use $W = \frac{1}{1 + 3N_{lvl}}$, reducing intensity by the same factor. While this model does not fully capture nonlinear saturation, it creates visually degraded conditions representative of exposure imbalance. This approach serves my goal of challenging detection robustness across different distortion at a controllable level. Figure 3.2 illustrates both effects on the same image, at different distortion levels.



Figure 3.2: **Overexposure** (top) and **underexposure** (bottom) degradations at 0% (original), 30%, 60%, and 100% (max) distortion levels

Additive Noise: This distortion simulates potential internal or external disruptions such as electro-magnetic interferences (EMI) or thermal noise originating from within the sensor. Low-intensity scenarios, such as nighttime conditions, can also lead to additive noise as image quality significantly decreases. Given how common these causes are, additive noise is likely the most frequent distortion encountered by autonomous vehicles, and it is also the most commonly simulated one.

While Gaussian convolutions can filter additive noise, high levels of distortion may either remain partially unfiltered or result in a completely blurred image. Recent studies have shown that Generative Adversarial Networks (GANs) can effectively reduce additive noise in images, even in cases of severe distortion (Goodfellow et al. 2020). However, such denoising methods can still benefit from prior knowledge about the expected noise level, as discussed in Chapter 2.

I simulate additive noise by generating a random $W \times H$ pixel grid drawn from a Normally distributed random variable $N(0, N_{lvl})$, with W and H being the width and height of the image, respectively. The resulting grid is then directly added to the image:

$$I' = I + \mathcal{N}(0, \sigma^2), \quad (3.4)$$

where I' is the noisy image, I the original image, and $\sigma = N_{lvl}$ is the standard deviation of the gaussian distribution. Here the noise dial N_{lvl} is being directly used to increase σ , ranging from 0.1 to 1. Figure 3.3 illustrates the additive Gaussian noise degradation on the same image, at different distortion levels.



Figure 3.3: Normally distributed **additive noise** degradation at 0% (original), 30%, 60%, and 100% (max) distortion levels

3.1.2 Radar Synthetic Degradation

One of my main contributions lies in developing the first realistic mm-wave radar degradation synthesizer. I design a physics-based SNR (signal-to-noise ratio) degradation model applied to an existing radar point cloud, operating solely with the information provided by the sensor and without requiring any additional data. To the best of my knowledge, such a synthesizer does not exist.

Structure of the Radar Data:

Before describing the radar data synthesizer, we will first examine the structure and characteristics of the radar point cloud. Each radar sensor outputs a 3D point cloud at a 13Hz sweep frequency, with the data stored in a specialized binary “.pcd” (Point Cloud Data) file format. To process this data, I developed a custom decoding function to decode and re-encode the PCD files, as no existing method was available for handling

the full range of data. Among the information contained in the file, we will mostly look at those described in Table 3.1.

	(x,y,z)	Dynamic properties	RCS	(v_x, v_y)	$(v_{x_{comp}}, v_{y_{comp}})$	Ambiguous state	invalid state	pdh0
Description	Cluster position. x front, y left, z top.	Dynamic property of cluster (moving / not moving).	Radar Cross Section	2D relative velocity.	2D motion-compensated radial velocity.	State of Doppler ambiguity solution.	Cluster validity state.	False alarm probability of cluster.

Table 3.1: Overview of the radar output data structure

We can separate the information in categories. The (x, y, z) , (v_x, v_y) and $(v_{x_{comp}}, v_{y_{comp}})$ channels contain the spatial information of the point. The dynamic properties, ambiguous state, invalid state, and pdh0 are information estimated by the radar concerning the validity of the point and its false alarm potential. Those can be assimilated to a confidence metric. Finally, the RCS value is an intrinsic property of the reflective surface the point originates from, and can be thought of as a measure of its reflectiveness capacity.

A few remarks can be made concerning the radar data:

- The radars are set with a 0-degree elevation angle in the nuScenes dataset, meaning the value of z is always 0.
- Radars measure the velocity of objects using the Doppler effect. Inherently, this means the measure made is only a relative radial velocity. Relative velocity indicates that if both the vehicle and the target move with the same speed and orientation, the radar will read the target velocity as 0 m/s. Radial velocity indicates the measured velocity is the projection of the velocity on the line between the radar’s center and the target.
- $\vec{V}_{comp} = (v_{x_{comp}}, v_{y_{comp}})$ is a compensated *radial* velocity.
- $\vec{V} = (v_x, v_y)$ is *not* a radial velocity, indicating the sensor does some additional tracking and processing step to estimate this vector. As each manufacturer has its

own method kept secret, we cannot know exactly how those values are extracted. $\vec{V}$ is, however, a relative velocity.

- Almost all other fields are also estimated internally by the sensor, using unknown methods.
- The points stored in the Point-Cloud-Data files are in fact clusters of points, with the output values being an average of the position and velocities of the individual points. This is most probably done to remove outliers and maximize confidence rates.

Mapping the Noise Level Dial N_{lvl} to Signal-to-Noise Ratio (SNR):

Similar to the camera noise synthesizer, the radar noise synthesizer includes a noise level dial denoted as N_{lvl} . Given I want to create a physics-based degradation model, I convert the noise level to a decrease of the signal-to-noise ratio (SNR) in dB scale, common in signal processing terminology:

$$SNR_{lin} = \frac{P_r}{P_N}. \quad (3.5)$$

Where P_r is the power of the received signal, and P_N the noise power. The subscripts “lin” and “dB” are used to denote linear scale and decibel scale, respectively.

Given the original P_r and P_N are unknowns, so is the original SNR. However, since my goal is to simulate the effect of an increase of N_{lvl} in noise, it is sufficient to work in terms of corresponding decrease in SNR. In practice, the conversion of the noise dial is designed so that:

- At $N_{lvl} = 0\%$, $SNR' = SNR$.

The new Signal-to-Noise-Ratio, denoted SNR' , is the same as the original.

- At $N_{lvl} = 10\%$, $SNR'_{dB} = SNR_{dB} - 2dB$.

The new SNR is a -2dB decrease from the original SNR.

- At $N_{lvl} = 100\%$, $SNR'_{dB} = SNR_{dB} - 20dB$.

The new SNR is a -20dB decrease from the original SNR.

The new Signal-to-Noise ratio, SNR' , is therefore related to the noise level, N_{lvl} , by:

$$\begin{aligned}
 SNR'_{dB} &= SNR_{dB} - 20N_{lvl}, \\
 &\equiv SNR'_{dB} = SNR_{dB} - 10\log_{10}(10^{-2N_{lvl}}), \\
 \implies SNR'_{lin} &= SNR_{lin} \cdot 10^{-2N_{lvl}}.
 \end{aligned} \tag{3.6}$$

Note: In previous iterations of the project the noise was set to range from 0dB to -10dB instead of -20dB, but this proved to be too narrow and did not encompass the full range of realistic degradation of radar data.

Impact of Noise on Radar Point Clouds:

A synthesizer aiming at realistically re-creating noise in radar data can be divided in three steps: **ghost points generation**, **increase of false negatives**, and **noise-induced range-Doppler shifts**.

Ghost Point Generation: Some urban environments may encourage the apparition of multipath artifacts more than in the nuScenes dataset, due to highly reflective surfaces for example, leading to the presence of more ghost points. While ghost points are unrelated to the background noise, they are a possibility in real-world scenarios, and may have a strong-enough RCS (Radar Cross Section) value to pass through the radar's filtering methods. I therefore aim at simulating higher apparition rates of ghost points to recreate highly reflective environment. Although some methods like Wu et al.

(2025) use GANs that implicitly re-create ghost points for downstream tasks such as sensor fusion, my physics-based approach allows for more accuracy and control on the generation.

I start by drawing a random number of new points from a uniform distribution, in my case from 0 to 4 new points. Then, for each new ghost point, I draw a random range/angle couple (r, θ) from two uniform random variables $\mathcal{U}_r(r_{min}, r_{max})$ and $\mathcal{U}_\theta(\theta_{min}, \theta_{max})$. r_{min} is set as the radar’s minimum range of 0.2 meters (see Appendix A.2). While the maximum range of the radar is 250m, most detections fall under 100m. To avoid generating points far away from the main cluster, which would make them easy to automatically detect and remove, I set r_{max} as the farthest point in the current frame, plus a 10 meters margin. θ_{min} and θ_{max} depend on the point’s range, following the radar range mode characteristics (see Appendix A.1).

Once the position is set, I create a synthetic velocity vector $\vec{V}$. However, since we cannot know how the radar extract this vector for real points, we can only sample from the current distribution of points. This will also prove useful to set the other parameters based on velocity for which we have no model.

From the resulting relative velocity vector, I can create a radial relative velocity vector $\vec{V}_r$, by projecting $\vec{V}$ of the line between the target and the sensor’s center. Finally, I calculate the corresponding compensated radial velocity vector $V_{comp}^{\rightarrow}$ of the point. I first calculate the velocity of the nuScenes car (ego velocity) by computing the difference in position between the current frame and the next one, as nuScenes does not provide a direct read of GPS data but does provide the coordinates of the

IMU. I then convert the ego velocity from the IMU coordinate system to the sensor's coordinate system. From there $V_{comp}^{\rightarrow}$ is calculated with:

$$V_{comp}^{\rightarrow} = (\mathbf{V}_r - \mathbf{V}_{ego}) \cdot \vec{V}_r, \quad (3.7)$$

with $\mathbf{V}_r$ and $\mathbf{V}_{ego}$ the magnitudes of the radial relative velocity vector and of the ego velocity vector, respectively, in m/s. $V_{comp}^{\rightarrow}$ is the compensated compensated radial velocity vector, and $\vec{V}_r$ the radial relative velocity vector, in m/s.

The final step is to generate a synthetic Radar Cross Section (RCS) value, in dBsm (decibels relative to a square meter). Realistically, the RCS value of a ghost point would most likely be small (e.g., -5dBsm), but, if the origin of the multipath had a very high RCS value, the resulting ghost point may still yield a high reflective power, leading to a high estimation of its RCS value by the radar (e.g., +5 dBsm). When generating a ghost point, I therefore consider there is a high chance of it having a low-RCS, and a small chance of it having a high-RCS value. To represent this, I create a half Gaussian random variable $X \sim |\mathcal{N}(0, 1/3)|$, bounded between 0 and 1 using rejection sampling. I then draw a random number following X , which is finally mapped to the ordered distribution of the known RCS values of the current frame. In case the frame has too few points, the RCS value is set to -5 dBsm by default, the lowest observed value.

I also set the invalid state value (see Table 3.1) by randomly selecting a value between one of the following: valid cluster with low RCS, valid cluster with high child probability, valid cluster with high probability of being a 50 deg artifact, valid cluster with no local maximum, and valid cluster with high artifact probability. While the cluster state has to be valid, otherwise it would simply be ignored, the sensor may still

be able to detect something wrong with a ghost point, which is represented here.

RCS-Ruled False Negatives: In autonomous navigation applications, the preferred radar sensors are mm-wave radars, due to their robustness to adverse weather, atmospheric conditions, and shocks. Furthermore, these sensors can operate under any light level, and offer a compact design making them easier to integrate to vehicles (Cadence Design Systems 2022).

Most traditional mm-wave radars use the CA-CFAR (Cell Averaging - Constant False Alarm Rate) algorithm (Raghavan 1992) to filter out the detections from the background noise using probabilistic guidance. As with all noise-removal methods however, an increase in the background noise level will lead to a higher rate of missed detections, as weaker reflections fall below the detection threshold. Whether a reflection is detected by the radar or not depends primarily on its SNR. Therefore, as the noise level increases, the new signal-to-noise ratio, denoted SNR' , will tell us if a given point would have been detected or not.

The radar equation tells us more on the relation between the received power and the parameters of a reflection:

$$P_r = \frac{P_t G_t A_{eff}}{(4\pi)^2 r^4} \sigma, \quad (3.8)$$

with P_r the received power, P_t the transmitted power, G_t the gain of the antenna, A_{eff} the effective aperture of the received antenna (which is transmitter/receiver in my case), σ the target RCS, and r the target range. Only σ and r are known as they

are measured by the radar. However, since they are properties of the radar itself, P_t , G_t and A_{eff} can be assumed to constants. Therefore, I can say that for each point:

$$P_r \propto \frac{\sigma}{r^4}. \quad (3.9)$$

This means I can fully describe the received power ratio between two points using only measured variables. From Eq. 3.5, and considering the noise power P_N to be constant between two points:

$$SNR \propto \frac{\sigma}{r^4}. \quad (3.10)$$

This means I can also describe the SNR ratio between two points using known variables. When increasing the noise, from Eq. 3.6 we get:

$$\begin{aligned} SNR'_{lin} &= SNR_{lin} \cdot 10^{-2N_{lvl}}, \\ \implies SNR'_{lin} &\propto \frac{\sigma}{r^4} \cdot 10^{-2N_{lvl}}. \end{aligned} \quad (3.11)$$

I can now fully describe the SNR ratio between two points, under different amounts of noise, using known variables only.

Assuming the lowest received power in a given point cloud is close to the minimum detection threshold set by the radar at that time, I want to know which detections would have been missed by the radar if the noise level was at N_{lvl} . I cannot know the exact value of the lowest received power (see Eq. 3.8), but I know it only depends on $\frac{\sigma}{r^4}$ (see Eq. 3.9). Therefore if I calculate $\beta = \frac{\sigma}{r^4}$, I know that the smallest β coefficient, denoted β_{min} , represents the lowest received power of the original cloud. This becomes my reference threshold under which I know a point would not have been detected by the radar.

As the noise level increases, I can calculate a new received power coefficient for each point of the scene using Eq. 3.11:

$$\alpha = \frac{\sigma}{r^4} \cdot 10^{-2N_{lvl}} + w, \quad (3.12)$$

with α the new power coefficient under a noise level of N_{lvl} , r the range of the point, and σ its RCS value. I also add $w \sim \mathcal{N}(0, \beta_{min})$, a small random noise, representing potential estimation errors, calculation errors, and physical uncertainties. Any point with an α value lower than β_{min} is dropped from the dataset. As N_{lvl} increases, the α values decreases, meaning more points get dropped. At a -20dB noise level, entire point clouds can get emptied, even though only a few points are missed at -2 dB, demonstrating that my range of noise levels covers the full spectrum of realistic possibilities.

Noise-Induced Shifts: Concurrently to the increase in missed points, the measurement accuracy of a sensor worsens (increases) as well with an increase in noise. The Cramér-Rao bound suggests the accuracy of a sensor is directly proportional to the inverted square root of the SNR (Mark A. Richards 2005):

$$acc \propto \frac{1}{\sqrt{SNR}}. \quad (3.13)$$

In the datasheet of a radar (see Appendix A.2), we are provided with the measurement accuracies of the range, orientation (azimuth), and velocity of the sensor. However, the values provided by a manufacturer are usually estimated in controlled conditions and for high-RCS targets. This means that even in the original data, since the conditions were non-ideal and the RCS of each target was lower, the measurement accuracy was different than the one provided in the datasheet of the radar. In this section, I want to quantify what the measurement accuracy of each point would have been if the noise

level was different. To achieve this, the first step is to define what the measurement accuracy was at the time of the measure, in the original dataset.

For a given point at a fixed range, I consider: $SNR \propto \sigma$, with σ the RCS. From Eq. 3.13, I can then calculate the measurement accuracy for each point of the cloud in the original dataset:

$$\begin{aligned} acc_{min} &\propto \frac{1}{\sqrt{\sigma_{max}}}, \\ \Rightarrow \frac{acc_{min}}{acc} &= \frac{\sqrt{\sigma}}{\sqrt{\sigma_{max}}} = \sqrt{\frac{\sigma}{\sigma_{max}}}, \\ \Rightarrow acc &= \frac{acc_{min}}{\sqrt{(\frac{\sigma}{\sigma_{max}})_{lin}}}, \end{aligned} \tag{3.14}$$

with acc_{min} the minimum (best) accuracy, which is computed at the maximum RCS value σ_{max} . $\sigma_{max} = 64$ dBsm, from the radar's characteristics.

From Eq. 3.5, I can directly take in account the new noise level in Eq. 3.14:

$$\begin{aligned} acc' &\propto \frac{1}{\sqrt{SNR'}} = \frac{1}{\sqrt{SNR}} \cdot \frac{1}{\sqrt{10^{-2N_{lvl}}}}, \\ \Rightarrow acc' &= \frac{acc_{min}}{\sqrt{10^{-2N_{lvl}} \cdot (\frac{\sigma}{\sigma_{max}})_{lin}}}. \end{aligned} \tag{3.15}$$

This gives me the new measuring accuracy, acc' , when taking in account both the new noise level and the original accuracy, different from the radar's theoretical minimal accuracy (denoted by acc_{min}). Note that in the equation, the accuracy only depend on the RCS value (σ), but since the minimal accuracies are given at different ranges and angles, they do also depend on the position of the point.

We are now able to calculate the range, angle, and velocity measurement accuracy of each point, meaning I know the error bounds of each measurements. For each point I can calculate the measurement accuracies at the new noise level, noted $[acc'_r, acc'_\theta, acc'_v]$. To visualize this, we can imagine a 3D ellipsoid around each point, of semi axes

$[acc'_r, acc'_\theta, acc'_v]$. At the time of the measure, the reading could have been anywhere in that error ellipsoid. To simulate this, I draw random values from three Gaussian random variables $[\mathcal{N}(0, acc'_r), \mathcal{N}(0, acc'_\theta), \mathcal{N}(0, acc'_v)]$, and add them to the original range, angle, and velocity values (respectively) to synthesize the effect of noise on the point's measurements. I also propagate the effect of the noise on velocity measurements from $\vec{V}$ to $V_{comp}^{\rightarrow}$ as follows:

$$\begin{aligned}
\vec{r} &= (x, y)^T, \\
\hat{r} &= \vec{r}/\mathbf{r}, \\
\vec{V}_r &= (\vec{V} \cdot \hat{r})\hat{r}, \\
\psi &= atan2(\vec{V} \times \vec{V}_r, \vec{V} \cdot \vec{V}_r), \\
\vec{V}' &= (\mathbf{V}_r + w_v)/cos(\psi), \\
\alpha_{v_{comp}} &= atan2(V_{y_{comp}}, V_{x_{comp}}), \\
\mathbf{V}_{comp}' &= \mathbf{V}_{comp} + w_v, \\
V_{comp}^{\rightarrow'} &= \begin{bmatrix} \mathbf{V}_{comp}' \cdot cos(\alpha_{v_{comp}}) \\ \mathbf{V}_{comp}' \cdot sin(\alpha_{v_{comp}}) \end{bmatrix},
\end{aligned} \tag{3.16}$$

with:

- $\vec{r}$: The position vector between the center of the sensor and the target at position (x, y) ,
- $\vec{V}_r$: The radial relative velocity vector (projection of $\vec{V}$ on $\vec{r}$),
- $\vec{V}$: The relative velocity,
- $\vec{V}'$: The synthetically modified relative velocity,
- ψ : The projection angle between $\vec{V}$ and $\vec{V}_r$,

- $\alpha_{v_{comp}}$: The conversion angle between the polar coordinate $\mathbf{V}_{comp}$ and its cartesian vector $V_{comp}^{\rightarrow}$,
- w_v : The velocity noise drawn from $\mathcal{N}(0, acc'_v)$.

And with the following convention: $\vec{X}$ is the vector $(x_1, x_2, \dots, x_n)$, $\hat{X}$ is the normalized vector of $\vec{X}$, and $\mathbf{X}$ is the magnitude of the vector $\vec{X}$ with $\mathbf{X} = \sqrt{\sum X_i^2}$.

Figures 3.5 and 3.4 illustrate the effect of noise levels on the same radar point cloud. Figure 3.4 shows the evolution of the same point cloud from the front radar, under four noise levels. Figure 3.5 is a 3D scan of the complete scene, aggregating the point clouds of all five sensors. The red cross in the scene represents the nuScenes car. We can recognize the output of the front radar on the right-side of the image. The visualized scene is the same as the one used in Figures 3.1, 3.2, and 3.3, so we can use the camera view to situate the objects in the point cloud. As noise level increases, we can see less and less points due to the RCS-ruled drops, along with an increasing deformation of the cloud's structure. While on the original image we can make out the sides of the road, the points belonging to the cars in front, and the cluster of points coming from the truck at the bottom left of the cloud, as the noise increases the shapes get increasingly deformed. At -20 dB SNR, the structure of cloud becomes so degraded it is difficult to recognize any elements of the scene.

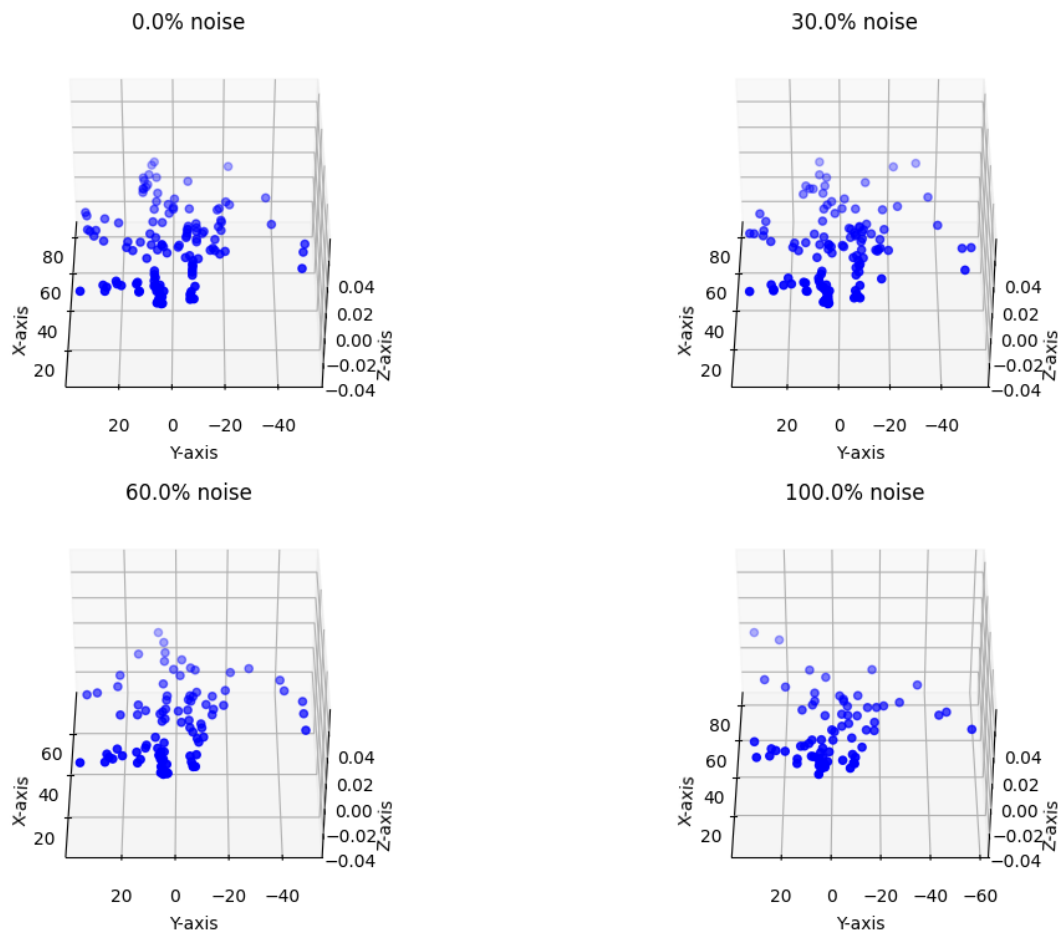


Figure 3.4: Effects of synthesized noise on a radar point cloud in a 3D scan at -0dB (original), -6dB, -12dB, and -20dB (max) noise levels

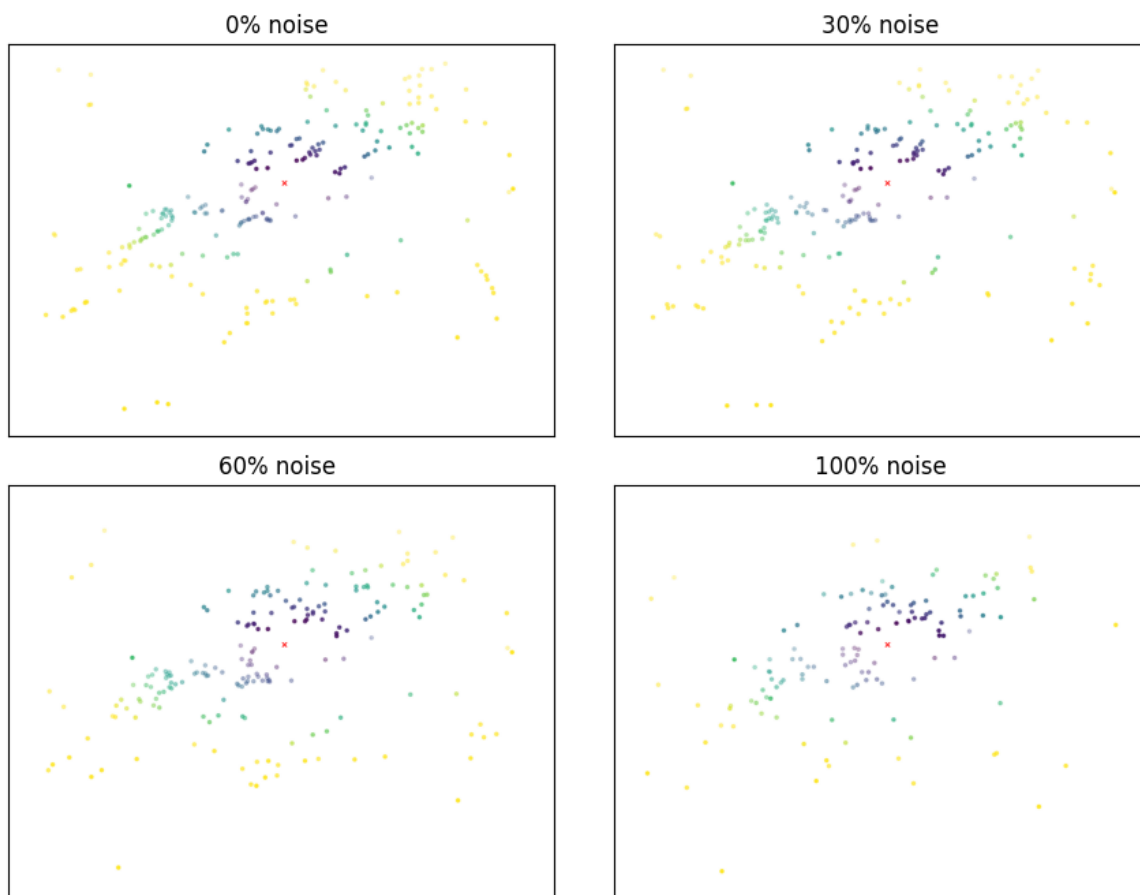


Figure 3.5: Effects of synthesized noise on the multi-radar point cloud scene reconstitution at -0dB (original), -6dB, -12dB, and -20dB (max) noise levels

3.2 Noise Recognition

With my realistic noise synthesizer in place, I can now focus on designing neural networks capable of recognizing the simulated levels of deformation.

3.2.1 Dataset Generation

The first step before designing my networks is to create a dataset on which I will train and test them.

Using nuScenes' mini dataset of ten scenes, I generate eleven noise levels, ranging from $N_{lvl} = 0\%$ to $N_{lvl} = 100\%$, with a step size of 10%. On the camera data, I generate these levels for all four types of degradation, leading to a dataset forty four times larger. However, I do not use the four nighttime scenes of the mini dataset for the image noise recognition, as they are already degraded at an unknown level. I also do not mix the four types of degradations to avoid generating too much data, and it is left for further work to expand on the camera noise recognizer with additional types of noise and their potential combinations.

Since radars are unaffected by light levels, all ten scenes are used and the resulting dataset is eleven times larger than the original one.

As mentioned in Chapter 2.4, the nuScenes mini dataset is composed of eight scenes from the 700-scenes training split and of two scenes of the 150-scenes validation split. This makes for an obvious split between training and test data for me, where all the data originating from the validation split will be used in testing, and all the data originating from the training split will be used in training and validation. For the camera data, the exclusion of nighttime data brings the total amount of training scenes to four, instead of eight for the radar data.

3.2.2 Noise Recognition Models

The core principle of my noise recognition module is to analyze the output data of camera and radar sensors, and estimate the amount of degradation within them. Directly quantifying the level of degradation instead of being a binary classification problem means my method can be easily integrated to other works. For example if I use a different camera-radar backbone in future experiments, I can leave the noise recognition module untouched, as it is independent, and simply change the switching thresholds, defined later on (see Chapter 6). It also makes my noise recognition useful for other applications such as de-noising pipelines, informing them on the amount of degradation to remove.

I create two lightweights noise recognition models, one for each sensors. To increase generalization, the data is shuffled at each training epoch. Both networks are trained using the Adam optimizer, cross-entropy loss function, and a dropout probability of 20% to reduce overfitting. Training and testing was done on an NVIDIA GeForce RTX 3060.

Camera Noise Recognition Model:

Using convolutional networks for the images seemed like an immediate and appropriate choice. My first model was a very simple three-layer ConvNet, that already showed promising results after twenty training epochs. The final architecture is a feature extracting model, with a first half composed of down-sampling Conv2D layers and increasingly larger channel dimensions, and a second half that further down-samples the images but also decreases the channel dimensionality. My Conv2D layers have 3×3 kernels, and are all followed by ReLU activation layers and 2×2 MaxPool down-sampling. In the second half of the network, reducing the output size is done to maintain a small

amount of parameters of the final dense activation layer. Figure 3.6 illustrates the architecture of this model.

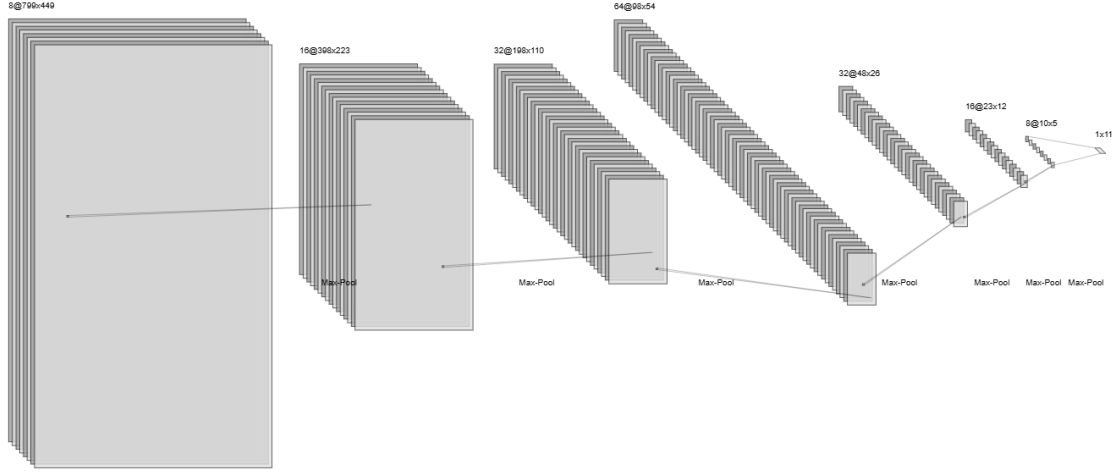


Figure 3.6: Camera noise recognition model architecture

Radar Noise Recognition Model:

The most challenging point I encountered with analyzing radar point clouds is their varying input size. All point clouds are of size $(N \times 18)$, with N the number of points, ranging between 1 and 100. This causes issues with neural networks as they expect a fixed-sized input data. To work around this issue, the options are limited.

I can either:

- Pad the input data to a fixed size. The problem with this method is padding changes the physical structure of the point cloud.
- Process the point cloud point-by-point, removing the input shape issue but losing spatial information.
- Encode the data in a different representation. Some methods, such as Svenningsson et al. (2021), represent the radar data in a graph, while others represent it

using custom “kernels” or even convert the cloud in an image, which may lead to information loss (Thomas et al. 2024; Pan et al. 2024). In alternative, Bang et al. (2024) encode the data in a “pillar” representation in RadarDistill. Due to time constraints and the complexity of encoding all 18 channels of the point cloud, these approaches were not tested, though I recognize they may be the most promising avenue for future work.

The architecture used in my final tests is the point-by-point model, illustrated in Figure 3.7. My network’s architecture is composed of three Conv1D layers with a kernel size of 1 (processing a single point) and of increasing channel dimension, all followed by a ReLU activation layer. Additional tests revealed a deeper network was not beneficial. Finally, the class prediction is done after a succession of three funneling dense layers. The loss of spatial information is the most probable reason for the low performance of this model. However, it is important to note this is the very first radar model to attempt radar point cloud noise recognition, and will serve as a baseline for future work.

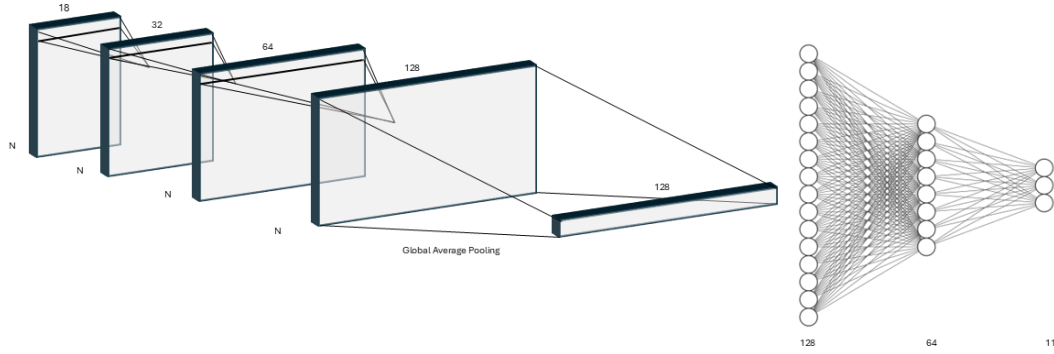


Figure 3.7: Radar noise recognition model architecture

While I initially wanted to avoid padding the data, the low performance of the point-by-point analysis lead me to explore this avenue anyway, hoping for improved performance. The architecture used in that case is closer to the camera noise recognizer, with a first half of three Conv2D layers each followed by ReLU activation and Max-Pooling down-sampling, and a second half of two layers further down-sampling while decreasing channel dimensionality. However, this model still did not show improvement.

3.2.3 Noise Recognition Results

In Figure 3.8, I present the confusion matrix of the camera noise recognition model. In a confusion matrix, the predictions of a neural network are compared with the true labels. The diagonal elements represent the number of correct predictions for each

class, while the off-diagonal elements indicate the misclassifications, showing which true labels were incorrectly predicted as other classes. Here the results are showing the model is learning the noise structure after only 20 training epochs.

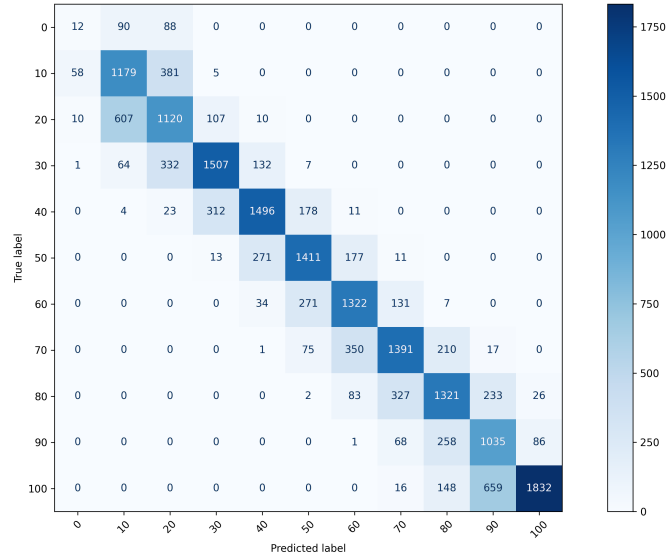


Figure 3.8: Camera noise recognition model confusion matrix on the synthetic data *test* split

In Figure 3.9, I present the confusion matrix of the radar noise recognition model. After 250 epochs, my model still struggles to classify the categories correctly, often guessing, with most predictions being either 0% or 100%. However, longer training showed a potential of overfitting on the data, even though it was shuffled at each epoch, which could indicate the task *can* be learned, but would require further experimentation with different architecture, training processes, or hyper-parameters values.

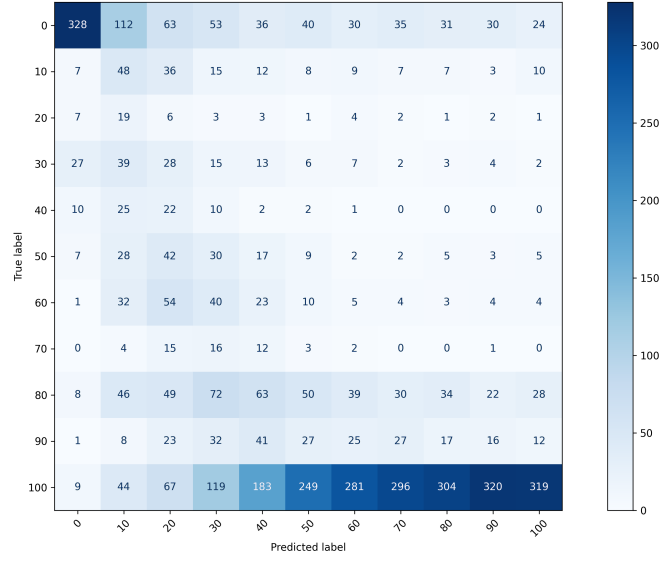


Figure 3.9: Radar noise recognition model confusion matrix on the synthetic data *test* split

Finally, Table 3.2 presents the noise recognition performance for both modules, across 11 categories. The camera noise recognition shows good results with only a few training epochs on my test dataset. In both cases, the inference time is relatively low, meeting my requirements of not impeding on the overall tracking inference speed.

Model	Training Epochs	Train Accuracy (final)	TP	FN	Test Accuracy (%)	Inference Time (sec/sample)
CameraNR	20	73.96	13626	5895	69.8	0.017
RadarNR	250	21.15	782	3673	17.6	0.005

Table 3.2: Noise recognition models results on *training* and *test* data splits

Chapter 4

VelACaR-MOT: Velocity-Aware Camera-Radar Multi-Objects Tracking Pipeline

My main objective is to use my noise recognition modules to evaluate the degradation in the camera and radar data, allowing switching between multiple backbones to increase tracking performance under sensor failure. In the previous chapter, we saw how I simulate sensor failures with a data degradation realistic synthesizer, and how I designed neural networks to recognize the simulated noise levels. In the following chapters, we will focus on the tracking pipeline, starting with its core aspects in this chapter, and then focusing on additional modules in the next one.

4.1 Core Concept

The idea behind this first tracking pipeline was to take advantage of the radar’s unique ability to accurately measure velocities, to improve on the prediction step of a Kalman filter (Kalman 1960). In practice, I take the estimated velocity from a Camera-Radar detection module and explicitly incorporate it to a Tracking-by-Detection pipeline, building on the AB3DMOT baseline method, from Weng et al. (2020).

The reason I chose Tracking-by-Detection (TBD) instead of more complex End-to-End Machine-Learning (ML) tracking methods is explicitly for the ability of TBD

methods to adapt to other backbones, datasets, and object categories. In contrast, ML-based tracking methods require extensive and computationally demanding retraining to adapt to such changes. Additionally, training a neural network for tracking requires a temporally ordered dataset, whereas TBD can leverage a detector trained on large, available, and unordered datasets. Furthermore, increased robustness to domain shifts (e.g., different environments or sensor configurations) in TBD methods can be achieved simply by using a more robust detection backbone, which is significantly faster to train than a complete tracking network. Moreover, TBD offers greater interpretability, facilitating modular improvements and quicker error analysis, whereas ML-based tracking often functions as a black box, making it harder to diagnose failures and to improve upon. As a result, guaranteeing robustness and stability is generally more challenging for ML-based methods than for TBD approaches.

Finally, the top-performing trackers on the nuScenes leaderboard are, in fact, all relying on various TBD methods, underlining their relevance (Wang et al. 2024; Li et al. 2024, 2023).

To achieve detection, I started by working with a simple camera-radar detection module, Radiant, from Long et al. (2023), before moving to a more recent and complex detection algorithm, CRN, from Kim et al. (2023). My tracking pipeline is adapted from AB3DMOT’s, reducing its dependence on the detection backbone and creating a plug-and-play application. In addition to a velocity awareness module, this lead to my Velocity-Aware Camera-Radar 3D Multi-Object Tracking (VelACaR-MOT) pipeline.

4.2 Detection Backbones

4.2.1 Radiant

The first detection method I used was Radiant, from Long et al. (2023). An older method, but its simplicity and ease of use provided a solid foundation to build the rest of the pipeline on, and to compare my later results.

Radiant works by extracting camera and radar features independently using a classic structure of Resnet, from He et al. (2016), followed by an FPN as described by Lin et al. (2017), finally feeding into multiple heads responsible for different predictions. The feature extraction of the camera data is largely inspired by FCOS3D from Wang et al. (2021), predicting objects position, size, orientation, velocity, and attribute. The camera features extracted by Resnet are concatenated with the radar features on three convolution levels before feeding to the radar FPN-to-heads structure, which does not predict the boxes sizes due to the radar point-cloud’s sparsity. Radiant then trains a Deep Weight Net to attribute a distance-driven weight to each branch, considering camera-estimated depth may be better than radar estimations in some cases.

4.2.2 CRN

The second detection backbone I chose to use is CRN, from Kim et al. (2023), a SOTA method at the time. CRN proposes a much more complex model that uses all five camera angles simultaneously. It feeds the data into a radar-assisted view transformation, followed by a Bird-Eye-View (BEV) pooling layer. Additionally, it leverages attention mechanisms to aggregate features and output detections.

The radar-assisted view transformation uses the same architecture as Radiant to extract image features: a Resnet followed by an FPN. Those features are then passed

through an additional convolution layer to extract a depth distribution and image context. The radar data is processed using Point-Net, from Qi et al. (2017), followed by a convolution layer to extract radar occupancy and context features. The depth distribution, image context, and radar occupancy are then all concatenated into a Frustum-viewed feature-map, before being transformed into a BEV feature map using the radar context features. The final step consists of a multi-modal deformable cross attention layer that outputs the bounding boxes (Zhu et al. 2021).

In both backbones, detections are represented with a set of bounding boxes with their global coordinate position, their size, rotation (in quaternions), velocity, name, confidence score, and attribute (e.g., a bicycle *with a rider*), stored in a single .json file, following nuScenes’ detection output format. One important remark is that detection evaluation is made on keyframes only (i.e., samples, as defined in Chapter 2.4), excluding sweeps. This means all detection methods will output a detection file for keyframes only, sampled at 2 samples per seconds.

4.3 Velocity Aware Tracking Pipeline

Following AB3DMOT, and for the reasons previously enunciated in Section 4.1, I built a Kalman filter-based Tracking-by-Detection pipeline. The Kalman filtering method consists in a series of steps: Prediction, Matching, and Update. Note that to increase performance, and following current SOTA methods, the object categories are processed separately.

4.3.1 Prediction

At a time t , the tracking pipeline takes all the objects in memory (if any) and predicts their future positions following an object motion model. In my method, I use a constant velocity model, which has been shown to be effective by Weng et al. (2020):

$$\begin{aligned}x' &= x + \Delta t \cdot v_x, \\y' &= y + \Delta t \cdot v_y, \\z' &= z + \Delta t \cdot v_z,\end{aligned}\tag{4.1}$$

where (x, y, z) are the position of the object, (v_x, v_y, v_z) its velocity, and δt the sampling period. As ground vehicles do not translate on the z axis (up), I consider $v_z = 0$ m/s. v_z is still included to simplify the state transition and measurement matrices.

As mentioned before, nuScenes imposes the output of detection methods to be for samples only, leading to a sampling period of $\Delta t = \frac{1}{2FPS} = 0.5$ seconds per frames. I accept this limitation as it aligns with the plug-and-play nature of my pipeline, which would be compromised if modifications to the detection methods were required. Although this results in a performance tradeoff, it also enhances the execution speed of my method and lowers its computational demands, aligning with the real-world applicability of my pipeline.

To account for all essential tracking information, the Kalman filter state includes the position (x, y, z) , orientation (θ) , size (l, w, h) , and velocity (v_x, v_y, v_z) of the objects. The resulting state is:

$$X = [x, y, z, \theta, l, w, h, v_x, v_y, v_z]^T.\tag{4.2}$$

As the nuScenes dataset imposes the orientation of the bounding boxes to be in quaternions, I convert it to a yaw angle before tracking, following Weng et al. (2020), to simplify calculations. Considering standard quaternion notations ($q = w + ix + jy + kz$), the applied transformation is:

$$\theta = \text{atan2}(2xy + 2zw, 1 - 2y^2 - 2z^2). \quad (4.3)$$

In this section, a small innovation of mine lies in using the high-accuracy radar-measured velocity to increase prediction accuracy. Previous methods, such as Weng et al. (2020), let the Kalman filter’s equation estimate the velocity of tracked objects. In our case, since the detection backbone has access to the Doppler-measured velocity of the radar, I add the velocity of the bounding boxes in the measurement matrix of the filter, which constitutes the velocity-awareness of my tracking pipeline. To the best of my knowledge, other camera-radar tracking-by-detection pipelines focus on using the object velocities for matching purposes, but do not explicitly use the velocity in the prediction step as I do (Zhou et al. 2020; Lee and Ang 2024; Kim et al. 2023).

At a time t , the predictions follow standard Kalman filtering methods:

$$\hat{X}_{t|t-1} = F\hat{X}_{t-1|t-1}, \quad (4.4)$$

$$P_{t|t-1} = FP_{t-1|t-1}F^T + Q, \quad (4.5)$$

where $\hat{X}_{t|t-1}$ is the estimated state at time t based on the previous state of time $t - 1$. $\hat{X}_{t-1|t-1}$ is the updated state estimate from time $t - 1$. P is the state estimate covariance matrix, representing uncertainty in the estimation of the new state. Q is the process noise covariance matrix, representing uncertainty in the transition model. F is the transition matrix from which we predict the new state (i.e., the constant velocity

model in our case). P and Q are initialized as custom covariance matrices from Chiu et al. (2021).

4.3.2 Data Matching

Following the prediction step, I match the detection from the backbone with predicted objects. This is achieved by computing a similarity matrix between all detections and predictions, using various similarity metrics that consider the unique characteristics of each object.

The metrics used include:

- 3D Euclidean distance: $d(box_a, box_b) = \sqrt{(x_a - x_b)^2 + (y_a - y_b)^2 + (z_a - z_b)^2}$, computed from the center of the 3D bounding boxes.
- 2D Euclidean distance: $d(box_a, box_b) = \sqrt{(x_a - x_b)^2 + (y_a - y_b)^2}$, computed from the center of the Bird-Eye-View projection of the bounding boxes (i.e., the bottom of the boxes).
- 2D Intersection over Union (IoU): $IoU_{2D} = \frac{B_a \cap B_b}{B_a \cup B_b}$, with B_a and B_b the areas of the bottom of the bounding boxes.
- 2D/3D Intersection over Union (IoU): $IoU_{3D} = IoU_{2D} \cdot \min(h_a, h_b)$ with h_a and h_b the heights of the bounding boxes.
- 2D/3D Generalized Intersection over Union from Rezatofghi et al. (2019):
 $IoU_{2D/3D} = \frac{C_{2D/3D} / |B_a \cup B_b|}{|C_{2D/3D}|}$, with C_{3D} the smallest convex hull that encloses the bounding boxes A and B, and C_{2D} the smallest convex hull that encloses the *bottom* of the two bounding boxes.

The association of new detections and previously memorized tracks is performed using either a greedy algorithm, or the Hungarian algorithm from Kuhn (1955).

4.3.3 Update

At the end of the data matching step, I distinguish three different possibilities:

- **Birth:** If a detection is not matched, it is considered to be a new object and is added to the memory with a new track ID.

If this detection should have been matched to an existing track, this is referred to as an Identity Switch(IDS).

- **Death:** If a track stays unmatched for a maximum time, it is considered to be from an object that left the field of view, and is therefore removed from memory. If the maximum age is set too low, it can lead to more identity switches in the case of sparse detections, as objects will be removed from memory as soon as they aren't matched. On the other hand, if the maximum age is set too high, it increases computational intensity, as these objects unnecessarily go through prediction steps.

- **Update:** If a prediction is matched to a detection, the state of the tracked object gets updated through the filter's equations from Kalman (1960):

$$\begin{aligned}
z_t &= H \cdot (F_t X_t + w_t) + v_t, \\
\hat{X}_{t|t} &= \hat{X}_{t|t-1} + K_t(z_t - H\hat{X}_{t|t-1}), \\
K_t &= P_{t|t-1}H^T(HP_{t|t-1}H^T + R)^{-1}, \\
P_{t|t} &= (I - K_tH)P_{t|t-1},
\end{aligned} \tag{4.6}$$

with w_t and v_t the process noise and the observation noise, respectively, both following a zero-mean multivariate normal distribution of covariances Q for w_t and R for v_t . R is the measurement noise covariance matrix, representing uncertainty and errors in the

measurements, and H is the measurement matrix, representing which part of the state are updated through matching and which are inferred on by the filter.

Finally, the update step includes a correction of the yaw angle θ , assuring the angles of predicted bounding boxes and those of detected bounding boxes are within $[-\pi, \pi]$, and that they form an acute angle, taking in account potential errors in the estimation of the orientation of detected objects.

Figure 4.1 illustrates my tracking pipeline across three samples. The tracking steps described in this chapter are visible in the middle of the diagram, under the blue “Kalman Filter” and “Data association” boxes. Furthermore, the left side of the diagram encompasses fusion of the camera and radar data and the output of the detection backbone in form of bounding boxes. At each timestep, the tracking pipeline receives new detections, predicts the new positions of previously memorized tracks, represented on the right side of diagram, matches the detections and the predicted tracks, updates its state and memory, and outputs the updated trajectories on the right. To track an object implies associating the objects of a given sample with those in memory, which is done by assigning them ID numbers, here represented by different colors. A consistent tracking, that shows no fragmentation of the tracks nor identity switches, is underlined by the absence of change in the colors of bounding boxes. We can make sure in the diagram that the colors of each bounding boxes on the tracking side stays the same, indicating good tracking.

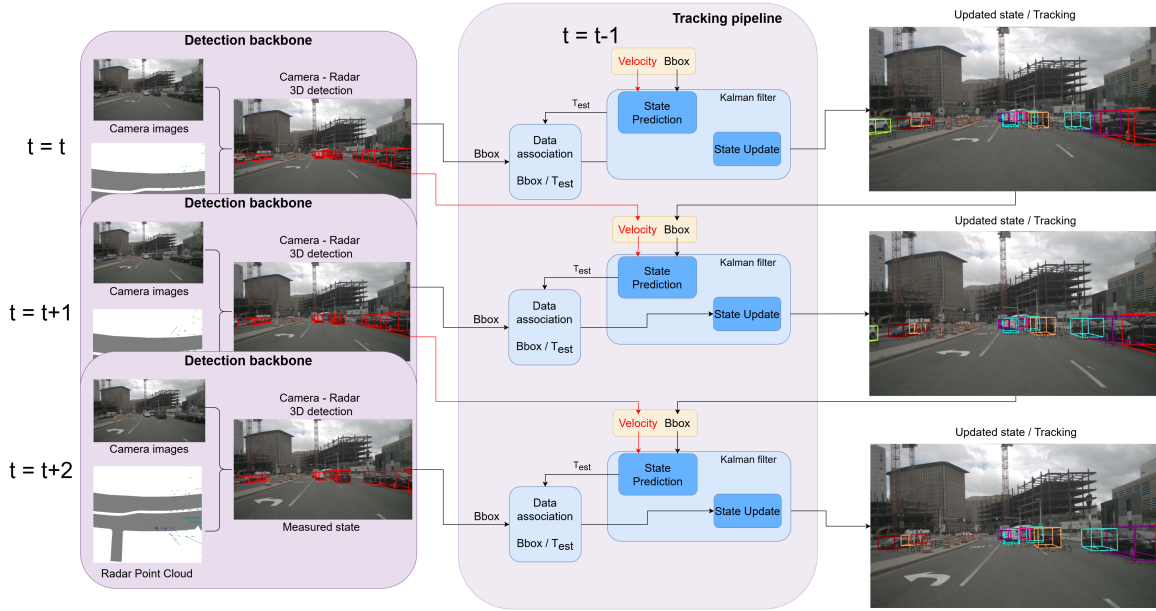


Figure 4.1: Overview of the VelACaR multi-objects tracking pipeline

4.4 Initial Results

I evaluate the performance of the VelACaR tracking pipeline on the nuScenes dataset, presented in Chapter 2.4. Among the metrics provided with the dataset, we will focus on the Average Multi-Object Tracking Accuracy (AMOTA) and Precision (AMOTP) metrics, as described in the nuScenes paper and website (Caesar et al. 2019, 2020):

$$AMOTA = \frac{1}{n-1} \cdot \sum_{r \in \{\frac{1}{n-1}, \frac{2}{n-1}, \dots, 1\}} \max(0, 1 - \frac{IDS_r + FP_r + FN_r - (1-r) \cdot P}{r \cdot P}),$$

$$AMOTP = \frac{1}{n-1} \cdot \sum_{r \in \{\frac{1}{n-1}, \frac{2}{n-1}, \dots, 1\}} \frac{\sum_{i,t} d_{i,t}}{\sum_t TP_t},$$
(4.7)

where $n = 40$ is the number of points in the n-point interpolation of the MOTA and MOTP curves, as described in Caesar et al. (2019). r represent multiple recall thresholds, and P is the number of ground truth objects. IDS_r , FP_r , and FN_r are

the number of identity switches, false positives, and false negatives, respectively, at the recall threshold r . $d_{i,t}$ is the position error of track i at time t , TP_t is the number of true positive at time t . For detailed definitions of AMOTA and AMOTP, I refer the reader to Caesar et al. (2019).

This results section consists first in a comparative analysis of my method’s average tracking performance with other camera-radar tracking methods, on the nuScenes test dataset. Secondly, I conduct an ablation study on the effect of the velocity awareness compared to the effect of using a custom observation noise covariance matrix, from Chiu et al. (2021). The reason I chose to compare the velocity awareness with the observation noise covariance matrix is that they are both linked to the observations used to update the model. Therefore, we can study the impact of an additional observation (i.e., the velocity awareness) compared to a better intrinsic knowledge of the observations (i.e., a dataset-custom noise covariance matrix).

Note that to obtain results from the test dataset, the tracking output JSON file must be sent to the nuScenes servers, where the annotations are stored and remain inaccessible to the public. Since the number of attempts is limited, only the best configuration was tested on this split, and the ablation studies are conducted on the validation dataset.

Table 4.1 shows VelACaR’s best tracking performance compared to SOTA Camera-Radar(CR) methods on the nuScenes test dataset. I compare my method with the highest ranking CR method HyDRa, from Wolters et al. (2024), and the tracking performance reported by CRN, both following CenterPoint, as described in Yin et al. (2021), and converting the detection into a Bird-Eye-View (BEV) representation, using velocity-based tracking-by-detection techniques. I also compare my results with the

ML-based method (Baumann et al. 2024), that uses a LSTM networks to learn the motion models, and notice tracking performance is higher with TBD methods.

Name	Tracking Method	AMOTA $\uparrow$	AMOTP $\downarrow$	FP $\downarrow$	FN $\downarrow$	IDS $\downarrow$
CR3DT	ML	0.306	1.339	12666	56082	3308
HyDRa	non-ML	0.584	0.950	13996	32950	791
CRN	non-ML	0.569	0.809	16822	41093	946
CRN-VelACaR (proposed)	non-ML	0.325	1.229	14205	52662	1668

Table 4.1: Tracking results of VelACaR on the nuScenes *test* dataset, using CRN’s detections, compared to other tracking methods

The tracking results remain below those of CRN and HyDRa, indicating that additional refinement is needed in this part of the pipeline. An illustration of the tracking between three frames is in Figure 4.2.

In Table 4.2, I show the results of my study comparing the effect of the velocity awareness and a custom noise covariance matrix, on the nuScenes validation dataset. Concerning the noise covariance matrix (R), I either use a default identity matrix, or use the tuned parameters from Chiu et al. (2021), enhanced with three covariances

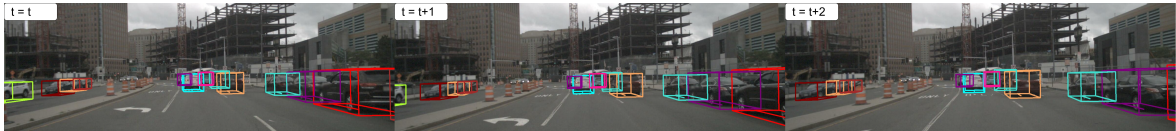


Figure 4.2: Illustration of the VelACaR tracking pipeline’s results with the CRN backbone

Tracking implies assigning ID numbers to bounding boxes, represented here by colors using a fixed table. Consistent tracking means a given object keeps the same ID number.

Parameters		Radiant		CRN	
R	Vel	AMOTA↑	AMOTP↓	AMOTA↑	AMOTP↓
×	×	0.244	1.530	0.286	1.359
×	✓	0.220	1.553	0.358	1.281
✓	×	0.284	1.479	0.355	1.273
✓	✓	0.238	1.525	0.360	1.216

Table 4.2: Ablation study on the effects of velocity awareness and tuned Noise Covariance Matrix (R) on the nuScenes *validation* dataset

Here the default covariance matrix is the Identity matrix and the velocity is estimated by the Kalman filter when not observed.

corresponding to the measured velocities (v_x, v_y, v_z) , set at 0.1 km/h, following the ARS-408-21 datasheet (see Appendix A.2).

The results show a 7.2% improvement in AMOTA on this dataset from the velocity awareness with CRN as backbone. Likewise, the AMOTA improvement made by using a custom noise covariance matrix is similar, at 6.9%. This may indicate that my noise awareness can effectively compensate for a lack of knowledge of the dataset, as would be the case in real-world deployments. When sufficient prior knowledge allows for the construction of custom noise covariance matrices, my velocity module still contributes a modest 0.5% accuracy improvement.

Regarding tracking precision (AMOTP), my velocity awareness module shows a 7.8 cm improvement compared to the baseline, while combining it with the custom matrix yields a 14.3 improvement in precision compared to the baseline (without R and without the velocity awareness), on the nuScenes validation dataset.

Interestingly, while the tracking performance of Radiant-VelACaR are improved when using the custom observation matrix, the velocity awareness has the opposite effect. This indicates that my pipeline benefits largely from better association of radar points with bounding boxes, and thus from better velocity estimation. In fact, if the velocity estimation error is high enough, velocity awareness will lead to wrong predictions and an increase of the Kalman filter’s uncertainty. This is underlined

by Radiant’s velocity error being 3.6 times higher than CRN’s, with 1.35 m/s mean Average Velocity Error compared to CRN’s 0.365 m/s (from nuScenes leaderboard on the test dataset). Due to this, the dataset-tuned noise covariance matrix from Chiu et al. (2021) will lead to improvement because it provides accurate information on the dataset, while the velocity measurement will not because it is not accurate enough.

Chapter 5

Extended VelACaR

To enhance the performance of the tracking pipeline, I integrated current data association SOTA methods, including double-round matching and position matching, following the approaches used by Lee and Ang (2024) in Cara-MOT and by Li et al. (2023) in Poly-MOT. Additionally, I sought to improve performance by using a more complex motion model. The Singer model, from Singer (1970), and mentioned in Shi et al. (2024), was a promising alternative as it is designed to take in account movement stochasticity. To further refine tracking, I designed an additional preprocessing module to remove clusters of objects and mitigate single-object multi-detections (i.e., when multiple detection bounding boxes overlap over a single object, with similar confidence score). Figure 5.1 illustrates the tracking pipeline before these modifications, while Figure 5.2 illustrates the added modules and the changes made to the pipeline.

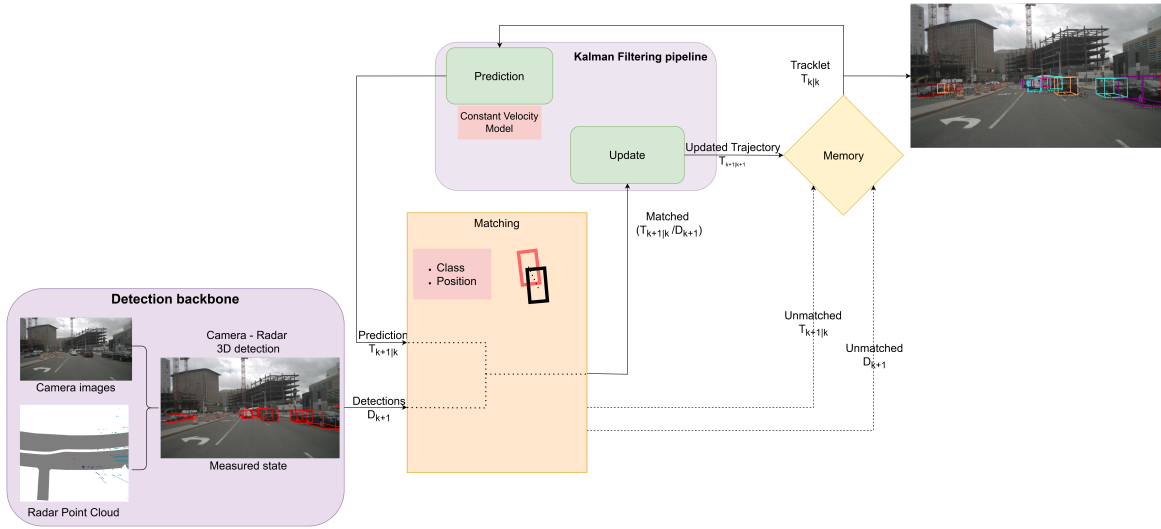


Figure 5.1: Diagram of the tracking pipeline **before** module improvements

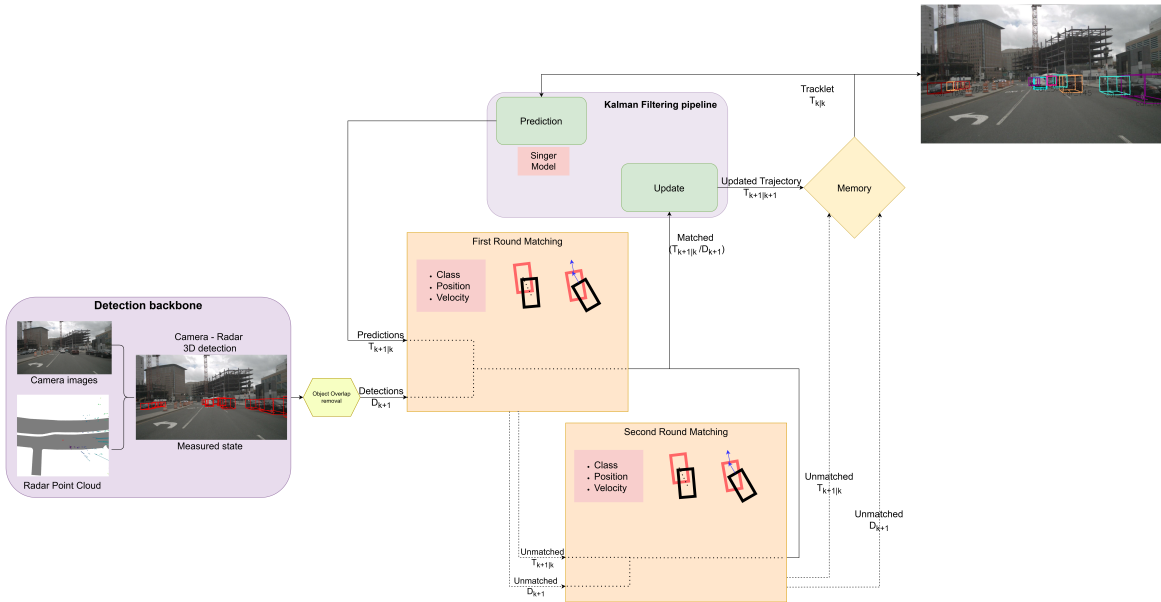


Figure 5.2: Diagram of the tracking pipeline **after** module improvements
The most visible addition is the second round of matching (at the bottom of the diagram), which takes place with the unmatched detections and tracks after the first round of matching. The velocity matching is represented in both matching rounds by a mention in writing and a pictogram of two bounding boxes and their velocity vector. The Singer model is mentioned under the prediction step of the Kalman filter to denote the additional motion model. The cluster removal module is visible after the detections, in yellow.

5.1 Singer Model

In my previous method, I modeled the motion of objects using a constant velocity model (see Eq. 4.1). Given our low sampling frequency of 2Hz, a more advanced model could theoretically improve tracking accuracy. The Singer model appeared to be a promising approach, as it accounts for both acceleration and turn rate dynamics. In tracking tasks, pedestrians pose the greatest challenge due to their narrow bounding boxes, which can rotate rapidly while moving at slower speeds, unlike vehicles, which have larger boxes, higher translation speeds, and slower turn rates. In consequence, given its specificity, the Singer model seemed particularly well-suited to improving pedestrian tracking accuracy.

The transition model for each axis direction (i.e., x, y, z) is defined by Singer (1970) as follows:

$$\phi(T, \alpha) = \begin{bmatrix} 1 & T & \frac{1}{\alpha^2}[-1 + \alpha T + e^{-\alpha T}] \\ 0 & 1 & \frac{1}{\alpha}[1 - e^{-\alpha T}] \\ 0 & 0 & e^{-\alpha T} \end{bmatrix}, \quad (5.1)$$

with $T = 1/2$ the sampling period and α the reciprocal of the time constant of the target maneuver, which represents the time over which the acceleration significantly changes. A large α (e.g., $\alpha = \frac{1}{20}$) indicates rapid acceleration changes, while a small one (e.g., $\alpha = \frac{1}{60}$) indicates slow acceleration changes and smooth maneuvers. For example, pedestrians may be represented with a higher α than cars, since they can rotate faster. We can note that if αT is very small, the Taylor expansion of $e^{-\alpha T} = 1 - \alpha T + \frac{(\alpha T)^2}{2}$ simplifies $\phi(T, \alpha)$ to approximately the Newtonian Matrix (i.e., the

constant acceleration model).

For a target defined in three dimensions, the state transition matrix becomes:

$$X = I_{3 \times 3} \otimes \phi(T, \alpha), \quad (5.2)$$

with $\otimes$ the Kronecker product, as described in Singer (1970).

The Singer model also changes the Process Noise Covariance matrix to:

$$Q = 2\alpha\sigma^2 \begin{bmatrix} T^5/20 & T^4/8 & T^3/6 \\ T^4/8 & T^3/3 & T^2/2 \\ T^3/6 & T^2/2 & T \end{bmatrix}, \quad (5.3)$$

with σ the standard deviation of the acceleration noise. Note that this matrix corresponds to the simplified version of Q from the Singer model, as described in the original paper.

5.2 Position and Velocity Matching

In Chapter 4, I used the position of objects as a metric to match predicted tracks with detections. The first step in position matching is to build a similarity matrix based on the objects' positions and orientations (if using IoU-type metrics). Then, I used a Hungarian algorithm, described in Kuhn (1955), to match predictions with detections. Since I consider CRN's estimation of the velocity to be accurate, I implement an additional velocity-matching module in the data-matching step of the Kalman filter. In practice, position-matched tracks and detections are re-matched based on their

velocity, using an additional affinity matrix built from the cosine similarity of their velocities:

$$S(v_A, v_B) = \frac{v_A \cdot v_B}{\|v_A\| \cdot \|v_B\|}. \quad (5.4)$$

Note: If both objects are static ($\|v_A\| = \|v_B\| = 0$), or barely moving, the similarity is set to 1. If either object is moving while the other is static, similarity is set to the minimum value (-1).

This additional metric helps identify False Positive matches, considering objects with matching positions but significantly different velocity vectors are most likely different. In position matching, I had a similarity threshold under which objects are automatically discarded, to avoid always having $N_{matched} = \min(N_{predictions}, N_{detections})$. Similarly for velocity matching, I apply a minimum similarity threshold, though it is usually set relatively low to avoid discarding valid matches. Naturally, if this tunable parameter is set to -1, the lower bound of cosine similarity, the velocity matching module is effectively disabled, as all position-matched couples will automatically be matched velocity-wise.

5.3 Double Round Matching

Current SOTA methods also improve the matching step with double-round matching (Lee and Ang 2024; Li et al. 2023). This module takes the detections and tracks discarded in the first round of matching and rematches them based on another metric. The purpose here is to reduce the number of False Negative matches.

The **first round** usually consists of high-confidence matching using high affinity thresholds, prioritizing strong matches with low uncertainty. In our case, the metric for the first round is a tunable parameter.

In the **second round**, previously unmatched detection and predictions are re-matched, using a 2D Generalized Intersection over Union (GIoU) metric in our case (see Chapter 4). This allows for more relaxed thresholds with a different matching criterion. This helps in recovering tracks that would have been otherwise lost.

In both rounds, setting the tunable similarity threshold to the upper bound of the used metric effectively deactivates the module, either forcing every track and detection to go through second-round matching, or preventing any match from being made in the second round.

5.4 Additional Modules

Chebyshev distance: In addition to the larger changes, I added the L_∞ norm, also called Chebyshev distance, as a potential position-matching metric:

$$d(x_1, x_2) = \|(x_1, x_2)\|_\infty = \max(|x_1|, |x_2|), \quad (5.5)$$

where x_1 and x_2 are the centers of two bounding boxes.

De-clustering method: I also devised a simple detection de-clustering method. As I noticed some objects were detected multiple times by CRN, with similar confidence scores, I used my matching modules to compute an affinity matrix between the detections. Then, the idea was to look for high matches that would indicate multiple bounding boxes in the same vicinity, suggesting single-object multi-detections. In that case, I could keep the detection with the highest score, and discard the other ones.

Inference speed improvements: Changes in data loading from the first method described in Chapter 4 lead to a substantial increase in inference speed, from 0.28

seconds per sample (3.5 fps) to 0.071 seconds per sample (14 fps).

Matching algorithm: Finally, while I previously used either a greedy or a Hungarian matching algorithm to perform data association, (see Section 4.3.2), I now exclusively use the latter, as it guarantees optimal results. Further experiments also no longer involve Radiant, as CRN largely outperforms it.

5.5 Hyperparameters Tuning

Adding the aforementioned changes to the tracking pipeline resulted in a large number of tunable hyperparameters. The resulting parameters are as follows:

- **Detection threshold:** To remove low-confidence detection. This threshold struck a balance between high-confidence false positives and low-confidence true positives.
- **Singer model parameters:** α and σ , as described in Section 5.1.
- **Single-round matching parameters:**
 - Matching metrics (2D/3D IoU, 2D/3D Euclidean distance, 2D/3D Euler distance, 2D/3D Chebyshev distance, 2D/3D GIoU).
 - Position affinity threshold.
 - Minimum hits.
 - Maximum age.
- **Velocity matching parameter:** Velocity affinity threshold.
- **Double-round matching parameters:** Additional position and velocity thresholds.

Detection confidence score threshold:

I calculate the optimal detection threshold using a weighted score selection based on the following evaluation criteria: recall, precision, translation error, velocity error, orientation error, and attribute error. Note that these variables are all normalized between $[0, 1]$.

Upon evaluating a detection method, nuScenes also provides performance metrics across 100 recall values and four distance threshold, which I use to tune the confidence thresholds for each object. Obviously, picking a low recall value gives the highest detection performance, as then only objects detected with high confidence are evaluated, but it also leads to many missed tracks. Conversely, picking a higher recall value yields more detected objects, but a too high recall threshold will lead to many being false positives. Therefore, we want to find a balance between high-recall values (i.e., detected most objects in the scene) and high performance (i.e., high-confidence detections).

For each object and at each distance threshold, I create a weight vector w , assigned to: $1 - r$ (with r the recall value), precision, translation error, velocity error, orientation error, and attribute error, respectively. The components of w are determined based on the importance of each metric for the tracking pipeline. Recall, precision, and translation error are the most important parameters for a tracking-by-detection pipeline. As we saw in previous results (see Table 4.2), good velocity estimation is important for the velocity awareness, so a higher weight is attributed to velocity error compared to the remaining parameters. In Chapter 4, I brute-forced the detection thresholds, which I used here as a comparison basis to set the weights attributed to orientation error and attribute error. I also increase the weight attributed to precision on the pedestrian category, as the shape of their bounding box makes it significantly more important.

Finally, I calculate the weighted sum of each row, resulting in a (100×1) matrix. I select the highest value, and extract its corresponding confidence score. Once averaged across all four distance metrics, I obtain the category’s final detection confidence score threshold.

Inter-dependent hyperparameter:

Unfortunately, the other hyper-parameters cannot be processed separately as they are all inter-dependent, resulting in a 7-dimensional optimization problem. Since we lack prior knowledge of the relationship between these parameters and cannot afford to settle for suboptimal solutions, I first tackle the problem using a brute-force approach. To evaluate a given set of parameters, I run the tracking pipeline with them and use the AMOTA (i.e., tracking accuracy, see Eq 4.7) value as an evaluation metric.

Since exhaustive search is very time-consuming, I reduce the search space within predefined ranges and step sizes. For instance, even though the bounds of GIoU metrics are $[-1, 1]$, matching affinity between a prediction and a detection is rarely above 0.2. Setting a threshold above this value would remove almost all matches, so testing them is unnecessary. Similarly, trying a minimum distance loss of less than 5 meters makes little sense, as true matches are never that far apart. To further simplify the problem, I assume that the performance function with respect to each individual parameter is concave. In other words, when all other parameters are held constant, the AMOTA score is expected to vary in a concave manner as the value of the selected parameter increases. Since exhaustive search is done using nested loops, only the innermost parameter changes at each step. This allows me to design a sliding window that measures when performance stop improving (i.e., when the AMOTA decreases as a single parameter increases) for each combination of fixed values of the other $N - 1$ parameters. If results do not improve over a predefined number of steps, I prematurely

stop the optimization of the innermost parameter N , increment the $N - 1$ parameter, and restart the optimization of the innermost parameter again. Finally, since objects exhibit different motion patterns and intrinsic properties, these thresholds need to be set individually for each category.

Due to the large number of parameters and the intensive nature of the described optimization method, I was unable to tune the parameters on the full 150-scenes (≈ 6000 samples) validation dataset. Instead, I perform the tuning on the smaller 4-scenes (≈ 160 samples) mini-val dataset. While this approach works well for categories with a large number of samples, such as pedestrians and cars, less-represented categories may suffer from sub-optimal parameters. This requires lengthy manual re-tuning with no guarantee of finding a better combination.

Note: More efficient optimization strategies were later developed and are described in Chapter 5. Given my timeline, exhaustive search was initially acceptable, as it was quick to implement and allowed me to focus on other parts of the project. Additionally, it enabled me to make key observations and draw important conclusions that informed the design of later methods. The parameters used in the remainder of this section were obtained using the aforementioned brute-force optimization approach.

5.6 Tracking Performance and Ablation Studies

I evaluate the improvements made by the additional modules on the nuScenes mini-dataset of 10 scenes, on which parameter optimization is done, as well as on the 150-scenes validation dataset. Since I only conduct my analysis on these two datasets, it

is important to note they are average results and while they give us an indication of general trends, they may not be statistically significant. Further statistical analysis on the effect of each module is left for further works.

5.6.1 Using a Constant Velocity Model

I begin by testing the base pipeline, CRN-VelACaR (i.e., VelACaR using the camera-radar CRN backbone), on the mini and validation datasets, as shown in Table 5.1, containing the tuned parameters and corresponding tracking results. We can immediately notice a large difference in performance between the two datasets, with a 19.5% gap in AMOTA. This could be explained by the difference in datasets size, with 685 annotated objects in the mini dataset against 14 556 in the validation dataset. This may cause certain categories to have promising tracking performance in the first one, but decreasing performance on more objects. This is the case with the trailer category, that goes from a 68% accuracy on 41 objects down to 0% on 2425, showing high penalization by nuScenes' evaluation functions.

Figure 5.3 illustrates potential issues with the CRN backbone on the trailer category. In the observed scene, what seems to be a large container is being detected over ten times by CRN, placing trailer bounding boxes with various orientations on a background object. These detections are made with a large-enough confidence score, above the tuned detection threshold, and are thus fed into the Kalman filter. Furthermore, since they appear for multiple frames, there is no way so filter them out with a minimum hits parameter. In the end, those artifacts will constitute a lot of false positives, reducing the AMOTA, and their proximity will also lead to identity switches

	Parameters				Performance Mini Dataset		Performance Validation Dataset	
Category	Metric	Position Threshold	Min Hits	Max Age	AMOTA $\uparrow$	AMOTP $\downarrow$	AMOTA $\uparrow$	AMOTP $\downarrow$
Car	2D GIoU	-0.2	1	2	0.641	0.800	0.607	0.804
Pedestrian	2D GIoU	-0.6	1	3	0.486	1.241	0.292	1.536
Truck	3D L_∞	-4	3	2	0.101	1.360	0.302	1.387
Trailer	3D L_∞	-4	3	1	0.680	1.704	0.000	1.490
Bus	3D GIoU	-0.7	1	3	0.828	0.845	0.515	1.005
Motorcycle	2D GIoU	-0.4	1	3	0.622	0.997	0.446	0.996
Bicycle	3D Euclidean	-3	3	1	0.318	1.189	0.361	1.294
Total	-	-	-	-	0.525	1.162	0.360	1.216

Table 5.1: Post-optimization **baseline** tracking results using constant velocity model and no additional matching modules, on the validation dataset

and track fragmentation, further reducing the AMOTA. Given how the AMOTA is calculated (see Eq 4.7), and since the trailer category is relatively under-represented, with low amounts of ground truth objects, having many false positives due to this sort of artifact can completely drive the tracking accuracy of this category to 0%.

Another probable explanation for the AMOTA difference between the mini dataset and the validation dataset is that my parameter values are not optimal. As there are few objects in the mini dataset, a wide range of possible parameters give the same results, but may not yield good results on a bigger dataset. Similarly, the parameters may be over-tuned to a small sample, not generalizing well to more objects.

This could also be simply due to the detection backbone. Indeed, CRN showed relative performance imbalance, with over 70% detection accuracy on some categories

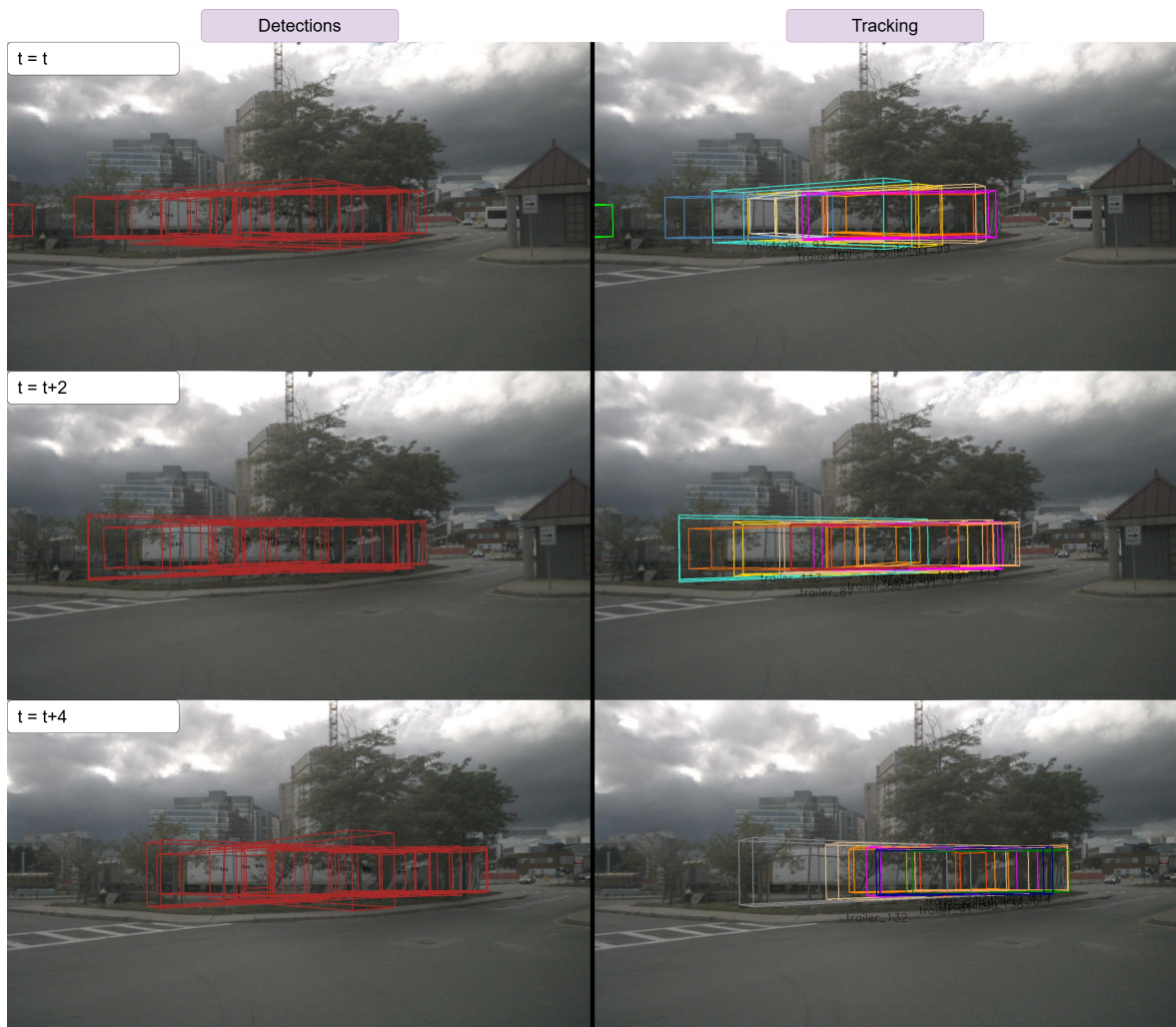


Figure 5.3: Illustration of the trailer category issue with the CRN detection method. On the left are detections from CRN on three samples that are one second apart, and on the right are the tracking-by-detection output. CRN places multiple trailer bounding boxes of various orientations on the same object, which seems to be a container. Because these detections have a high confidence score and appear throughout multiple samples, they are being tracked by the Kalman filter, and are at the origin of multiple false positives (FP) and identity switches (IDS), illustrating both the difficulties CRN has with detecting this class and the impact it can have on a tracking-by-detection pipeline. These kinds of artifacts are why I believe better detection backbones will lead to higher tracking performance.

like cars, while having a 48% detection accuracy on trailers. As detection is a common bottleneck in Tracking-by-Detection, a detection method with less performance variation may provide better and more consistent tracking results.

Finally, we can notice certain trends, with larger objects like trucks and trailer providing better results on this dataset with a Chebyshev distance matching metric, while smaller categories like pedestrians are better tracked using a GIoU. Similar to Li et al. (2023), the bicycle category stands apart, requiring a different matching metric. The selected thresholds effectively reflect the tracking difficulty for each category. Cars have a much higher threshold than pedestrians, suggesting that most true matches for cars exhibit high affinity. The selected minimum hits parameter provides insight into which categories are more prone to False Positives detections from CRN, with trucks, trailers, and bicycles showing better performance if tracking only begins after three successive detections.

In Table 5.2 are the tracking results from adding the velocity matching module. As observed, the optimal selected value for all object categories (except trailers) is the lowest bound of cosine similarity (i.e., -1), indicating that the module is effectively deactivated in these cases. For the trailer category, optimum value for accuracy was also -1 , but a small tradeoff of 0.2% in accuracy for a 65 mm gain in precision could be achieved by activating velocity-matching. Overall, this indicates that the velocity matching module does not improve tracking performance on this dataset when using CRN as a detection backbone. The results here are likely not due to the tuning test space but rather to CRN-VelACaR exhibiting relatively low levels of False Positive matches. A False Positive match would typically result in an Identity Switch, yet we

	Parameters			Performance Mini		Performance Validation	
Category	Metric	Position Threshold	Velocity Threshold	AMOTA $\uparrow$	AMOTP $\downarrow$	AMOTA $\uparrow$	AMOTP $\downarrow$
Car	2D GIoU	-0.2	-1	0.641	0.800	0.607	0.804
Pedestrian	2D GIoU	-0.6	-1	0.489	1.247	0.292	1.536
Truck	3D L_∞	-4	-1	0.101	1.360	0.302	1.387
Trailer	2D Euclidean	-3	-0.7	0.678	1.639	0.000	1.427
Bus	3D GIoU	-0.7	-1	0.828	0.845	0.515	1.005
Motorcycle	2D GIoU	-0.4	-1	0.622	0.997	0.446	0.996
Bicycle	3D Euclidean	-3	-1	0.318	1.189	0.361	1.294
Total				0.525	1.154	0.360	1.207

Table 5.2: Post-optimization **velocity matching** tracking results using a constant velocity model, on the validation dataset

observe only 1 330 IDS out of 14 556 annotated tracks in the validation dataset (9%).

The results of the final analysis conducted using the base model (i.e., a constant velocity motion model) are presented in Table 5.3, which reports the performance of the double-round matching module. Since velocity matching did not yield any improvement, it is fully deactivated in this configuration.

Overall, tracking performance is only marginally improved by this module, with a 0.1% accuracy increase on mini dataset on average and a 0.3% average increase on the validation dataset. While this is most likely due to sub-optimal parameters, we do have deactivation values in more than half of the categories, indicating the module simply does not help in those cases. Namely, the car category exhibits a clear deactivation of the first-round matching, as a 0-meter threshold for the Euclidean distance similarity metric prevents all matches. This forces all detections and predictions to be matched in the second round using the GIoU metric. We can point out that in single-round

matching, the car category also uses a GIoU affinity metric, with the same threshold that the optimization found in double-round matching. Similarly, a 0.7 IoU threshold for the motorcycles category will filter out every first-round matches. Furthermore, the trailer and bicycle categories show a clear deactivation value of the second matching round, as +1 is the upper bound of the GIoU metric.

	Parameters			Performance Mini		Performance Validation	
Category	Metric (round 1)	Position Threshold 1	Position Threshold 2	AMOTA $\uparrow$	AMOTP $\downarrow$	AMOTA $\uparrow$	AMOTP $\downarrow$
Car	2D Euclidean	0	-0.2	0.641	0.800	0.607	0.804
Pedestrian	2D Euclidean	-1	-0.7	0.490	1.244	0.293	1.535
Truck	3D L_∞	-1	0.1	0.102	1.367	0.318	1.346
Trailer	3D L_∞	-4	1	0.680	1.704	0.000	1.490
Bus	2D Euclidean	-2	-0.6	0.828	0.845	0.518	1.005
Motorcycle	2D IoU	0.7	-0.4	0.622	0.997	0.446	0.996
Bicycle	3D Euclidean	-3	1	0.318	1.189	0.361	1.294
Total				0.526	1.164	0.363	1.210

Table 5.3: Post-optimization **double-rounds matching** tracking results using a constant velocity model, on the validation dataset

Note: Here the max age of the truck category was increased to 3.

5.6.2 Using the Singer Model

The next analyses focus on evaluating the impact of the Singer model, followed by assessing how the velocity and double-round matching modules influence tracking performance when used in conjunction with it.

Table 5.4 shows the results when using the Singer motion model and its Q matrix, instead of a constant velocity model and the Q matrix from Chiu et al. (2021).

These first results were slightly discouraging as the overall performance is lower than the baseline. However, the model showed promising improvement in the pedestrians category I was interested in improving, with tracking accuracy gaining 0.8% on the validation dataset.

	Parameters						Performance Mini		Performance Validation	
Category	Metric	Position Threshold	Min Hits	Max Age	σ	α	AMOTA $\uparrow$	AMOTP $\downarrow$	AMOTA $\uparrow$	AMOTP $\downarrow$
Car	2D GIoU	-0.1	1	3	4.00	0.02	0.628	0.785	0.593	0.813
Pedestrian	2D GIoU	-0.5	1	4	0.80	0.05	0.493	1.205	0.306	1.500
Truck	3D L_∞	-4	3	4	1.00	0.02	0.086	1.382	0.296	1.385
Trailer	3D L_∞	-4	3	2	0.10	0.02	0.803	1.867	0.000	1.423
Bus	3D GIoU	-0.8	1	2	4.00	0.05	0.751	1.043	0.521	1.080
Motorcycle	2D Euclidean	-2	1	4	0.20	0.03	0.654	0.938	0.422	0.963
Bicycle	3D L_∞	-4	3	1	0.20	0.03	0.142	1.264	0.324	1.376
Total							0.508	1.212	0.352	1.220

Table 5.4: Post-optimization tracking results using the **Singer model** with no additional matching modules, on the validation dataset

Contrary to results with the constant velocity model, adding double-round matching and the velocity matching modules first showed large improvements on the mini-dataset tests, as shown in Table 5.5 and Table 5.6. Notably, there was a 0.9% accuracy increase from the double-round matching module, and a further 3.2% accuracy increase by combining it with the velocity matching module. However, upon testing on the validation dataset, that tendency reverted, with accuracy decreasing with each module. This heavily implied sub-optimal optimization, or over-fitted tuning parameters to the mini-dataset. The increase of performance on the first dataset still leads me to believe the model could be useful if tuned properly. However, as the Singer model was originally developed for tracking aircraft, it is also very possible it is overly complex

for certain categories. A hybrid approach could therefore be more suited, but is left for future work.

	Parameters					Performance Mini		Performance Validation	
Category	Metric (round 1)	Position Threshold 1	Position Threshold 2	Min Hits	Max Age	AMOTA $\uparrow$	AMOTP $\downarrow$	AMOTA $\uparrow$	AMOTP $\downarrow$
Car	3D IoU	0.1	-0.1	1	3	0.630	0.785	0.593	0.812
Pedestrian	3D Euclidean	-2	-0.5	1	3	0.494	1.257	0.293	1.534
Truck	2D IoU	0.3	0.1	2	3	0.102	1.340	0.325	1.347
Trailer	3D Euclidean	-2	-0.2	2	2	0.814	1.812	0.000	1.327
Bus	2D IoU	0.1	-0.7	1	2	0.751	1.043	0.520	1.079
Motorcycle	2D Euclidean	-2	-0.3	1	4	0.654	0.938	0.419	0.961
Bicycle	2D IoU	0.2	0.2	3	3	0.177	1.478	0.292	1.365
Total						0.517	1.236	0.349	1.204

Table 5.5: Post-optimization tracking results using the **Singer model** and **double-rounds matching**, on the validation dataset

	Parameters						Performance Mini		Performance Validation	
Category	Metric (round 1)	Position Threshold 1	Position Threshold 2	Velocity Threshold	Min Hits	Max Age	AMOTA $\uparrow$	AMOTP $\downarrow$	AMOTA $\uparrow$	AMOTP $\downarrow$
Car	3D IoU	0.1	-0.1	-1	1	3	0.630	0.785	0.593	0.812
Pedestrian	3D Euclidean	-2	-0.5	-1	1	3	0.494	1.257	0.293	1.534
Truck	2D L_∞	-1	0.0	0.4	3	3	0.163	1.582	0.154	1.701
Trailer	2D IoU	0.7	-0.6	0.0	3	3	0.817	1.782	0.000	1.577
Bus	2D IoU	0.1	-0.7	-1	1	2	0.751	1.043	0.520	1.079
Motorcycle	2D Euclidean	-2	-0.3	-1	1	4	0.654	0.938	0.419	0.961
Bicycle	3D Euclidean	-2	-0.4	0.8	3	3	0.334	1.252	0.084	1.815
Total							0.549	1.234	0.295	1.354

Table 5.6: Post-optimization tracking results using the **Singer model**, **double-rounds matching**, and **velocity matching**, on the validation dataset

In conclusion of this chapter, I implemented various additional modules to try to increase the tracking accuracy of my pipeline. However, looking at the nuScenes

validation dataset, the observed tendency is that the modules do not seem to provide much better average tracking accuracy and precision, measured with the AMOTA and AMOTP metrics, when using my camera-radar detection backbone. That being said, additional experiments could yield better results, and more in-depth statistical analyses may uncover additional information and lead to different conclusions. In most cases, hyperparameter optimization would require a larger dataset and a more efficient search strategy, which I implemented later in the next chapter. Additionally, using a different camera-radar detection backbone with more stable detection performance could lead to improved tracking outcomes.

However, since the core contributions of my work lie in the following chapters, I determined that pursuing these improvements would be too time-intensive given my time-frame and have left them for future work.

For the same reasons, I also chose not to refine the de-clustering module. Tuning it proved tedious, and although it appeared effective visually, it did not yield improvements in AMOTA. I believe this is primarily because duplicate detections are relatively sparse and are already well-handled by the tracking pipeline through the minimum hits parameter. In contrast, my de-clustering approach led to more missed tracks on average and increased track fragmentation.

Chapter 6

SwitchTrack: Switching Sensor-Failure Robust

Tracking Pipeline

In Chapter 3, I described how I created a realistic noise synthesizer to simulate the effect of noise of the existing data from the nuScenes dataset. In Chapters 4 and 5, I described my tracking pipeline and its additional modules. In this chapter, we will cover how these two parts are combined to form my final adaptive tracking pipeline, named SwitchTrack.

6.1 Methodology

The final goal of my project is to develop a tracking pipeline that can adapt to sensor deterioration, while ensuring that the method remains lightweight and computationally efficient for real-world applications. An overview of my complete pipeline is illustrated in Figure 6.1, with the two sensor modalities connected to the noise recognition models, which in turns decides which detection backbone is used by the common tracking module.

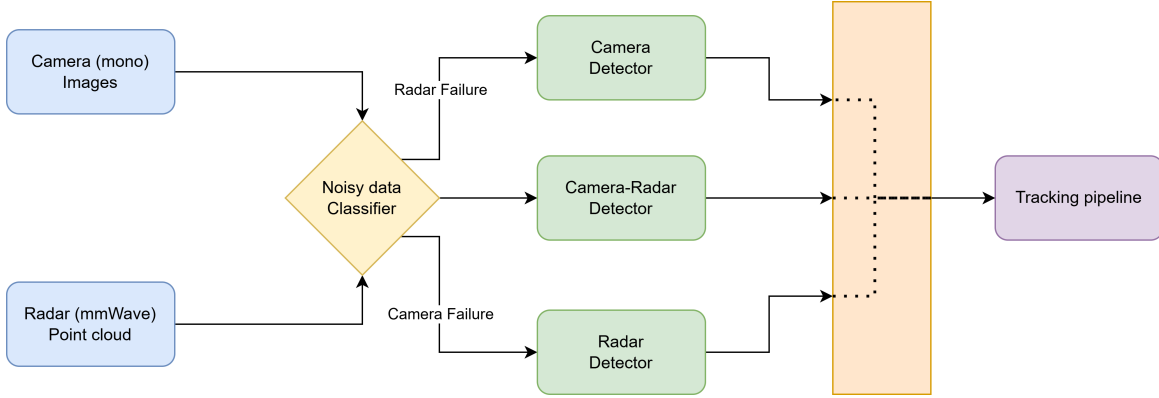


Figure 6.1: Overview of the tracking pipeline and its switching mechanism

6.1.1 Approach

To achieve robustness to sensor failure, my main approach is to use the trained noise recognition models described in Section 3.2.2 to infer on the noise levels of each frame. Since multiple sensor views are available and CRN utilizes all of them, I compute the average noise level separately for the six cameras and the five radars, resulting in one aggregated value per sensor type. If the averaged noise of either is higher than a threshold, the camera-radars backbone is deactivated and the pipeline switches to a single-sensor detection instead. To the best of my knowledge, no existing noise-aware adaptive pipelines such as this have been developed for camera-radar automotive tracking datasets.

The main strength of my approach is that the backbones can be easily replaced by higher performing ones as research advances. This way, the performance of my pipeline can scale up with minimal re-works, the only requirements being finding the right tracking hyper-parameters and characterizing the maximum noise levels of the detection backbone (see Section 6.4).

6.1.2 Camera-only Backbone

The camera-only detection backbone used in this work is the state-of-the-art (SOTA) Sparse4D, from Lin et al. (2023). More specifically I use Sparse4D-v3-offline, achieving 66.8 mAP on the nuScenes test dataset. Notably, Sparse4D outperforms CRN in detection accuracy. Given the rapid advancements in 3D detection, methods do not remain SOTA for long, and current camera-radar SOTA, such as Lin et al. (2024), also largely outperforms CRN. However, integrating this newer backbone was beyond the scope of this work due to time constraints and implementation costs, and is left for future exploration.

Sparse4D-v3-offline is the third iteration of the Sparse4D pipeline. Built as an End-to-End 3D Detection and Tracking pipeline this method uses a mix of decoupled self-attention mechanism and encoding-decoding pipelines to achieves reliable detection of 3D objects using 2D images.

6.1.3 Radar-only Backbone

Finally, the radar-only method I used is the current SOTA method RadarDistill, from Bang et al. (2024). Due to the challenging nature of 3D radar-only tracking and the relatively lower research interest in this area, the field remains in its early stages, and existing methods have yet to achieve high accuracy. Notably, only three radar-only detection methods exist for the nuScenes dataset, compared to 116 camera-only methods. Furthermore, no radar-only tracking methods have been developed for this dataset, highlighting the scarcity of research in radar-based detection and tracking.

RadarDistill relies on transfer learning techniques, leveraging a LiDAR-only detection model and distilling its knowledge to a radar-only model. This is possible because the data types from both sensors are fairly similar, as both yield point clouds. It is also

important to remark that before RadarDistill, radar-only detection methods accuracies did not go over 5%. We can hope the performance gap achieved by Bang et al. (2024) will encourage more research in that area.

6.2 Improved Hyper-Parameter Tuning

For both backbones, another hyper-parameter optimization process is required to find good matching threshold, given the different inherent ability of each method to detect objects. Since the exhaustive search described in Section 5.5 is not adapted anymore, I implemented a faster optimization method instead. Furthermore, results of Chapter 5 indicated it was better not to use the best parameters of the mini dataset, to avoid overfitting, alleviating my previous concerns expressed in Section 5.5.

My first optimization strategy was a **random hill-climbing grid search algorithm** described in Algorithm 1. The idea is to randomly initialize the parameters, to select a parameter to explore, computing the forward (+step) and backward (-step) accuracies with this parameter, and selecting the best. If no best parameter has been found for a predetermined number of steps, I consider we found a local minima.

While this first algorithm greatly reduces the optimization time compared to the exhaustive search described in Section 5.5, there is a high risk of the optimizer falling in a local minima, or of not moving in a flat region. Building on this first algorithm, I created my second and current optimization strategy: a **multi-start multi-direction random hill climbing grid search**, described in Algorithm 2.

Given the computational intensity of evaluating the pipeline to retrieve the AMOTA value, Bayesian optimization presents a promising alternative for this tuning problem.

However, due to time constraints, I was unable to implement it, and it is left as a direction for future work.

Algorithm 1 Random Hill-Climbing Grid Search

```

for cat in categories do
  for metric in metrics do
    initialize hyper_params  $\leftarrow$  random_init
    initialize best_amota  $\leftarrow$  TRACKING(hyper_params)
    initialize IMPROVEMENT  $\leftarrow$  TRUE
    initialize cnt  $\leftarrow$  0
    while IMPROVEMENT do
      prev_amota  $\leftarrow$  best_amota
      parameter  $\leftarrow$  RANDOM (parameters)
      forward_param  $\leftarrow$  hyper_params[parameter] + step
      backward_param  $\leftarrow$  hyper_params[parameter] - step
      hyper_params[parameter]  $\leftarrow$  forward_param
      forward_acc  $\leftarrow$  TRACKING(hyper_params)
      hyper_params[parameter]  $\leftarrow$  backward_param
      backward_acc  $\leftarrow$  TRACKING(hyper_params)
      best_amota  $\leftarrow$  MAX(best_amota, forward_acc, backward_acc)
      if prev_amota = best_amota then
        cnt  $\leftarrow$  cnt + 1
      else
        cnt  $\leftarrow$  0
      end if
      if cnt > max_cnt then
        IMPROVEMENT  $\leftarrow$  FALSE
      end if
    end while
  end for
end for

```

Algorithm 2 Random Hill-Climbing Grid Search

```
for cat in categories do
  for metric in metrics do
    for  $i = 1$  to  $N\_restarts$  do
      initialize hyper_params  $\leftarrow$  random_init
      initialize best_amota  $\leftarrow$  TRACKING(hyper_params)
      initialize IMPROVEMENT  $\leftarrow$  TRUE
      initialize cnt  $\leftarrow$  0
      while IMPROVEMENT do
        prev_amota  $\leftarrow$  best_amota
         $Ndirections \leftarrow$  random(2, 6)
        directions  $\leftarrow$  RANDOM_CHOICE(parameters,  $Ndirections$ )
        initialize all_movement_acc  $\leftarrow$  []
        for parameter in directions do
          movement  $\leftarrow$  RANDOM_CHOICE(forward, backward)
          movement_param  $\leftarrow$  hyper_params[parameter]  $\pm$  step
          hyper_params[parameter]  $\leftarrow$  movement_param
          movement_acc  $\leftarrow$  TRACKING(hyper_params)
          append movement_acc to all_movement_acc
        end for
        best_amota  $\leftarrow$  MAX(best_amota, all_movement_acc)
        if prev_amota = best_amota then
          cnt  $\leftarrow$  cnt + 1
        else
          cnt  $\leftarrow$  0
        end if
        if cnt > max_cnt then
          IMPROVEMENT  $\leftarrow$  FALSE
        end if
      end while
    end for
  end for
end for
```

6.3 Dataset Generation

To measure the performance gain from the switching pipeline under degraded data, I use my noise synthesizer to create a custom, noisy nuScenes validation dataset.

I start by separating the nighttime scenes from the rest, since I consider night data to be already degraded (see Section 3.2.1). I split the remaining dataset in four parts, each applying data degradation differently:

- 20% of the dataset is **random noise**. A random noise level between 0% and 100% is applied to each sample. For cameras, one of the four types of noises is randomly selected at every sample as well.
- 30% of the dataset is **ramping-up noise**. The scene starts with a noise level of 0%, which gradually increases until reaching 100%. This is done by dividing the scene’s samples into eleven even bins, corresponding to the eleven noise levels I simulate. For cameras, one of the four types of noises is randomly selected for the entire scene.
- 30% of the dataset is **constant noise**. A random noise level between 30% and 100% is applied uniformly to all the samples throughout the scene. The minimum value is set to 30% to avoid having low-intensity noise levels applied on entire scenes. For cameras, one of the four types of noises is randomly selected for the entire scene.
- 20% of the dataset is **unchanged**. No noise is applied to these scenes, providing a control subset.

In random noise, I randomly draw which sensor is going to be degraded at each sample, while for increasing and constant noise generation, I further split the scenes in two,

50% of scenes receive a degradation on the images, and the remaining 50% receive a degradation on the radar point clouds. All random draws follow a uniform distribution.

Note that this dataset is different from the one used in Chapter 3. To train and test the noise recognition models, I had generated all possible noises on the 10-scenes mini-dataset, subdivided in a train/val/test split. The dataset described here, and used to test the tracking pipelines on, is a noisy version of the nuScenes validation dataset, with the same number of scenes (150) and samples (6019).

6.4 Switching Thresholds

Using my noise recognition models, I can infer on the amount of the noise in the camera and radar data. However, to be able to effectively switch between backbones when the degradation is too important, we must first define what level of degradation is considered as too high for each sensor.

6.4.1 Using Detection Performance Across Noise Levels

As an initial approach, I consider the decrease in detection performance of CRN as noise increased. Figures 6.2 and 6.3 show the evolution of the camera-radar detection performance on increasingly degraded versions of the camera and the radar data, respectively, of the nuScenes mini dataset. While the performance of the backbone may vary depending on the dataset used, I considered that, on average, the evolution of accuracy versus noise levels should follow the same trend.

I evaluate CRN recursively on each noise level, looking at the mean Average Precision (mAP) and the NuScenes Detection Score (NDS) metrics, as defined by Caesar et al. (2019). The Average Precision is computed by calculating the 2D distance between the centers predictions and the ground truths, and averaging across all detections.

The NDS score is an weighted average of all detection metrics, with a weight of one attributed to the translation, scale, orientation, velocity, and attribute errors, and a weight of five attributed to the average precision metric. This gives us two curves per figure. I focus on the NDS curve since it encompasses all detection metrics, but the average precision curve can be useful in case of uncertainty. Looking at the results, we notice the expected decrease in NDS and mAP performance as noise levels increase.

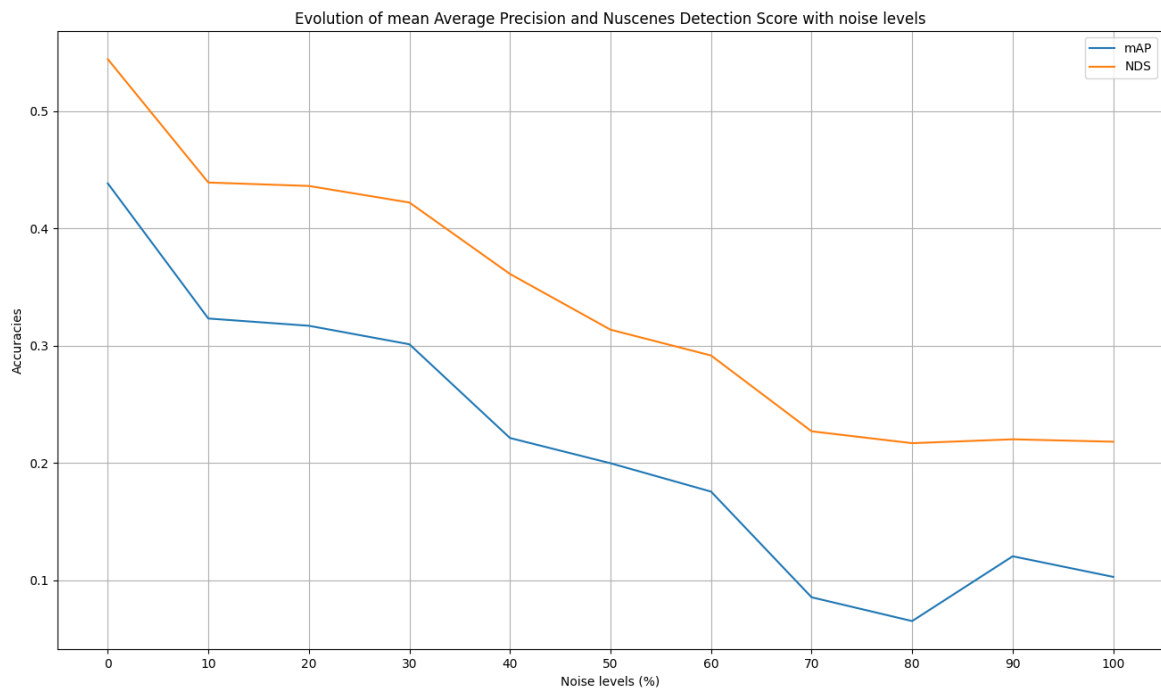


Figure 6.2: CRN's performance evolution with increasing **camera** noise levels

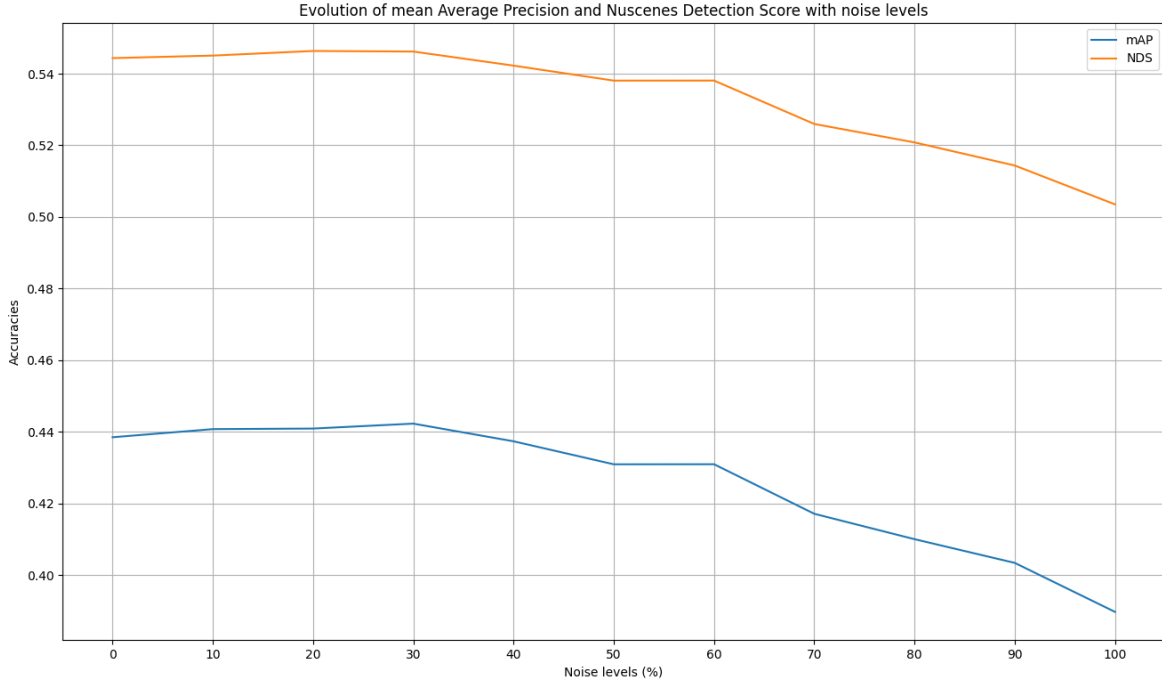


Figure 6.3: CRN’s performance evolution with increasing **radar** noise levels

To select a potential switching threshold on the camera data, I compare the evolution of CRN’s accuracy in Figure 6.2 with the known accuracy of the radar-only RadarDistill method, which is 20.5% mAP and 43.7% NDS. At 40% degradation level on the camera data, the mAP and NDS performance of CRN are below those of RadarDistill, indicating a potential good camera noise switching threshold.

For the radar data, we cannot compare with Sparse4D directly, since it outperforms CRN even at a 0% noise level. To set a threshold, I instead look at CRN’s performance evolution across the degradation levels, and notice a large drop in performance at 70%, which will constitute the radar noise threshold.

To verify if these thresholds are correct, I evaluate the performance of SwitchTrack (my switching pipeline) and compare them to those of CRN-VeLACaR, on the noisy dataset described in Section 6.3. To avoid bias due to my noise recognition models,

I instead use an “ideal” noise recognizer, using the noisy dataset generation logs to know exactly what noise level was applied to each sample. For the night-time data, I compare the images with both the underexposure and gaussian noise levels and find that 80% is a reasonable noise level estimation.

The comparative results for the selected thresholds, reported in Table 6.1, indicate that, on average, they do not show improvement on the validation dataset. We can draw very important conclusions from this, the first being that a decreased detection mAP or NDS does not linearly translate to a decrease in AMOTA and AMOTP. In reality, accurate and precise tracking relies on more than detection performance. While detection performance mainly depends on labeling and position accuracy, the tracking pipeline also requires a good velocity and orientation estimation to correctly match the detection and maintain consistent tracking.

Furthermore, a difference in performance between each backbone will translate to even greater differences in tracking. For instance, Sparse4D shows few differences in cross-category accuracy, further increasing average tracking accuracy, even under noisy data. Furthermore, CRN’s larger cross-category differences (e.g., trailers are detected with 48.3 % accuracy while cars are detected with 78.4% accuracy) and lower average detection accuracy lead to low tracking accuracy on some categories, reducing the average performance. Finally, while RadarDistill represents the state-of-the-art in radar-only detection, it still achieves a relatively low overall accuracy (20.5% mAP), which translates into poor tracking performance. This performance gap between radar-only methods and those leveraging other sensor modalities places significant limitations on the effectiveness of the proposed pipeline, underscoring the need for further research in this area.

Finally, the performance gap between the three detection backbones also complicates the interpretation of the results and limits our ability to draw definitive conclusions.

Method	AMOTA $\uparrow$	AMOTP $\downarrow$	FP $\downarrow$	FN $\downarrow$	IDS $\downarrow$
CRN-Extended VelACaR	0.275	1.357	21465	51766	795
SwitchTrack (initial)	0.205	1.407	23029	49068	1388

Table 6.1: Preliminary tracking performance comparison between SwitchTrack and a static CRN-VelACaR tracker, on synthetically degraded data

The noise thresholds are estimated using detection performance evolution under increasing noise levels. The noise level for cameras is 30% and for radars is 60%. To avoid bias, noise recognition is ideal.

On a separate note, Figures 6.2 and 6.3 also show how much camera data is important compared to radar data for CRN, as even a degradation by a 100 factor (-20 dB) of the radar data only results in a 4% drop in precision on this dataset.

As mentioned in Chapter 2, Kim et al. (2023) did an ablation study on the robustness of CRN, but only by removing camera or radar views. The results in Figures 6.2 and Figure 6.3 also show that my assumption that noisy data is worse than no data is correct. Indeed, the synthetic noise effectively decreases CRN’s accuracy under 10% mAP if applied on the cameras, and under 40% applied on the radar. On the other hand, CRN noted a drop down to 12.8% and 43.8% accuracy if they completely removed the camera or radar input data, respectively, on the same dataset.

6.4.2 Using Tracking Performance Across Noise Levels

In the previous section, I considered that when CRN’s average performance are lower than RadarDistill’s, which is at camera noise levels above 30%, we should immediately switch to radar-only tracking. However, as the initial results suggested, this would be premature, as other parameters come into play.

To find a better switching threshold, I instead evaluate the adaptive tracking pipeline using different noise thresholds, comparing the tracking performance directly, thus considering the noise thresholds as hyper-parameters. In this experiment, the maximum noise threshold of one of the sensors is set to 100%, effectively deactivating its switching mechanism, and making sure the results only depend on the switching threshold of the other sensor.

Figure 6.4 shows the evolution of the tracking performance when increasing the switching threshold of the camera noise level. Here the maximum radar noise threshold is set to 100%, meaning the camera-only backbone (i.e., Sparse4D) is deactivated, only allowing switching from the camera-radar (i.e., CRN) backbone to the radar-only (i.e., RadarDistill) backbone. I repeat the same experiment on the radar noise level, deactivating RadarDistill by setting the camera noise level threshold to 100% and progressively increasing the radar noise level threshold. Those results are in Figure 6.5. In both figures, we can see the accuracy of SwitchTrack for each category, across multiple noise levels represented by different colored bars. In each case, we want to find the noise threshold that yields the highest AMOTA for as many categories as possible. Since noise is never applied to both sensors simultaneously in the noisy dataset (see Section 6.3), I am able to determine the optimal threshold for each sensor independently before evaluating the complete system with both thresholds. From Figure 6.4 and

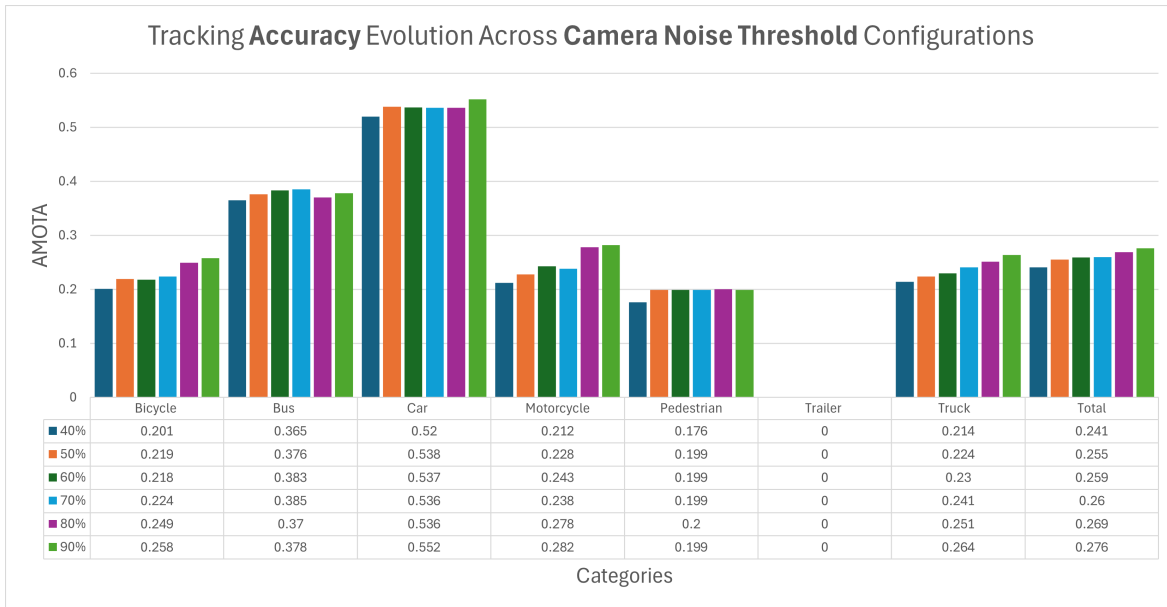


Figure 6.4: Tracking performance evolution as the maximum **camera noise threshold** before switching is gradually increased

Figure 6.5, I conclude that the best thresholds are **90% for the camera noise** and **40% for the radar noise**.

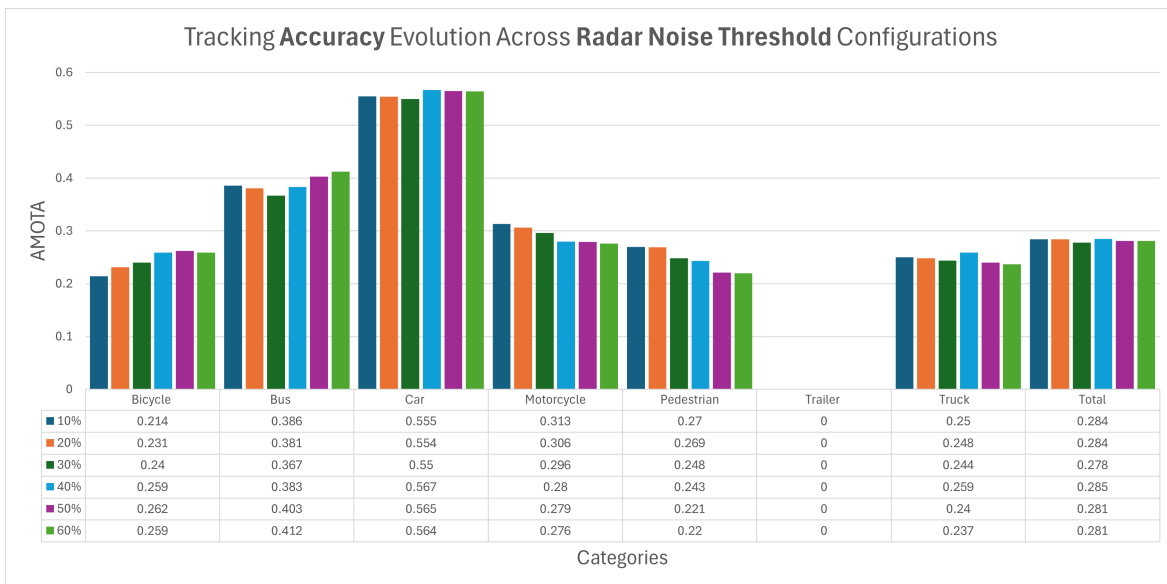


Figure 6.5: Tracking performance evolution as the maximum **radar noise threshold** before switching is gradually increased

Chapter 7

Experimental Results

In the previous sections I described my adaptive tracking pipeline, along with the thought process behind my choices and my takeaways from negative results. In this section, we will analyze the final performance results of the proposed method under sensor degradation, and compare them to the tracking performance of “static” pipelines (i.e., performance when using a single detection backbone). Since the results are averaged performance across 150 scenes, they provide general trends and allow us to draw initial conclusion, but may not be statistically significant.

While my tracking pipeline is deterministic, meaning the same input yields the same output, performance can still vary across scenes due to differences in content, object count, and degradation impact. Thus, statistical evaluation across scenes (e.g., via paired t-tests or rank tests on per-scene AMOTA scores) could help assess the robustness of observed improvements. However, in this work, we follow the standard practice in the tracking literature, where AMOTA and AMOTP are commonly reported without formal statistical significance testing (Weng et al. 2020; Caesar et al. 2020; Nabati et al. 2021; Baumann et al. 2024; Kim et al. 2023; Lei et al. 2023; Wolters et al. 2024; Bang et al. 2024). I emphasize that the evaluation specifically targets performance degradation under controlled noise, and the reported metrics are consistent across all scenes, reinforcing the observed trends. Given my time-frame, additional statistical analysis is left for further works.

We will start by observing the average performance results of each static pipeline on the default and noisy datasets, before comparing the performance of the adaptive pipeline (SwitchTrack), with the static trackers, first assuming ideal noise recognition and then using the noise recognition models described in Section 3.2.2.

7.1 Static Pipelines

To observe the impact of data degradation on traditional single-backbone (static) tracking pipelines, I evaluate the AMOTA and AMOTP (defined in Eq. 4.7) performance of VelACaR using CRN(camera-radar), RadarDistill(radar-only), and Sparse4D(camera-only) as detection backbones. As I concluded in Section 5.6.2 that the Singer model was not beneficial in its current state and on this dataset, I use a constant velocity motion model instead, as described in Section 4.3.1. Likewise, I do not use the clutter-removal module described in Section 5.4, for the same reasons, but I kept all the other modules described in Chapter 5 since optimization will automatically activate, or deactivate, them for each category. For the CRN backbone, the hyper-parameters I use are those optimized through exhaustive search, defined in Section 5.6.1. When using RadarDistill and Sparse4D, the hyperparameters are optimized using Algorithm 2 defined in Section 6.2.

In Figure 7.1, Figure 7.2 and Figure 7.3, I report the tracking accuracy of each category, along with the total accuracy averaged across all categories, for CRN, Sparse4D, and RadarDistill, respectively. Results in blue are those on the default dataset, and those in orange are on the synthetically degraded dataset. For all three backbones, we observe a performance drop across all categories between the default dataset and the synthetic one. However, not all methods are equal. On the car category for instance,

CRN shows a 6.9% AMOTA decrease (on average, across all 150 scenes), while for the same category Sparse4D only shows a 4.2% accuracy decrease, and RadarDistill a much larger 14.3% decrease. This could indicate that Sparse4D is more robust to noise than CRN, or it could be simply due to half of the deformations being applied on the radar data, which Sparse4D does not use. In any case, it does underline that by using two sensor modalities, CRN may be more accurate, but it is exposed to twice as many sensor failures as a single sensor backbone. This underlines the importance of having backup plans, either denoising or switching, when taking in account possible data degradation. Furthermore, the large difference in tracking accuracy between the radar-only RadarDistill and the other tracking pipelines concurs with my analysis in Section 6.4.1, and will pose limits on my method’s performance since radar-only tracking shows a 0% accuracy for multiple categories. For this reason I will pay a particular attention to the car category, as it is the only one that is properly tracked in radar-only modality on the synthetic dataset.

The reason for the lower tracking performance of radar-only methods can be traced back to the lack of semantic information the radar sensor offers, which is further deteriorated by the noise. Where CRN and Sparse4D can still make out objects shapes and colors for example, the radar point cloud degradation can be enough to make current radar-only methods unable to extract meaningful spatial features.

Finally, it is also interesting to notice that, even though detection performance of Sparse4D are about 10% higher than CRN’s, tracking performance of the latter are higher on the default validation dataset.

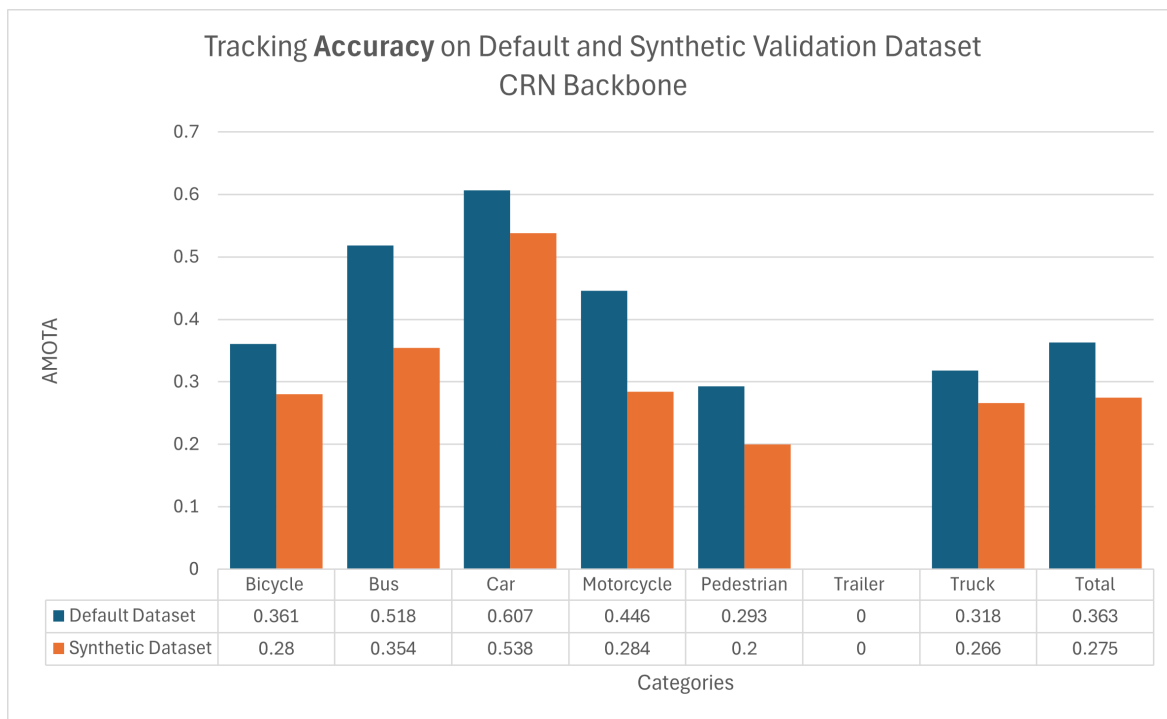


Figure 7.1: Tracking performance drop induced by the synthetic data degradation with the **CRN** (**camera-radar**) backbone

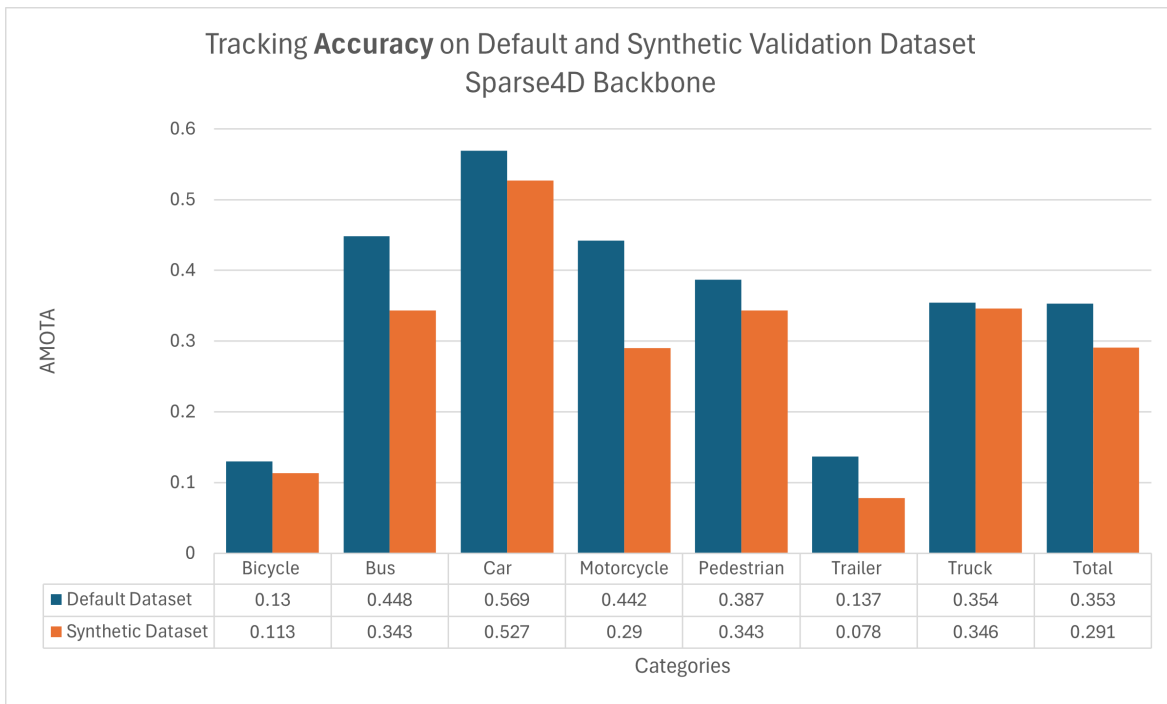


Figure 7.2: Tracking performance drop induced by the synthetic data degradation with the **Sparse4D** (camera-only) backbone

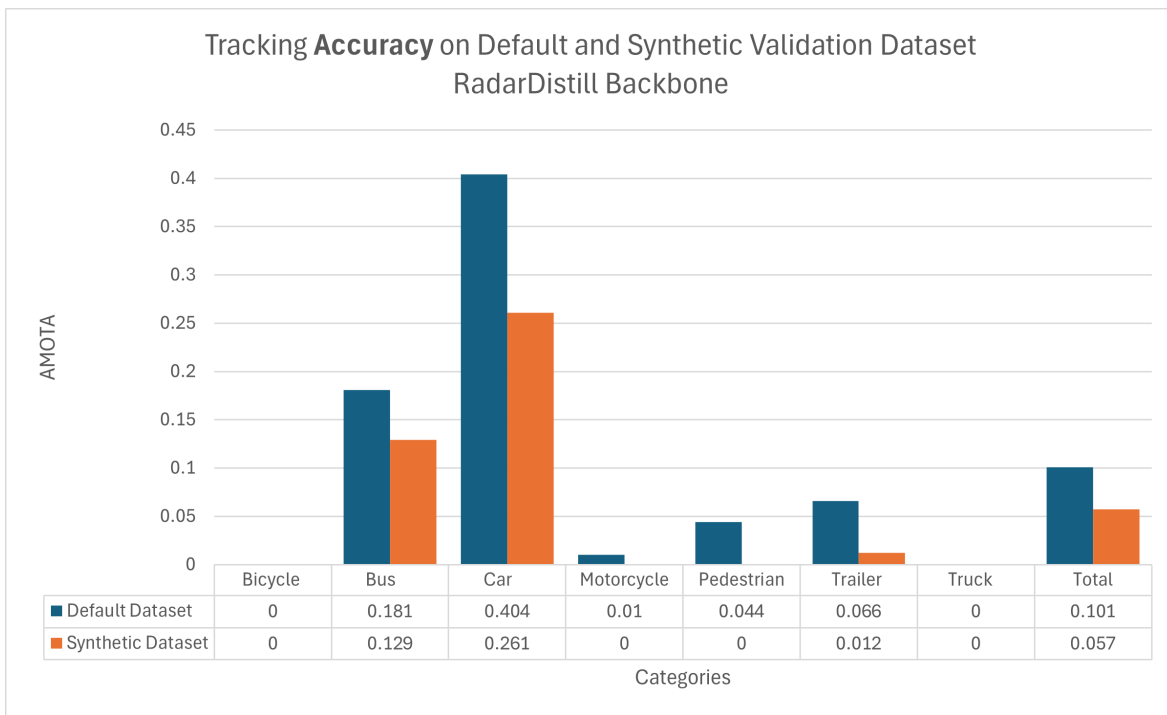


Figure 7.3: Tracking performance drop induced by the synthetic data degradation with the **RadarDistill (radar-only)** backbone

7.2 Adaptive Pipeline

Ideal Switch:

Since my noise recognition models are prone to errors, my first experiments with the adaptive pipeline are aimed at finding out what performance we can envision with an ideal switch that would identify the noise level with 100% accuracy. To this effect, and as described in Section 6.4.1, I use the data generation logs to get the ground-truth value of the data degradation at each sample.

I evaluate my tracking pipeline’s performance using the maximum noise thresholds defined in Section 6.4.2, and compare to the static pipelines. These results are reported in Table 7.1. Compared to CRN-VelACaR, we observe a slight improvement in average accuracy of 0.4% across the scenes of my synthetic validation dataset, with a gain of 10mm in precision. When comparing SwitchTrack to the static Sparse4D-VelACaR tracker, we note that while accuracy is 1.2% lower, precision improves by 56mm, on this dataset.

These results suggest that integrating a stronger radar-only detection backbone could significantly enhance my method’s overall performance on this dataset. Additionally, they demonstrate that with a sufficiently accurate noise recognition model, my system is capable of improving both precision and accuracy under noisy sensor conditions.

Method	AMOTA↑	AMOTP↓	FP↓	FN↓	IDS↓
CRN-VelACaR	0.275	1.357	21465	51766	795
Sparse4D-VelACaR	0.291	1.403	8901	46272	798
SwitchTrack	0.279	1.347	26778	57005	1507

Table 7.1: Tracking performance comparison between static trackers and SwitchTrack with improved noise thresholds and **ideal noise recognition**, on the synthetic dataset I compare the tracking performance of my switching method, under ideal noise recognition, with the static CRN-based tracker and the static Sparse4D-based tracker. Evaluation is done on the synthetic dataset.

To further quantify the potential improvements of the adaptive pipeline with an ideal noise recognizer, Figures 7.4 and 7.5 compare the tracking accuracy and precision, respectively, across all static pipelines and my proposed method. Results from CRN-VelACaR are in dark blue, those from Sparse4D-VelACaR are in orange, those from RadarDistill-VelACaR are in green, and those from SwitchTrack are in light-blue (on the right end of each category).

On the two categories for which the RadarDistill backbone shows consistent tracking, the *bus* and the *car* categories, we observe a substantial improvement of the AMOTA with my method on the synthetic validation dataset. On both categories, CRN-VelACaR has the highest tracking accuracy (AMOTA) among the static pipelines, while SwitchTrack reaches even higher average accuracies, with 5.4% increase on the **bus** category, and a 2.6% increase on the *car* category. Precision-wise, SwitchTrack also shows a 4.3 cm AMOTP gain on the *car* category for this dataset, outperforming all other static pipelines.

The general trend suggests that when all three detection backbones achieve reasonably consistent tracking, my switching mechanism improves accuracy beyond what

any single static backbone achieves. However, if one or more backbones exhibit poor tracking accuracy for a given category, overall performance tends to degrade. This is expected, as my pipeline relies on contributions from all three backbones—each needs to provide at least some reliable input for the combined result to be beneficial. This observation also points to a potential refinement: applying the switching mechanism on a per-category basis, selectively excluding backbones with poor performance for specific categories. I leave this direction for future work.

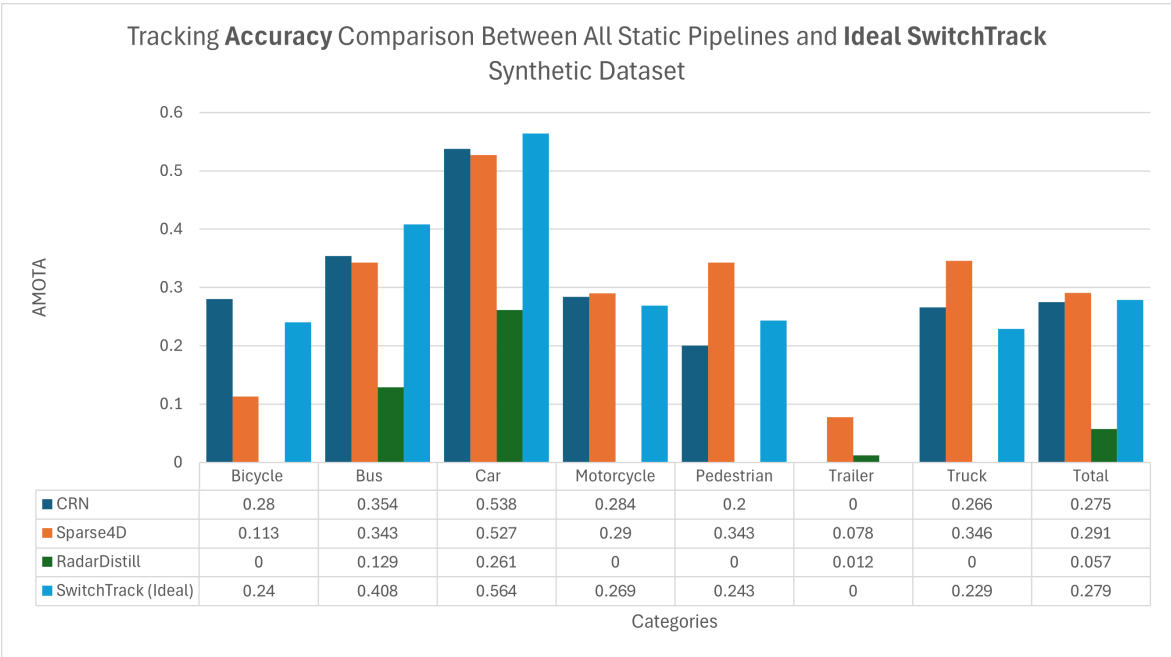


Figure 7.4: Category-wise tracking **accuracy** comparison between all static models and an SwitchTrack, **ideal noise recognition**

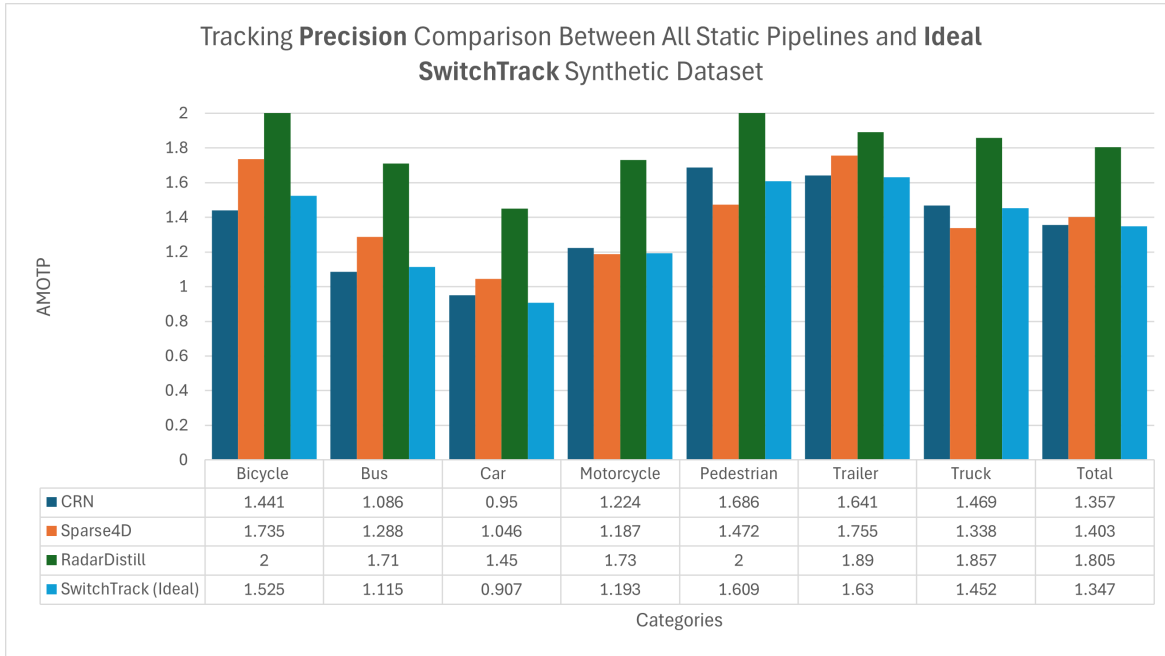


Figure 7.5: Category-wise tracking **precision** comparison between all static models and an SwitchTrack, **ideal noise recognition**

Using Noise Recognizers:

Finally, I evaluate the performance of the tracking pipeline when using the noise recognition models trained in Chapter 3, and report the results in Table 7.2. This time, we observe a small accuracy tradeoff of 0.1% AMOTA, for a 1.4 cm AMOTP gain on the noisy dataset. This underlines that, even with a low-accuracy radar noise recognizer, and a low accuracy radar-only tracking pipeline, my method shows a small improvement on average across the 150 scenes of the validation dataset.

I also compare the results of the adaptive pipeline with the static one in Figure 7.6 and Figure 7.7, corresponding to category-wise AMOTA and AMOTP performance, respectively, on the synthetic dataset. In blue is the performance of the CRN-VelACaR static tracker, and in orange those of SwitchTrack, using the noise recognizers. Once again, we observe an improvement in AMOTA on the *car* and the *bus* categories with

Method	AMOTA $\uparrow$	AMOTP $\downarrow$	FP $\downarrow$	FN $\downarrow$	IDS $\downarrow$
CRN-VelACaR	0.275	1.357	21465	51766	795
SwitchTrack	0.274	1.343	23766	43927	957

Table 7.2: Tracking performance between static CRN-VelACaR and SwitchTrack using the **noise recognition models**, on the synthetic dataset

Here I compare my default static pipeline, CRN-VelACaR, with my adaptive method when using my noise recognizers from Chapter 3, on the synthetically degraded nuScenes validation dataset.

my method, of 2.2% and 2.1%, respectively, above all static tracking performance on the synthetic dataset.

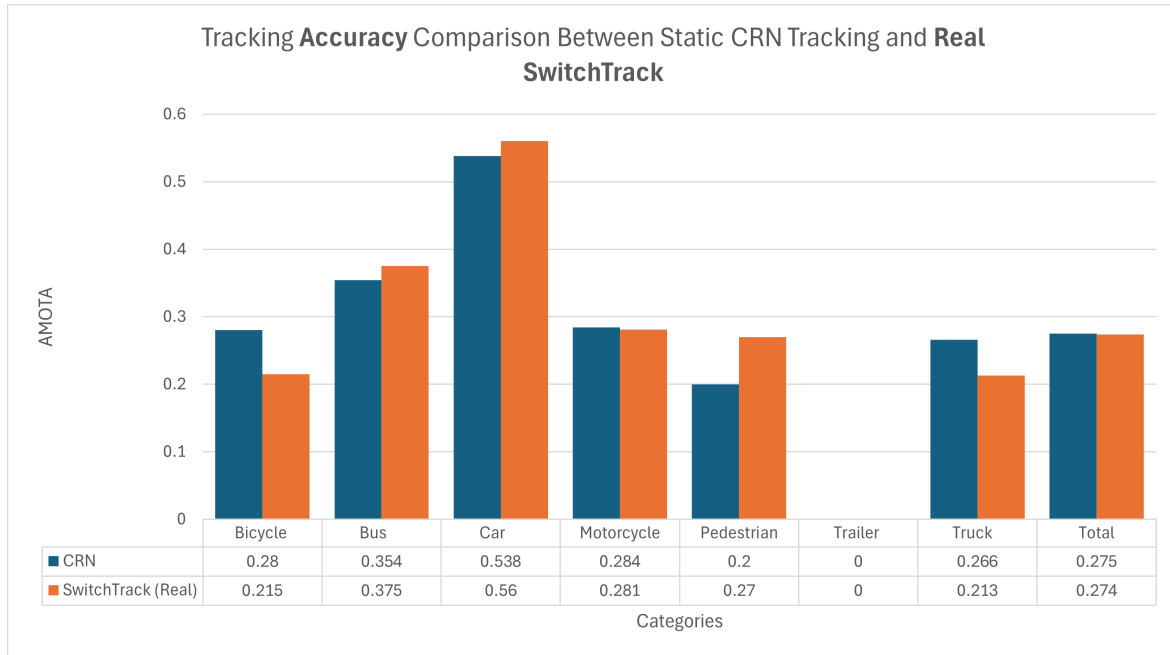


Figure 7.6: Category-wise tracking **accuracy** comparison between the static CRN-VelACaR tracker and SwitchTrack, using **noise recognition models**

Furthermore, to provide a more in-depth analysis of the performances, I isolate the scenes that received a camera degradation and those that received a radar degradation, and evaluate the performance of the static pipelines and of SwitchTrack on them. Given

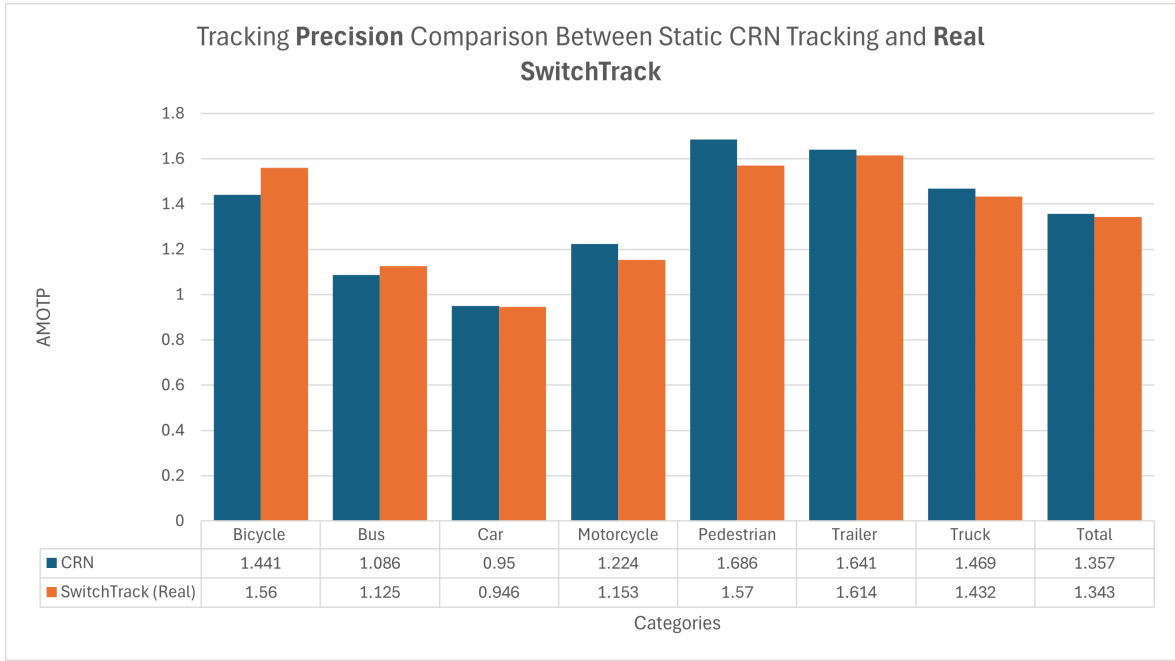


Figure 7.7: Category-wise tracking **precision** comparison between the static CRN-VelACaR tracker and SwitchTrack, using **noise recognition models**

the radar-only backbone (RadarDistill) performs very poorly on most categories, it would not be fair to compare tracking performance beyond the *bus* and *car* categories, which are the only ones reliably tracked by the radar-only system.

In Table 7.3 we see my method has increased accuracy on these two categories, as a well as on average across all categories of the nuScenes dataset and all camera-degraded scenes. On the *car* category, SwitchTrack shows an increase of 4.4% AMOTA compared to using a static camera-radar tracking pipeline. We can notice the camera-only Sparse4D-VelACaR still maintains a 50% AMOTA, which is impressive given the distortion, but SwitchTrack again shows an increase of 1.3% in comparison. Note that on the evaluated scenes, Sparse4D is never used by SwitchTrack since high camera noise levels will lead to switching to RadarDistill instead. On the *bus* category, SwitchTrack shows an increase of 8.2% AMOTA compared to CRN-VelACaR. These results indicate

that, on this radar-deformed dataset, my method achieves higher accuracy than each of the individual static trackers.

In Table 7.4, SwitchTrack achieves an improvement of 3.2% in AMOTA on the *car* category when the radar sensor fails, with an average increase of 4.8%. Note that since Sparse4D is better than CRN, SwitchTrack’s results on the radar category are lower, which is expected given the nature of the switching mechanism. If one of the backbone outperforms the default detector, the switching mechanism will have a hard time providing even higher performance.

Method	Car	Bus	Total
CRN-VelACaR	0.469	0.265	0.197
RadarDistill-VelACaR	0.374	0.194	0.103
Sparse4D-VelACaR	0.500	0.133	0.183
SwitchTrack	0.513	0.347	0.199

Table 7.3: AMOTA performance on the **camera**-degraded scenes, with SwitchTrack using a real noise recognizer

Method	Car	Bus	Total
CRN-VelACaR	0.532	0.279	0.303
RadarDistill-VelACaR	0.028	0.000	0.004
Sparse4D-VelACaR	0.145	0.272	0.352
SwitchTrack	0.564	0.278	0.351

Table 7.4: AMOTA performance on the **radar**-degraded scenes, with SwitchTrack using a real noise recognizer

These final results are encouraging, showing only a minimal decrease in accuracy despite the suboptimal radar noise recognizer. Notably, we observe even better precision compared to the ideal switching pipeline. This improvement may stem from increased reliance on the default CRN detection backbone, which offers the highest precision among the evaluated methods, and from a reduction in false positives, as a lower switching frequency results in less fragmentation.

Fragmentation during switching primarily stems from the varying detection characteristics of each backbone, leading to detection “jumps” that are more pronounced than in static pipelines. This issue could be mitigated by using more permissive matching thresholds or more consistent backbones. When fragmentation occurs, remnants of previously tracked object persist in the filter’s memory and may be incorrectly re-matched, further impacting tracking performance.

To further visualize the benefits of SwitchTrack on a degraded data sample, in Figure 7.8, I compare tracking with a static CRN tracker versus my switching method, on a blurry sample from the synthetic dataset. CRN produces numerous false positives, resulting in hallucinated tracks and degraded performance. In contrast, our switching mechanism detects significant degradation in the camera data and switches to radar-only tracking. As a result, it maintains reliable performance by correctly tracking only the bus in front of the ego vehicle.

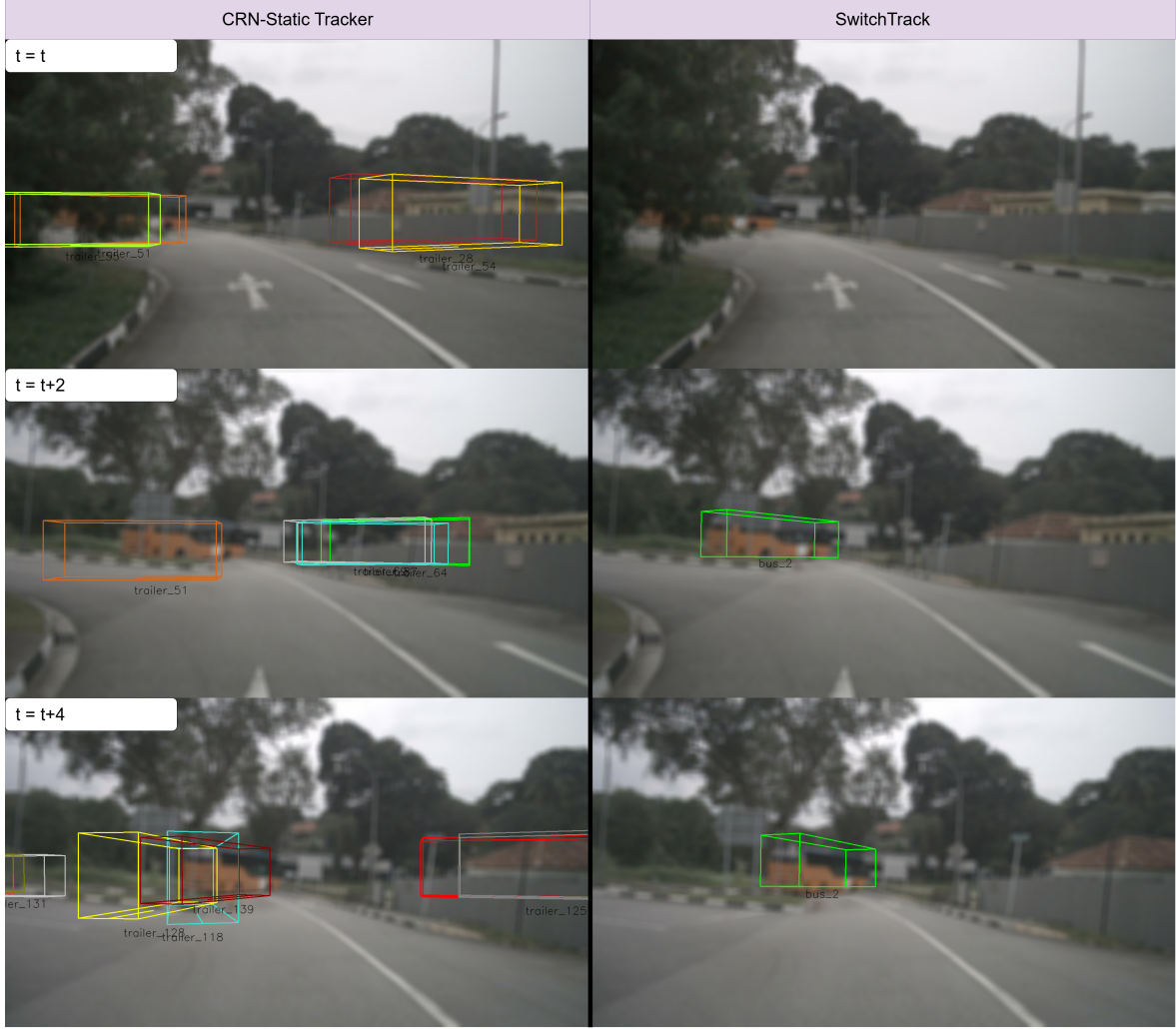


Figure 7.8: Visual comparison between a static CRN-VelACaR tracker and SwitchTrack on a noisy camera data sample

On the left is the visualization of the tracking using a CRN-static tracker, and on the right is the visualization of the tracking using SwitchTrack, under a heavily blurred image. Here the camera noise is too high and the backbone switched to RadarDistill for SwitchTrack, allowing it to only track the bus, while the distortion created artifacts throughout CRN's detections resulting in many false positives in tracking.

In Table 7.5, I also report how much time SwitchTrack spends using each backbone on my dataset. As we could expect, SwitchTrack uses CRN most of the time, which is expected given that 20% of the dataset is untouched, and on the remaining 80% not every noise levels reach the maximum thresholds. Furthermore, given the high camera noise threshold, it is also not surprising to see that RadarDistill is only used during 4.5% of the synthetic dataset. However I noticed that during 7 frames out of 40 in scene 0012, SwitchTrack uses RadarDistill, which corresponds to the frames displayed in Figure 7.8.

	CRN	Sparse4D	RadarDistill	Total
Samples	3460	2289	270	6019
Proportion	57.5%	38%	4.5%	100%

Table 7.5: Distribution of data samples across backbones

Samples denote the how many samples across all 150 scenes are spent using each backbones. Proportion is the relative proportion the numbers of samples represents over all 6019 samples of the synthetic validation dataset.

Finally, I also evaluated the impact of the noise recognition module on the pipeline’s inference speed using the default dataset, on intel core i7 processors. Performance decreased slightly from the original 0.071s (14.1 fps) to 0.09s per sample (11 fps), showing minimal impact from the noise recognition models. However, on the synthetic dataset, inference speed dropped significantly to 0.91s per sample (1.1 fps), probably due to an increase in tracked objects caused by fragmentation. This opens the door to further research on the necessary trade-off between noise recognition accuracy and model efficiency.

Chapter 8

Summary and Conclusions

In summary, I presented my Camera-Radar 3D Multi Object Tracking Robust to Sensor-Failures. I designed a dual noise recognition camera-radar model, plugged into a versatile tracking pipeline that switches between multiple detection backbones.

My most important contribution is most likely my noise synthesizer, the first to realistically simulate degradations on camera and radar data. Furthermore, because I simulate noise on an existing dataset, using only the output data of the sensors, my simulated degradations can theoretically be applied to any camera-radar datasets.

I use this synthesizer to train the first camera-radar noise recognizer, estimating the amount of noise in each sensor's output data. More specifically, while there are existing additive noise recognizer for camera data, mine is the first to propose an estimation of noise levels for additive noise, blur, and exposure issues. It is also the first to propose noise recognition on mm-wave radar point clouds.

I present a Tracking-by-Detection pipeline, leveraging its versatility to dynamically change detection backbone. This allows the method to switch from camera-radar sensor fusion detection to single-sensor, camera-only or radar-only, detection, in case the degradation in a sensor is judged to be too high. I also leverage high-accuracy Doppler-measured velocity to increase prediction accuracy.

Those three components (i.e., noise synthesizer, noise recognition, adaptive tracking) constitute a complete and functional development platform for future research in

robust three-dimensional multi-object detection and tracking.

The modular approach of my work allows it to be easily used for further research. The tracking pipeline requires minimal adjustments to adapt to an additional backbone, and can constitute a baseline for future works. My realistic noise generator can be used on other datasets since it does not depend on additional or external information.

Furthermore, my work on synthesizing realistic sensor data deterioration has application in robustness training and de-noising pipelines. My noise-aware tracking pipeline, on the other hand, has applications beyond autonomous driving, including drone dynamic landing. This field was initially considered as primary focus of this thesis, but was later set aside after an extensive literature review, as it was deemed too ambitious given my time-frame and available resources.

Current limitations of the project are its tracking performance and its radar noise recognition accuracy. Current state-of-the-art radar-only detection methods do not allow us to fully benefit from the switching pipeline, but results are encouraging continued research in this direction.

To improve my tracking pipeline, mixing my current approach with CRN and HyDra’s BEV tracking method, described in Kim et al. (2023) and Wolters et al. (2024), seems to be a promising avenue, while further tests with the Singer model and hyperparameters optimization could also prove beneficial.

My radar noise recognition model showed encouraging training results, even though it did not generalize well to new data. Testing more advanced models like GNNs or attention-based models, as described by Vaswani et al. (2017), should provide much better results. The varying input size could also be handled by grid-encoding or pillar-encoding, as seen in Guo et al. (2023).

Finally, given I have not conducted statistical analysis, I want to remind the reader that my results and interpretations are, for now, only for my specific dataset. Conducting in-depth statistical analyses on the performance of my modules and switching pipelines is part of the future works.

Further works also include:

- Using a more recent and more stable Camera-Radar detection backbones, potential candidates being HVDETFusion from Lei et al. (2023) or RCBEVDET++ from Lin et al. (2024), as they show small differences in category-wise accuracy, all the while having low velocity errors.
- Implementing Bayesian optimization for hyperparameter tuning.
- Measuring real-world noisy data could also be interesting to compare with the data degradation generated by SynthCRAV.
- Increasing the realism of simulated camera degradation using generative AI models.
- Mixing different noise categories together on the camera data.
- Exploring scenarios in which both sensors experience data degradation at the same time.
- Implementing category-dependent backbone switching, only activating the switching mechanism on relevant categories.

Finally, it is worth noting that using RadarDistill with my tracking pipeline effectively resulted in the very first radar-only tracking pipeline of the nuScenes dataset.

The tracking pipeline described in Chapter 4 was submitted to the WACV 2025 conference, but was not accepted. The feedback received provided valuable insights,

leading me to explore a different project direction, which ultimately shaped the main contributions of this thesis.

Some aspects of Chapter 5 were part of a paper submitted to the IEEE Intelligent Vehicles Symposium (IV), which was not accepted.

Most of the material covered in Chapter 3 is part of a paper currently under review for the IEEE IROS 2025 conference.

Reference List

- Affi, M., K. G. Derpanis, B. Ommer, and M. S. Brown, 2021: Learning multi-scale photo exposure correction. *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 9153–9163, <https://doi.org/10.1109/CVPR46437.2021.00904>.
- Bang, G., K. Choi, J. Kim, D. Kum, and J. W. Choi, 2024: RadarDistill: Boosting radar-based object detection performance via knowledge distillation from LiDAR features. *2024 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 15 491–15 500, <https://doi.org/10.1109/CVPR52733.2024.01467>.
- Bansal, K., K. Rungta, and D. Bharadia, 2022: RadSegNet: A reliable approach to radar camera fusion. URL <https://arxiv.org/abs/2208.03849>, accessed 2025-03, 2208.03849.
- Baumann, N., M. Baumgartner, E. Ghignone, J. Kühne, T. Fischer, Y.-H. Yang, M. Pollefeys, and M. Magno, 2024: CR3DT: Camera-radar fusion for 3D detection and tracking. URL <https://arxiv.org/abs/2403.15313>, accessed 2024-07, 2403.15313.
- Cadence Design Sytems, 2022: Mmwave radar applications and advantages. URL <https://resources.system-analysis.cadence.com/blog/msa2022-mmwave-radar-applications-and-advantages>, accessed 2025-04.
- Caesar, H., V. Bankiti, A. H. Lang, S. Vora, V. E. Liong, Q. Xu, A. Krishnan, and Y. Pan, 2019: nuScenes: A multimodal dataset for autonomous driving. URL <https://www.nuscenes.org/>, accessed 2024-01, Website.
- Caesar, H., and Coauthors, 2020: nuScenes: A Multimodal Dataset for Autonomous Driving . IEEE Computer Society, Los Alamitos, CA, USA, URL <https://doi.ieeecomputersociety.org/10.1109/CVPR42600.2020.01164>, 11618-11628 pp., <https://doi.org/10.1109/CVPR42600.2020.01164>.
- Chen, A., X. Wang, K. Shi, S. Zhu, B. Fang, Y. Chen, J. Chen, and Y. Huo, 2023: ImmFusion: Robust mmWave-RGB fusion for 3D human body reconstruction in all weather conditions. *2023 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2752–2758, <https://doi.org/10.1109/icra48891.2023.10161428>, URL <http://dx.doi.org/10.1109/ICRA48891.2023.10161428>.
- Chiu, H.-K., J. Li, R. Ambruş, and J. Bohg, 2021: Probabilistic 3D multi-modal, multi-object tracking for autonomous driving. *2021 IEEE International Conference on Robotics and Automation (ICRA)*, 14 227–14 233, <https://doi.org/10.1109/ICRA48506.2021.9561754>.

- Chuah, J. H., H. Y. Khaw, F. C. Soon, and C.-O. Chow, 2017: Detection of gaussian noise and its level using deep convolutional neural network. *TENCON 2017 - 2017 IEEE Region 10 Conference*, 2447–2450, <https://doi.org/10.1109/TENCON.2017.8228272>.
- Ding, S., E. Rehder, L. Schneider, M. Cordts, and J. Gall, 2023: 3dmotformer: Graph transformer for online 3d multi-object tracking. *2023 IEEE/CVF International Conference on Computer Vision (ICCV)*, 9750–9760, <https://doi.org/10.1109/ICCV51070.2023.00897>.
- Dreissig, M., D. Scheuble, F. Piewak, and J. Boedecker, 2023: Survey on LiDAR perception in adverse weather conditions. Accessed 2024-07, 2304.06312.
- Goodfellow, I., J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, and S. Ozair, 2020: Generative adversarial networks. *Commun. ACM*, **63** (11), 139–144, <https://doi.org/10.1145/3422622>, URL <https://doi.org/10.1145/3422622>.
- Guo, D., G. Yang, and C. Wang, 2023: PillarNet++: Pillar-based 3-d object detection with multiattention. *IEEE Sensors Journal*, **23** (22), 27 733–27 743, <https://doi.org/10.1109/JSEN.2023.3323368>.
- He, K., X. Zhang, S. Ren, and J. Sun, 2016: Deep residual learning for image recognition. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 770–778, <https://doi.org/10.1109/CVPR.2016.90>.
- Kalman, R. E., 1960: A new approach to linear filtering and prediction problems. *Transactions of the ASME–Journal of Basic Engineering*, **82** (Series D), 35–45.
- Kaneko, T., and T. Harada, 2021: Blur, Noise, and Compression Robust Generative Adversarial Networks . *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, IEEE Computer Society, Los Alamitos, CA, USA, 13 574–13 584, <https://doi.org/10.1109/CVPR46437.2021.01337>, URL <https://doi.ieeecomputersociety.org/10.1109/CVPR46437.2021.01337>.
- Kim, Y., J. Shin, S. Kim, I.-J. Lee, J. W. Choi, and D. Kum, 2023: CRN: Camera Radar Net for Accurate, Robust, Efficient 3D Perception. *2023 IEEE/CVF International Conference on Computer Vision (ICCV)*, 17 569–17 580, <https://doi.org/10.1109/ICCV51070.2023.01615>.
- Kuhn, H. W., 1955: The Hungarian method for the assignment problem. *Naval Research Logistics (NRL)*, **52**, URL <https://api.semanticscholar.org/CorpusID:9426884>.
- Lang, P., X. Fu, M. Martorella, J. Dong, R. Qin, X. Meng, and M. Xie, 2020: A comprehensive survey of machine learning applied to radar signal processing. URL <https://arxiv.org/abs/2009.13702>, accessed 2024-07, 2009.13702.

- Lee, M. Y., and M. H. Ang, 2024: CaRA-MOT: 3D multi-object tracking with memory mechanism using camera and radar. *2024 10th International Conference on Control, Automation and Robotics (ICCAR)*, 334–339, <https://doi.org/10.1109/ICCAR61844.2024.10569794>.
- Lei, K., Z. Chen, S. Jia, and X. Zhang, 2023: HVDetFusion: A simple and robust camera-radar fusion framework. URL <https://arxiv.org/abs/2307.11323>, accessed 2025-02, 2307.11323.
- Li, X., D. Liu, Y. Wu, X. Wu, L. Zhao, and J. Gao, 2024: Fast-Poly: A fast polyhedral framework for 3D multi-object tracking. URL <https://arxiv.org/abs/2403.13443>, accessed 2024-11, 2403.13443.
- Li, X., T. Xie, D. Liu, J. Gao, K. Dai, Z. Jiang, L. Zhao, and K. Wang, 2023: Poly-MOT: A polyhedral framework for 3D multi-object tracking. *2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 9391–9398, <https://doi.org/10.1109/IROS55552.2023.10341778>.
- Liang, M., and F. Meyer, 2022: Neural enhanced belief propagation for data association in multiobject tracking. *2022 25th International Conference on Information Fusion (FUSION)*, 1–7, <https://doi.org/10.23919/FUSION49751.2022.9841258>.
- Lin, T.-Y., P. Dollar, R. Girshick, K. He, B. Hariharan, and S. Belongie, 2017: Feature pyramid networks for object detection. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.
- Lin, T.-Y., M. Maire, S. Belongie, L. Bourdev, R. Girshick, J. Hays, P. Perona, and D. Ramanan, 2015: Microsoft COCO: Common objects in context. URL <https://arxiv.org/abs/1405.0312>, accessed 2025-03, 1405.0312.
- Lin, X., Z. Pei, T. Lin, L. Huang, and Z. Su, 2023: Sparse4D v3: Advancing end-to-end 3D detection and tracking. URL <https://arxiv.org/abs/2311.11722>, accessed 2024-10, 2311.11722.
- Lin, Z., Z. Liu, Y. Wang, L. Zhang, and C. Zhu, 2024: RCBEVDet++: Toward high-accuracy radar-camera fusion 3D perception network. URL <https://arxiv.org/abs/2409.04979>, accessed 2025-03, 2409.04979.
- Long, Y., A. Kumar, D. Morris, X. Liu, M. Castro, and P. Chakravarty, 2023: RADIANT: Radar-image association network for 3D object detection. *Proceedings of the AAAI Conference on Artificial Intelligence*, **37**, 1808–1816, <https://doi.org/10.1609/aaai.v37i2.25270>, URL <https://ojs.aaai.org/index.php/AAAI/article/view/25270>.
- Mark A. Richards, J. A. H., William L. Melvin, 2005: Precision and accuracy in radar measurements. *Principles of Modern Radar*, Scitech.

- Nabati, R., L. Harris, and H. Qi, 2021: Cftrack: Center-based radar and camera fusion for 3D multi-object tracking. *2021 IEEE Intelligent Vehicles Symposium Workshops (IV Workshops)*, 243–248, <https://doi.org/10.1109/IVWorkshops54471.2021.9669223>.
- Pan, Z., F. Ding, H. Zhong, and C. X. Lu, 2024: RaTrack: Moving object detection and tracking with 4D radar point cloud. *2024 IEEE International Conference on Robotics and Automation (ICRA)*, 4480–4487, <https://doi.org/10.1109/ICRA57147.2024.10610368>.
- Patole, S. M., M. Torlak, D. Wang, and M. Ali, 2017: Automotive radars: A review of signal processing techniques. *IEEE Signal Processing Magazine*, **34** (2), 22–35, <https://doi.org/10.1109/MSP.2016.2628914>.
- Qi, C. R., L. Yi, H. Su, and L. J. Guibas, 2017: PointNet++: Deep hierarchical feature learning on point sets in a metric space. Accessed 2024-10, 1706.02413.
- Raghavan, R., 1992: Analysis of CA-CFAR processors for linear-law detection. *IEEE Transactions on Aerospace and Electronic Systems*, **28** (3), 661–665, <https://doi.org/10.1109/7.256288>.
- Rezatofighi, H., N. Tsoi, J. Gwak, A. Sadeghian, I. Reid, and S. Savarese, 2019: Generalized Intersection Over Union: A metric and a loss for bounding box regression. *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*.
- Shi, K., S. He, Z. Shi, A. Chen, Z. Xiong, J. Chen, and J. Luo, 2024: Radar and camera fusion for object detection and tracking: A comprehensive survey. URL <https://arxiv.org/abs/2410.19872>, accessed 2024-11, 2410.19872.
- Singer, R. A., 1970: Estimating optimal tracking filter performance for manned maneuvering targets. *IEEE Transactions on Aerospace and Electronic Systems*, **AES-6** (4), 473–483, <https://doi.org/10.1109/TAES.1970.310128>.
- Svenningsson, P., F. Fioranelli, and A. Yarovoy, 2021: Radar-PointGNN: Graph based object recognition for unstructured radar point-cloud data. *2021 IEEE Radar Conference (RadarConf21)*, 1–6, <https://doi.org/10.1109/RadarConf2147009.2021.9455172>.
- Thomas, H., Y.-H. H. Tsai, T. D. Barfoot, and J. Zhang, 2024: KPConvX: Modernizing kernel point convolution with kernel attention. *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 5525–5535.
- Vasuki, P., C. Bhavana, S. Mohamed Mansoor Roomi, and E. Lakshmi Deebikaa, 2012: Automatic noise identification in images using moments and neural network. *2012 International Conference on Machine Vision and Image Processing (MVIP)*, 61–64, <https://doi.org/10.1109/MVIP.2012.6428761>.

- Vaswani, A., N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. u. Kaiser, and I. Polosukhin, 2017: Attention is all you need. *Advances in Neural Information Processing Systems*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds., Curran Associates, Inc., Vol. 30, URL https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf.
- Wang, T., X. Zhu, J. Pang, and D. Lin, 2021: FCOS3D: Fully Convolutional One-Stage Monocular 3D Object Detection. *2021 IEEE/CVF International Conference on Computer Vision Workshops (ICCVW)*, 913–922, <https://doi.org/10.1109/ICCVW54120.2021.00107>.
- Wang, X., S. Qi, J. Zhao, H. Zhou, S. Zhang, G. Wang, K. Tu, and S. Guo, 2024: MCTrack: A unified 3D multi-object tracking framework for autonomous driving. URL <https://arxiv.org/abs/2409.16149>, accessed 2024-11, 2409.16149.
- Wei, K., Y. Fu, J. Yang, and H. Huang, 2020: A Physics-Based Noise Formation Model for Extreme Low-Light Raw Denoising . *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, IEEE Computer Society, Los Alamitos, CA, USA, 2755–2764, <https://doi.org/10.1109/CVPR42600.2020.00283>, URL <https://doi.ieeecomputersociety.org/10.1109/CVPR42600.2020.00283>.
- Weng, X., J. Wang, D. Held, and K. Kitani, 2020: 3D Multi-Object Tracking: A Baseline and New Evaluation Metrics.
- William L. Melvin, J. A. S., 2005: *Principles of Modern Radar*. Scitech.
- Wolters, P., J. Gilg, T. Teepe, F. Herzog, A. Laouichi, M. Hofmann, and G. Rigoll, 2024: Unleashing HyDRa: Hybrid fusion, depth consistency and radar for unified 3D perception. URL <https://arxiv.org/abs/2403.07746>, accessed 2024-07, 2403.07746.
- Wu, R., Z. Li, J. Wang, X. Xu, H. Yu, Z. Zheng, K. Huang, and G. Lu, 2025: Diffusion-based mmWave radar point cloud enhancement driven by range images. URL <https://arxiv.org/abs/2503.02300>, accessed 2025-04, 2503.02300.
- Yin, T., X. Zhou, and P. Krähenbühl, 2021: Center-based 3D object detection and tracking. *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 11 779–11 788, <https://doi.org/10.1109/CVPR46437.2021.01161>.
- Zamir, S. W., A. Arora, S. Khan, M. Hayat, F. S. Khan, M.-H. Yang, and L. Shao, 2020: Cycleisp: Real image restoration via improved data synthesis. *CVPR*.
- Zhang, P., X. Li, L. He, and X. Lin, 2023: 3D multiple object tracking on autonomous driving: A literature review. URL <https://arxiv.org/abs/2309.15411>, accessed 2024-11, 2309.15411.

- Zhou, X., V. Koltun, and P. Krähenbühl, 2020: Tracking objects as points. *Computer Vision – ECCV 2020*, A. Vedaldi, H. Bischof, T. Brox, and J.-M. Frahm, Eds., Springer International Publishing, Cham, 474–490.
- Zhu, X., W. Su, L. Lu, B. Li, X. Wang, and J. Dai, 2021: Deformable DETR: Deformable transformers for end-to-end object detection. Accessed 2024-10, 2010.04159.
- Zong, Z., G. Song, and Y. Liu, 2023: DETRs with collaborative hybrid assignments training. *2023 IEEE/CVF International Conference on Computer Vision (ICCV)*, 6725–6735, <https://doi.org/10.1109/ICCV51070.2023.00621>.

1 Appendix A

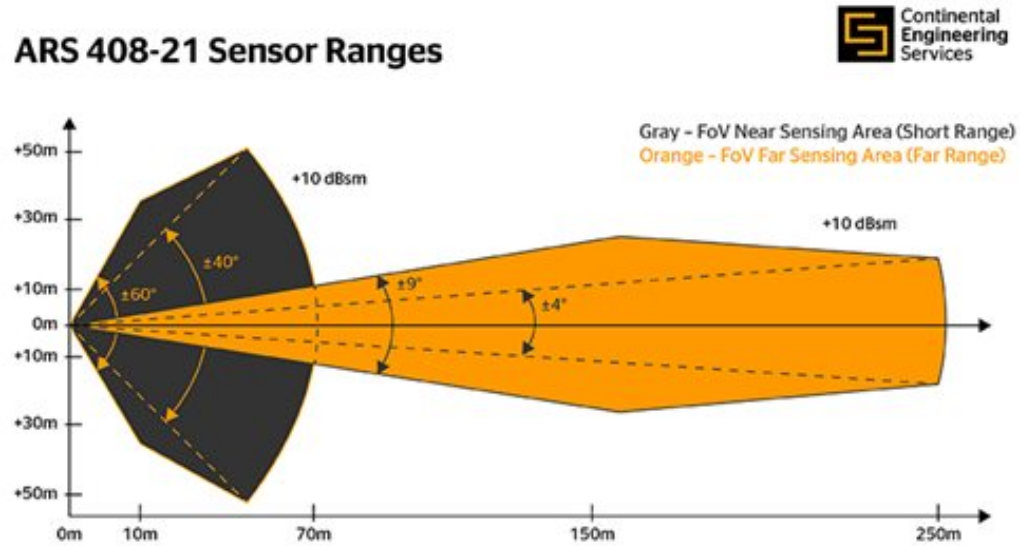


Figure A.1: The different range modes of the ARS-408-21



ARS 408-21 Long Range Radar Sensor 77 GHz

Safe - reliable - robust - very small design

Continental offers a new type of radar sensor, the **ARS 408-21**, as a possible adaption in different application and as entry version of the series 40X.

Measuring procedure

The rugged **ARS 408-21** sensor from Continental measures independent the distance and velocity (Doppler's principle) to objects without reflector in one measuring cycle due basis of FMCW (Frequency Modulated Continuous Wave) with very fast ramps, with a real time scanning of 17 / sec.. A special feature of the device is the simultaneously measurement of great distances up to 250 m, relative velocity and the angle relation between 2 objects.

Typical areas of application

- › Anti-collision protection for vehicles of every description (particul. autonomous)
- › Headway control also for far range (vehicles of every description, particularly autonomous)
- › Area monitoring system for far range, e.g. of hazardous or non-accessible areas
- › Classification of objects
- › Object detection, e.g. in confusing or unclear areas
- › Unremarkable object detection by affix a protection cover before it (radome)

Advantages

- › **Fast and safe:** The **ARS 408-21** dispels with the apparent contradiction between excellent great measuring performance and a high degree of operational safety. The rugged **ARS 408-21** radar sensor is capable of determining the distance to an object in real time scanning and dependent on the driving speed a possible risk of collision.
- › **Reliable:** The **ARS 408-21** radar sensor is fail-safe and able to recognize troubles of the sensor and sensor environment and display it automatically.
- › **Robust and very small design:** By using a radar technology with less complex measuring principle and the development and mass production in automotive supply industry, the design is kept very robust and very small.

ARS 408-21 Long Range Radar Sensor 77 GHz - Data Sheet

Measuring performance	Comment	to natural targets (non-reflector targets)
Distance range		0.20 ... 250 m far range, 0.20...70m/100m@0...±45° near range and 0.20...20m@±60° near range
Resolution distance measuring	point targets, no tracking	Up to 1.79 m far range, 0.39 m near range
Accuracy distance measuring	point targets, no tracking	±0.40 m far range, ±0.10 m near range
Azimuth angle augmentation	(field of view FoV)	-9.0°...+9.0° far range, -60°...+60° near range
Elevation angle augmentation	(field of view FoV)	14° far range, 20° near range
Azimuth beam width (3 dB)		2.2° far range, 4.4°@0° / 6.2°@±45° / 17°@±60° near range
Resolution azimuth angle	point targets, no tracking	31.6° far range, 3.2°@0° / 4.5°@±45° / 12.3°@±60° near range
Accuracy azimuth angle	point targets, no tracking	±0.1° far range, ±0.3°@0°/ ±1°@±45°/ ±5°@±60° near range
Velocity range		-400 km/h...+200 km/h (- leaving objects...+approximation)
Velocity resolution	target separation ability	0.37 km/h far field, 0.43 km/h near range
Velocity accuracy	point targets	±0.1 km/h
Cycle time		app. 72 ms near and far measurement
Antenna channels / -principle	microstripe	4TX/2x6RX = 24 channels = 2TX/6RX far - 2TX/6RX near / Digital Beam Forming
Operating conditions	Comment	to natural targets (non-reflector targets)
Radar operating frequency band	acc. ETSI & FCC	76...77 GHz
Mains power supply	at 12 V DC / 24 V DC	+8,0 V...32 V DC
Power consumption	at 12 V DC / 10 A fuse	6.6 W / 550 mA typ. and 12 W / 1.0 A @max. peak power
Load dump protection internal		disconnection >60 V and re-start returning to <60 V
Operating-/ storage temperature		-40°C...+85°C / -40°C...+90°C
Life time	acc. LV124 part 2 - v1.3	10000 h or 10 years (for passenger cars)
Shock	mechanical	500 m/s ² @6 ms half-sine (10 x shock each in +/-X/Y/Z dir)
Vibration	mechanical	20 [(m/s ²)/Hz]@10 Hz / 0,14 [(m/s ²)/Hz]@1000Hz (peak)
Protection rating	ISO 16750 Classification (Trucks) for vibration	IP 6k 9k (dust, high-pressure cleaning) IP 6k7 (10 cm under water), ice-water shock test, salt fog resistant, mixed gas EN 60068-2-60

ARS 408-21 Long Range Radar Sensor 77 GHz - Data Sheet

Connections	Comment	to natural targets (non-reflector targets)
Monitoring function		self monitoring (fail-safe designed)
Interface	up to 8 ID	1 x CAN - high-speed 500 kbit/s
Housing	Comment	to natural targets (non-reflector targets)
Dimensions / weight	L * W * H (mm) / (mass)	138 * 91 * 31 / app. 320 g
Material	housing front / backcover	PBT GF 30 black (BASF-Ultradur B4300G6 LS sw 15073) / AC-47100 (AlSi12Cu1(Fe)) die cast aluminium or EN AW 5754 (3.535) AlMg3 pressed-formed aluminium
Miscellaneous		
Measuring principle (Doppler's principle) in one measuring cycle due basis of FMCW with very fast ramps independent measurement of distance and velocity Version ARS 408-21 sensor for the industry CAN protocol for free communication The version -21 allows to set maximum 8 ID's and maximum 8 collision avoidance regions and to change the sensitivity between low and high sensitivity by the user continuously Interfaces: The device is fitted with one CAN bus interface. Further interfaces as converter, software adaption are possible on demand and in case of assumption of costs. A special software version supports a measurement range up to 1200m (For objects with high RCS and a free field of view)		



Print ID: 091018_FS_ARS_408_21_EN_H5

Continental Engineering Services
Graf-Vollrath-Weg 6 • D-60489 Frankfurt/Main
Tel. +49 69 67 86 96 -0 • Fax +49 69 67 86 96 -070
www.conti-engineering.com



Page 3 of 3

Figure A.2: Continental™'s ARS-408-21 Long Range Radar Sensor 77 GHz datasheet