Radar-Camera Fusion with Deep Neural Networks for Automotive Object Detection

Thesis approved by
the Department of Computer Science
University of Kaiserslautern-Landau
for the award of the Doctoral Degree
Doctor of Engineering (Dr.-Ing.)

to

Lukas Stefan Stäcker

Date of Defense	09.05.2025
Dean	Prof. Dr. Christoph Garth
Reviewer	Prof. Dr. Didier Stricker
Reviewer	Prof. Dr. Christoph Stiller

DE-386

Abstract

Reliable environment perception is critical for advanced driver assistance systems (ADAS) and automated driving systems, which rely heavily on sensor data from cameras, LiDARs, and radars. This dissertation focuses on the promising yet under-researched area of radar-camera fusion via deep neural networks. Radars and cameras complement each other well, with radars providing robust measurements in challenging conditions and cameras offering high-resolution visual data for object classification and accurate localization.

Radar-camera fusion presents significant challenges due to the difference in semantic level. Integrating sparse radar point clouds with detailed images raises questions about the optimal fusion point within the neural network. This thesis introduces Fusion Point Pruning, a technique that identifies ideal fusion points during training. Additionally, a novel projection method maps enriched radar data onto the image plane that increases data density based on azimuth uncertainty, which improves 2D object detection.

Another challenge is the different measurement coordinates of radars and cameras. Traditional fusion methods either lose radar's geometric information when merged on the image plane or require high computational resources to fuse in 3D space. This thesis proposes RC-BEVFusion, a new approach that merges radar and camera data in the bird's eye view plane, leveraging camera-based object detection advancements. This method utilizes two radar encoders and integrates seamlessly with existing image-only detection networks, significantly enhancing 3D object detection performance.

An additional cross-dataset study on sensor quality and scale indicates that image networks benefit from large-scale, varied visual inputs, while radar networks rely more on resolution than volume.

To showcase the real-world applicability of the proposed algorithms, a data processing pipeline for a test vehicle equipped with multiple sensors was developed. This includes extensive data pre-processing, calibration, and real-time data management. A new dataset was generated using a proposed automated labeling pipeline based on LiDAR data. Domain adaptation techniques were employed to enhance network performance on the generated dataset. The optimized networks were then tested in real-time on an Edge AI device, balancing computational efficiency and detection accuracy. The findings contribute valuable insights into sensor fusion and automotive object detection.

Acknowledgements

I want to take the opportunity to express my gratitude to all the people that supported me on this journey over the last four years. First, I want to thank the German Research Center for Artificial Intelligence and the RPTU Kaiserslautern-Landau for the opportunity to pursue my PhD. Special thanks go out to my doctoral advisor Prof. Didier Stricker and my supervisor Jason Rambach for the fruitful discussions and their invaluable scientific guidance. I would also like to thank Prof. Christoph Stiller for taking his valuable time as a second examiner.

Next, I want to thank Stellantis and Nikolas Wagner for hiring me in their PhD program as well as Frank Bonarens for his guidance at crucial moments of this journey. My special thanks go to my supervisor Philipp Heidenreich for his inexhaustible advice, which is only topped by his sense of humor on the workplace.

Finally, I would like to thank my friends and family for all the joy they brought me. Most notably my parents, whose support lead me to this position and continues to open up possibilities. Further, I would like to thank my friend Tristan Eckart for helping me find the right topic for my PhD. Last but not least I want to thank my lovely wife Milène, for enduring me during the times when it was stressful, cheering me up when it was difficult, and motivating me when I needed it most.

Without all of you, this work would not have been possible!

June 13, 2025, Lukas Stefan Stäcker

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Imagine a world with autonomous vehicles. A world, in which the elderly and the impaired re-gained their mobility. A world, in which up to 94 % of accidents formerly caused by human error could be avoided [1]. A world, in which cities could use space formerly occupied by parked vehicles to realize new qualities of urban life [2].

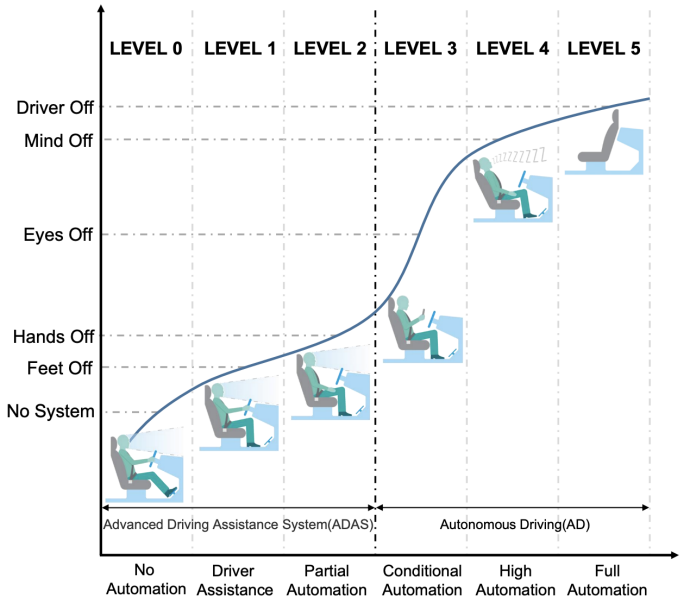


Figure 1.1.: *SAE taxonomy describing the levels of automation [3].*

While the potentials of autonomous vehicles seem endless, so do the challenges. The automotive industry has been working on them for many years. According to the SAE taxonomy depicted in Figure 1.1, there are five levels of automation that can be achieved. Most cars produced today include at least some form of advanced driver assistant system (ADAS) like adaptive cruise control or lane keeping assistant, but are limited to Level 1 or 2, where the driver has to remain fully engaged. Although there are first regulatory ap-

proved Level 3 systems for traffic jam chauffeurs up to 60 km/h and even a Level 4 parking assistant, there is still a long way to achieve extensive Level 4 or fully autonomous Level 5 systems [3], [4]. In addition to the technical difficulties, many challenges like safety validation, regulation, liability and societal acceptance have to be solved.

In this thesis, we aim to contribute towards this development in the realm of environment perception, which plays a key role in the automated driving pipeline shown in Figure 1.2. We choose the sub-task of object detection to research the potential of radar-camera fusion systems, aiming for more reliable perception and increasing the overall safety of automated driving systems.

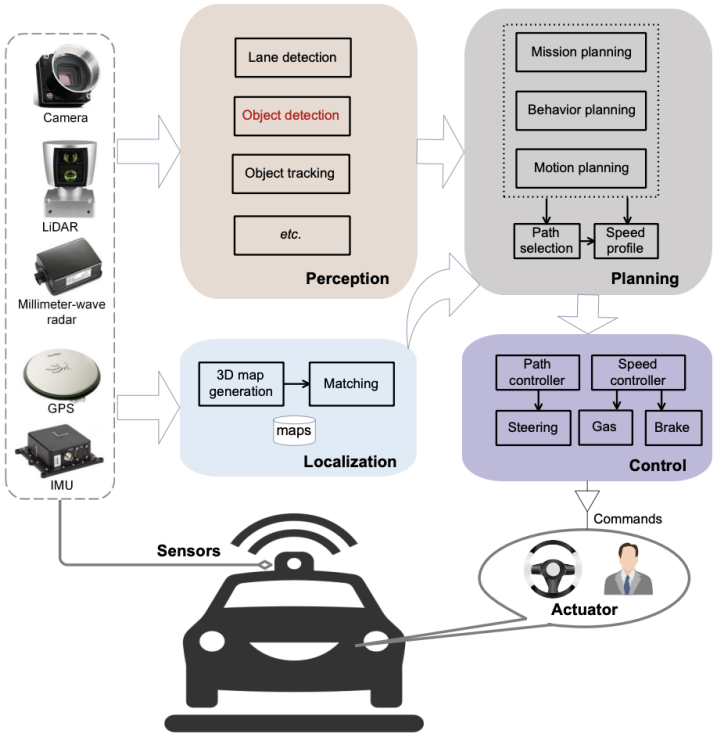


Figure 1.2.: *Overview of an automated driving system [5]. This thesis works on object detection, a sub-task of environment perception.*

1.1. Motivation

The reason for combining radar and camera systems is based on their different strengths and weaknesses. Cameras provide detailed visual information, which is crucial for recognizing objects, seeing colors, and reading traffic signs. However, cameras do not perform well in bad weather or low light. On the other hand, radar is reliable in these difficult conditions and is good at measuring the distance and speed of objects, but it cannot classify objects as accurately as cameras [6]. The combination of the detailed images from cameras and the reliable measurements from radar promises more robust object detection in a

wide range of conditions. The goal of radar-camera fusion is to gain a result that offers more than just the sum of two systems by leveraging the inherent synergies.

Given the potential of radar-camera fusion, there has been surprisingly little research in the field. Looking at recent surveys on automotive object detection, radar is, if mentioned at all, merely addressed as a side-note [3], [4], [7]. This is partially due to a lack of public datasets covering radar and camera data, which has only been addressed in recent years and was soon followed by a surge of publications in the field. Another reason is that a lot of research focus on sensor fusion lies in the combination of LiDAR and camera. While this combination can offer good detection performance in many conditions, it can fail for adverse weather conditions and is less suited for automotive series integration. In this thesis, we aim to close this research gap by exploring various approaches to radar-camera fusion and demonstrate their real-world applicability by integrating them into our own AI Sensing test vehicle. The aim is to provide a clear and thorough understanding of the technical details, challenges and practical applicability of the technology and to contribute to its advancement in making cars safer and smarter.

1.2. Data-based perception

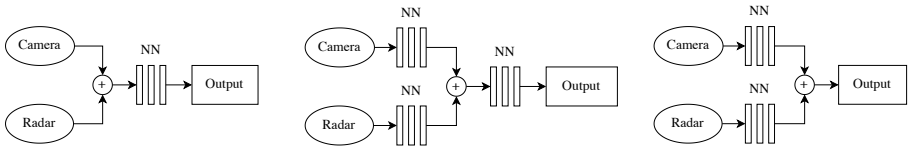
The field of automotive environment perception has undergone a significant change, moving away from traditional rule-based perception systems to modern data-driven techniques. Rule-based perception systems were initially popular because they use clear, predefined rules and models to detect objects. These systems depend heavily on the expertise of the people who program them, applying specific algorithms designed to interpret sensor data within defined parameters. While effective in controlled settings, these systems struggle with the unpredictability and complexity of real-world driving environments, leading to limitations in their flexibility and performance [8, p. 3].

The landscape began to change with the introduction of deep neural networks (DNNs), a development that started making waves with the creation of AlexNet [9]. Unlike their rule-based predecessors, DNNs learn to identify features and patterns directly from data, without needing explicit instructions. Their ability to learn and improve from examples has made them particularly powerful for detecting objects in the varied and dynamic scenarios encountered on the roads.

DNNs for environment perception are typically trained using supervised learning [8, p. 163], which relies on large amounts of labeled data. The performance of these networks is closely tied to the volume and quality of data they are trained on. The more accurate and diverse the training data, the better the network can learn to detect objects under different conditions. Thus, research of DNNs for automotive environment perception depends on the availability of public datasets. However, the need for extensive datasets introduces its own set of challenges, including the efforts required to collect and label this data accurately.

1.3. Sensor fusion topology

Sensor fusion involves combining data from multiple sensors to improve the accuracy, robustness, and reliability of systems. As shown in Figure 1.3, sensor fusion with DNNs can be broadly categorized into three types: early fusion, deep fusion, and late fusion [6], [10].



(a) *Early fusion* **(b)** *Deep fusion* **(c)** *Late fusion*

Figure 1.3.: *Sensor fusion topology.*

Early fusion refers to the combination of raw sensor data, like pixels from cameras or point clouds from radar or LiDAR sensors, before applying detection algorithms. The main advantage of early fusion is its theoretical ability to fully utilize the synergies when combining the strengths of different sensor types. However, it faces several challenges: Merging data from sensors with inherently different coordinate systems poses a significant challenge and early fusion can suffer from temporal misalignments and calibration errors. Additionally, there is a challenge in combining data with different semantic levels, such as raw camera pixels with very low-level information and radar point clouds, which are closer to object level. Finally, validating early fusion systems can be problematic because the full pipeline depends on various modalities, which makes it difficult to exchange components [10].

In contrast, late fusion involves combining data after applying detection algorithms, usually at the final regression layer or at the object level. This method is more modular and therefore has the advantage of being easier to validate because each sensor's data is processed separately. The downside is that late fusion may not fully utilize the potential strengths of combining different sensors. Objects that are missed by each sensor individually will also be missed by the sensor fusion system [6].

Deep fusion provides a balance, combining data within the layers of a deep neural network, using features extracted from each sensor. This approach allows for more flexibility in adjusting the semantic level of features for fusion, while still utilizing the synergies of the different sensors after the fusion. However, deep fusion still poses challenges in validation because the network partially relies on inputs from multiple sensors. Empirically, deep fusion tends to yield the best performance, which is why most recent research has focused on deep sensor fusion [6].

1.4. Scope

In this thesis, we research how to ideally fuse radar and camera data with deep neural networks for automotive object detection. Because of the poten-

tial synergies when combining the complementary strengths of cameras and radars, we focus on fusing these two sensor modalities and exclude fusion with other modalities like LiDAR or ultrasound. Further, we focus on radar data from typical automotive millimeter-wave radars. We therefore use radar point cloud data as provided by most automotive radars and exclude raw radar data formats. Similarly, we focus on monocular cameras and exclude stereo or thermal cameras.

While automated driving functions ultimately require 3D information, much of the research in image-based automotive object detection deals with the task of 2D object detection on the image plane. We therefore decide to add a contribution on 2D object detection with radar-camera fusion, before focusing the remainder of the thesis on 3D object detection. We research solely sensor fusion methods utilizing deep neural networks, as they provide state-of-the-art results for object detection. Consequently, we exclude late or object level fusion approaches, which do not use neural networks to leverage the synergies of the sensor modalities. Instead, we focus on deep fusion techniques.

In some aspects of this thesis, we use techniques that extend the scope of radar-camera fusion with deep neural networks. This includes the data processing pipeline of our experimental AI Sensing test vehicle and an automated labeling technique using LiDAR-based object detection with subsequent tracking and post-processing. We note that we use these techniques merely as a tool that ultimately allows us to research the practicability and generalization capabilities of our proposed radar-camera fusion algorithms.

1.5. Outline

This thesis is organized as follows: In Chapter 2, we introduce the foundational knowledge required to understand the contributions of our work. We give an overview on the sensor modalities, deep learning and object tracking techniques as well as the recent related work in the field of object detection with cameras, radars and sensor fusion. In Chapter 3 we present the Fusion Point Pruning (FPP) technique, our contribution to enhance the performance of 2D object detection models with radar-camera fusion. We further propose a novel projection technique of radar points onto the image plane, which takes azimuth angle uncertainty into account. Chapter 4 deals with our contributions in the realm of 3D object detection. We present a novel radar-camera fusion network architecture based on Bird's Eye View (BEV) features and propose two novel radar encoder branches that can be used as a plug-in system in various state-of-the-art image-only object detection algorithms. We further use the proposed architecture for a study on generalization capabilities across two datasets with different sensor setups including a recent high-performance radar sensor. In Chapter 5, we present our AI Sensing test vehicle along with the data preparation steps and the real-time data processing pipeline. Chapter 6 presents and evaluates an automized labeling pipeline using a LiDAR-based 3D object detection network and subsequent tracking and post-processing steps. This technique is used in Chapter 7 to create an

automatically labeled dataset with data from our AI Sensing vehicle. We conduct a study on algorithm deployment on an Edge AI system in the vehicle, which we use to evaluate the performance of our proposed algorithms in terms of their generalization capability and real-world applicability. In Chapter 8, we conclude the findings of this thesis.

To summarize, the main contributions of this dissertation include:

- Fusion Point Pruning: a technique to automatically find the best fusion points in a neural network.
- RC-BEVFusion: A flexible radar-camera fusion architecture based on BEV features.
- The AI Sensing test vehicle setup.
- An automized labeling algorithm.
- A case study on network deployment and domain adaptation in real world conditions.

1.6. Publications

Most of the work presented in this thesis has been accepted and presented in peer-reviewed conferences or journals. In the following, we provide a list of the publications derived from this work in the order they appear in the thesis:

1. F. Engels, P. Heidenreich, M. Wintermantel, **L. Stäcker**, M. Al Kadi, and A. M. Zoubir, "Automotive Radar Signal Processing: Research Directions and Practical Challenges," *IEEE Journal of Selected Topics in Signal Processing*, vol. 15, no. 4, 2021, pp. 865–878 [11].
2. **L. Stäcker**, P. Heidenreich, J. Rambach, and D. Stricker, "Fusion Point Pruning for Optimized 2D Object Detection with Radar-Camera Fusion," in *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, 2022, pp. 3087–3094 [12].
3. **L. Stäcker**, S. Mishra, P. Heidenreich, J. Rambach, and D. Stricker, "RC- BEVFusion: A Plug-In Module for Radar-Camera Bird's Eye View Feature Fusion," in *DAGM German Conference on Pattern Recognition*, Springer, 2023, pp. 178–194 [13].
4. **L. Stäcker**, P. Heidenreich, J. Rambach, and D. Stricker, "Cross-Dataset Experimental Study of Radar-Camera Fusion in Bird's-Eye View," in *31st European Signal Processing Conference*, IEEE, 2023, pp. 810–814 [14].
5. **L. Stäcker**, J. Fei, P. Heidenreich, F. Bonarens, J. Rambach, D. Stricker, and C. Stiller, "Deployment of Deep Neural Networks for Object Detection on Edge AI Devices with Runtime Optimization," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2021, pp. 1015–1022 [15].

Chapter 2

Related work

Contents

In this chapter, we cover the technical foundations as well as recent related work in research needed to understand the contributions of this thesis. We start by introducing the three sensor modalities used in this thesis: camera, radar and LiDAR. In the following, we describe the basic principles of deep learning as well as the components of modern deep convolutional networks, followed by a short introduction of object tracking. We continue by giving an overview of the recent research results in object detection using images, radar

data as well as sensor fusion approaches. Finally, we list and categorize public datasets that can be used for object detection with radar-camera fusion. Some contents of Sections 2.1.2, 2.4, 2.5 and 2.6 have been published in [11], [12] and [13].

2.1. Sensor modalities

For automotive environment perception, several sensor modalities with different physical properties as well as their inherent advantages and disadvantages are used. In this chapter, we give an overview of the three most widely used sensors for automotive object perception, that are also relevant to this thesis: Cameras, Radars, and LiDARs [10]. Figure 2.1 shows the different advantages and disadvantages of each of these sensors. While cameras are cheap and well suited for object classification and precise localization within the image, radar sensors are more robust in adverse wheather and lighting conditions and directly measure depth and velocity of objects. Note that the characteristics of radar and camera complement each other well, indicating that sensor fusion with radar and camera has high potential. LiDARs on the other hand provide the most accurate 3D perception and are robust to different lighting conditions, but compared to radar they are less robust to adverse weather conditions. Also, at the time of writing, LiDARs are more expensive and therefore less suited for automotive series integration [6].

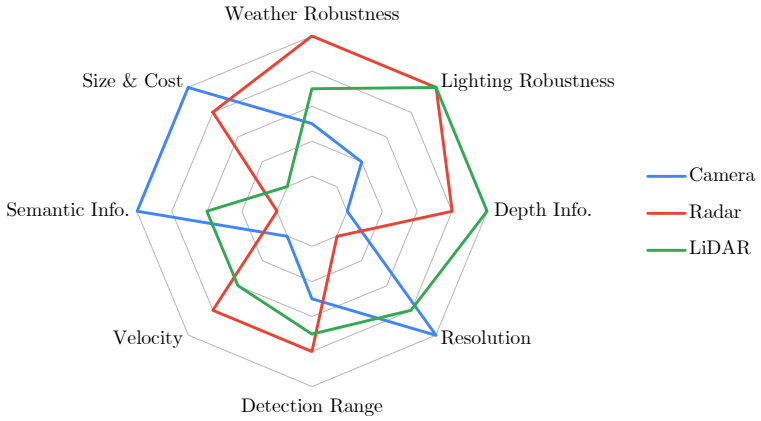


Figure 2.1.: *Sensor characteristics for automotive perception based on [6]. Radar-Camera fusion has high potential due to complementary characteristics.*

2.1.1. Camera

Cameras are passive, optical sensors that capture light from a scene. In a digital camera, a lens is used to focus light onto a photosensitive surface, typically a Charge-Coupled Device (CCD) or Complementary Metal-Oxide-Semiconductor (CMOS) image sensor, which converts the light rays into elec-

trons. These electrons are transformed into voltage, amplified, and converted to a digital image using an Analog-Digital-Converter (ADC) [16, p. 80].

Real lenses introduce distortion onto images as shown in Figure 2.2a. Note how the buildings on the left side and the poles on the right side of the image appear bent. It is necessary to calibrate cameras in order to rectify such distorted images as shown in Figure 2.2b. Calibration further serves as a prerequisite for accurately projecting 3D points onto an image or fusing data from depth sensors with image features. Camera calibration can be divided into extrinsic and intrinsic calibration.

(a) *Distorted image* **(b)** *Rectified image*

Figure 2.2.: *Example of a distorted image being rectified.*

The extrinsic calibration matrix $\boldsymbol{R}_{w\to c}$ is needed to project a point from the world coordinate system $\boldsymbol{p}_w$ into the camera coordinate system $\boldsymbol{p}_c$. Using homogeneous coordinates, the rotation and translation can be described in a single matrix multiplication as:

$$\boldsymbol{p}_c = \begin{bmatrix} x_c \\ y_c \\ z_c \\ 1 \end{bmatrix} = \boldsymbol{R}_{w\to c} \begin{bmatrix} x_w \\ y_w \\ z_w \\ 1 \end{bmatrix} = \boldsymbol{R}_{w\to c}\boldsymbol{p}_w \tag{2.1}$$

In order to project the point to 2D image coordinates, an intrinsic calibration matrix $\boldsymbol{R}_{c\to\text{img}}$ is needed. The underlying camera model is the widely used pinhole camera model [16, p. 55] shown in Figure 2.3. For an ideal, distortion-free lens, a point in the camera coordinate system $\boldsymbol{p}_c$ is mapped onto the image plane as $\boldsymbol{p}_\text{img}$ as:

$$s_\text{img}\boldsymbol{p}_\text{img} = s_\text{img} \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \boldsymbol{R}_{c\to\text{img}} \begin{bmatrix} x_c \\ y_c \\ z_c \\ 1 \end{bmatrix} = \boldsymbol{R}_{c\to\text{img}}\boldsymbol{p}_c \tag{2.2}$$

Here, s_img is a scalar factor and $\boldsymbol{R}_{c\to\text{img}}$ is the intrinsic camera calibration matrix that maps the point from camera to pixel coordinates and can be described as:

$$\boldsymbol{R}_{c\to\text{img}} = \begin{bmatrix} f_x & s & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \tag{2.3}$$

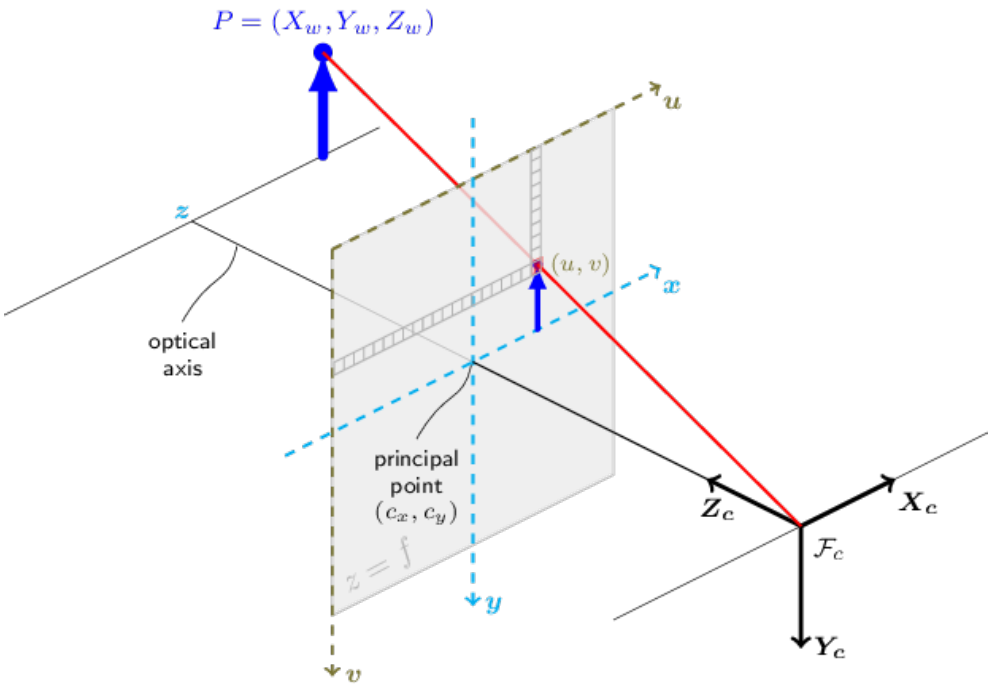


Figure 2.3.: *Pinhole camera model [17].*

Here, f_x and f_y are the focal lengths, c_x and c_y are the principal point or optical center, and s is a skew coefficient, which can be discarded for modern CCD cameras, where the image axes are perpendicular [16, p. 45]. Using these factors, the mapping from camera to pixel coordinates can be re-written as:

$$\begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} f_x x' + c_x \\ f_y y' + c_y \end{bmatrix} \tag{2.4}$$

Here, $x' = x_c/z_c$ and $y' = y_c/z_c$. When dealing with real lenses, distortion factors have to be taken into account. In this work, up to twelve distortion parameters are used: Six radial coefficients k_i, two tangential coefficients p_i and four thin prism coefficients s_i [17]. Given $r^2 = x'^2 + y'^2$, these distortion coefficients are used to extend the model as:

$$\begin{bmatrix} x'' \\ y'' \end{bmatrix} = \begin{bmatrix} \frac{1+k_1r^2+k_2r^4+k_3r^6}{1+k_4r^2+k_5r^4+k_6r^6} x' + 2p_1x'y' + p_2(r^2+2x'^2) + s_1r^2 + s_2r^4 \\ \frac{1+k_1r^2+k_2r^4+k_3r^6}{1+k_4r^2+k_5r^4+k_6r^6} y' + p_1(r^2+2y'^2) + 2p_2x'y' + s_3r^2 + s_4r^4 \end{bmatrix} \tag{2.5}$$

Consequently, for real lenses, the undistorted coordinates x'' and y'' are used for the projection to the image plane:

$$\begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} f_x x'' + c_x \\ f_y y'' + c_y \end{bmatrix} \tag{2.6}$$

2.1.2. Radar

Radar is an acronym for radio detection and ranging. Radar sensors are active electromagnetic sensors that detect, range, and map objects by emitting radio waves and analyzing the echoes reflected back. The transmit antenna emits a burst of radio or microwave frequency waves, which, upon hitting an object, reflect back to the sensor. The time taken for these waves to return is precisely measured, enabling the calculation of the object's distance based on the speed of light. Additionally, the direction of the object is determined by the direction of the reflected waves, and any change in their frequency, due to the Doppler effect, indicates the object's relative speed. Automotive radars use a microwave frequency band of 76 GHz to 81 GHz. Depending on the application, a trade-off between detection range and azimuth coverage has to be made, which classifies radars into short range (SRR), mid range (MRR), and long range radars (LRR) [18].

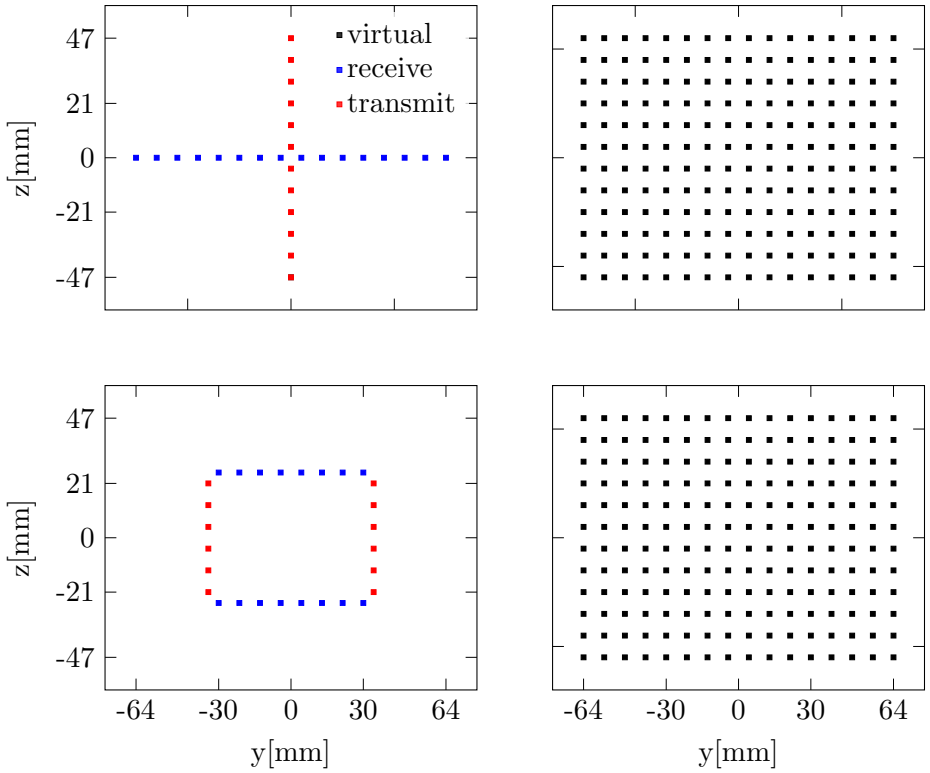


Figure 2.4.: *Two examples of MIMO virtual arrays based on [11]. Note how the lower configuration can be used with approximately half of the physical space.*

In current state-of-the-art automotive radar processing, Frequency Modulated Continuous Wave (FMCW) radars, particularly employing chirp sequence modulation with stretch processing, have emerged as a standard due to their hardware efficiency and cost-effectiveness [18]. Notably, the integration of Multiple-Input Multiple-Output (MIMO) antenna designs enhances

the number of effective receive antennas, facilitating larger array apertures [19]. An exemplary virtual array achievable with 12 transmit and 16 receive antennas is shown in Figure 2.4. Note that the same array can be achieved with different physical configurations, the lower being preferable due to its approximately 50 % lower space requirement [11].

The processing chain in these advanced radar systems is depicted in Figure 2.5. First, a MIMO signal model is used to mix the transmit signal with the receive signal and convert it to baseband samples. Due to the high likelihood of interference with automotive radar signals from other cars in the environment, an important part of the radar processing is interference mitigation [20]. Next, several pre-processing steps are used to convert the samples into a 4D radar cube with dimensions range, relative radial velocity, azimuth angle, and elevation angle. To this end, the Fast-Fourier-Tranform (FFT) algorithm is employed several times. First, a fast-time FFT is used, corresponding to the range dimension, before the MIMO signal is demodulated. Then, a slow-time FFT extracts the relative radial velocity correspondents. Finally, a 2D FFT along the horizontal and vertical dimensions of the MIMO array is used for the beamforming, corresponding to the azimuth and elevation angles [11].

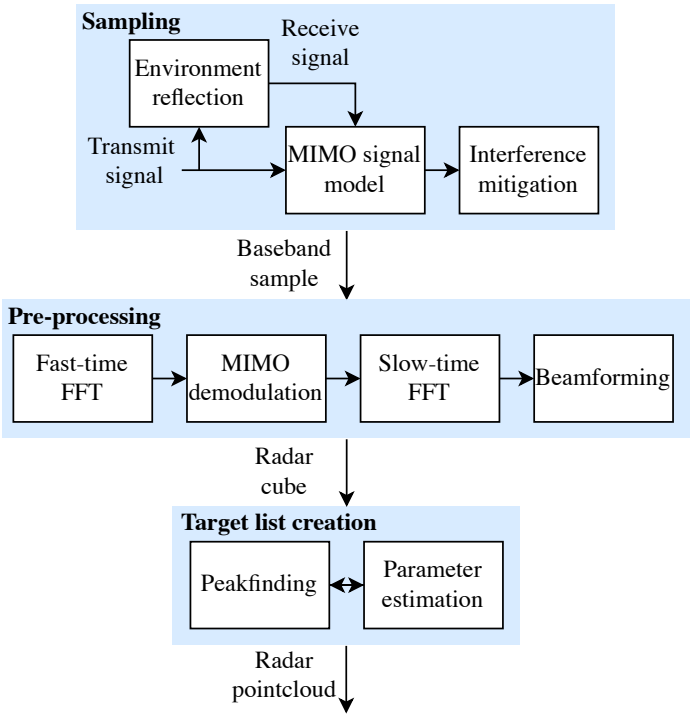


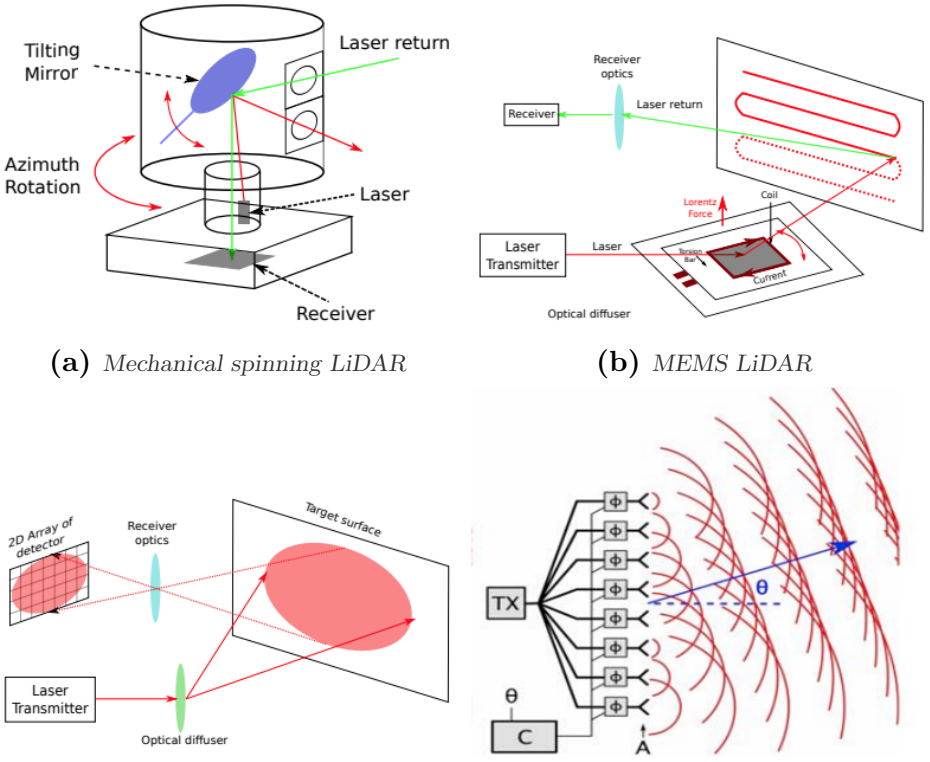
Figure 2.5.: *Overview of the radar signal processing chain based on [11].*

While the calculated 4D radar data cube is a useful intermediate raw data interface, it requires a lot of memory, and is generally not feasible to be sent via network to a processing unit at high frequencies. Therefore, a more compact target list is typically extracted from the radar data cube using constant false alarm rate (CFAR) power detection. This is often combined with peak

finding and sidelobe thresholding, to exclude artifacts from strong targets [11]. Using the identified peaks, the parameters of the targets can be estimated. In addition to estimating the four dimensions of the radar cube, its power values are used to estimate the radar cross section (RCS), a measure indicating how detectable an object is by radar, which depends on a target's size, shape and material. The parameter estimation finally results in the target list, also referred to as the radar point cloud. The radar point cloud is often used for further processing steps such as grid mapping or object detection, and is a useful interface for data fusion with data from other sensor modalities as performed in this thesis.

2.1.3. LiDAR

LiDAR is an acronym for Light Detection and Ranging. LiDARs are active optical sensors that emit near-infrared laser pulses and receive their reflections from the environment. A LiDAR sensor can be broadly separated into two parts: a laser rangefinder and a scanning system.



(a) *Mechanical spinning LiDAR* **(b)** *MEMS LiDAR*

(c) *Flash LiDAR* **(d)** *Optical Phased Array*

Figure 2.6.: *Different types of LiDAR scanning systems [21], © 2020, IEEE.*

The laser rangefinder consists of a laser transmitter, optics, a photodetector, and signal processing electronics. Most automotive LiDARs use a pulsed laser transmitter and measure the time-of-flight of the pulse reflections. Recent

LiDAR variants emit an FMCW signal, which can be used to simultaneously measure distance and relative radial velocity based on the Doppler effect. The optics are designed to collimate the emitted laser and focus the reflected photons. The photodetector captures them and performs optical processing and photoelectric conversion to generate an electronic signal. Finally, fast signal processing electronics estimate the distance between the sensor and the reflected target [21].

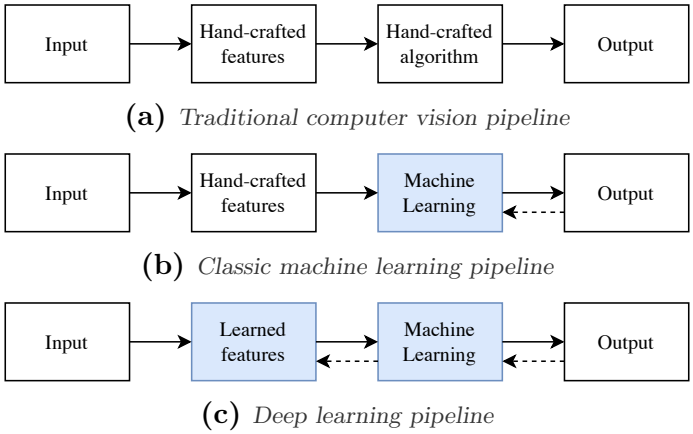
The scanning system is used to steer laser beams along different azimuth and elevation angles. As shown in Figure 2.6, there are different types of scanning systems, which can be grouped into mechanical spinning and solid state [21]. Mechanical spinning systems use a rotating element like a mirror or prism, which is controlled by a motor to re-direct laser beams and increase the field of view [22]. While mechanical spinning LiDARs are popular due to their high signal to noise ratio in combination with 360° horizontal field of view, they cannot be seamlessly integrated into the vehicle and are too fragile for automotive. In contrast, solid state LiDARs are more robust, but typically have a lower field of view. Different variants of solid state LiDARs include Micro-Electro-Mechanical Systems (MEMS) [23], which are miniature mirrors embedded on a chip, flash LiDARs [24], which illuminate the scene with an optically diffused laser and then act as a camera, and Optical Phased Arrays (OPA) [25], which use phase modulators to steer the laser beams.

2.2. Deep Learning

Deep learning describes a set of modern machine learning algorithms, that are based on deep neural networks. As shown in Figure 2.7, computer vision processing pipelines have evolved from traditional, hand-crafted feature and algorithm design to more data-based methods. In classic machine learning pipelines, a set of hand-crafted features was used to train a machine learning algorithm. In deep learning techniques, all components including intermediate feature representations are learned from data [16, p. 238]. Deep learning has become an essential technique for state-of-the-art environment perception algorithms. In this section, we give an overview on the basics of convolutional neural networks (CNNs) and introduce examples of common components of a modern deep neural network in the realm of computer vision: backbones, necks and detection heads. Further, we describe loss functions, optimizers and metrics used to train and evaluate the neural networks.

2.2.1. Convolutional neural networks

A CNN is a neural network that has one or more convolutional layers and is used mainly for image processing but can also be used for other auto-correlated data. Typically, modern CNNs are deep neural networks with many layers of different types. In the following, we present the most common components of CNNs.



(a) *Traditional computer vision pipeline*

(b) *Classic machine learning pipeline*

(c) *Deep learning pipeline*

Figure 2.7.: *Evolution of computer vision processing pipelines based on [16, p. 238].*

Convolutional layer The convolutional layer is the core building block of a CNN. The layer's parameters consist of a set of learnable filters or kernels, which have a small receptive field but extend through the full depth of the input volume. As illustrated in Figure 2.8, each filter is convolved across the width and height of the input volume, computing the dot product between the filter and the input, and producing an activation map of that filter. As a result, the network learns filters that activate when they see specific types of features at some spatial position in the input [16, p. 292].

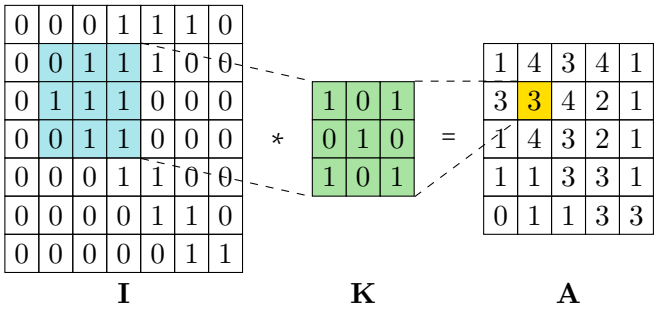


Figure 2.8.: *Convolution operation. The element-wise multiplication and summation between the filter* $\boldsymbol{K}$ *convolved over the input map* $\boldsymbol{I}$ *produces the activation map* $\boldsymbol{A}$*.*

A 2D convolution can be mathematically described using 3D tensors for input $\boldsymbol{I} \in \boldsymbol{R}^{w_I \times h_I \times c_I}$ and activation map $\boldsymbol{A} \in \boldsymbol{R}^{w_A \times h_A \times c_A}$ with width w, height h and feature channels c_I and c_A, respectively, as well as a 4D tensor for the kernel weights $\boldsymbol{K} \in \boldsymbol{R}^{k \times k \times c_I \times c_A}$ with uneven kernel size k. For a stride of $s_{conv} = 1$ and same padding $p = \frac{k-1}{2}$, an element of $\boldsymbol{A}$ at row m, column n and channel a can be calculated by:

$$\boldsymbol{A}[m, n, a] = \sum_{i=1}^{c_I} \sum_{u=1}^{k} \sum_{v=1}^{k} \boldsymbol{K}[u, v, i, a] \cdot \boldsymbol{I}[m + u - \frac{k+1}{2}, n + v - \frac{k+1}{2}, i] \quad (2.7)$$

Activation function Activation functions are crucial in CNNs as they introduce non-linear properties to the network. The most common activation function in CNNs is the Rectified Linear Unit (ReLU). ReLU is defined as $f(x) = max(0, x)$. In comparison with the traditionally popular sigmoid activation function, ReLU has two key advantages when used in deep neural networks: It solves the vanishing gradient problem that occurs when multiplying gradients with absolute values < 1 across many layers and it is computationally cheap which speeds up training. Variants of ReLU, like Leaky ReLU and Parametric ReLU, also exist to address specific issues like the dying ReLU problem [16, p. 273].

Pooling layer Pooling layers are used to reduce the spatial dimensions of the input volume for the layers that follow. The most common form of pooling is max pooling, where the maximum element from a region of the input is selected. Pooling helps in making the representation approximately invariant to small translations of the input. It also reduces the number of parameters and computation in the network [16, p. 295].

Fully connected layer In fully connected layers, neurons have full connections to all activations in the previous layer, as seen in multi-layer perceptrons (MLPs). Typically, fully connected layers are placed at the end of CNN architectures. Their purpose is to use the calculated high-level features for the final task of e.g. image classification, object detection, or semantic segmentation [16, p. 271].

2.2.2. Backbones

A backbone architecture is a part of a deep neural network that is used to encode high-level features from the raw input data. Due to the need of extracting many different semantic cues from the raw data, a common practice involves pretraining backbones on extensive datasets like ImageNet [9] before fine-tuning them for the respective task. In this section, we introduce two of the most common backbone architectures, which are also used in this work: ResNet [26] and SwinTransformer [27].

ResNet The most common backbone in computer vision is ResNet [26], short for residual network. It belongs to the family of convolutional backbones along with VGG [28], efficient models like MobileNet [29] and EfficientNet [30], as well as the more recent ConvNeXt [31]. This architecture emerged as a key innovation, particularly in computer vision, by enabling the training of networks with unprecedented depth. At the heart of ResNet's innovation is the concept of residual learning as shown in Figure 2.9, which fundamentally alters the approach of training deep networks. Traditional CNNs attempted to learn direct mappings from inputs to outputs, but ResNet layers are designed to learn residual functions with reference to the layer inputs. This change is operationalized through the introduction of skip connections, which effectively

allow the output of one layer to bypass one or more layers by being added to the output of a later layer. Mathematically, this translates to approximating the residual function $F(x) = H(x) - x$, thereby reformulating the original function as $F(x) + x$. These skip connections play a critical role in addressing the vanishing gradient problem, as they provide an alternate route for the gradient to flow through during backpropagation, thus aiding in the training of deep networks.

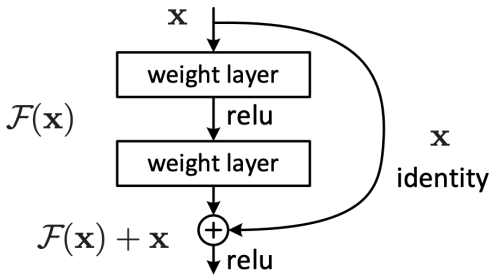


Figure 2.9.: *ResNet building block with residual connection [26], © 2016, IEEE.*

ResNet's architecture is characterized by a series of residual blocks, each comprising several convolutional layers followed by batch normalization and ReLU activation. The block's output is combined with its input through element-wise addition before undergoing the subsequent ReLU activation. Depending on the amount of residual blocks, the architecture comes in various configurations between 18 and 152 layers, which allows a trade-off between performance and computational requirements. A more modern version of ResNet is ResNeXt [32].

SwinTransformer SwinTransformers [27] represent a novel architectural development in the field of deep learning. This architecture, along with the VisionTransformer (ViT) [33], integrates the principles of transformers, traditionally used in natural language processing [34], into the realm of computer vision. Unlike standard transformers that operate on sequences of text tokens, SwinTransformers handle 2D images by partitioning them into non-overlapping patches and treating each patch as a token. These patches are then processed through a series of transformer blocks, which involve self-attention mechanisms and MLPs.

A unique aspect of the SwinTransformer is its use of shifted windows illustrated in Figure 2.10. In each successive layer of the network, the partitioning of patches into windows is shifted, allowing for cross-window connections and enhancing the ability of the model to capture relationships across different parts of the image. This approach addresses one of the key limitations of applying transformers to vision tasks: the computational complexity associated with global self-attention. By focusing on local windows and then gradually shifting these windows, SwinTransformers efficiently manage computational resources while still capturing the global context of the image.

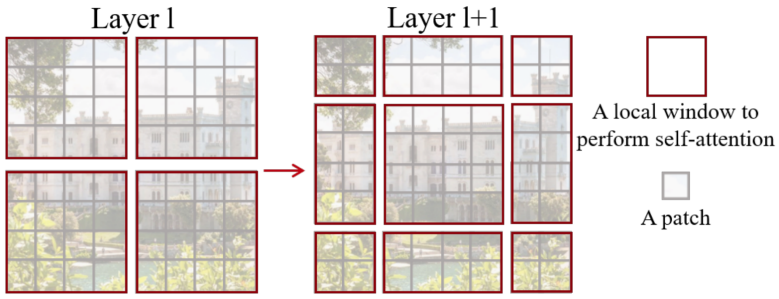


Figure 2.10.: *Shifted window approach utilized in the SwinTransformer [27], © 2021, IEEE.*

In contrast to previous approaches which apply transformers to vision like ViT [33], SwinTransformers [27] are designed to produce hierarchical feature representations, much like traditional CNNs. As shown in Figure 2.11, this hierarchical structure is achieved by progressively merging patches and reducing their number, effectively increasing the receptive field as the network gets deeper. Such a design allows SwinTransformers to maintain high-resolution feature maps in the initial layers, which are crucial for detailed feature extraction, while also capturing more abstract and global features in the deeper layers.

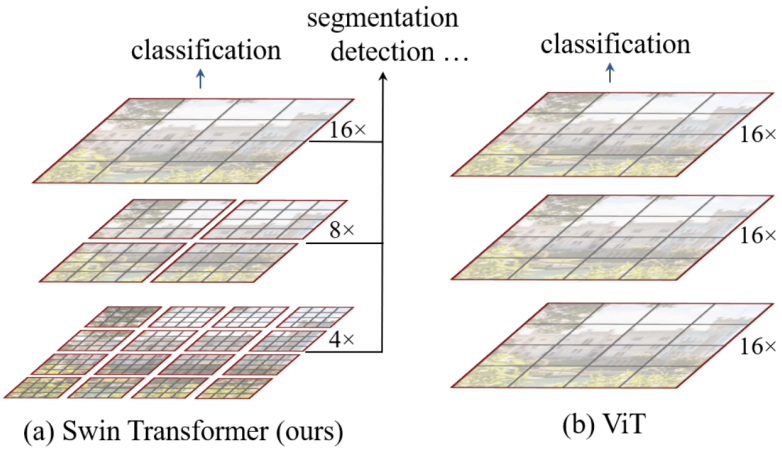


Figure 2.11.: *SwinTransformer makes use of hierarchical feature maps, which allows it to extract both high-resolution feature maps as well as abstract, global features in deeper layers [27], © 2021, IEEE.*

2.2.3. Necks

Neck structures in deep CNNs are used to bridge between the backbone used for feature extraction and the head used to produce the final outputs. Typically, necks are used to aggregate multi-scale feature maps and their architecture follows Feature Pyramid Networks (FPNs) [35].

Backbone architectures often produce feature maps at different levels, each corresponding to different scales of the input image. However, these feature maps diminish in resolution as the network gets deeper, making them less effective for detecting small objects. As shown in Figure 2.12, FPNs tackle this issue by augmenting the standard CNN architecture with a top-down pathway and lateral connections. The top-down pathway upsamples spatially coarser, semantically stronger features from higher levels of the network. These are then enhanced with spatially finer, but semantically weaker, features from lower levels through lateral connections. The result is a set of feature maps at multiple levels, each combining deep, semantically rich information with high-resolution details.

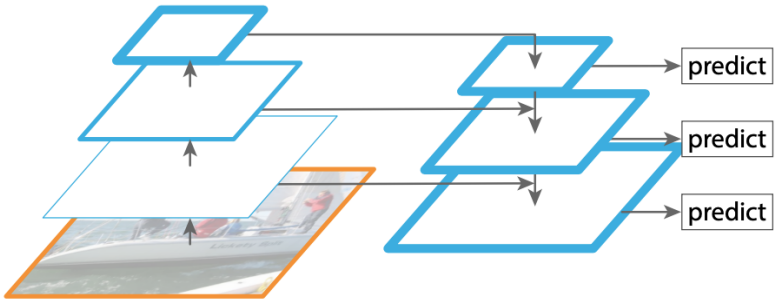


Figure 2.12.: *FPNs aggregate multi-scale feature maps produced from a backbone structure [35], © 2017, IEEE.*

2.2.4. Detection Heads

The detection head is a component of the network architecture that specifically focuses on interpreting the encoded features obtained from backbone and neck to predict the outputs needed to solve the respective task of e.g. semantic segmentation or object detection. Depending on the specific requirements of the task and the overall architecture of the network, detection heads generally consist of a series of convolutional layers, sometimes accompanied by fully connected layers. In the following, we focus on the most important detection heads used for object detection.

R-CNN [36] was among the first to introduce the concept of a dedicated detection head in CNNs. This architecture utilizes region proposal methods to identify areas of interest in an image and applies a detection head to classify and refine the bounding boxes around these proposed regions. Single Shot MultiBox Detector (SSD) [37] streamlined the detection process by eliminating the need for separate region proposals, integrating the detection head directly into the backbone network. This allowed for real-time detection performances, making the process more efficient. An approach that scales better for 3D detection was introduced in CenterNet [38]. Unlike the previous methods that focus on bounding box predictions, CenterNet adopts a keypoint-based approach. As shown in Figure 2.13, it detects objects as keypoint peaks in a heatmap, and regresses to all other object properties, such as size, depth, and

orientation. This approach simplifies the detection pipeline and has shown effectiveness in improving detection accuracy, especially in crowded scenes.

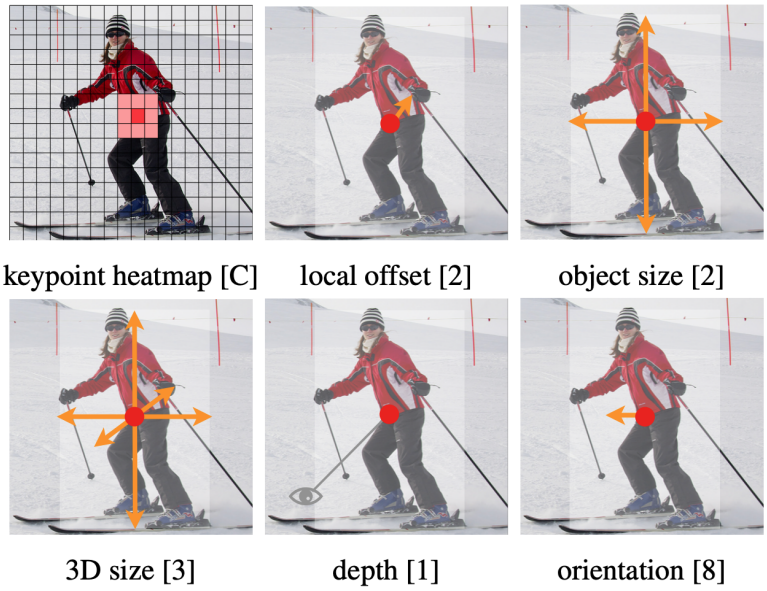


Figure 2.13.: *CenterNet treats objects as points and regresses to all other attributes needed to decode bounding boxes [38].*

2.2.5. Loss functions

The primary role of a loss function in CNN training is to measure the discrepancy between the predicted outputs of the network and the actual target values. This measure of error is used by optimization algorithms, like gradient descent, to update the network's weights in a direction that minimizes this error. The choice of an appropriate loss function is crucial as it directly influences how the weights of the network are adjusted during the training process. Therefore, the design of a loss function must reflect the specific objectives of the task at hand, such as penalizing incorrect classifications or inaccuracies in the localization of objects [16, p. 280].

A widely used loss function in classification tasks is the cross-entropy loss, which measures the performance of a classification model whose output is a probability value between 0 and 1. It is effective in cases where the model needs to output the probability of a class label being true and is particularly useful for training multi-class classification problems. For regression tasks, when the model needs to predict continuous values, L1 loss or L2 loss are commonly used, which correspond to the sum of all absolute or squared errors, respectively.

In the context of object detection, a frequent problem is the imbalance of classes in the training dataset. To address this, Focal Loss was introduced [39]. Focal Loss modifies the standard cross-entropy criterion by adding a factor that reduces the loss contribution from easy examples and focuses more on correcting misclassified examples. This adjustment allows the model to focus

on learning from the hard, misclassified examples, which is particularly useful when there is a significant imbalance between the background and foreground classes.

2.2.6. Optimizers

An optimizer is an algorithm used to change network weights to reduce the losses. To this end, backpropagation is used to determine the gradients of the network weights with respect to the loss function [8, p. 180]. Optimizers determine how quickly a network learns from the gradients and how it converges to minimize the loss function.

One of the foundational optimizers in the field of deep learning is Stochastic Gradient Descent (SGD). It uses mini-batches of training data to iteratively move in the direction of the steepest gradient descent. While SGD is simple and efficient, it can be sensitive to the choice of the learning rate and may show slow convergence on complex error surfaces common in deep learning tasks. Variants of SGD, such as SGD with momentum, have been introduced to overcome these limitations. The momentum term helps to accelerate SGD in the relevant direction and dampens oscillations, making it more robust to the choice of hyperparameters [8, p. 260].

A more recent, popular optimizer is Adam [40], which combines the advantages of two other extensions of SGD: Adaptive Gradient Algorithm (AdaGrad) [41] and Root Mean Square Propagation (RMSProp) [42]. Adam computes adaptive learning rates for each parameter. It stores an exponentially decaying average of past squared gradients and an exponentially decaying average of past gradients. These terms are estimates of the mean and variance of the gradients, which are corrected by the algorithm to counteract their bias towards zero. This bias correction helps Adam perform well in practice even with little tuning of hyperparameters [8, p. 267].

Building upon the success of Adam [40], the AdamW [43] optimizer presents a refinement that addresses specific shortcomings in the original Adam optimizer, particularly regarding weight decay regularization. Weight decay is a form of L2 regularization that shrinks weights during the training process to prevent overfitting, a crucial aspect in training deep neural networks. AdamW decouples the weight decay from the optimization steps, applying it directly to the weights similarly to SGD. This subtle yet impactful change allows AdamW to preserve the benefits of adaptive learning rates from Adam while improving the effectiveness of weight decay regularization.

2.2.7. Object Detection Metrics

To evaluate the performance of algorithms, metrics are needed that represent all important aspects of the downstream task. In this section, we introduce the most relevant metrics in the realm of object detection. A recent study has shown that these metrics, especially the nuScenes detection score (NDS) [44] and the mean average precision (mAP), correlate well with the driving score

and the number of collisions when deploying object detection algorithms in a driving simulation [45].

The performance of an object detection algorithm is evaluated based on two sets of bounding boxes: the predictions and the labeled ground truth. The first important step is to match predictions with ground truth boxes to determine so called True Positives (TP). Predictions that have no matching ground truth box are called False Positives (FP). Similarly, ground truth boxes that have no matching prediction are called False Negatives (FN).

To determine a TP match, many datasets like KITTI [46], and notably for this work also View-of-Delft [47], utilize the intersection-over-union (IoU) as a criterion. IoU quantifies the overlap between the predicted bounding box and the ground truth, offering a direct measure of localization accuracy for a single pair of boxes. It is calculated by dividing the area of overlap between the bounding boxes by the area of their union. For 2D or BEV object detection, the IoU is calculated using 2D areas on the image or BEV plane, respectively. For 3D object detection, it is calculated based on 3D overlaps and unions. IoU values range from 0 to 1, where 1 signifies perfect overlap and 0 denotes no overlap. In order to count a prediction as TP, its IoU with one of the ground truth bounding boxes has to exceed an IoU threshold. Typical values are 0.7 for *Cars* and 0.5 for smaller objects like *Bicycles* or *Pedestrians*.

A problem with 3D IoU arises for distant, small objects. An average pedestrian bounding box is approximately 70 cm long and wide. Assuming perfect size estimation as well as horizontal and vertical positioning, the depth estimation error still has to be lower than 23 cm to achieve an IoU greater than 0.5. For the maximum pedestrian detection distance of 40 m in nuScenes [44], this corresponds to less than 1 % relative error, which is very difficult to achieve with image-based object detection. Therefore, nuScenes [44] instead relies on BEV center distance to determine TPs. It uses several distance thresholds of [0.5, 1, 2, 4] m and the results are averaged at a later stage.

Using the gathered TP, FP and FN statistics, we can calculate precision and recall. Precision measures the accuracy of the positive predictions as:

$$\text{precision} = TP/(TP + FP) \tag{2.8}$$

In contrast, recall assesses the algorithm's completeness as:

$$\text{recall} = TP/(TP + FN) \tag{2.9}$$

A trade-off between precision and recall can be achieved by varying the confidence threshold, which indicates the minimum confidence score needed to count a network prediction. A low score threshold leads to high recall but low precision, whereas a high score threshold leads to high precision but low recall.

To extract a single number that captures this trade-off that is achievable with a given algorithm, we calculate the Average Precision (AP). First, we plot precision as a function of recall at different confidence thresholds, known as the precision-recall curve shown in Figure 2.14. The AP is defined as the

area under the curve and takes values between 0 and 1. For nuScenes [44], the AP is calculated for each distance threshold separately and then averaged.

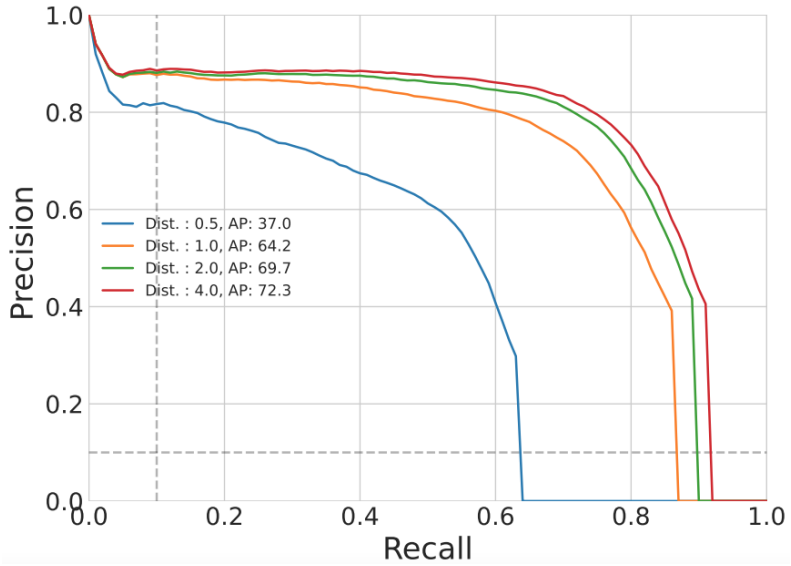


Figure 2.14.: *The AP is defined as the area under the Precision-Recall curve and is often averaged over multiple distance or IoU thresholds.*

The mAP extends the concept of AP to a broader scale by averaging the AP scores across n_l different classes as:

$$\text{mAP} = \frac{1}{n_l} \sum_{l=1}^{n_l} \text{AP}_l \tag{2.10}$$

This metric is particularly useful for assessing performance in multi-class object detection scenarios, offering a holistic view of an algorithm's effectiveness. It is therefore widely regarded as the gold standard for comparing the performance of object detection algorithms, encapsulating both detection accuracy in terms of the achievable trade-off between precision and recall as well as class-wide detection capabilities in a single number between 0 and 1. For datasets with heavy class imbalances, the mAP can be strongly influenced by small deviations in classes with few objects. Therefore, it can be more fair to use a weighted mean Average Precision (wmAP), which assigns each AP_l of class l a weight depending on the objects per class $n_{o|l}$. It is then normalized by the total number of objects n_o as:

$$\text{wmAP} = \frac{1}{n_o} \sum_{c=1}^{n_l} n_{o|l} \text{AP}_l \tag{2.11}$$

To get a more detailed view on the performance in several sub-tasks of object detection, nuScenes [44] additionally defines TP metrics, which are average errors calculated for each TP match: The Average Translation Error (ATE) calculates the Euclidian ground center distance in meters, the Average Scale Error (ASE) is calculated as $1 - IoU$ after alignment of the object center

and rotation, the Average Orientation Error (AOE) is the yaw angle difference in radians, the Average Velocity Error (AVE) is the absolute velocity error in m/s and the Average Attribute Error (AAE) is calculated as $1 - acc$, where acc is the classification accuracy of the nuScenes attribute, a sub-classification task for some of the classes indicating their movement status. Each of the TP metrics can, similar as for the mAP, be further aggregated across multiple classes, yielding the mean TP metrics: mATE, mASE, mAOE, mAVE and mAAE.

Finally, the NDS is used as an aggregated metric to encapsulate the mAP and the mean TP metrics in a single number between 0 and 1. First, it calculates scores for the mean TP metrics defined as:

$$\text{mean\_TP\_score} = max(1 - \text{mean\_TP\_error}, 0) \tag{2.12}$$

It then assigns a weight of 5 to the mAP and 1 to each of the mean TP scores and calculates a weighted average.

In summary, object detection is a multi-faceted problem, which requires complex and nuanced metrics to capture all necessary aspects. The mAP yields a good overview of the object detection performance, while the TP metrics allow more detailed understanding of the strengths and weaknesses of different algorithms, which can be especially useful when comparing algorithms based on different sensor modalities. The NDS is a final aggregated measure which has the highest correlation with the downstream driving task [45].

2.3. Object Tracking

Object tracking, a fundamental domain in the field of sensor data analysis, refers to the continuous observation and determination of an object's state over time. It can be realized through various sensor modalities, including cameras, radars and LiDARs. Based on a series of noisy object detections, the aim of a tracking filter is to estimate the object's trajectory. The task becomes more complicated when dealing with multi-object tracking, as required in automotive environment perception. In addition to the state updates and predictions required for each track, new detections have to be associated to the right tracks. This is especially challenging because detections are noisy and objects can be temporarily occluded [48]. In the following, we present an overview on track management, the commonly used Kalman filter [49], which can be used for probabilistic state updates and predictions, as well as approaches to associate new detections with tracks.

2.3.1. Track management

Track management deals with the problem that it is unknown how many objects need to be tracked at a given point in time. New detections can belong to an already identified track, a new track, or correspond to false positives or noise. Further, objects can be temporarily not detected due to occlusion or false negatives. Therefore, it is needed to manage the track list and decide

when to start a new track and when to remove an old track, as well as how to update existing tracks.

2.3.2. Kalman Filter

The Kalman filter [49] is a powerful recursive algorithm used in a wide range of engineering and scientific applications for estimating the states of a dynamic system based on a series of noisy measurements. If the noise is Gaussian and the model is linear, the Kalman filter is the optimal estimator.

For the linear Kalman filter, the dynamic system is modeled using linear state space equations. Given the state transition matrix $\boldsymbol{F}$, the control input matrix $\boldsymbol{B}$, an optional control input vector $\boldsymbol{u}_t$ and a random process noise $\boldsymbol{w}_t \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{Q}_t)$, the current state vector $\boldsymbol{x}$ at iteration t is modeled as:

$$\boldsymbol{x}_t = \boldsymbol{F}\boldsymbol{x}_{t-1} + \boldsymbol{B}\boldsymbol{u}_t + \boldsymbol{w}_t \tag{2.13}$$

In addition, given the observation matrix $\boldsymbol{H}$ and a random measurement noise $\boldsymbol{v}_t \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{V}_t)$, the measurement $\boldsymbol{z}_t$ is modeled as:

$$\boldsymbol{z}_t = \boldsymbol{H}\boldsymbol{x}_t + \boldsymbol{v}_t \tag{2.14}$$

Given initial estimates of the object state $\boldsymbol{x}_0$ and state covariance $\boldsymbol{P}_0$, the Kalman filter iterates between prediction steps and measurement updates. The prediction extrapolates the state as:

$$\boldsymbol{x}_{t|t-1} = \boldsymbol{F}\boldsymbol{x}_{t-1|t-1} + \boldsymbol{B}\boldsymbol{u}_{t-1} \tag{2.15}$$

Further, the state covariance is predicted as:

$$\boldsymbol{P}_{t|t-1} = \boldsymbol{F}\boldsymbol{P}_{t-1|t-1}\boldsymbol{F}^T + \boldsymbol{Q}_{t-1} \tag{2.16}$$

Given a new measurement, the filter is updated to correct the predicted estimate. First, the Kalman gain is computed as:

$$\boldsymbol{K}_t = \boldsymbol{P}_{t|t-1}\boldsymbol{H}^T(\boldsymbol{H}\boldsymbol{P}_{t|t-1}\boldsymbol{H}^T + \boldsymbol{V}_t)^{-1} \tag{2.17}$$

Then, the new measurements are used to update the state:

$$\boldsymbol{x}_{t|t} = \boldsymbol{x}_{t|t-1} + \boldsymbol{K}_t(\boldsymbol{z}_t - \boldsymbol{H}\boldsymbol{x}_{t|t-1}) \tag{2.18}$$

Finally, the state covariance is updated taking the measurement noise into account:

$$\boldsymbol{P}_{t|t} = (\boldsymbol{I} - \boldsymbol{K}_t\boldsymbol{H})\boldsymbol{P}_{t|t-1} \tag{2.19}$$

2.3.3. Association

Track association deals with the problem that a set of new detections needs to be assigned to a set of existing tracks. A common assumption is that each new detection belongs to at most one existing track, and each existing track leads to at most one new detection [48]. In this case, the detections can be greedily

associated to the existing tracks, based on a distance metric which needs to be lower than a pre-defined threshold. The distance metric is calculated based on the detection's parameters as well as the predicted state of the tracks.

State spaces generally cover attributes of different units and scales, such as an object's position, velocity, size or orientation. In order to account for these different scales as well as the correlations of variables in the state space, we can normalize the innovation term $\boldsymbol{z}_t - \boldsymbol{H}\boldsymbol{x}_{t|t-1}$ with its covariance matrix $\boldsymbol{S}_t$, which can be derived as:

$$\boldsymbol{S}_t = \boldsymbol{H}\boldsymbol{P}_{t|t-1}\boldsymbol{H}^T + \boldsymbol{R}_t \tag{2.20}$$

The normalized and squared innovation yields the resulting distance metric, known as the Mahalanobis distance:

$$d_{\text{mahalanobis}} = (\boldsymbol{z}_t - \boldsymbol{H}\boldsymbol{x}_{t|t-1})^T(\boldsymbol{S}_t)^{-1}(\boldsymbol{z}_t - \boldsymbol{H}\boldsymbol{x}_{t|t-1}) \tag{2.21}$$

2.4. Image-based object detection

The use of deep neural networks for object detection in images has lead to a significant increase in detection accuracy. The early methods are characterized by a two-stage approach: First, a region proposal network (RPN) is used to identify regions of interest (RoIs) in the image. Second, an image classification network is used to classify the object in each RoI. An exemplary algorithm using this approach is Faster R-CNN [50]. To reduce the computational cost of these approaches, one-stage object detectors were introduced. They eliminate the need for a RPN by replacing the RoIs with pre-defined anchor boxes. Examples of this technique include SSD [37] and RetinaNet [39]. More recent approaches use transformers instead of purely convolutional networks [51]–[54]. In Chapter 3.1 of this thesis, we build an architecture based on a modified variant of the RetinaNet architecture.

Image-based 3D object detection is a difficult task in the field of computer vision because it involves identifying and localizing objects in 3D space using only a single camera as a sensor. This is in contrast to LiDAR and radar systems, which provide depth measurements and can more accurately determine the 3D location of objects. Early approaches to image-based 3D object detection focused on using known geometric information to estimate 3D bounding boxes from 2D detections [55]. More recent techniques have extended existing 2D object detection models with additional detection heads specifically designed for 3D object detection [38], [56]–[58]. Some approaches have also used predicted depth maps as auxiliary features [59] or to create a pseudo-LiDAR point cloud, which is then processed using LiDAR-based object detectors [60].

The most recent research in this area has mainly followed two directions. The first is the use of transformer-based techniques, which leverage the ability of transformer models to process sequences of data and perform self-attention to learn more complex relationships between features [61]–[64]. The second direction is the development of BEV-based object detectors. To this end, the

features need to be transformed from the image plane to the BEV plane. A pioneering work uses orthographic feature transform [65], where a voxel grid is projected to the image to extract features. To reduce memory consumption, the voxel grid is then collapsed along the vertical axis to create BEV features. More recent methods are based on the Lift-Splat-Shoot (LSS) view transformer [66] or similar work [67]. As shown in Figure 2.15, LSS lifts image features into a 3D pseudo point cloud via dense depth prediction, before again collapsing the vertical dimension to create BEV features. A recent review on BEV-based perception is given in [68]. The idea is first incorporated into a 3D object detection network in the BEVDet series [69]–[71], and later refined with LiDAR-based depth supervision [72] and temporal stereo [73], [74]. In [75], a more efficient view transformer in terms of memory and computations is introduced by formulating LSS into matrix operations with decomposed ring and ray matrices, and compressing image features and the estimated depth map along the vertical dimension. Another approach to BEV-based object detection is the BEVFormer series [76], [77], which uses spatio-temporal transformers to create BEV features from multiple images. Both of these approaches are finally combined in recent work [78]. In Chapter 4.1 of this work, we build upon these BEV-based object detectors and integrate our radar-camera fusion as a plug-in module.

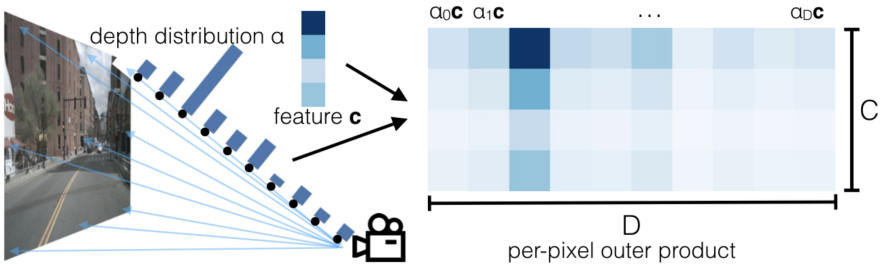


Figure 2.15.: *In the LSS view transformer, image features are lifted into 3D via dense depth prediction, before the vertical dimension is collapsed to create BEV features [66], © 2020, Springer Nature Switzerland AG.*

2.5. Radar-based object detection

Automotive radars are a common sensor used in autonomous vehicles for detecting objects in the environment. These radars typically provide a preprocessed list of detections, but the sparsity and lack of semantic information make it difficult to use the data for stand-alone 3D object detection. As a result, much of the research in this area has focused on either semantic segmentation of radar point clouds [79] or experimental setups using the raw radar cube [80], [81].

Recently, there have been some point cloud-based techniques for object detection as well. These can be broadly divided into convolutional and graph-based approaches. Some convolutional approaches assign each point in the point cloud to a cell in a BEV grid and include feature layers such as the

maximum RCS and Doppler values [82], [83]. Since the BEV grid is similar in structure to an image, traditional convolutional networks can be applied for object detection. Other convolutional approaches use variants of PointPillars [84] to create a pillar grid automatically from the point cloud [47], [83]. In contrast, graph neural network based approaches perform object detection directly on the radar point cloud [85], [86]. Recent work combines both approaches by first extracting features with a graph neural network and then mapping the features to a BEV grid for further processing [87]. In Chapter 4.2 of this work, we examine radar feature encoders inspired by these ideas.

2.6. Sensor fusion for object detection

Sensor fusion aims at leveraging the strengths of a diverse sensor combination. Most sensor fusion research focuses on LiDAR-camera fusion, as LiDAR provides accurate 3D information and cameras provide high semantic value, while sharing the same optical propagation principles. There are several techniques for fusing LiDAR and camera data. Some approaches are based on unprojecting 2D detections from the camera into a frustum and matching them with LiDAR points to refine the 3D detections [88], [89]. Other techniques augment the LiDAR points with semantics from the image and use LiDAR-based object detectors to perform detection [90]–[92]. The most recent approaches to LiDAR-camera fusion involve extracting BEV features from both the LiDAR and camera data and fusing them on the BEV plane before applying a joint BEV encoder to perform object detection [76], [93], [94].

When compared to LiDAR, automotive radar uses a different wavelength and measurement principle. As a consequence, radar typically shows strong RCS fluctuations and reduced resolution in range and angle, resulting in less dense point clouds. Moreover, whereas modern radars also measure elevation, many conventional radars only provide detections on the BEV plane. These differences make the fusion especially challenging and prevent the simple replacement of LiDAR with radar processing.

A recent review of automotive radar-camera fusion is given in [95]. Most of the early radar-camera fusion approaches for 2D object detection use a modified image-only detection network that is enhanced by the integration of the radar. To enhance two-stage image-based object detectors, the radar is often used to find RoIs in the image, which can then be used for further image processing. In [96], the radar is used to determine square RoIs in the image, which are then classified by a custom CNN with a contrastive loss function. In [97], the authors show that the RPN can be replaced by a radar-based region proposal to reduce computational cost and obtain better results on the nuScenes dataset [44]. One of their key ideas is to generate multiple anchor boxes per radar detection, which have different sizes, scales and alignments with the radar point either in the center, to the left, right or bottom of the box. In [98], the authors further develop this idea by generating 3D proposals for each radar detection, mapping them to the image and refining them using radar and image features to generate 2D radar proposals. These are then fused

with 2D image proposals created by a RPN and used as input for the detection stage of the algorithm. A limitation of these sequential approaches is that the performance at each stage is limited by the corresponding sensor, rather than exploiting the synergies of multiple sensor inputs used at the same stage.

For the improvement of one-stage image-based 2D object detectors with radar-camera fusion, most algorithms first create additional image channels by projecting the radar points onto the image. The idea is to enable the network to learn how to make the best use of the complementary inputs. John & Mita [99] introduce the RVNet, a 2D object detection algorithm with two input and two output branches. The input branches are for the radar and camera input, respectively. The radar detections are transformed to the image plane to form a three channel radar image using the measured depth and the lateral and longitudinal velocity. The two output branches are used to detect small and large obstacles. The authors also develop the SO-Net [100], which extends the RVNet by adding another output branch for semantic segmentation, making it a multi-task learning algorithm. Chadwick et al. [101] show that the radar is especially useful for the detection of distant, small vehicles. They project the radar detections as small circles to reflect their uncertainty and use the range and the range-rate as channels of the radar image. In contrast, Nobis et al. [102] argue that the radar detections should rather be projected as a vertical line, since the radar is mainly uncertain in terms of the object's elevation. They assume a fixed height for each object and use the distance and RCS as the radar channels. They also introduce a new training technique BlackIn, which blacks out the camera inputs at a fraction of the training steps, to encourage the network to rely more heavily on radar data. All of the above methods fuse radar and camera features by concatenation. In contrast, spatial attention fusion [103] creates an attention matrix by applying convolutional filters on the radar features, which is then multiplied with the image features. Thus, the network can learn which parts of the image are most relevant based on the radar detections. In Chapter 3.2, we improve upon above mentioned techniques by incorporating azimuth uncertainty in the radar projection and introducing a method to automatically find the best fusion points in the network architecture.

More recent methods have moved away from this 2D approach and instead focus on fusing based on 3D information. One approach is to refine image-based 3D detections with associated radar data [104], [105]. This can be sub-optimal because it discards the possibility of radar-only detections. Another approach is to project 3D RoIs to the image and BEV plane to extract features from each sensor [106]. Finally, many recent papers use cross-attention between image and radar features to align and fuse the features in 3D. CramNet use cross-attention to assist camera-based depth estimation with radar features [107], while CRAFT [108] uses it to exchange spatio-contextual information between camera and radar. MVFusion [109] injects semantic image features into the radar encoder to generate semantic-aligned radar features, before using these in a cross-attention fusion with the image features. In TransCAR [110], a transformer architecture with multiple cross-attention fu-

sion layers enables a soft association of radar and camera features which is arguably more beneficial than the hard association via geometric fusion. In contrast, FUTR3D[111] projects 3D query points into the native coordinate systems of radar, camera and LiDAR to retrieve features for a fusion transformer. In Chapter 4.1, we propose a novel type of radar-camera fusion using joint BEV features. This BEV-based radar-camera fusion scheme has been adopted in several subsequent works, confirming its effectiveness [112]–[115].

2.7. Radar-camera datasets

The training and evaluation of a machine learning based algorithm requires a well suited dataset. In the past, many radar algorithms were evaluated on private datasets, which leads to less reproducible and comparable results. To combat this trend, a number of publicly available datasets, some including benchmarks and leaderboards, have been introduced in the recent years [116]. In this section, we give an overview on publicly available datasets in Table 2.1. We focus on datasets which contain radar and camera data in an automotive setting as well as bounding box labels required for object detection.

We group the datasets by the type of radar data they cover. First, there are datasets with conventional, 2+1D (range, azimuth and Doppler) sparse radar point cloud data, as they are built for automotive series production. In this category, the nuScenes dataset [44] is the largest and most popular dataset. It covers data from five low resolution radars mounted on the four corners and the front of the vehicle, as well as 6 cameras for full surround view coverage, and a LiDAR sensor on the rooftop. It has 3D bounding box labels for 40000 annotated frames from mostly urban scenes with diverse weather and lighting conditions, recorded in Boston and Singapore. The nuScenes [44] benchmark for 3D object detection is used in many research papers to compare the performance of camera-based, LiDAR-based, radar-based and sensor fusion algorithms. Other datasets with conventional radar data include PixSet [117] and Pointillism [118]. The main focus of PixSet [117] is to showcase a novel type of solid-state LiDAR, but it also has a front-facing radar and camera, so it can be used to evaluate sensor fusion algorithms for 3D object detection. Pointillism [118] has a unique setup with two front facing radars to improve the resolution, as well as a front camera and a LiDAR. It further provides 3D bounding box annotations for scenes in different weather conditions. Recently, aiMotive [119] was introduced, which covers front and back facing radars as well as a mix of fisheye and pinhole cameras and a LiDAR.

In the second group, there are datasets with raw radar data recorded before the radar target list is created. The datasets CARRADA [120] and RADDet [121] both use a stationary radar in short range mode to provide raw Range-Azimuth-Doppler (RAD) tensors covering the near field up to 50 m distance. CARRADA [120] captures sequences on an empty test track with only two objects and provides 2D annotations for the Range-Azimuth (RA) and Range-Doppler (RD) representations of the data, respectively. On the other hand, RADDet [121] captures sequences from main roads and provides

Table 2.1.: *Radar-camera datasets for object detection, adapted from [116]. LR = Low Resolution Radar, HR = High Resolution Radar, SP = Spinning Radar. PC = Point cloud, RA[E][D] = Range, Azimuth[, Elevation][, Doppler] Pre-CFAR tensor, ADC = Analog Digital Converter raw data. BB = Bounding Box.*

Category	Dataset	Radar Data Type	Camera Type	Annotation	Size
Conventional Radar	nuScenes [44]	LR, 5x Sparse PC	1600×900 70°/110°, 5x	3D BB, TrackID	L
	PixSet [117]	LR, 1x Sparse PC	1440×1080 90/180°, 4x	3D BB, TrackID	M
	Pointillism [118]	LR, 2x Sparse PC	1920×1080 69°, 1x	3D BB	M
	aiMotive [119]	LR, 2x Sparse PC	2896×1876, 2x Fisheye, 2x	3D BB	M
Pre-CFAR Radar	CARRADA [120]	LR, 1x RAD	1238×1028 1x	2D BB, TrackID	M
	RADDet [121]	LR, 1x RAD	1440×1080 2x	3D BB	M
	RaDICaL [122]	LR, 1x ADC	1920×1080 69°, 1x	2D BB	L
4D Radar	Astyx HiRes [123]	HR, 1x Dense PC	2048×618 1x	3D BB	S
	View-of-Delft [47]	HR, 1x Dense PC	1936×1216 64°, 1x	3D BB, TrackID	M
	TJ4DRadSet [124]	HR, 1x Dense PC	1280×960 67°, 1x	3D BB, TrackID	M
	K-Radar [125]	HR, 1x RAED	1280×720 110°, 4x	3D BB, TrackID	L
Spinning Radar	RADIATE [126]	SP, 1x RA	672×376 2x	BEV BB, TrackID	M
	BOREAS [127]	SP, 1x RA	2448×2048 81°, 1x	3D BB	L

3D annotations for the RAD tensors, and is therefore more realistic. Data from even further up the processing chain is provided in the RaDICaL dataset [122]. In indoor, highway, parking lot and moving pedestrian scenes, raw radar data from the ADC is recorded along with RGB-Depth images.

The third group of datasets cover the recent developments of high performance radars, which have more transmit and receive antennas than conventional radars and therefore allow for higher resolution 3+1D radar data with measurements of the elevation angle. A first sample of this data was provided in the Astyx HiRes [123] dataset. However, it only covers 500 annotated frames, which is very small. Recently, several larger datasets with high performance radars were introduced. The View-of-Delft [47] dataset covers urban scenes with 3+1D dense radar point clouds, as well as front camera images, LiDAR data and 3D bounding box annotations. TJ4DRadSet [124] has a similar setup, but unfortunately, at the time of writing, the camera images could not yet be released due to data privacy restrictions. K-Radar [125] is the first dataset to provide raw radar cube data for a high performance 3+1D radar along with four wide-angle stereo cameras for surround view, LiDAR data and

3D bounding box annotations. It is a large-scale dataset that covers adverse weather conditions and various road types.

Finally, the datasets RADIATE [126] and BOREAS [127] include data from special spinning radars that provide RA scans. RADIATE provides BEV annotations directly on the RA maps and focuses on scenes with varying weather conditions. BOREAS on the other hand covers seasonal variations of the same route driven over the course of a year and provides 3D bounding boxes along with a pose ground truth for the localization task.

In this thesis, we focus on radar point cloud data and the task of object detection. After careful examination of available datasets, we have selected the nuScenes dataset [44] from the conventional radar datasets for most of our experiments due to it's large scale, high data variety and complete sensor suite with full surround view coverage. Further, we have selected the View-of-Delft [47] dataset for a study on high-performance radar point clouds because it is currently the largest dataset in its category that provides 3+1D radar point cloud data and camera images.

Chapter 3

Fusion Point Pruning: Enhancing 2D object detection with radar-camera fusion

Contents

One of the main challenges of radar-camera fusion is to handle the heterogeneity of the data sources. While the camera provides raw sensor information in the form of RGB images, the radar typically provides a pre-processed list of detections, containing measurements of the distance, relative velocity, azimuth angle, and RCS. Note that the information from the radar is sparse when compared with the dense camera images. For the task of 2D object detection on the image plane, most radar-camera fusion techniques fuse the different representations by projecting the radar point cloud onto the image plane to construct additional channels with parameters of the radar detections. The radar detections can be represented in the form of a small pixel area around the projected image location [101], or a vertical line of fixed height [102] to reflect the elevation uncertainty of the radar measurement. In this chapter, we propose novel projection techniques that additionally incorporate

the azimuth uncertainty when constructing the radar channels. In particular, we assume that the measured azimuth angle follows a Gaussian distribution around the actual measurement, where the standard deviation corresponds to the accuracy value from the sensor's technical data sheet. Thereby, we are able to effectively use the additional uncertainty information and create visually denser radar channels. An example of the obtained representation is shown in Figure 3.2.

When designing a neural network to process both radar and camera channels, it is challenging to determine the ideal method for how and when to fuse the branches of radar and camera. In the literature, fusion strategies are often categorized into early, deep and late fusion [10], [99], [103]. When choosing one of these fusion strategies, the main concerns are that fusing too early may not be ideal considering the heterogeneity of the data sources, while fusing too late may not enable the network to take advantage of potential synergies. To solve this problem, the CRF-Net [102] uses a network architecture that concatenates radar and camera features at multiple early, deep and late stages in the network, to which we refer as fusion points. The idea is to enable the network to adjust its weights for each fusion point during training and thereby to implicitly find the ideal fusion points. However, this approach does not lead to significant performance improvements compared to an image-only baseline, which shows that the network is not able to fully leverage the advantages of the added radar modality. In contrast to [102], we show that, surprisingly, too many fusion points can limit the performance of the network. We introduce a technique that is inspired by network pruning and automatically selects the ideal fusion points during training, effectively optimizing the network architecture and performance.

The contents of this chapter have been published in [12]. To summarize, the main contributions of this chapter are:

- Two new projection techniques to create dense radar channels by incorporating elevation and azimuth uncertainty.
- A fusion point pruning technique to automatically optimize the network architecture.
- An overall radar-camera fusion pipeline that achieved state-of-the-art performance in 2D object detection evaluated using the nuScenes [44] dataset.

3.1. Network architecture

The model architecture is shown in Figure 3.1. It is based on a RetinaNet [39] architecture with a ResNet [26] backbone, a subsequent FPN [35] neck and classification and regression detection heads that produce the 2D bounding boxes. Additionally, we use a radar branch with max pooling layers to adjust the shape of the radar inputs according to the different stages in the network, similar to the CRF-Net introduced by [102]. The radar features are fed into

the network on several different fusion points at early, deep and late stages of the network. The fusion operation is a concatenation of the respective feature tensors. The network is thereby expected to learn the ideal stages for the radar fusion.

In contrast to the CRF-Net [102], we add flexibility to the network architecture by introducing binary hyperparameters to enable or disable each of the fusion points. We can use these hyperparameters to experimentally examine the effectiveness of each fusion point. Also, we can dynamically change the network architecture during training. We use pretrained weights from ImageNet [9] for each layer that is identical with the RetinaNet architecture. The layers after each fusion point have additional input channels, whose weights are randomly initialized and get discarded when the corresponding fusion point is eliminated.

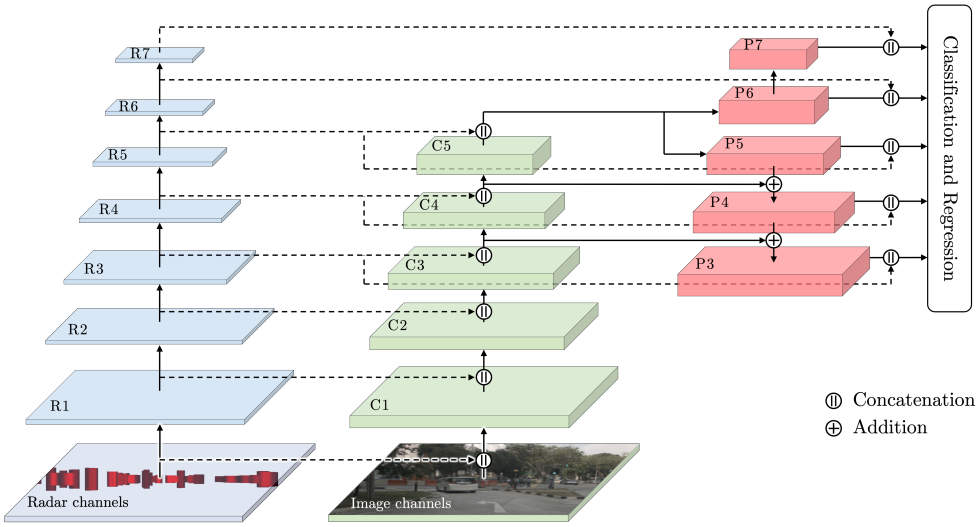


Figure 3.1.: *Model architecture showing the fusion points as concatenations of radar and image features. Dashed lines indicate that the connections are optional depending on the fusion point hyperparameters [12].*

3.2. Radar input generation

We project each radar detection onto the image plane using the measured azimuth angle and range, as well as the camera's extrinsic and intrinsic calibration parameters. The corresponding entries in the resulting radar channels can be filled with information from the detections like the measured distance, relative velocity, or RCS. To reflect the measurement uncertainty of the radar, we model the elevation and azimuth uncertainty in our projection technique. We follow [102] in the generation of vertical lines assuming a height of 3 m for each detection to represent the elevation uncertainty. Additionally, we introduce new techniques to further reflect the detection azimuth uncertainty. We

propose the following two techniques which have been experimentally evaluated in this work.

3.2.1. Uncertainty channel

The first method is an additional radar channel that contains information on the azimuth uncertainty of each detection. We assume a Gaussian distribution for the measurement of the azimuth angle and calculate the density values using the measured azimuth angle as mean and the accuracy from the technical data sheet as standard deviation. Instead of a vertical line with a width of one pixel, we now generate a visually denser radar input where the corresponding entries of each detection are horizontally spread across several pixels, according to the Gaussian density curve. In the case of overlapping entries, we keep the highest value.

3.2.2. Uncertainty weighted RCS channel

The second method makes use of the idea of the first method but further enriches the radar input channel by its RCS information. To this end, we calculate the uncertainty channel as described in the previous section and multiply the density values of each detection with its measured RCS value. Since the RCS is a measure that represents object reflectivity and is often used for object size classification, we expect this to be a meaningful weighting. A visualization of the camera image, augmented with the uncertainty weighted RCS channel values highlighted in red, is given in Figure 3.2.

Figure 3.2.: *Visualization of the camera image from nuScenes [44], augmented with the uncertainty weighted RCS channel values highlighted in red [12].*

3.3. Fusion point pruning

We propose a novel network pruning technique to improve the capability of the network to find the best fusion points. We note that this technique is not limited to radar-camera fusion as shown in this chapter and can be applied to other fusion problems as well. After several epochs of training, we evaluate the relative importance of each fusion point to decide if it should be kept in the architecture.

As shown in Section 2.2.1, a 2D convolution can be described using 3D tensors for input $\mathbf{I}$ and activation map $\mathbf{A}$ with channels c_I and c_A, respectively, and a 4D tensor for the kernel weights $\mathbf{K}$ with uneven kernel size k.

For regular network pruning, the importance of neuron connections is usually determined by examining the magnitude of their corresponding weights [128], [129]. However, since we are dealing with heterogeneous data sources of different magnitude, this does not appear meaningful to us. Instead, we calculate the relative importance of the input channels at our fusion points using their impact on the next layer's activation map $\mathbf{A}$. To this end, we separately calculate the activations $\mathbf{A}_i$ for each of the input channels i in the first layer after a fusion point:

$$\mathbf{A}_i[m,n,a] = \sum_{u=1}^{k}\sum_{v=1}^{k} \mathbf{K}[u,v,i,a] \cdot \mathbf{I}[m + u - \frac{k+1}{2}, n + v - \frac{k+1}{2}, i] \quad (3.1)$$

We then take the magnitude in terms of the L1-norm of each channel's activation and normalize it to determine the relative impact_i of each input layer i in the feature tensor at a fusion point:

$$\text{impact}_i = \frac{\|\mathbf{A}_i\|_1}{\sum_{j=1}^{c_I} \|\mathbf{A}_j\|_1} \quad (3.2)$$

We sum up the relative impacts of the radar channels that were concatenated at each fusion point to measure the importance of fusion taking place at this point in the network. We iteratively eliminate the least effective fusion points and their corresponding weights in the following layer after a predefined number of training epochs and save the corresponding model checkpoint. This is repeated until a single fusion point remains. Finally, we use the model checkpoints to evaluate the performance and select the best performing model with the optimal set of fusion points.

3.4. Results

As discussed in Section 2.7, the nuScenes dataset [44] is used to train and evaluate our network. It is currently the largest dataset for autonomous driving which has conventional automotive radar sensors. The nuScenes dataset contains 3D bounding box annotations of 27 classes. As in [102] and [98], we obtain 2D bounding box annotations by projecting the 3D bounding boxes onto the image plane, and apply a similar class mapping to obtain five high-level classes: *Car*, *Pedestrian*, *Bicycle*, *Motorcycle* and *Truck*. We note that

Table 3.1.: *Quantitative evaluation with aggregated metrics.*

	Camera	Radar	mAP	wmAP	Night wmAP	Runtime[‡]
Faster R-CNN* [50]	✓		34.95	43.78	—	—
RRPN* [97]	✓	✓	35.45	—	—	—
Nabati & Qi* [98]	✓	✓	35.60	44.49	—	—
RetinaNet [39]	✓		34.75	45.65	43.24	36.4ms
CRF-Net [102]	✓	✓	32.93	44.91 (43.95[†])	46.42	37.6ms
Ours - UC	✓	✓	35.64	45.99	49.56	37.1ms
Ours - UwRCS	✓	✓	36.16	46.33	48.11	36.8ms
Ours - UC + FPP	✓	✓	36.14	46.43	48.56	37.0ms
Ours - UwRCS + FPP	✓	✓	**36.78**	**46.73**	**50.53**	36.7ms

[*] The results in these rows were calculated by Nabati & Qi [98] and use a slightly different class mapping, which leads to the mAP metric being less comparable.
[†] Nobis et al. [102] only report this wmAP based on a slightly different class mapping. We therefore additionally list results that we calculated ourselves with the CRF-Net using our class mapping.
[‡] Average inference runtime per frame was calculated on NVIDIA GeForce RTX 2080.

the projected 2D bounding boxes are generally too wide, which is not ideal. However, we choose to follow previous work [98], [102] to ensure comparability.

3.4.1. Training and evaluation details

We train and evaluate the algorithms using the front camera and radar with the official train and val scene splits containing 28130 training and 6019 validation frames. Additionally, we evaluate them using a subset of the validation scenes consisting of 608 frames that were taken at night. The networks are trained using the Adam optimizer [40] with an initial learning rate of 10^{-5} that is reduced by a factor of 10 when the optimization reaches a plateau.

The most common metric to evaluate object detection performance is the mAP introduced in Chapter 2.2.7, which is calculated as the mean of the AP across each class of the dataset. Working with the nuScenes dataset, many researchers choose individual class mappings to reduce the 27 classes to a set of selected classes. A downside of the mAP is its dependency on the class mapping, where even small changes can lead to substantial differences in the calculated mAP. Hence, in recent work [102], [98], the wmAP is additionally reported, which is calculated as a weighted mean of the AP for each class, using the amount of objects per class as weights and divided by the total amount of objects. This metric is less dependent on the underlying class mapping. In turn, the classes with more objects tend to dominate the results of the wmAP.

We calculate the mAP and wmAP using an IoU threshold of 0.5 and report both to ensure the best comparison possible. Additionally, we provide the night scene wmAP, which highlights the benefits of radar-camera fusion in adverse conditions for the camera. We list the results of two image-only networks, RetinaNet [39] and Faster R-CNN [50], which are used as baselines for three radar-camera fusion algorithms, RRPN [97], Nabati & Qi [98] and CRF-Net [102], as well as our own results with the uncertainty channel (UC) and uncertainty weighted RCS channel (UwRCS), with or without FPP. The

results from Faster R-CNN, RRPN and Nabati & Qi were calculated in [98] and use a slightly different class mapping. Their mapping includes a *Bus* class that we chose to merge with the *Truck* class in our class mapping. We compute the remaining results with the same training and evaluation settings and based on the same RetinaNet [39] as core architecture to ensure comparability. In our experiments, we choose to work with a small ResNet-18 backbone due to the real-time requirements in automotive systems. However, the results should be applicable to larger backbones as well. In the CRF-Net paper, Nobis et al. [102] only report the wmAP based on a different class mapping, which we additionally list. They further list results which were produced by filtering data based on ground truth information that is not available in a real-world scenario, which we therefore discard.

3.4.2. Quantitative Results

Looking at the results in Table 3.1, we see that all of our own methods outperform the other methods in terms of mAP and wmAP, leading to a new state of the art in radar-camera fusion for 2D object detection. Our best performing model is the UwRCS channel in combination with FPP. It should be noted that the mAP and wmAP of all methods are in a relatively close range. The best mAP is 3.85 above the lowest, which is a relative increase of 11.7 %, while the relative increase in terms of the wmAP is even less. We can observe that the Nabati & Qi [98] model only slightly outperforms its image-only baseline Faster R-CNN, while the CRF-Net [102] does not outperform its image-only baseline RetinaNet [39] in our evaluation. From our own experiments, we find that this is due to the many fusion points in the CRF-Net architecture, which hinders training and can lead to a suboptimal detection performance.

For our own methods, we started experimenting with the UC and UwRCS techniques and found that we significantly outperform the original CRF-Net [102] when reducing the amount of fusion points in the network architecture experimentally. To reduce the amount of experiments needed to find the optimal architecture, we developed the FPP technique. Out of the 11 initial fusion points shown in the architecture in Figure 3.1, FPP determined that the best fusion point for the UC + FPP technique is the concatenation of the $R3$ radar feature with the $C3$ ResNet feature, while for UwRCS + FPP it is best to have two fusion points at $R3$ with $C3$ and at $R4$ with $C4$, which are all located in the center of the neural network. This indicates that a surprisingly strong reduction of fusion points is beneficial for training. The fusion points late in the network, after the FPN, have been eliminated first in our experiments, indicating that a regression level fusion is not ideal to exploit the complementary features. The fusion points early in the network, including the direct concatenation of the radar and image input channels, are also eliminated, indicating that it is beneficial to separately extract some semantic information before fusing the feature tensors.

To further underline the potential of radar-camera fusion in adverse conditions for the camera, we study the performance for scenes that were recorded at night. We can observe a more significant increase in detection performance

of all fusion methods with respect to the RetinaNet [39] baseline, with up to 16.8 % relative increase in night wmAP for the UwRCS + FPP technique. Note that the night wmAP can be higher than the overall wmAP due to mostly *Cars* present at night which have high AP.

The UwRCS + FPP technique further increases the detection performance and performs consistently well across all classes as shown in Table 3.2. It has the highest AP for *Car*, *Pedestrian* and *Bicycle* and second highest for *Truck* and *Motorcycle*, where it is only outperformed by the RRPN [97]. In fact, the RRPN [97] performs particularly well for these classes but equally poorly for the other classes. This method seems to have some particular strengths and weaknesses that differ from all other methods. It should be noted, that the wmAP of RRPN was not reported by Nabati & Qi [98] but should be low due to the weak performance of the *Car* and *Pedestrian* classes, which have the highest number of annotated objects.

Table 3.2.: *Quantitative evaluation with class-wise AP.*

	Camera	Radar	Car	Truck	Ped.	Bus	Bicycle	Motorcycle
Faster R-CNN [50]	✓		51.46	33.26	27.06	47.73	24.27	25.93
RRPN [97]	✓	✓	41.80	**44.70**	17.10	**57.20**	21.40	**30.50**
Nabati & Qi [98]	✓	✓	52.31	34.45	27.59	48.30	25.00	25.97
RetinaNet [39]	✓		55.69	31.89	37.10	—	21.80	27.27
CRF-Net [102]	✓	✓	53.71	33.72	36.50	—	18.62	22.09
Ours - UC	✓	✓	55.40	34.15	37.23	—	23.27	28.14
Ours - UwRCS	✓	✓	55.46	35.26	37.63	—	25.02	27.43
Ours - UC + FPP	✓	✓	55.69	35.42	37.36	—	23.42	28.78
Ours - UwRCS + FPP	✓	✓	**55.94**	35.60	**37.77**	—	**25.74**	28.84

The inference times of RetinaNet [39], CRF-Net [102] and our models are very similar because the radar branch is inexpensive relative to the rest of the network. The runtime slightly increases with the amount of fusion points in the network, so FPP can help to reduce it.

3.4.3. Qualitative Results

In Figure 3.3, we show some qualitative results of our proposed method with UwRCS and FPP, compared with the RetinaNet [39] image-only baseline. The color of the bounding boxes relates to the different classes. In the left column, we have a very crowded urban scene with multiple *Cars*, *Trucks* and *Pedestrians*. We can see that the performance is relatively similar and most of the objects in the scene are correctly detected. The most notable difference is the detection of the *Car* and the construction worker *Pedestrian* below the road sign on the right side, which both are partially occluded by a road barrier and thus missed by the RetinaNet [39]. The same happens for the *Car* that is to the right of the *Truck* on the left side of the image. In these circumstances, the radar can help to detect such occluded objects. Another example of this observation is shown in the center column, where the radar can help to detect a *Car* that is occluded by bushes and would have been missed by the RetinaNet [39]. In the right column, we have a night scene with

difficult lighting conditions. There is only one *Car* on the right of the image which is hardly visible in the camera image and thus missed by the RetinaNet [39]. However, our optimized fusion network is able to detect and localize the *Car* quite well.

Figure 3.3.: *Qualitative comparison of detection results for nuScenes [44]. Yellow: Car, Orange: Truck, Blue: Pedestrian [12].*

3.5. Conclusion

In this chapter, we presented novel approaches to optimize radar-camera fusion for 2D object detection. First, we presented new projection techniques that take the radar's elevation and azimuth uncertainty into account and create a visually denser radar input. Second, we developed the FPP technique that automatically selects the best fusion points in the network architecture, effectively solving the problem of when to fuse camera and radar data. We note that this technique can be applied to other fusion problems as well. As shown in the quantitative evaluation, our contributions outperform previously published radar-camera fusion systems. While the improvements compared to image-only algorithms like RetinaNet [39] are still relatively small, we can demonstrate the potential of radar-camera fusion for night scenes, where the camera is less reliable. A possible drawback for the performance of the fusion models is the quality of the radar data in nuScenes. The radar point clouds are less dense than what can be expected from current automotive radars, so the full potential of radar can not be demonstrated using this dataset. Also, the task of 2D object detection on the image plane is less suited for radar, which can add more value in the field of 3D object detection.

Chapter 4

RC-BEVFusion: A flexible architecture for radar-camera fusion with bird's eye view features

Contents

Sensor fusion has the potential to overcome limitations of individual sensors. As discussed in Section 2.1, the combination of radar and camera data arguably offers the most complementary features. However, a key challenge is how to associate radar and camera features given that conventional radars provide data on the BEV plane, whereas cameras provide data on the image plane. Thus, the data representations only share one common dimension. Projecting radar points to the image can provide useful augmentations for the task of 2D object detection on the image plane as shown in Chapter 3. However, for 3D object detection, the projection discards too much geometric information. In contrast, projecting camera features to the sparse radar points discards too much semantic information [76].

Recent advancements in camera-only networks utilizing view transformers [66], [69] have paved the way for a new form of fusion on the BEV plane, ideally suited for radar data. In this study, RC-BEVFusion is introduced, representing a novel radar-camera fusion architecture on the downstream task-friendly BEV plane [130]. This architecture, influenced by [76], is depicted in Figure 4.1. Unlike prior radar-camera fusion techniques [104], [108], [110], this architecture facilitates equal contribution from both radar and camera features to the final detections. This balanced contribution empowers the network to detect obstacles that might be overlooked by a single modality. Characterized by its flexibility, the architecture inherits several components from an interchangeable camera-only baseline: a camera encoder, a camera-to-BEV view transformer, a BEV encoder, and a detection head. In addition to these modules, two radar encoder branches, namely RadarGridMap and BEVFeatureNet, are proposed.

We extensively evaluate our models quantitatively and qualitatively on the nuScenes dataset [44], which indicates that the proposed radar branches can function as plug-in modules in diverse camera-based architectures, significantly elevating their performance. We further conduct a study on the performance of the image-only baseline, a radar-only variant and the radar-camera fusion model across nuScenes and the View-of-Delft [47] dataset, which has high-performance radar data. We study generalization capabilities of the models and outline aspects of each dataset that drive the performance of the network.

The contents of this chapter have been published in [13] and [14]. In summary, this chapter includes the following contributions:

- A flexible, modular radar-camera fusion architecture based on BEV features.
- Two novel radar encoders that can be used as a plug-in system in various camera-only baseline networks.
- Extensive qualitative and quantitative evaluation of the proposed networks on the nuScenes dataset [44].
- A cross-dataset study with nuScenes [44] and View-of-Delft [47] on generalization capabilities and important factors that elevate the performance of the proposed architectures.

4.1. Network architecture

In this study, we introduce a novel radar branch and use it as a plug-in module on various camera-based 3D object detection networks to enhance their performance. The prerequisite for the proposed radar-camera fusion is that the camera-only network utilizes BEV features as an intermediate representation. The general architecture is depicted in Figure 4.1. The block highlighted in grey is inherited from an exchangeable camera-only baseline, whereas the block highlighted in blue represents the proposed radar plug-in module.

The modular architecture contains the following components: Initially, BEV features are extracted from the images and the radar point cloud independently. For this purpose, a backbone network is used to extract features from each image, which are then transformed into joint BEV features using a view transformer. The radar encoder is designed to produce BEV features that match the shape and geometric orientation of the camera BEV features. These features are subsequently fused by concatenation and then processed through a 1×1 convolutional layer, which reduces the embedding dimension to match the original dimension of the camera BEV features. Following this, the same BEV encoder and 3D detection heads can be used as in the respective camera-only network, thus introducing minimal overhead with the fusion.

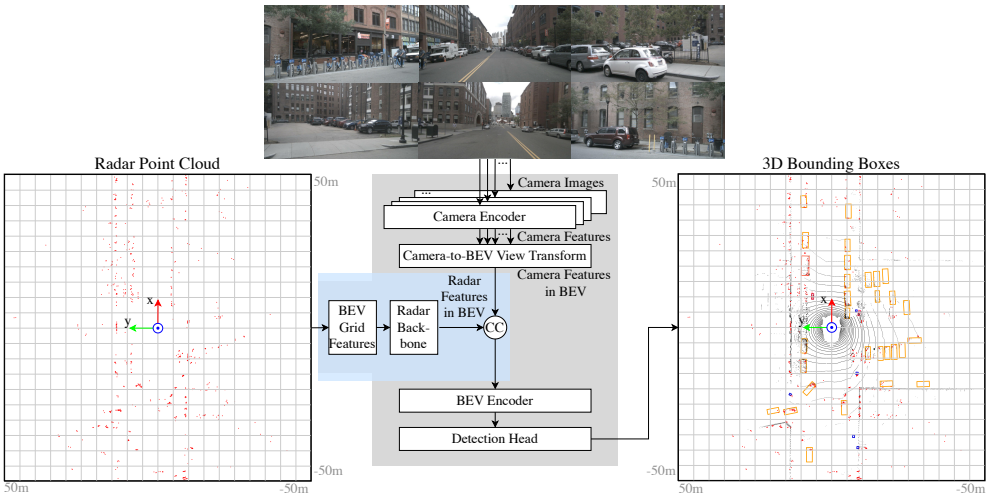


Figure 4.1.: Overview of RC-BEVFusion network architecture. The block marked in gray is inherited from an exchangeable camera-only baseline, while the block marked in blue shows the proposed radar-camera fusion plug-in module [13].

4.2. Radar encoders

In this study, we propose two radar encoders, namely RadarGridMap and BEVFeatureNet. Their structure and functionality is presented in the subsequent sections. Both encoders include two main stages: Initially, a regular,

structured BEV grid is created from the sparse radar point cloud. Subsequently, a convolutional backbone is applied to further encode the BEV features.

4.2.1. RadarGridMap

Motivated by [82], we design a hand-crafted radar BEV grid. Each detection is mapped to a cell on the grid, which is then populated with four channels: the number of detections per cell, the maximum RCS value, and the minimum and maximum signed compensated Doppler values. We chose this setup as a good representative choice and note that it is the best setup we found experimentally. We included some ablative studies for the hand-crafted radar grid in Section 4.4.5.

Following the grid mapping, we use a generalized ResNet [26] with 16 layers, grouped into residual blocks with BatchNorm and ReLU, as the radar backbone. Two downsampling stages are used, which double the channels but decrease the resolution of the BEV grid. The BEV grid size is chosen to ensure that the output of the radar backbone matches the shape of the camera BEV features.

4.2.2. BEVFeatureNet

The design of BEVFeatureNet, illustrated in Figure 4.2, is influenced by the pillar feature encoding of PointPillars [84], but adapted for radar data. Each radar detection is mapped to a cell in a predefined BEV grid. In the conventional pillar feature encoding, every point in the LiDAR point cloud has coordinates x, y, and z, along with the measured reflectivity. Additionally, the point is enriched with distances to the arithmetic mean of all points in its pillar, denoted as x_c, y_c, and z_c, and the offsets to the pillar center, x_p and y_p. The 2+1D radar point cloud in nuScenes [44] lacks a z-coordinate and reflectivity, but contains a radial velocity v_d, measured via the Doppler effect, and an RCS value. Consequently, we omit the z-axis and its augmented feature, substitute the reflectivity with the RCS, and incorporate the radial velocity values. We further aggregate multiple radar sweeps, appending the timestamp difference to the latest sweep Δt to each point. This yields the 9-dimensional feature set:

$$\boldsymbol{f} = [x, y, \mathrm{RCS}, v_d, \Delta t, x_c, y_c, x_p, y_p] \tag{4.1}$$

Following the approach in [84], a set of non-empty BEV grid cells n_b with a predetermined number of points per cell n_p is used to create a dense tensor of dimensions (n_f, n_b, n_p) with $n_f = 9$. If the count of non-empty BEV grid cells or the points per cell is lower or higher than the designated number, we apply zero-padding or random sampling, respectively, to ensure that the dense tensor has fixed dimensions. Each point is then processed through a simplified PointNet [131], consisting of a 1×1 convolutional layer followed by BatchNorm and ReLU. This results in a tensor of shape (c, n_b, n_p), before applying a max operation over the points per cell, decreasing its dimensions to (c, n_b).

Subsequently, the c-dimensional features are mapped back to their respective positions on the BEV grid. Lastly, a convolutional backbone identical to the one used for RadarGridMap is applied.

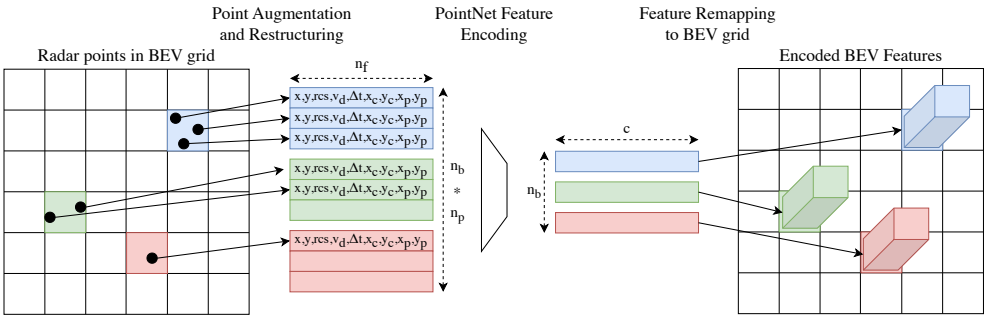


Figure 4.2.: BEVFeatureNet radar encoder. The radar detections are mapped to BEV grid cells for augmentation and restructuring into a dense tensor. After applying a simplified PointNet, the encoded features are remapped to the BEV grid [13].

4.3. Camera-only baselines

Various camera-only baselines are selected to study the plug-in character of the radar-camera fusion module. In this section, we list the details regarding the examined camera baselines. All architectures share the same detection head and loss, which is detailed separately. We provide a concise overview of the modules in Table 4.1.

Table 4.1.: *Camera-only network configurations. For FPN-LSS, SECOND-FPN and Generalized ResNet, the number indicates the output channels of the module. For LSS and MatrixVT, the numbers indicate the channels and the resolution of the output BEV feature in meters. MF denotes multi-frame temporal fusion.*

Model	Camera Encoder	View Transformer	BEV Encoder	Det. Head	Input Res.	MF
BEVDet [69]	Swin-Tiny FPN-LSS-256	LSS 80-0.4-0.4	Gen. ResNet-512 FPN-LSS-256	Center-Point	704x256	
BEVDepth [72]	ResNet-50 SECOND-FPN-128	LSS 80-0.8-0.8	ResNet-18 SECOND-FPN-64	Center-Point	704x256	✓
BEVStereo [73]	ResNet-50 SECOND-FPN-128	LSS 80-0.8-0.8	ResNet-18 SECOND-FPN-64	Center-Point	704x256	✓
MatrixVT [75]	ResNet-50 SECOND-FPN-128	MatrixVT 80-0.8-0.8	ResNet-18 SECOND-FPN-64	Center-Point	704x256	

4.3.1. BEVDet

BEVDet [69] is the first image-only network that uses BEV-based features for object detection on the nuScenes dataset. Initially, high-level features from

each of the n_{img} input images, each with a shape of $(h_{\text{img}}, w_{\text{img}})$, are extracted independently using a SwinTransformer-Tiny [27] backbone. The resulting multi-scale feature maps are then processed by the feature pyramid network (FPN-LSS) from [66], which upscales low-resolution feature maps to align with high-resolution ones. After concatenating the upscaled multi-scale feature maps, they are run through a ResNet [26] block, resulting in a feature map of shape $(n_{\text{img}}, h_{\text{img}}/8, w_{\text{img}}/8, n_c)$ with n_c feature channels. Subsequently, a 1×1 convolution paired with a Softmax function is used for a dense depth classification into d pre-defined depth bins. An outer product across feature and depth classification channels yields a large tensor of shape $(n_{\text{img}}, h_{\text{img}}/8, w_{\text{img}}/8, d, c)$, which contains the aggregated features and depth classification scores for each pixel of the feature map. Utilizing the intrinsic and extrinsic calibration matrices for each camera, the feature tensor can be un-projected into a pseudo point cloud. To reduce the computational demands, the vertical dimension of the pseudo point cloud is condensed by summing features from all points inside the same cell of a predetermined BEV grid with shape $(h_{\text{bev}}, w_{\text{bev}})$. We follow the implementation from [76], providing efficient BEV pooling and deploying a much heavier view transformer for enhanced depth estimation, which is crucial for associating features from radar and camera. The fused BEV features are further encoded using the generalized ResNet [26] structure from [66] as a BEV encoder and again processed by FPN-LSS. Finally, the CenterPoint [132] detection head is used to produce the 3D bounding box outputs.

4.3.2. BEVDepth

BEVDepth [72] uses a similar structure as BEVDet [69], but aims for more precise depth estimation. To this end, the depth estimation convolutional layer in BEVDet [69] is substituted with a camera-aware DepthNet module. This module concatenates and flattens the camera's intrinsic and extrinsic calibration parameters, and uses an MLP to rescale them to the dimension of the image features n_c. The calibration vector is then utilized to re-weight the image features using a Squeeze-and-Excitation module [133]. During training, BEVDepth [72] uses the depth value from projected LiDAR points on the image for direct supervision of the depth estimation through binary cross-entropy loss. The released configuration encodes the images with a ResNet-50 [26] backbone followed by the feature pyramid net from [134], known as SECOND-FPN, which upsamples and concatenates the multi-scale feature maps. The BEV encoder is a ResNet-18 followed by SECOND-FPN. The view transformer has a lower resolution than the one we use for BEVDet [69]. BEVDepth also uses a multi-frame fusion with one previous keyframe, thus two images from each camera taken 500 ms apart are used to create the BEV features, resulting in more accurate velocity estimation.

4.3.3. BEVStereo

BEVStereo [73] builds upon BEVDepth [72] and maintains the same configuration for most modules. Besides monocular depth estimation with DepthNet,

a temporal stereo depth estimation module is introduced, which is based on multi-view-stereo techniques [135]. For each pixel in the current image feature map, depth candidates are predicted and used to retrieve corresponding features from the previous image features through homography warping. The confidence level of each candidate is evaluated based on the similarity between current and previous features, contributing to the iterative optimization of depth candidates. After three iterations, the depth candidates are used to construct the stereo depth. A convolutional WeightNet module is necessary for pixels lacking corresponding units in the previous image, combining the current and previous monocular depth estimations with the stereo depth estimation to generate the final depth estimates.

4.3.4. MatrixVT

MatrixVT [75] introduces an alternative view transformer that is computationally more efficient. Initially, the view transformation step of LSS [66] is generalized into matrix operations, which leads to a large and sparse feature transportation tensor of shape $(h_{\text{bev}}, w_{\text{bev}}, n_{\text{img}}, h_{\text{img}}/8, w_{\text{img}}/8, d)$. This tensor transforms the image feature tensor with depth classification scores to BEV features. To address its extensive size, the image feature and dense depth prediction are compressed along the vertical dimension, resulting in prime feature and prime depth matrices. This step is deemed feasible due to the low response variance in the image's height dimension. To further reduce its sparsity, the feature transportation tensor is decomposed orthogonally into a ring tensor, encoding distance information, and a ray tensor, encoding directional information. With efficient mathematical operations, the resulting view transformer reduces computational cost and memory requirements. We use a configuration with the same elements as in BEVDepth [72], only substituting the view transformer with MatrixVT [75] and excluding multi-frame fusion.

4.3.5. Detection Head and Loss

All examined camera baselines use the CenterPoint [132] detection head, so we can apply the same loss in all architectures. For each class l, CenterPoint [132] predicts a BEV heatmap with peaks at object center locations. The heatmap is trained using a ground-truth heatmap $\boldsymbol{M}$ filled with Gaussian distributions around ground truth object centers. Given the heatmap score $\hat{\boldsymbol{M}}_{lij}$ at position i, j in the BEV grid and the ground truth $\boldsymbol{M}_{lij}$, we can compute the Gaussian focal loss [136] as:

$$l_{\text{hm}} = -\frac{1}{n_o} \sum_{l=1}^{n_l} \sum_{i=1}^{h_{\text{bev}}} \sum_{j=1}^{w_{\text{bev}}} \begin{cases} (1 - \hat{\boldsymbol{M}}_{kij})^{\alpha} \log(\hat{\boldsymbol{M}}_{kij}) & \boldsymbol{M}_{kij} = 1 \\ (1 - \boldsymbol{M}_{kij})^{\beta} (\hat{\boldsymbol{M}}_{kij})^{\alpha} \log(1 - \hat{\boldsymbol{M}}_{kij}) & \text{otherwise} \end{cases} \tag{4.2}$$

where n_o is the number of objects per image, n_l is the number of classes, h_{bev} and w_{bev} are the height and width of the BEV grid, and α and β are hyperparameters. In addition, CenterPoint [132] has regression heads that output all parameters needed to decode 3D bounding boxes: a sub-pixel location refinement, a height above ground, the dimensions, the velocity, and the sine

and cosine of the yaw rotation angle. The regression heads are trained with L1 loss.

4.4. nuScenes experiments

In this section, we perform extensive experiments on the nuScenes dataset [44] using the proposed RC-BEVFusion architecture with different components. First, we list the training details including important hyperparameters settings. We then use the validation set to compare the performance of our radar-camera fusion networks with their respective image-only baselines. Subsequently, we select a model for submission on the nuScenes test benchmark, so it can be compared with other published radar-camera fusion algorithms. Finally, we provide ablations regarding our augmentation strategies, which play a pivotal role for the performance of the networks.

4.4.1. Training and evaluation details

For our radar-camera fusion networks, we adopt the configurations from the camera-only baselines presented in Table 4.1 to ensure a fair comparison. We use a BEV grid of $x, y \in [-51.2\,\text{m}, 51.2\,\text{m}]$ with a cell size of 0.1 m. Furthermore, we design the radar encoder branch to ensure that the BEV features align in shape and orientation with the camera BEV features. To enhance point cloud density while minimizing the reliance on outdated data, we aggregate five radar sweeps, corresponding to 300 ms of past data. The resulting aggregated radar point cloud remains sparse relative to LiDAR data, allowing for a significant reduction in the number of non-empty grid cells and points per grid cell of the BEVFeatureNet, especially when compared to the pillar feature encoding in [84]. We empirically determine that setting $n_b = 2000$, $n_p = 10$, and $c = 32$ is sufficient, which minimizes the computational overhead of the BEVFeatureNet radar encoder. For the Gaussian focal loss, we follow the configuration from [136], setting $\alpha = 2$ and $\beta = 4$. We train for 20 epochs with an AdamW [43] optimizer, a base learning rate of 2×10^{-4}, and weight decay of 10^{-2} for batch size 32 and a cyclic learning rate policy. To account for potential class imbalances, class-balanced grouping and sampling (CBGS) [137] is applied. For evaluation, we use the official nuScenes [44] metrics introduced in Chapter 2.2.7: the mAP, the true positive metrics for translation, scale, orientation, velocity and nuScenes attribute, as well as the condensed NDS. Note that the NDS correlates best with the overall driving task [45].

4.4.2. NuScenes validation results

We show the results for our radar-camera fusion networks with respect to the camera-only baseline BEVDet on the nuScenes validation split in Table 4.2. For both proposed radar encoders, the fusion offers significant performance increases, with the mAP increasing up to 24 % and the NDS 28 %. The up to 23 % reduced translation error shows how the direct depth measurements provided by the radar can lead to more precise location predictions. The most

significant improvement of 55 % is achieved for the velocity error, which is enabled by the direct velocity measurement of the radar. This effect also helps determining whether an object is currently moving or stopped, thus reducing the attribute error. The two metrics that remain relatively unaffected by the fusion are scale and orientation error. This is as expected since the sparse radar detections do not help to determine an object's size or orientation. Overall, we observe similar results for the RadarGridMap and the BEVFeatureNet encoder. This demonstrates the effectiveness and modularity of the proposed BEV-based feature fusion. While the RadarGridMap offers slightly better runtime, we generally recommend using the BEVFeatureNet because it requires less hand-crafted input, is more scalable, and achieves slightly better results. Thus, we focus the remainder of our experiments on BEVFeatureNet.

Table 4.2.: *Experimental results for the proposed radar-camera fusion based on BEVDet with varying radar encoder branches on the nuScenes val split. The inference latency T is measured on a Nvidia RTX 2080 Ti.*

	Radar encoder		mAP ↑	NDS ↑	mATE ↓	mASE ↓	mAOE ↓	mAVE ↓	mAAE ↓	T [ms]
[69]	None		0.350	0.411	0.660	0.275	0.532	0.918	0.260	122
Ours	RadarGridMap		0.429	0.525	0.523	0.272	0.507	0.412	0.183	132
		Δ_r	23%	28%	-21%	-1%	-5%	-55%	-30%	
Ours	BEVFeatureNet		0.434	0.525	0.511	0.270	0.527	0.421	0.182	139
		Δ_r	24%	28%	-23%	-2%	-1%	-54%	-30%	

In Table 4.3, we use the BEVFeatureNet encoder as a plug-in branch in different camera-only baselines. We observe significant performance increase for all examined architectures, again confirming the modularity of the proposed architecture. There are two potential reasons for the difference in relative performance increase between the camera-only baselines. First, BEVDepth and BEVStereo use temporal fusion and therefore achieve better velocity prediction and overall scores, leading to smaller margins for the radar-camera fusion. Second, we use a BEVDet variant with a heavier view transformer especially designed for fusion on the BEV space. This modification may explain the relatively high performance gains.

Finally, we also measure the inference latency on an Nvidia RTX 2080 Ti GPU to demonstrate that our fusion approach introduces only small computational overhead due to the efficient radar encoder design.

In addition to the aggregated metrics in Table 4.3, we present more detailed, class-wise results in Table 4.4. To reduce the amount of data, values of the three most common dynamic classes are listed: *Car*, *Pedestrian*, and *Truck*. The results show that the average precision for important classes increased across the board, with an even greater increase for the larger classes that are more easily detectable by radar: *Car* and especially *Truck*. Translation errors decreased most notably for *Cars*, while scale errors remained relatively unchanged, except for a slight decrease for the proposed model based on BEVDepth [72], which may be due to statistical effects. Orientation errors improved for models without temporal fusion, but increased for the proposed

Table 4.3.: *Experimental results for the proposed radar-camera fusion with our BEVFeatureNet radar encoder used in different architectures on the nuScenes val split. The inference latency T is measured on a Nvidia RTX 2080 Ti. *The implementation of BEVDet-Tiny with a heavier view transformer from [76] was used. †The results are listed as reported by the authors.*

	Camera model	Radar fusion		mAP ↑	NDS ↑	mATE ↓	mASE ↓	mAOE ↓	mAVE ↓	mAAE ↓	T [ms]
[69]	BEVDet*			0.350	0.411	0.660	0.275	0.532	0.918	0.260	122
Ours	BEVDet*	✓		0.434	0.525	0.511	0.270	0.527	0.421	0.182	139
			Δ_r	24%	28%	-23%	-2%	-1%	-54%	-30%	
[72]†	BEVDepth			0.359	0.480	0.612	0.269	0.507	0.409	0.201	132
Ours	BEVDepth	✓		0.405	0.521	0.542	0.274	0.512	0.309	0.181	146
			Δ_r	13%	9%	-11%	2%	1%	-24%	-10%	
[73]†	BEVStereo			0.372	0.500	0.598	0.270	0.438	0.367	0.190	308
Ours	BEVStereo	✓		0.423	0.545	0.504	0.268	0.453	0.270	0.174	322
			Δ_r	14%	9%	-16%	-1%	3%	-26%	-8%	
[75]	MatrixVT			0.319	0.400	0.669	0.281	0.494	0.912	0.238	54
Ours	MatrixVT	✓		0.386	0.495	0.549	0.275	0.539	0.423	0.193	64
			Δ_r	21%	24%	-18%	-2%	9%	-54%	-19%	

model based on BEVDepth, possibly indicating difficulty in estimating the orientation of some additionally detected *Cars* and *Trucks*. Velocity errors saw a significant reduction, especially for *Cars* and *Trucks* and for models without temporal fusion. Even with temporal fusion, there was a considerable decrease in velocity error. Finally, the attribute error decreased most for *Pedestrians*, with radar making it easier to determine if a *Pedestrian* is moving or standing.

4.4.3. Results for rain and night scenes

In this section, the performance of the proposed RC-BEVFusion is evaluated in adverse conditions for the camera. To this end, the scenes are filtered according to the scene descriptions in the nuScenes [44] validation set for the terms "rain" and "night", respectively, to obtain 27 rain and 15 night scenes. These scenes are then evaluated for BEVDet [69] and the corresponding RC-BEVFusion algorithm with BEVFeatureNet. Since not all classes are represented in the rain and night scenes, the averaged metrics across all classes are less meaningful. Therefore, again class-wise results are presented of the three most common dynamic classes: *Car*, *Pedestrian*, and *Truck*. The results in Table 4.5 show that compared to the overall AP increase across all scenes listed in Table 4.4, there were higher improvements for rain and especially for night scenes. In conclusion, the camera-only model struggles in these adverse conditions, particularly in detecting *Pedestrians* and *Trucks* at night. This is where the proposed radar-camera fusion can add the most value. Note that the true positive metrics for *Pedestrians* and *Trucks* at night should be treated with caution due to the low number of matches for the camera-only model. As discussed above, a significant decrease in translation, velocity, and attribute errors can be observed, while the scale error remains relatively unchanged.

Table 4.4.: *Experimental class-wise results for the proposed radar-camera fusion with our BEVFeatureNet radar encoder used in different architectures on the three most common dynamic classes of the nuScenes val split. *The implementation of BEVDet-Tiny with a heavier view transformer from [76] is used. †The results are listed as reported by the authors.*

	Camera model	Radar fusion	Class		AP ↑	ATE ↓	ASE ↓	AOE ↓	AVE ↓	AAE ↓
[69]	BEVDet*		car		0.538	0.529	0.157	0.128	0.992	0.232
			ped.		0.377	0.700	0.304	1.383	0.872	0.759
			truck		0.290	0.679	0.209	0.165	0.911	0.252
Ours	BEVDet*	✓	car		0.700	0.315	0.156	0.106	0.395	0.196
				Δ_r	30%	-40%	-1%	-17%	-60%	-16%
			ped.		0.468	0.528	0.300	1.016	0.701	0.329
				Δ_r	24%	-25%	-1%	-27%	-20%	-57%
			truck		0.405	0.493	0.206	0.131	0.329	0.202
				Δ_r	40%	-27%	-1%	-21%	-64%	-20%
[72]†	BEVDepth		car		0.559	0.475	0.157	0.112	0.370	0.205
			ped.		0.363	0.690	0.297	0.831	0.491	0.244
			truck		0.270	0.659	0.196	0.103	0.356	0.181
Ours	BEVDepth	✓	car		0.661	0.356	0.162	0.134	0.289	0.193
				Δ_r	18%	-25%	3%	20%	-22%	-6%
			ped.		0.410	0.585	0.295	0.732	0.434	0.208
				Δ_r	13%	-15%	-1%	-12%	-12%	-15%
			truck		0.332	0.563	0.214	0.162	0.254	0.198
				Δ_r	23%	-15%	9%	57%	-29%	9%
[73]†	BEVStereo		car		0.567	0.457	0.156	0.104	0.343	0.204
			ped.		0.402	0.653	0.297	0.803	0.479	0.249
			truck		0.299	0.650	0.205	0.103	0.321	0.197
Ours	BEVStereo	✓	car		0.687	0.324	0.159	0.106	0.250	0.192
				Δ_r	21%	-29%	2%	2%	-27%	-6%
			ped.		0.469	0.530	0.295	0.694	0.413	0.197
				Δ_r	17%	-19%	-1%	-14%	-14%	-21%
			truck		0.364	0.516	0.208	0.106	0.214	0.184
				Δ_r	22%	-21%	1%	3%	-33%	-7%
[75]	MatrixVT		car		0.517	0.529	0.162	0.155	1.049	0.221
			ped.		0.309	0.746	0.300	1.204	0.813	0.465
			truck		0.244	0.713	0.213	0.154	0.917	0.219
Ours	MatrixVT	✓	car		0.658	0.346	0.162	0.141	0.400	0.190
				Δ_r	27%	-35%	0%	-9%	-62%	-14%
			ped.		0.386	0.618	0.298	1.071	0.695	0.335
				Δ_r	25%	-17%	-1%	-11%	-15%	-28%
			truck		0.320	0.547	0.214	0.133	0.337	0.201
				Δ_r	31%	-23%	0%	-14%	-63%	-8%

This time, there is also a considerable improvement in orientation error. This finding suggests that the camera-only model struggles to accurately predict orientation in these adverse conditions, and further emphasizes the potential of radar-camera fusion in such scenarios.

Table 4.5.: *Experimental class-wise results for the proposed radar-camera fusion based on BEVDet and our BEVFeatureNet radar encoder on the three most common dynamic classes of the nuScenes val split, filtered by rain and night scenes, respectively. *The implementation of BEVDet-Tiny with a heavier view transformer from [76] is used.*

Split		Camera model	Radar fusion	Class		AP ↑	ATE ↓	ASE ↓	AOE ↓	AVE ↓	AAE ↓
Rain	[69]	BEVDet*		car		0.517	0.548	0.158	0.133	0.693	0.151
				ped.		0.218	0.748	0.360	1.679	1.043	0.725
				truck		0.276	0.751	0.216	0.153	0.479	0.120
	Ours	BEVDet*	✓	car		0.723	0.304	0.160	0.121	0.277	0.141
					Δ_r	40%	-45%	1%	-9%	-60%	-7%
				ped.		0.338	0.516	0.360	1.011	0.849	0.332
					Δ_r	55%	-31%	0%	-40%	-19%	-54%
				truck		0.449	0.539	0.200	0.120	0.235	0.108
					Δ_r	63%	-28%	-7%	-22%	-51%	-10%
Night	[69]	BEVDet*		car		0.403	0.527	0.137	0.111	1.619	0.485
				ped.		0.045	0.664	0.296	1.509	0.675	0.469
				truck		0.057	0.630	0.221	0.151	2.795	0.582
	Ours	BEVDet*	✓	car		0.611	0.310	0.137	0.092	0.538	0.469
					Δ_r	52%	-41%	0%	-17%	-67%	-3%
				ped.		0.191	0.262	0.283	0.810	0.592	0.037
					Δ_r	324%	-61%	-4%	-46%	-12%	-92%
				truck		0.265	0.304	0.181	0.127	0.616	0.697
					Δ_r	365%	-52%	-18%	-16%	-78%	20%

4.4.4. NuScenes test results

For the nuScenes test benchmark, the results achieved in this work are compared to other previously published radar-camera fusion models in Table 4.6. It should be noted that many methods tune their models for the benchmark submission by enlarging their network and the input image resolution and by using test time augmentation. All of these techniques trade off smaller performance gains for large computational cost and are therefore not helpful in

Table 4.6.: *Experimental results for published radar-camera fusion models on the nuScenes test benchmark. Experiment conducted with the model based on BEVFeatureNet and BEVDet.*

Model	mAP ↑	NDS ↑	mATE ↓	mASE ↓	mAOE ↓	mAVE ↓	mAAE ↓
CenterFusion [104]	0.326	0.449	0.631	0.261	0.516	0.614	0.115
CRAFT [108]	0.411	0.523	0.467	0.268	0.456	0.519	**0.114**
TransCAR [110]	0.422	0.522	0.630	0.260	**0.383**	0.495	0.121
Ours	**0.476**	**0.567**	**0.444**	**0.244**	0.461	**0.439**	0.128

an automotive application in which fast decisions are required. For instance, the authors of BEVDet achieve 15.6 frames per second with the tiny configuration similar to the one used in this work, while the base configuration used for the benchmark achieves only 1.9 frames per second [69]. In this work, we choose to combat this trend by only retraining the model with scenes from the training and validation set for the test submission. As shown in Table 4.6, the proposed RC-BEVFusion with BEVFeatureNet and BEVDet significantly outperforms previously published radar-camera fusion networks in all metrics except the orientation error, even without tuning for the benchmark, while achieving 7.2 frames per second. A key advantage of this architecture compared to existing methods [104], [108], [110], is that radar and camera features can equally contribute to the final detections, allowing the model to detect objects that might be missed by each single modality.

4.4.5. Ablations

In Table 4.7 we show the impact of different augmentation strategies on the proposed RC-BEVFusion with BEVFeatureNet and BEVDet. As in [69], we use a two-stage augmentation. First, 2D augmentation is applied by rotating, resizing and horizontally flipping the images. In the BEV view transformer, the 2D augmentations are reversed so that the original orientation is retained. Then, 3D augmentations are applied to the BEV features, radar points and ground truth bounding boxes. They include scaling, rotating, as well as horizontal and vertical flipping. The results show that 3D augmentation is very important, while 2D augmentation helps to improve the results only slightly. This is due to the camera encoder branch being shared across the six cameras, leading to more variety in the images than in the BEV plane. Therefore, the BEV encoder is more prone to overfitting and requires more augmentation [69].

Table 4.7.: *Ablation study for 2D and 3D augmentations. All experiments conducted with the model based on BEVFeatureNet and BEVDet.*

3D	2D	mAP ↑	NDS ↑	mATE ↓	mAVE ↓
		0.380	0.453	0.595	0.676
✓		0.419	0.513	0.520	0.478
✓	✓	**0.434**	**0.525**	**0.511**	**0.421**

Further, we examine two potential augmentation strategies for the proposed RadarGridMap encoder, which have been proposed based on a similar setup in [83]. To combat the sparsity of the radar grid, a blurring filter spreads values from the reference cell to neighboring cells depending on the number of detections per reference cell. Further, the authors find that the distribution of compensated Doppler values is heavy-sided towards zero and introduce a Doppler skewing function to spread the distribution for values close to zero. The results are shown in Table 4.8. The grid mapping variant without extra augmentation works quite well, confirming the effectiveness of the approach.

The blurring filter has little impact on the results and thus is not deemed necessary. The Doppler skewing function leads to higher velocity errors and should therefore be omitted.

Table 4.8.: *Ablation study for augmentation strategies of the RadarGridMap encoder: a blurring filter (BF) and a Doppler skewing (DS) technique. All experiments conducted with the model based on BEVDet.*

BF	DS	mAP ↑	NDS ↑	mATE ↓	mAVE ↓
		0.429	**0.525**	0.523	**0.412**
✓		0.424	0.524	0.517	0.439
	✓	0.425	0.508	0.523	0.482
✓	✓	0.423	0.487	**0.516**	0.650

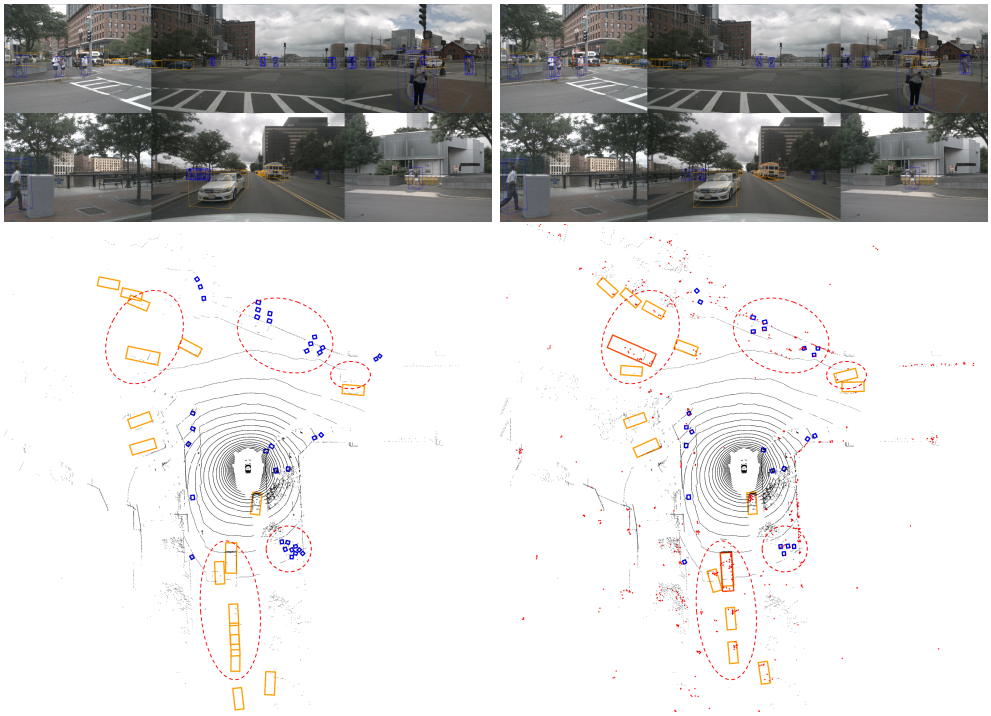
4.4.6. Qualitative Evaluation

To get a better impression of the difference in object detection performance, an inference example is presented for BEVDet [69] and RC-BEVFusion based on BEVFeatureNet and BEVDet in Figure 4.3. The camera-only inference results and radar-camera fusion results are shown in the left and right subfigure, respectively. Further, the full surround view with all six cameras is given, the top row shows the front left, center and right camera, while the bottom row shows the back right, center and left camera. On the bottom, a BEV perspective is shown with LiDAR points in black for reference and radar points in red for the fusion network only. Each perspective includes the projected 3D bounding boxes predicted by the networks, where the color indicates the class: *Pedestrians* are blue, *Cars* are yellow and *Trucks* are orange.

The selected scene is a crowded intersection with lots of *Cars* and *Pedestrians*. At first, the visual impression when looking at the camera images is that most objects are well detected. However, comparing the dashed red ellipses in the BEV perspective on the right, it can be observed that the radar-camera fusion enables much more accurate detection of the *Pedestrians* in the front and back right area, as well as the *Cars* in the front left, front right and back area.

4.5. Cross-dataset study

After evaluating the proposed RC-BEVFusion model on the nuScenes dataset [44], we conduct an additional study across datasets by additionally using the View-of-Delft dataset [47]. While nuScenes provides conventional radar and camera data with full surround view, View-of-Delft includes high-performance radar data limited to front view. Thus, for better comparability between the datasets, in this study we use only the front camera and the front-facing radars in nuScenes as well as the respective 3D bounding box annotations that fall inside the field-of-view of the front camera. To adapt our models for the front view, we re-define the BEV grid as $x \in [0\,\text{m}, 51.2\,\text{m}]$ and $y \in$



(a) BEVDet **(b)** Proposed RC-BEVFusion

Figure 4.3.: Inference example from nuScenes [44] at daytime. Predicted 3D bounding boxes projected to all six cameras (top) and BEV plane (bottom) with LiDAR points (black) and radar points (red) for reference. The proposed fusion network more accurately detects *Pedestrians* in the front and back-right area andvehicles in the front-left, front-right and back area (s. dashed red ellipses) [13].

$[-25.6\,\mathrm{m}, 25.6\,\mathrm{m}]$ with a step size of $0.1\,\mathrm{m}$. Further, the considered classes are reduced to *Pedestrian*, *Cyclist* and *Car*, since these are the only classes with sufficient annotations in View-of-Delft.

In addition to the presented radar-camera fusion, the proposed object detection network is used also in radar-only and camera-only configurations, where the encoded camera and radar feature inputs are removed, respectively. Since View-of-Delft [47] has less camera frames and shows a smaller visual variability, transfer learning of the camera feature encoding is investigated. To this end, the camera-only network is pretrained with nuScenes [44] before the camera-only and radar-camera fusion networks are fine-tuned with View-of-Delft to improve the performance.

4.5.1. Training and evaluation details

The networks are trained and evaluated with the official training and validation splits of nuScenes [44] and View-of-Delft [47]. Except the dimensions of the BEV grid, we keep most of the settings as in Section 4.4. We aggregate 5 sweeps of radar point clouds and train the models with an AdamW [43] optimizer and applying CBGS [137]. Due to the different dataset sizes, the models

are trained for 20 epochs on nuScenes and for 80 epochs on View-of-Delft, before the best-performing model on the validation set is selected.

To evaluate the 3D object detection performance, the commonly used AP and mAP metrics introduced in Chapter 2.2.7 are considered, where the latter corresponds to an average over relevant classes. For reproducibility, the suggested metric implementations of the datasets are used, which approximate the area under the curve at sampling points. For the association of predicted and ground truth bounding boxes, the BEV center point distance is used for nuScenes [44], and the intersection-over-union ratio in 2D, BEV and 3D is used for View-of-Delft [47]. For nuScenes, some additional metrics are calculated for the true-positive detections, including translation, scale, orientation, velocity, and attribute errors.

4.5.2. Quantitative results

Table 4.9 shows the AP results as well as detailed true positive metrics on nuScenes [44] for classes *Pedestrian*, *Cyclist*, and *Car*. It can be observed that the radar-only network is able to detect *Cars* to a certain degree but it can barely detect *Pedestrians* and *Cyclists*, as these often do not have a single radar detection and are difficult to distinguish from the environment. The camera-only network does better across all classes and shows that with a large dataset like nuScenes, monocular 3D object detection is possible. When fusing radar and camera, the performance for all classes improves significantly, demonstrating how well the combination of the semantic value from the camera and the geometric value from the radar can work together.

Table 4.9.: *Experimental validation results for RC-BEVFusion on nuScenes front-view. All experiments conducted with the model based on BEVFeatureNet and BEVDet.*

Modality	mAP ↑	Class	AP ↑	ATE ↓	ASE ↓	AOE ↓	AVE ↓	AAE ↓
Radar	0.125	ped.	0.031	0.53	0.31	**0.71**	0.67	**0.11**
		cyc.	0.054	0.52	0.34	**0.45**	**0.11**	0.02
		car	0.290	0.44	0.20	0.23	0.60	**0.11**
Camera	0.277	ped.	0.267	0.77	**0.30**	1.40	0.83	0.73
		cyc.	0.173	0.76	**0.28**	1.13	0.52	**0.01**
		car	0.391	0.64	**0.17**	0.36	1.72	0.31
Radar-camera	**0.406**	ped.	**0.366**	**0.49**	**0.30**	0.88	**0.63**	0.27
		cyc.	**0.264**	**0.45**	**0.28**	0.93	0.29	0.03
		car	**0.589**	**0.31**	**0.17**	**0.14**	**0.44**	0.23

Looking at the true positive metrics, It should be noted that the radar-only metrics for *Pedestrian* and *Cyclist* are not meaningful due to the very low AP values for these classes. A comparison of radar-only and camera-only performance for *Cars* reveals that despite the lower AP, the radar does better in translation, orientation, velocity, and nuScenes attribute estimation. This is likely due to the direct distance and velocity measurement. The radar-camera

fusion further improves all metrics, again showing the synergies of the different modalities.

Table 4.10.: *Experimental validation results for RC-BEVFusion on View-of-Delft with optional pretraining on nuScenes front-view. All experiments conducted with the model based on BEVFeatureNet and BEVDet.*

Modality	Pretrain	mAP 2D	mAP BEV	mAP 3D	Class	AP 2D	AP BEV	AP 3D
Radar		0.485	0.491	0.437	ped.	0.341	0.320	0.294
					cyc.	0.683	0.666	0.654
					car	0.432	0.487	0.363
Camera		0.321	0.142	0.121	ped.	0.245	0.098	0.096
					cyc.	0.405	0.142	0.125
					car	0.313	0.185	0.140
Radar-Camera		0.531	0.535	**0.480**	ped.	0.432	**0.406**	**0.372**
					cyc.	0.715	0.677	0.672
					car	0.446	0.523	0.397
Camera	✓	0.496	0.215	0.169	ped.	0.392	0.126	0.114
					cyc.	0.585	0.266	0.216
					car	0.512	0.254	0.177
Radar-Camera	✓	**0.582**	**0.557**	0.477	ped.	**0.500**	0.394	0.350
					cyc.	**0.720**	**0.692**	**0.677**
					car	**0.527**	**0.585**	**0.404**

For View-of-Delft [47], the AP results in terms of 2D, BEV, and 3D metrics are shown in Table 4.10. With the high-performance radar, the radar-only approach works quite well for all classes, and even achieves the best results for *Cyclists*, which were difficult to detect in nuScenes [44]. The camera-only network performs worse than radar-only for all classes. On the 2D image plane, the results are more competitive, but the difficulty of the monocular depth estimation can be observed in the BEV and 3D metrics, given the limited amount of training data in View-of-Delft. The fusion network once again outperforms the single-modality networks. To provide more variety in the training images used for the camera branch, the camera-only network is pretrained with nuScenes before fine-tuning the camera-only and radar-camera fusion networks with View-of-Delft. For camera-only, significant improvements can be observed in all metrics. The radar-camera fusion network benefits from the pretraining mostly in the 2D metrics, since the BEV and 3D metrics are dominated by the high-performance radar in View-of-Delft.

4.5.3. Exemplary qualitative results

To highlight the differences of the datasets and demonstrate the performance of the best-performing radar-camera fusion models, selected qualitative examples for each dataset are shown in Figure 4.4. In each subfigure, the frontal camera image is shown with the projected radar detections and a BEV representation of the same frame with radar detections in red and the LiDAR point cloud in black. Note that the LiDAR point cloud is shown as a geometric reference only. The colored boxes are the projected 3D bounding boxes, predicted by the

proposed fusion models, where blue is used for *Pedestrians*, red for *Cyclists*, and yellow for *Cars*.

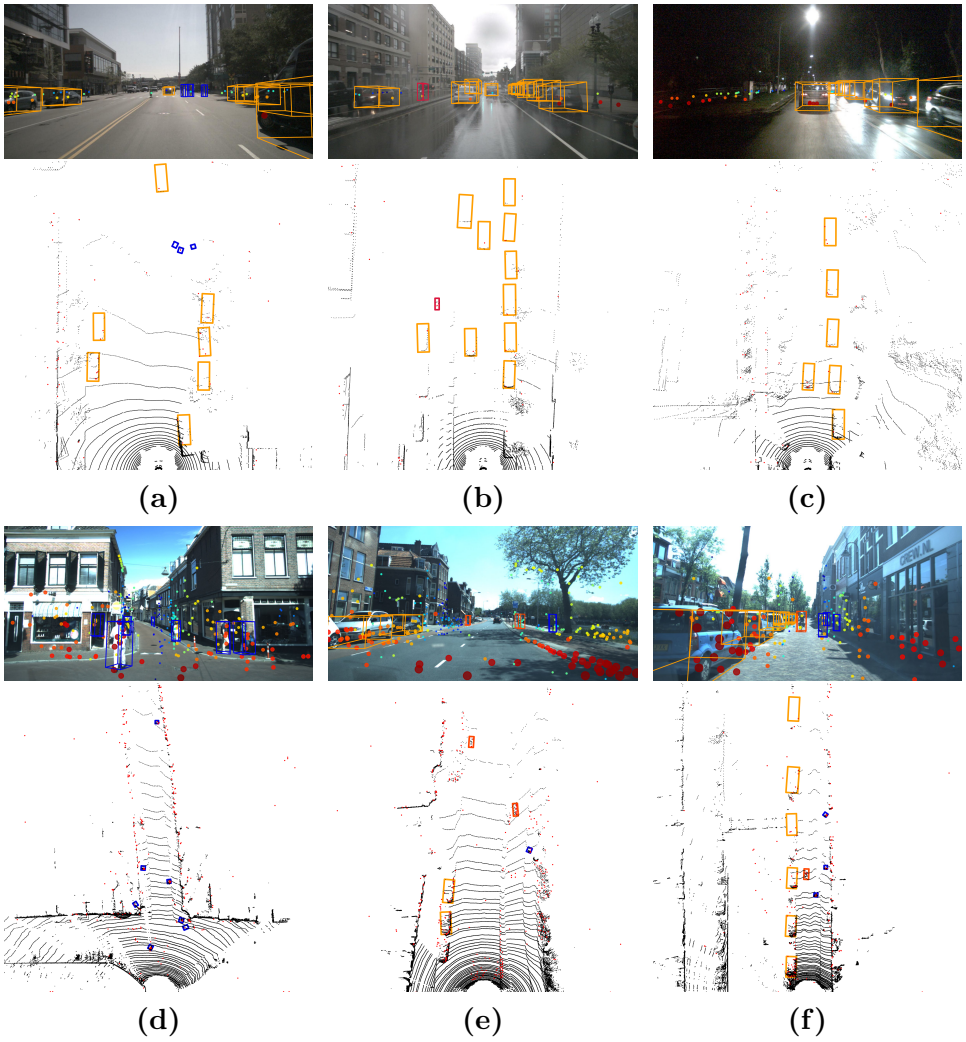


Figure 4.4.: *Qualitative radar-camera fusion results from nuScenes [44] (a)–(c) and View-of-Delft [47] (d)–(f). The respective upper and lower plots show the camera image and the BEV grid, including projected radar detections and predicted bounding boxes, blue for Pedestrian, red for Cyclist, and yellow for Cars. The LiDAR point cloud in the BEV grid is shown for reference only [14].*

The examples in 4.4a–4.4c are from nuScenes [44], whereas the examples in 4.4d–4.4f are from View-of-Delft [47]. When looking at the raw data, the strengths and weaknesses of the two datasets can be observed. In particular, nuScenes offers a large variety in the images, since the scenes are recorded in two different countries, and at different daytimes and weather conditions. However, the radar point cloud is very sparse and does not contain elevation angles. In contrast, View-of-Delft has less variety in the images, but a much denser radar point cloud with elevation angles.

All six frames show how well the proposed RC-BEVFusion models perform. Even though there is dense traffic with many *Cars*, *Pedestrians* and some *Cyclists*, the network is able to detect and localize all the relevant objects in the scenes. Especially in the challenging situation for the camera in 4.4c, the radar can help to accurately localize the *Cars*.

4.6. Conclusion

In this chapter, we have presented a novel radar-camera fusion architecture on the BEV plane. We proposed two radar encoders and showed that they can be integrated into several camera-based architectures that use BEV features. In our experiments, the proposed radar-camera fusion outperform the camera-only baselines by a large margin, demonstrating its effectiveness. The NDS metric, which correlates well with the downstream driving task [45], increases up to 28 %. Without tuning the model for the test submission to avoid unrealistic computational cost, we outperformed previously published radar-camera fusion networks. Our qualitative evaluation shows improved localization accuracy at daytime and higher recall at nighttime. We have further used the proposed radar-camera fusion model to study differences in the nuScenes [44] and View-of-Delft [47] datasets. Our results show that camera-only 3D object detection requires a large dataset with reasonable visual variability, as it is available in the nuScenes dataset. In contrast, the radar-only network profits more from the high-performance radar in View-of-Delft and can cope with a smaller dataset. Also, it can also classify vulnerable road users like *Pedestrians* and *Cyclists*. We conclude that the full potential of radar-camera fusion could be achieved when combining the needs for radar and camera perception, with a dense point cloud and a large visual variability, respectively. Until such a dataset is publicly available, we have shown that pretraining the camera-only network with a large dataset like nuScenes can help to improve the performance of camera-only and radar-camera fusion networks in smaller datasets like View-of-Delft.

Chapter 5

Data preparation

Contents

In the preceding chapters, the enhancement of object detection through the development of novel radar-camera fusion algorithms has been demonstrated, with evaluations conducted using benchmarks from publicly available datasets. However, for the current development of AI-based perception methods with public datasets, often impractical computational resources and excessive dataset tuning are used. It is often overlooked that a perception method is only practically useful if it can be deployed with limited computational resources and generalizes well on unseen data in a real-world scenario. This chapter therefore introduces an experimental vehicle, the AI Sensing test vehicle, utilized to validate the performance of the previously discussed algorithms. Through this exploration, insights into the nuances of real-world deployment are highlighted, contributing to a more grounded comprehension of the intersection between theoretical development and practical application in the field of radar-camera fusion and object detection.

Initially, the sensor configuration used in the AI Sensing test vehicle is detailed, providing an overview of the integral components. This is followed by the real-time computation environment in the Robot Operating System (ROS) designed for data streaming, processing, and recording. Finally, considerations regarding the synchronization of measurements from different sensors are presented.

In summary, this chapter covers:

- An overview of the AI Sensing test vehicle architecture and technical details.
- The intrinsic and extrinsic sensor calibration.
- The real-time data processing system in ROS.
- Data synchronization efforts.

We note that this chapter covers some collaborative work, which we include for completeness, but do not claim individual contribution. This includes:

- The mechanical and electrical setup of the components in the AI Sensing test vehicle.
- The software for camera and camera-to-LiDAR calibration.

5.1. AI Sensing test vehicle

The AI Sensing test vehicle is a Peugeot 3008 equipped with a set of additional sensors and processing systems. The sensors include 5 cameras, 5 radars, 1 LiDAR and one Global Navigation Satellite System (GNSS) sensor. The sensor coverage is shown in Figure 5.1.

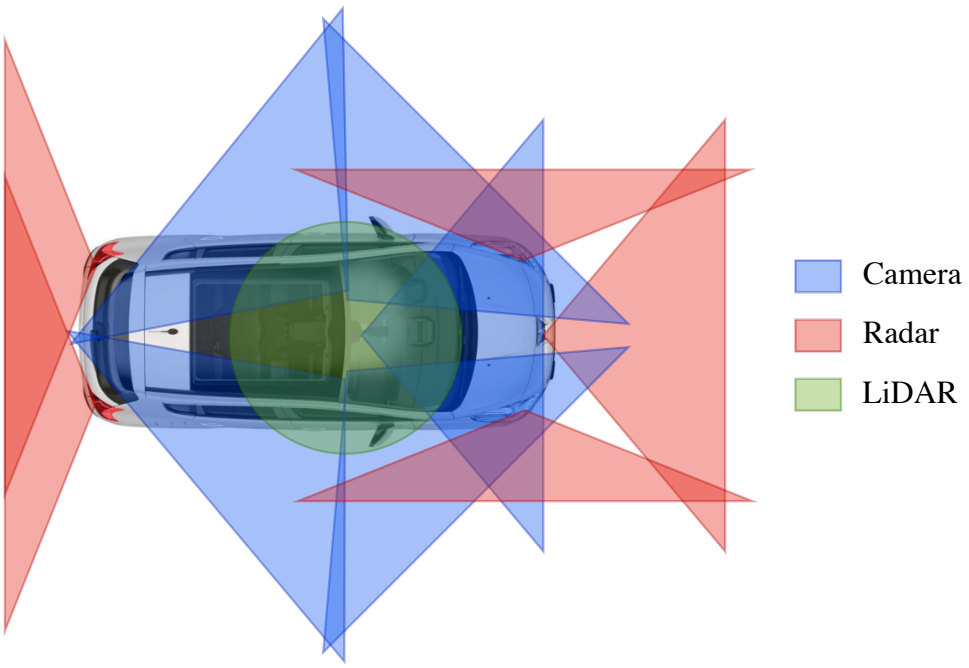


Figure 5.1.: *AI Sensing test vehicle sensor setup and field of view.*

The cameras are STURDeCAM20 Gigabit Multimedia Serial Link (GMSL) cameras from e-con Systems with a resolution of 1920×1080 and a horizontal

field-of-view of 98.5° and a vertical field-of-view of 53.5°. The cameras are mounted on the rooftop to avoid reflections of the windshield and positioned to cover 360° surround view. They use an electronic rolling shutter and stream uncompressed images in the 16 bit UYVY format at 30 Hz.

The radar belt covers 76 GHz FMCW radars from Aptiv, including four short range SRR5 radars mounted at the corners of the vehicle, as well as one mid range MRR5 radar mounted at the front center of the vehicle. The SRR5 radars have 2×4 MIMO antennas, a detection range of up to 93 m at an accuracy of ± 0.3 m, an azimuth angle coverage of ± 75° at an accuracy of ± 1° and do not measure the elevation angle. The MRR5 radar has 3×4 MIMO antennas, a detection range of up to 200 m at an accuracy of ± 0.25 m, an azimuth angle coverage of ± 50° at an accuracy of ± 0.5° and has minimal support for the elevation angle with an accuracy of ± 2°. The data from all radars is pre-processed in an Aptiv radar belt system and streams point cloud data at 20 Hz.

The LiDAR sensor is a Velodyne VLP-32c, which is mounted on the rooftop and rotates around the vertical axis for full 360° azimuth angle coverage at an resolution of 0.4°. It has 32 non-linearly distributed channels which cover a vertical field-of-view of 40° with resolution of 0.33° in the central area. The detection range is up to 200 m with an accuracy of ± 3 cm and it streams point cloud data at 20 Hz.

The GNSS sensor is a ublox EVK-M8N which streams a Global Positioning System (GPS) signal at a rate of 1 Hz. In addition, the vehicle's on-board sensors provide measurements of the current vehicle speed and yaw rate at 100 Hz via Controller Area Network (CAN). The computing platform is a NVIDIA Jetson AGX, which allows for up to 32 TOPS computing performance.

5.2. ROS environment and data preparation

In order to use the data from the AI Sensing test vehicle, several preparation steps are needed. First, the sensors need to be calibrated in order to process their data in a common coordinate system. Then, a real-time environment is needed to read, pre-process and record the sensor data.

5.2.1. Calibration

The sensors need to be calibrated to enable sensor fusion. First, we follow the common convention in automotive engineering when defining the vehicle coordinate system at the center of the rear axis, which is the turning point of the vehicle in the vehicle model. The x-, y- and z-axes point to the front, left and up, respectively. The vehicle coordinate system is useful whenever ego motion of the vehicle has to be subtracted from data points. In order to find the extrinsic calibration parameters of the sensors with respect to the vehicle coordinate system, the LiDAR is used as an intermediate step because its accurate 3D measurements enable a good calibration with the cameras and radars. In order to determine the extrinsic LiDAR to vehicle transformation,

the vehicle is placed on a flat plane and the ground plane is estimated in the LiDAR point cloud to derive the height, pitch and roll angle. The remaining parameters are measured manually.

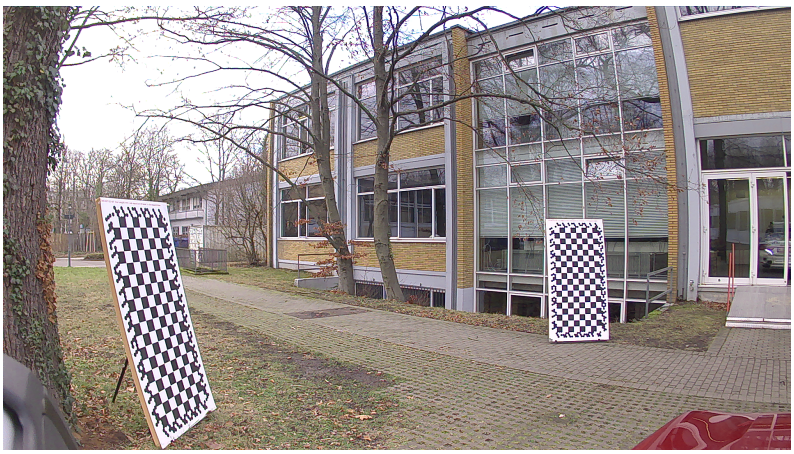


Figure 5.2.: *Camera calibration setup with rectangular checkerboards.*

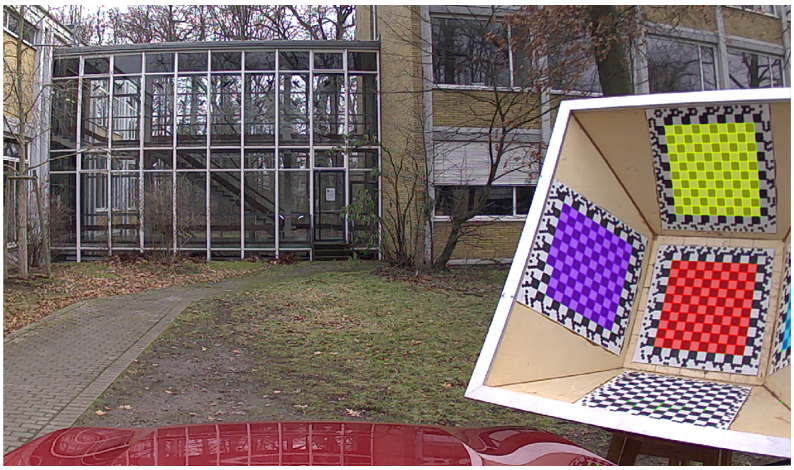


Figure 5.3.: *Camera calibration setup with truncated pyramid target. The calibration software detects the respective checkerboard patterns.*

Intrinsic and extrinsic camera calibration

The next step is the intrinsic and extrinsic calibration of the cameras according to the model and parameters described in Section 2.1.1, which is performed using the approach and tools provided by [138]. To this end, two different types of targets are used to calibrate the cameras. The first targets are four large boards with rectangular checkerboard patterns which are placed statically in the surroundings of the vehicle as shown in Figure 5.2. In this setup, the vehicle is driven around the boards to cover typical positions of traffic participants around the vehicle. A sequence of 52 images with each camera

is taken. The second target is a truncated pyramid shape with a total of five squared checkerboard patterns as shown in Figure 5.3. This target is moved around the static vehicle to cover all possible angles of the cameras. In this setup, another 248 measurements are taken, for each of which the target is visible in one or two cameras.

Figure 5.4 shows the results of the intrinsic camera calibration. For each camera, a circle with a radius of one pixel is used to visualize the re-projection errors. The vast majority of the re-projection errors is below half a pixel.

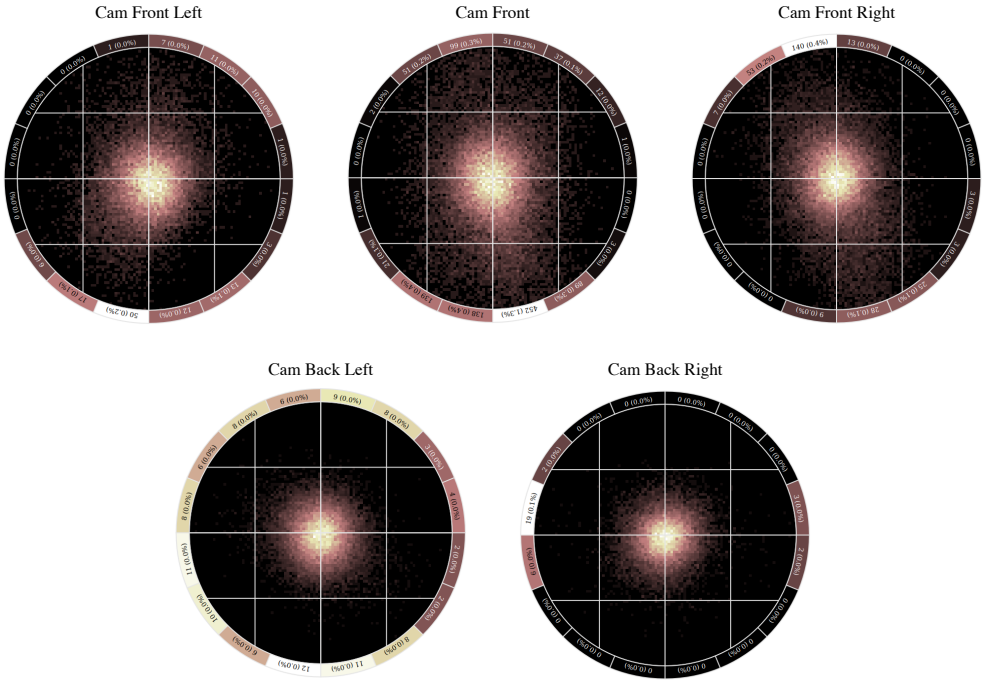


Figure 5.4.: *Intrinsic camera calibration results. The visualization shows the re-projection errors for each of the five cameras. The circle radius is one pixel.*

From the overlap in the camera's fields of view (FoV), the software is able to further calculate the extrinsic parameters between the cameras. The result is shown in Figure 5.5. The position and viewing angle of each of the cameras is visualized in 3D, along with an exemplary detection of the truncated pyramid target.

Extrinsic camera-to-LiDAR calibration

The extrinsic camera to LiDAR calibration is performed based on the work by [139]. The calibration target used is a styrofoam ball in front of a black flag. This target can easily be detected by both sensor modalities. No matter which part of the ball is observed in the LiDAR point cloud, the center can always be estimated from the curvature. The black flag behind the white ball leads to high contrast, which helps the camera to accurately detect the ball center via Hough transformation. The detections from LiDAR and camera

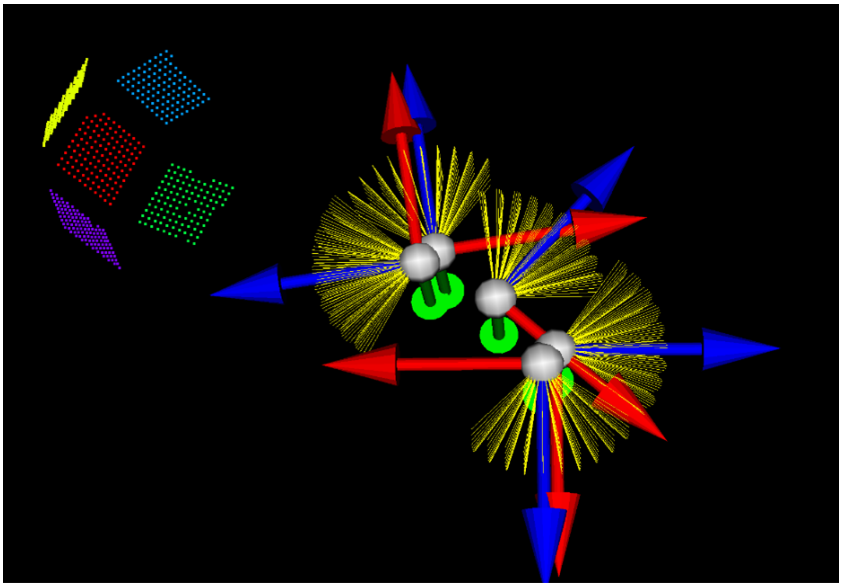


Figure 5.5.: *Visualization of the extrinsic camera calibration results. The calibration software estimates the 3D position and orientation of each camera and visualizes them including a detected truncated pyramid target.*

can then be used as correspondences for the calibration. Figure 5.6 shows the calibration target and the projected point cloud onto the front camera image using the estimated calibration parameters. The LiDAR points are colored according to their distance. Generally, there is a good correspondence between the sensor modalities, showing the quality of the calibration result. The small discrepancies can be explained by the parallax effect between the sensors. It is noted that only the front camera is directly calibrated to the LiDAR, because the remaining parameters for the other cameras can be inferred from the extrinsic camera to camera calibration, which is more precise.

Figure 5.6.: *Extrinsic camera-to-LiDAR calibration target and results.*

Extrinsic radar-to-LiDAR calibration

For the extrinsic radar to LiDAR calibration, another target with a styrofoam ball is created, which also has a radar corner reflector inside. For the calibration measurements, the vehicle is parked in a flat, empty area, and the target is moved in a large ellipsis around the vehicles. During this sequence, regular measurements of the LiDAR and radar point clouds are taken. The extrinsic radar to radar calibration was already performed by the supplier. In addition, an initial guess for the calibration between the radar system and the LiDAR coordinates is measured manually. Further, the initial position of the calibration target in the first LiDAR point cloud is estimated. Then, Algorithm 1 is applied to obtain the calibration parameters. Note that the radars do not measure the elevation, hence the aim of the algorithm is to estimate a 2D position on the ground plane as well as the yaw angle for each of the five radar sensors with respect to the LiDAR sensor. To obtain the remaining parameters, the height is measured manually, while the roll and pitch angles are set to zero.

Algorithm 1: Radar-to-LiDAR calibration

Data: Set of measurements with corresponding LiDAR and radar point clouds, initial calibration guess, calibration target radius and initial target position guess.

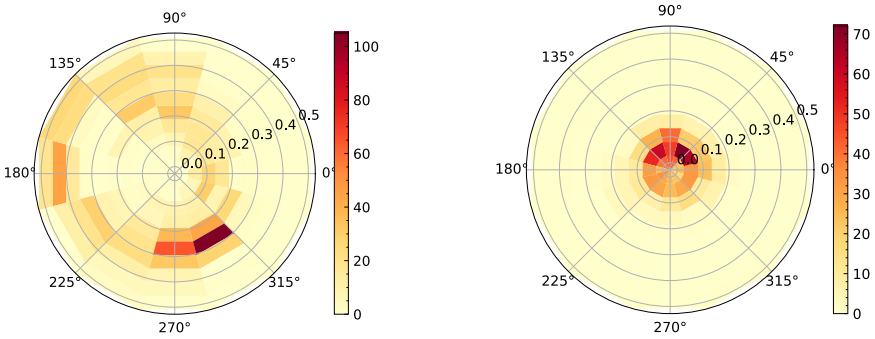
for *all measurements* **do**
- Crop the LiDAR point cloud near the calibration target position guess;
- Determine the center point of the calibration target by fitting a spherical model;
- Improve the center point iteratively by removing outliers;
- Transform the center point to the radar coordinate system using the initial calibration guess;
- Determine the position of the corresponding radar detection;
- Save the corresponding pair of LiDAR and radar positions of the calibration target;
- Use tracking to guess the position of the calibration target for the next iteration;

end

Create a non-linear equation system from the corresponding pairs;

Run an optimization algorithm to obtain the calibration parameters;

Figure 5.7 shows the results of the radar to LiDAR calibration as a polar histogram of the re-projection errors for the calibration target. The mean re-projection error was reduced from 35 cm for the initially measured calibration to 8 cm. Given the angular accuracy of the corner radar sensors of ± 1°, which is about 17 cm for a distance of 10 m, the remaining inaccuracy can be explained by the inherent technical limitations of the radar sensors.



(a) *Initial calibration guess based on radar supplier and manual measurements.* **(b)** *After applying the described radar-to-LiDAR calibration.*

Figure 5.7.: *Extrinsic radar-to-LiDAR calibration results. The visualization shows the re-projection error histogram for the calibration target. The circle radius is* 0.5 m.

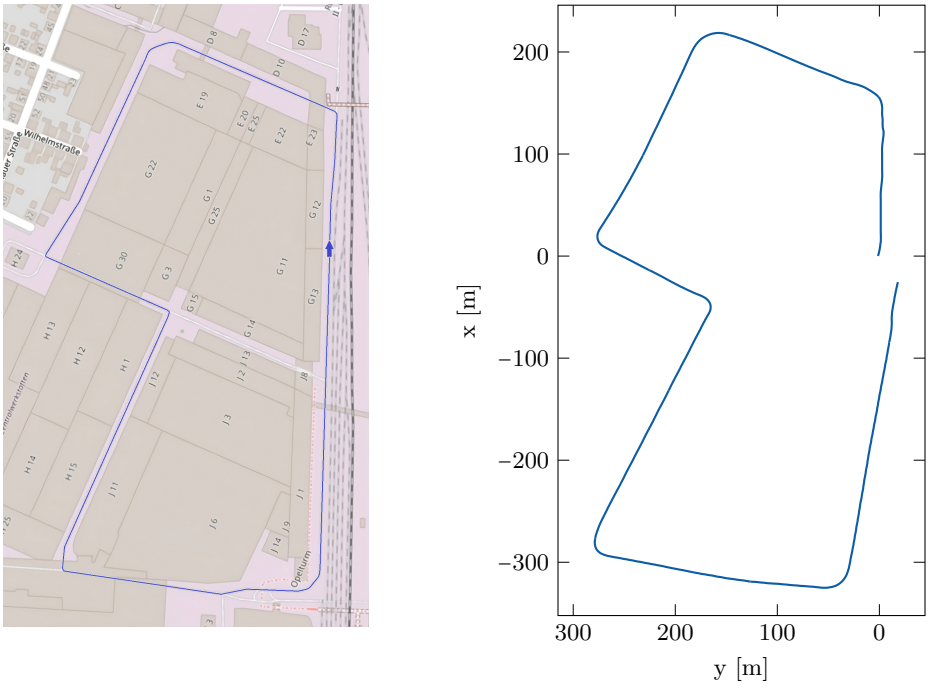
5.2.2. Real-time environment

In this chapter, the ROS enrivonment needed to process data in real-time is described. In order to maximize performance, the necessary ROS nodes are written in C++.

Vehicle CAN and ego vehicle tracking To process sensor data recorded at different timestamps in a joint coordinate frame, we have to know the pose of the ego vehicle at each timestamp. The coordinate transformation between vehicle poses at different timestamps is often referred to as ego motion compensation. A popular model for tracking the poses of the ego vehicle is the Constant-Turn-Rate-and-Velocity (CTRV) model, which estimates the pose based on a series of radial and angular velocity measurements [140]. To provide these measurements, we need a ROS node that processes Vehicle CAN messages. The structure of the CAN messages is described in DBC files, thus a header file is created accordingly to allow the node to interpret the messages and read the radial and angular velocity values. The node then performs the CTRV tracking and streams the tracked position and heading angle along with the measured radial and angular velocity in a custom message.

To verify the estimations, a 1.5 km roundway is driven, after which an error of about 20 m accumulates, as shown in Figure 5.8. Since this data is mainly used to subtract movement within fractions of a second, the achieved accuracy meets the requirements.

LiDAR streaming The LiDAR manufacturer provides a ROS node, which reads the raw data packets and streams them into ROS as the commonly used sensor_msgs/PointCloud2 messages, which is the standard datatype for point cloud data. The ego-vehicle tracking described above can be used to eliminate another problem occurring in dynamic scenes. During the recording of a LiDAR point cloud, the sensor spins 360° within 50 ms and finally assigns

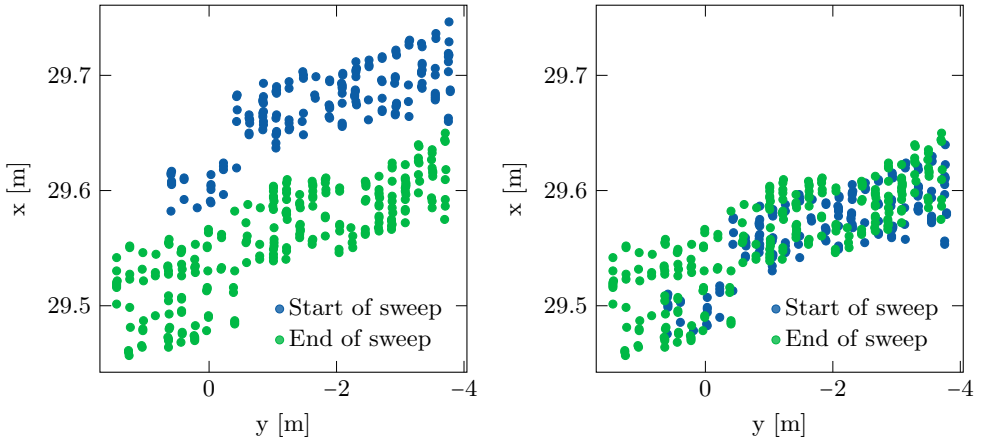


(a) *Ground truth path on map* **(b)** *Tracked ego-vehicle path*

Figure 5.8.: *Ego-vehicle tracking results.*

the point cloud the timestamp when the last point was recorded. When the ego vehicle is in motion, this leads to a misalignment of the LiDAR points within a point cloud. The effect can be visualized when driving towards a wall and visualizing the detections on the wall in birds-eye-view as shown in Figure 5.9. The blue points were recorded at the beginning of the sweep, the green points at the end of the sweep. In the left plot in Figure 5.9a, the misalignment due to ego motion during the sweep can be observed. The ego motion can be compensated by calculating the time offset of each point in the point cloud and subtracting the movement according to the ego vehicle tracking. The right plot in Figure 5.9b shows that the point cloud aligns well after applying motion compensation.

Camera streaming The camera manufacturer provides a ROS data streaming node, which streams Full HD images in UYVY format at a rate of 30 Hz in a proprietary image message type. This leads to a significant load on the network when streaming data from all five cameras. Thus, we change the node to reduce the stream rate to 10 Hz. Further, the images are directly rectified, which reduces their size by roughly another 20 %. Finally, the images are originally in UYVY format and a proprietary image message type. To facilitate subsequent processing, they are converted to RGB format and the standard sensor_msgs/Image datatype.



(a) *Original LiDAR point cloud* **(b)** *After motion compensation*

Figure 5.9.: *LiDAR motion compensation results. Detections in the original point cloud are misaligned due to ego motion during the LiDAR sweep.*

Radar streaming The radar manufacturer does not provide a ROS node. Instead, the raw data packets are sent via Ethernet in User Datagram Protocol (UDP). To incorporate them into ROS, a data parsing node is written, which reads data packets from the UDP Socket and converts the point cloud data into sensor_msgs/PointCloud2 ROS messages, which also ensures consistency with the LiDAR messages.

GNSS streaming The GNSS manufacturer provides a ROS node that fits all requirements. It streams messages of type NavSatFix, which is common for GPS positions.

Data recording tools Data messages in a ROS environment can easily be recorded in a ROSBAG file which can be used to re-create the scene as if it was recorded live. However, for deeper analysis of the data and to use it to train and evaluate neural networks, static frames need to be extracted. Therefore, a tool is written that extracts images as PNG files, point clouds as Point Cloud Data (PCD) files and ego vehicle tracking and GPS data into CSV files. The files are named after the timestamps when the data was recorded, so that the ego vehicle tracking data can be used to take the ego motion between two files into account.

5.2.3. Synchronization

To ensure precise fusion of data captured by various sensors, it is important that the measurements are well synchronized. Ideally, one sensor acts as the master, dictating timing through trigger signals to coordinate the other sensors operating in slave mode. Unfortunately, the hardware of the AI Sensing test vehicle does not allow for this ideal setup because the rolling shutter cameras can not be triggered.

Consequently, additional measures must be taken during post-processing to synchronize the data. To this end, we record data when driving next to static landmarks like traffic signs or trees. Using the calibration parameters obtained in Section 5.2.1, we project the LiDAR and radar point clouds to the images while accounting for vehicle ego motion between the timestamps of each measurement. By observing possible misalignment for the static landmarks, we can identify timestamp offsets between the sensors. Note that this approach assumes constant delays between the reported timestamps of each sensor, which holds well in our manual examination. Using these timestamp offsets, we can accurately subtract ego motion between measurements from different sensors, aligning all static objects. However, misalignment of dynamic objects like other traffic participants remains an issue. To address this problem for the training and evaluation of our proposed algorithms using data from the AI Sensing test vehicle, we select well synchronized keyframes as described in Section 7.1.

5.3. Conclusion

In this chapter, we have presented our AI Sensing test vehicle used to test the practicability and real-time applicability of our proposed algorithms. We have introduced the sensor setup including five radars, five cameras, a LiDAR sensor and a GNSS sensor. Further, we have outlined our data pre-processing steps including sensor calibration and have described our real-time data processing environment with ROS on an Edge AI platform. Finally, we have presented efforts to synchronize data recorded from various sensors.

Chapter 6

Automized Labeling

Contents

In AI and machine learning, the process of training deep neural networks strongly depends on the availability of accurately labeled data. However, the manual process of labeling is not only cumbersome but also cost-intensive, presenting a considerable bottleneck in the rapid development and deployment of neural network models. To address this challenge, this chapter introduces an automated labeling procedure.

Given the task of 3D object detection, LiDAR sensors typically yield the highest accuracy due their spatial resolution and precision. Therefore, the proposed automated labeling pipeline is based on LiDAR data. The generated pseudo-labels can then be used to train and evaluate models using data from cameras and radars, aiming at bridging the accuracy gap inherent in these sensing modalities. While the proposed automated labeling procedure may not fully replace manual labeling, it can improve generalization by leveraging powerful ground truth sensors and extensive offline processing.

In the following, the automized labeling procedure is described, starting with the utilization of a LiDAR-based object detector for inference on the dataset to be labeled. Note that a prerequisite for the labeling procedure is the availability of a public dataset with similar data as the one to be labeled, so that the object detector can be trained. The object detection is followed by the application of a Kalman filter tracking algorithm and additional post-processing steps, ensuring the refinement of the pseudo-labels. The effectiveness of the

proposed labeling technique is evaluated by re-creating the validation labels of the nuScenes dataset [44]. We note that the generalization capability of the labeling approach to data from a different domain is investigated at a later stage stage of this thesis in Chapter 7.1.

In summary, the contributions of this chapter are:

- A novel automized labeling technique based on object detection and tracking using LiDAR data
- Post-processing steps which refine the tracks to create pseudo-labels
- Quantitative and qualitative evaluation of the labeling pipeline based on the nuScenes dataset [44]

6.1. LiDAR-based object detection

In the recent years, LiDAR-based 3D object detection has been a widely studied field due to the high accuracy that can be achieved. LIDARs typically provide a 3D point cloud with x, y and z coordinates as well as the measured intensity of the reflection. Several algorithms are dedicated to processing this 3D point cloud data. Early algorithms voxelize the point cloud into a grid and apply 3D convolutions [141], [142]. However, these 3D convolutional operations are computationally intensive and use significant memory. In response to this, a trend has emerged in research that compresses the voxel grids into BEV [84] or range view (RV) [143]. Each of these views has its unique advantages and challenges. In BEV, objects are displayed from a top-down perspective. This ensures that there are no overlaps between objects and maintains a consistent object size regardless of their distance from the viewpoint. Closer to the native representation of LiDAR, RV produces compact and dense features. However, it also comes with the limitation of potential loss of spatial information. A recurring theme in current research is the idea of multi-view fusion [144], [145]. The rationale is that by integrating multiple perspectives, it becomes possible to achieve a more accurate and robust 3D object detection.

In this work, the state-of-the-art method VISTA (Dual CrossVIew SpaTial Attention) [146] shown in Figure 6.1 is used. VISTA uses attention mechanisms to enable multi-view fusion of BEV and RV features. First, a 3D sparse convolutional backbone is applied to the voxelized input point cloud. This results in a feature map of shape (b, c, h, w, d) with batch size b, c channels, as well as the height h, width w and depth d of the voxel grid. The feature map is then collapsed into a BEV of shape $(b, c \times h, w, d)$ and a RV of shape $(b, c \times d, h, w)$, respectively. The BEV and RV features are further processed using separate 2D necks similar to UNet [147]. These features are then unrolled to a sequence of feature vectors and used as queries and keys for the attention module. In contrast to normal attention modules utilizing a linear layer to map the feature vector sequence to queries and keys, VISTA uses 3×3 convolutional kernels to exploit the local context in both views. The subsequent cross-view attention mechanism allows VISTA to capture global context across

BEV and RV. The cross-view attention features are then processed by a feed forward network to obtain the final outputs. VISTA further decouples the attention for semantic and geometric attention modeling by essentially doubling the attention module and propagating only the loss for object classification and regression, respectively.

In this work, VISTA [146] is trained on the train split of the nuScenes dataset [44], before performing inference on the validation split. The point cloud is voxelized within [−51.2 m, 51.2 m] for x and y, as well as [−5.0 m, 3.0 m] for the z dimension with a resolution of 0.1 m. For training augmentation, the point clouds are randomly flipped along the x- and y-axes, rotated around the z-axis between [−0.3925 rad, 0.3925 rad], scaled with a factor between [0.95, 1.05] and translated within 0.2 m in each dimension. CBGS [137] is used to balance the classes during training.

Since the automized labeling can be performed offline, it is not time-critical. Therefore, several adjustments are made that allow for higher detection accuracy but lead to more computational cost. First, ten LiDAR sweeps are aggregated into a single point cloud, increasing the density and thus making it easier to detect and classify objects, especially if they are partially occluded. The ego movement is subtracted when aggregating the sweeps, accounting for the movement of static objects across sweeps. Since the movement of dynamic objects is unknown a priori, it cannot be subtracted. Instead, the timestamp difference with respect to the most recent sweep is appended to each point in the aggregated point cloud, so that the VISTA algorithm can learn to adjust its prediction according to the movement of dynamic objects.

Further, double-flip test time augmentation is applied to increase the detection accuracy during inference. To this end, the input point cloud is subjected to a series of spatial transformations during model inference: original orientation, flipped x-axis, flipped y-axis, and a combined flip of x-axis and y-axis. After obtaining predictions for each transformed version, the augmentations are reversed, and the resulting predictions are averaged to produce the final output. This method seeks to enhance model robustness and accuracy by leveraging multiple perspectives of the same input during testing.

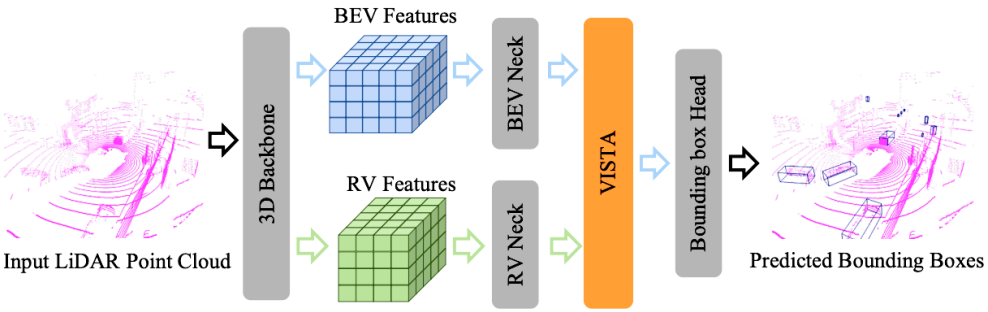


Figure 6.1.: *VISTA network architecture [146], © 2022, IEEE.*

6.2. Post-processing

The LiDAR-based detection outputs obtained by the object detection algorithm build a good starting point, but they are still prone to several types of errors. First, there are false positive detections, when the algorithm misclassifies an object or detects an object that is not actually there. Then, there are false negative detections, when the object detection algorithm fails to detect an object. Finally, even the true positive detections are prone to errors in terms of their precise localization, size, orientation and velocity. In the following chapters, a post-processing technique is proposed that uses a tracking algorithm and subsequent heuristics to mitigate these errors and enable better quality pseudo-labels.

6.2.1. Tracking

The first step of the post-processing pipeline is to apply a tracking algorithm on the LiDAR-based 3D object detections. The goal of the tracking algorithm is to assign each detection to a track, so that the tracks can later be used to refine the pseudo-labels. In this work, the tracking problem is simplified by modeling each object with its center point and applying a linear Kalman filter [49] as described in Section 2.3.2 to track the center points in the bird's eye view. The objects movement is modeled with a constant acceleration model. Consequently, the state space is:

$$\boldsymbol{x} = \begin{bmatrix} x \\ \dot{x} \\ \ddot{x} \\ y \\ \dot{y} \\ \ddot{y} \end{bmatrix} \tag{6.1}$$

The corresponding state transition matrix is:

$$\boldsymbol{F} = \begin{bmatrix} 1 & \Delta t & 0.5\Delta t^2 & 0 & 0 & 0 \\ 0 & 1 & \Delta t & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & \Delta t & 0.5\Delta t^2 \\ 0 & 0 & 0 & 0 & 1 & \Delta t \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{6.2}$$

This decouples the state covariance for x and y, yielding the state covariance:

$$\boldsymbol{P} = \begin{bmatrix} p_x & p_{x\dot{x}} & p_{x\ddot{x}} & 0 & 0 & 0 \\ p_{\dot{x}x} & p_{\dot{x}} & p_{\dot{x}\ddot{x}} & 0 & 0 & 0 \\ p_{\ddot{x}x} & p_{\ddot{x}\dot{x}} & p_{\ddot{x}} & 0 & 0 & 0 \\ 0 & 0 & 0 & p_y & p_{y\dot{y}} & p_{y\ddot{y}} \\ 0 & 0 & 0 & p_{\dot{y}y} & p_{\dot{y}} & p_{\dot{y}\ddot{y}} \\ 0 & 0 & 0 & p_{\ddot{y}y} & p_{\ddot{y}\dot{y}} & p_{\ddot{y}} \end{bmatrix} \tag{6.3}$$

The control input of the tracked objects is unknown, therefore it is set to zero:

$$\boldsymbol{u}_t = \mathbf{0} \tag{6.4}$$

Since the model assumes a constant acceleration, the process noise is modeled as a random noise in acceleration σ_a to cover potential changes. It propagates to the state space as follows:

$$\boldsymbol{Q} = \begin{bmatrix} \frac{\Delta t^4}{4} & \frac{\Delta t^3}{2} & \frac{\Delta t^2}{2} & 0 & 0 & 0 \\ \frac{\Delta t^3}{2} & \Delta t^2 & \Delta t & 0 & 0 & 0 \\ \frac{\Delta t^2}{2} & \Delta t & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{\Delta t^4}{4} & \frac{\Delta t^3}{2} & \frac{\Delta t^2}{2} \\ 0 & 0 & 0 & \frac{\Delta t^3}{2} & \Delta t^2 & \Delta t \\ 0 & 0 & 0 & \frac{\Delta t^2}{2} & \Delta t & 1 \end{bmatrix} {\sigma_a}^2 \tag{6.5}$$

The object detection algorithm estimates an object's position and velocity, but not its acceleration. Therefore, the measurement space is:

$$\boldsymbol{z} = \begin{bmatrix} x \\ \dot{x} \\ y \\ \dot{y} \end{bmatrix} \tag{6.6}$$

The corresponding observation matrix is:

$$\boldsymbol{H} = \begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 \end{bmatrix} \tag{6.7}$$

The measurement variables are assumed independent with measurement noises σ_i, respectively. Consequently, the measurement uncertainty is:

$$\boldsymbol{V}_t = \begin{bmatrix} {\sigma_x}^2 & 0 & 0 & 0 \\ 0 & {\sigma_{\dot{x}}}^2 & 0 & 0 \\ 0 & 0 & {\sigma_y}^2 & 0 \\ 0 & 0 & 0 & {\sigma_{\dot{y}}}^2 \end{bmatrix} \tag{6.8}$$

In the next step, a logic is needed for the association of new detections to existing tracks. Since the object detection algorithm outputs many detections with low confidence scores, the first step is to remove detections below a confidence threshold. Then, the Mahalanobis distance introduced in 2.3.3 is computed for each combination of existing track and new detection. If the Mahalanobis distance is below a distance threshold, a detection is eligible to be assigned to the respective track. However, it may be the case that a detection could fit to several nearby tracks or several detections could match the same track. To account for this problem, the eligible combinations of tracks and detections are sorted by Mahalanobis distance and assigned in a greedy manner starting with the lowest distance. Detections that can not be assigned to an existing track are used to initialize new tracks. If a track cannot be confirmed for several iterations, it is closed and can no longer be matched with new detections.

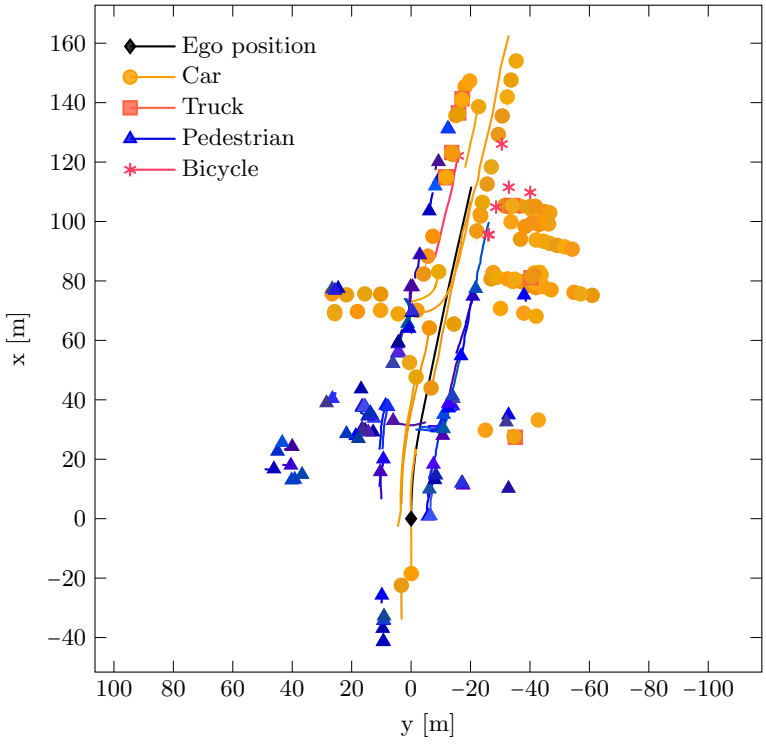


Figure 6.2.: *Tracking result for scene-0103 of the nuScenes validation split.*

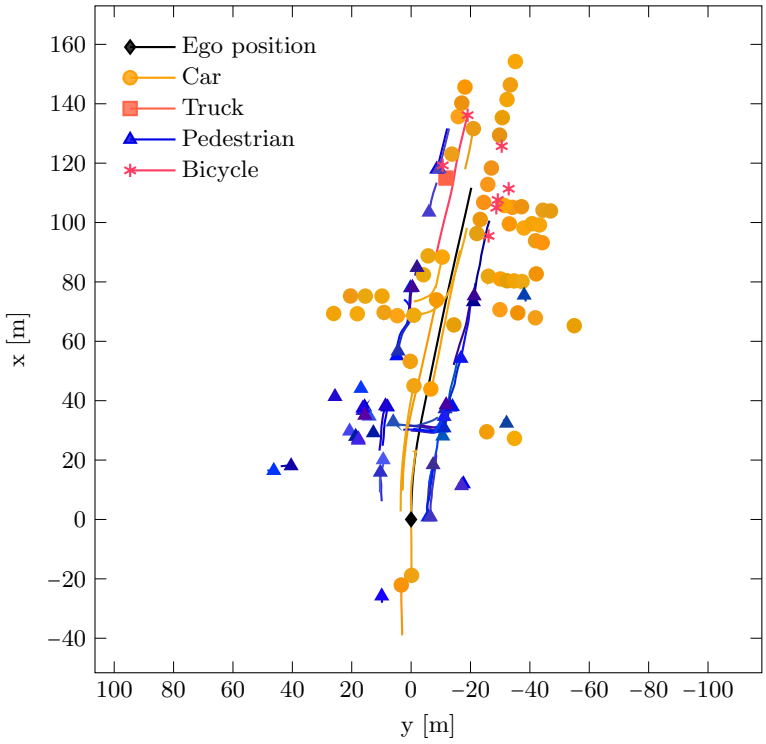


Figure 6.3.: *Ground truth tracks of scene-0103 of the nuScenes validation split.*

To verify that the tracking works properly, scene-0103 from the nuScenes validation split is used. In Figure 6.2, the result of the object detection algorithm and subsequent tracking is shown. The figure shows the positions of each detected and tracked object across its lifetime in the scene. A global coordinate system is used, where the origin is the origin of the ego vehicle in the first frame of the scene. Each object's starting position is marked with a circle, and during the scene it moves along the line in the same color. The tracking result can be compared with the ground truth shown in Figure 6.3. It can be observed that there are many similar tracks that have correctly been identified by the tracking algorithm. Further, the tracks found by the tracker are more noisy and there are more tracks than labeled in the ground-truth, which may be due to false positives. These problems will be addressed in the following.

6.2.2. Track interpolation

Following the application of the tracking algorithm, detections attributed to each track are post-processed to eliminate errors and ensure track consistency. A selected track from a *Car* in the scene depicted in Figure 6.2 is used to visualize the process. In this section, we describe the track interpolation for individual parameters, including centroid, dimension and rotation of objects, and present the used principles for track fusion, the insertion of false negatives, and the removal of false positives.

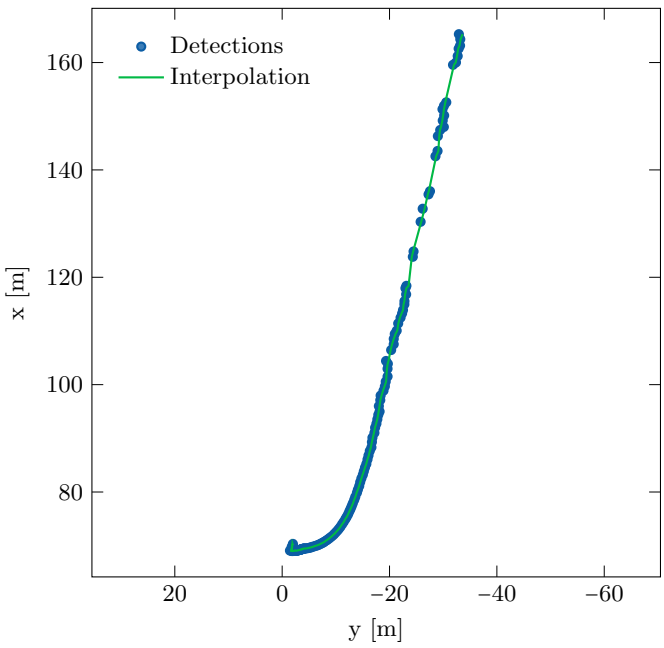


Figure 6.4.: *Exemplary interpolation of an object's centroid estimates.*

Centroid

The object's centroid is initially addressed, necessitating noise filtering from the predicted 3D bounding box detections. While the tracked state of the Kalman filter could be used to get a smoother representation of the object's centroid, we argue that this is not ideal since it only has access to past estimates at each iteration. Instead, it is preferable to interpolate the centroid utilizing both past and future detections. To this end, centroid estimations within a local window around each iteration are used to fit a second-degree polynomial, from which the interpolated centroid is derived. This leads to a robust and smooth track. In Figure 6.4, the centroid interpolation of the selected *Car* track is shown. The tracked *Car* initially waits at an intersection in the left of the figure. It then takes a left turn and continues to drive further away from the ego vehicle. When the *Car* is far away, the detections become more noisy and are sometimes missing.

Dimension

Subsequently, the dimensions of the object are considered. Given the assumption that an object's dimensions remain unchanged throughout its lifetime, the median of all dimension estimations is used as a reliable metric for the object's extent. In Figure 6.5, the dimension estimation of the selected *Car* are shown over time. Using the median value for each dimension, a robust and consistent estimate is found.

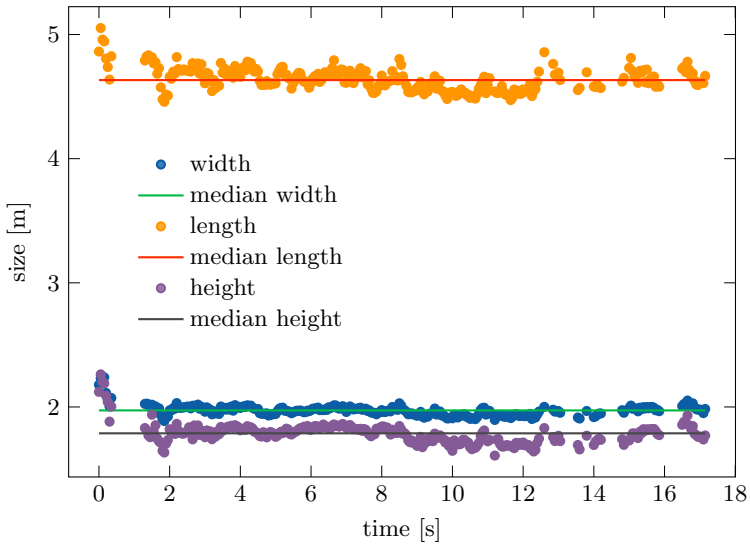


Figure 6.5.: *Exemplary interpolation of an object's dimension estimates.*

Rotation

Next, the object's rotation around the vertical axis is addressed. This is a more challenging problem due to various prevalent issues: Firstly, LiDAR-

based object detectors can struggle to correctly identify the driving direction. Hence, there are frequent errors of 180° or sometimes ±90° relative to the true yaw angle, which result in significant deviations between successive detections. Secondly, the typical smaller-scale noise has to be addressed. Lastly, the interpolation has to take the cyclic behavior of the rotation angle into account. To tackle these issues, Algorithm 2 is proposed. Initially, four hypotheses are formulated which shift the rotation angle estimates by increments of 0°, 90°, 180°, and 270°. Subsequent angle estimations are then iteratively matched to the most appropriate hypothesis. Upon completion, the hypothesis with the highest number of matched detections is selected, which is now filtered from deviations of 90° or 180°. To counteract the smaller-scale noise, the median angle within a local window around each iteration is used. Note that the angles have to be mapped to sine and cosine to account for the cyclic behavior. The proposed algorithm produces a stable and smooth curve of the rotation angle along each track.

Algorithm 2: Rotation post-processing

Data: Series of rotation estimates.
Set shifts = [0°, 90°, 180°, and 270°];
Initialize array of aligned rotation estimates;
Initialize array of hypothesis votes;
for *all rotation estimates* **do**
    Calculate median of previous iterations in local window;
    Find shift that aligns new rotation estimate with local median;
    Append shifted rotation to aligned rotation estimates;
    Append index of shift to hypothesis votes;
end
Find the shift of the most voted hypothesis;
Adjust aligned rotation estimates by most voted shift;
Smooth the aligned rotation estimates using a sliding window median of their sine and cosine values;

In Figure 6.6, the rotation angle estimates of the *Car* are shown over time. Several characteristic jumps of about 90° or 180° can be seen, which are eliminated using the proposed algorithm.

Velocity

Finally, the object's velocity is estimated. Given the inherently noisy nature of the velocity estimates provided by the object detection model and the previously computed smooth interpolated centroid for each track, we find that the velocity is most accurately derived from the change in centroid position.

Track fusion

In general, the object detection algorithm should only output one detection per object since nearby detections with lower confidence are filtered by non-

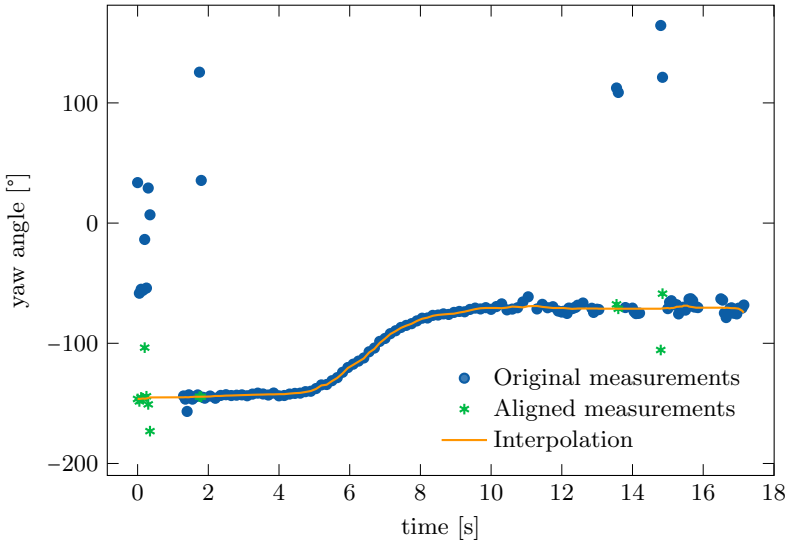


Figure 6.6.: *Exemplary interpolation of an object's rotation estimates.*

maximum suppression. However, it sometimes happens that there are two detections just outside each other's non-maximum suppression radius, that still belong to the same object. In this case, only one detection is assigned to the existing track while a new track is started for the other. This can lead to two tracks belonging to the same object. To address this problem, Algorithm 3 is suggested to fuse tracks in post-processing. To this end, the distance between each pair of tracks is calculated for every overlapping timestamp to find potential matches. The necessary condition for fusing the tracks is that the minimal distance is below a distance threshold. However, this might simply be the case if two object's tracks cross close to each other. Therefore, the simultaneous detections assigned to both tracks are extracted to compute their IoUs. If at least half of the simultaneous detections exceed a threshold, the sufficient condition for track fusion is met. To fuse the tracks, the detections from both tracks are assigned to a single track. In the case of simultaneous detections, the detection with the higher confidence score is used.

False negatives

A false negative detection occurs when an object is missed by the detection algorithm at one iteration. Using the information of the tracked objects, this problem can be mitigated. If an existing track is temporally not confirmed but later picked up again, the missing detections can be interpolated and inserted. Bilinear interpolation between the closest past and future detections is used to fill these gaps. It should be noted that the false negatives may have been due to temporary occlusions, in which case they should not be used as labels to train machine learning algorithms. To mitigate this issue, we only include interpolated bounding boxes if the number of contained LiDAR points is sufficiently large.

Algorithm 3: Track fusion in post-processing

```
Data: Set of tracks, each with series of detections.
for each pair of tracks do
    if minimum distance between tracks > distance threshold then
        Continue with next pair;
    end
    Find all detection pairs with simultaneous timestamps;
    for each pair of simultaneous detections do
        Calculate and store IoU of detections;
    end
    if at least half of calculated IoUs > IoU threshold then
        Fuse detections of both tracks to form single track;
    end
end
```

False positives

A false positive detection occurs when the detection algorithm outputs a detection where there is no actual object of the respective class. This problem can again be mitigated using the information from the tracks. If a track does not contain sufficiently many detections, we interpret it as false positive and discard it.

6.3. Results

In the following, the proposed labeling is evaluated quantitatively and qualitatively using the validation split of the nuScenes dataset [44]. We experimentally adapted the hyperparameters of the labeling algorithm, leading to the following setup: For the Kalman Filter, we set the accelerations noise to $\sigma_a = 0.5\,\mathrm{m/s^2}$, and the measurement uncertainties to $\sigma_x = \sigma_y = 0.2\,\mathrm{m}, \sigma_{\dot{x}} = \sigma_{\dot{y}} = 4\,\mathrm{m/s}$. Note that the velocity measurement uncertainty is high due to the noisy velocity estimates of the object detection network. We further choose a distance threshold for the association via the Mahalanobis distance of two, corresponding to two standard deviations. We deactivate tracks when they are unconfirmed for 1 s. For the post-processing, we choose an interpolation window of ± 500 ms to smoothen the centroids and the smaller scale noise of the rotation. To fuse tracks in post-processing, we use an IoU threshold of 0.2 for at least half of the overlapping detections. To mitigate false negatives while considering temporary occlusion, we insert detections into tracks when the bounding box contains at least two LiDAR points. We remove tracks as false positives when they contain less than three detections.

In Table 6.1, we separately evaluate the contribution of each of our proposed post-processing steps. In the first row, we list the results for the raw VISTA object detections when compared with the manual labels of nuScenes. These results are already quite good, showing the efficacy of VISTA when applying

the settings with 10 aggregated LiDAR sweeps and double-flip test-time augmentation. In the following rows, we use the extracted tracks to perform our suggested post-processing steps. When smoothing the object centroid position along each track, we can slightly increase the mAP. This indicates that we are able to correct some centroid outliers to an extent that they are additionally counted as true positives. This comes with a small cost of higher mATE. This is likely due to the newly inserted detections having higher average translation error than the ones that could directly be detected without post-processing. Next, averaging the objects dimension along each track leads to a slight reduction of the mASE. More significantly, the proposed algorithm to correct the rotation estimates leads to a 19 % reduction of the mAOE, showcasing the effectiveness of the approach. Similarly, calculating the velocity from the change of position leads to a significant reduction of the mAVE by 24 %. As a side-effect, this also helps to determine when an object is moving and thus better attribute detection.

Possibly the most effective post-processing step is the reduction of false negatives by inserting missing detections along each track. It leads to a significant increase of 6 % mAP. Again, the increasing mAP comes with a cost of small increments in the error metrics, indicating that the new true positives have higher average errors. This is because they are likely hard detections which may be heavily occluded or far away. Finally, the small increments in the true positive errors can be reverted again by eliminating false negatives by deleting detections without confirmed tracks. Unfortunately, this has to be applied conservatively, because deleting too many detections is punished heavily in the mAP. Overall, each post-processing step contributes towards better quality pseudo-labels. However, some of the contributions seem small in this quantitative evaluation. To get a more in-depth understanding of the improvements, we have therefore inspected a large number of frames and visually compared the resulting pseudo-labels with the ground truth. We believe that the presented post-processing steps result in an important qualitative improvement, which we present using a representative frame in the following qualitative evaluation.

Table 6.1.: *Evaluation of the post-processing steps for the automated labeling pipeline.*

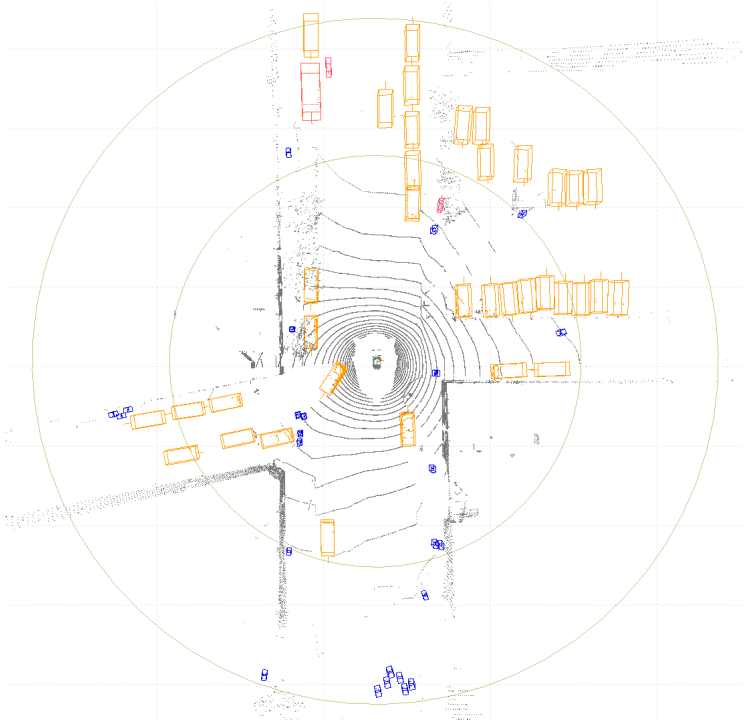
Smooth				Reduce		mAP	mATE	mASE	mAOE	mAVE	mAAE	NDS
pos	dim	rot	vel	FN	FP	↑	↓	↓	↓	↓	↓	↑
						0.643	0.199	0.205	0.174	0.203	0.187	0.726
✓						+1%	+2%					
	✓							-4%				
		✓							-19%			
			✓							-24%	-3%	+1%
				✓		+6%	+2%	+1%	+5%	+4%	+3%	+3%
					✓	0%	-2%	-1%	-6%	-4%	-2%	0%
✓	✓	✓	✓	✓	✓	0.686	0.202	0.197	0.139	0.155	0.183	0.755
						+7%	+2%	-4%	-20%	-24%	-2%	+4%

For the qualitative evaluation, a selected frame from the nuScenes validation split is shown in Figure 6.7a. The figure shows a bird's eye view representation of the LiDAR point cloud along with the created 3D bounding boxes. As a comparison, the corresponding ground truth is shown in Figure 6.7b. In the inner circle, which corresponds to 30 m distance from the ego vehicle, the generated labels are almost identical to the ground truth. However, in the outer circle, up to 50 m distance from the ego vehicle, there are some differences. Although it is difficult to see based on this image alone, we evaluate the differences looking at past and future frames. For example, the automated pseudo-labels include two false positive *Pedestrians* in the lower part of the image, and one false positive *Pedestrian* on the top right part. They further contain a false negative *Pedestrian* in the far front and some false negative *Cars* far to the sides. However, the manual labels from nuScenes [44] also contain errors. For instance, there are multiple parked *Cars* on the right side which are missing, as well as three *Pedestrians* on the left side and one *Pedestrian* in the back.

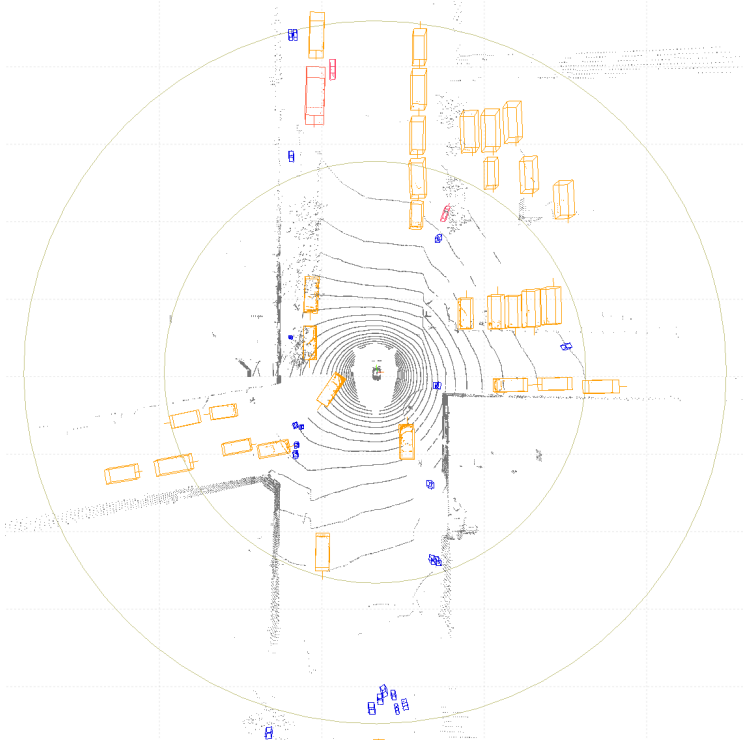
To summarize, the qualitative evaluation shows that the automated pseudo-labels are visually of similar quality as the manual labels. They can include failure cases of inaccurate bounding boxes, false negatives and false positives. However, this is also true for manually labeled data. Therefore, the labeling can not be purely evaluated based on quantitative comparison between automated labels and ground-truth. In conclusion, we argue that the presented approach can be used as quick and cheap way to generate large amounts of pseudo-labeled data, which can in turn be used to train machine learning methods with less computational complexity or based on cheaper, automotive series-level sensors. We note that this does not hold true for edge cases, which still require careful human labeling.

6.4. Conclusion

In this chapter, we have presented an automized labeling technique based on LiDAR data. We use the nuScenes dataset [44] to validate the performance of our approach by re-creating the labels of the validation split. Starting with a LiDAR-based object detector, VISTA [146], trained on the training split of nuScenes, we run inference on the validation data. We then track the bounding box detections over time using a Kalman filter [49]. Finally, we apply post-processing steps along each track to get pseudo-labels which are smooth in terms of the centroids, size, orientation and velocity values. Using quantitative and qualitative evaluation, we prove the effectiveness of each step of our proposed labeling technique. We conclude that the proposed technique can be used as a quick and cheap method to generate a large annotated dataset, with labels that are visually close to manual label quality. However, we advise that edge cases should still be labeled manually.



(a) *Automated labeling result.*



(b) *Ground truth labels.*

Figure 6.7.: *Qualitative comparison of automated labeling with ground truth for exemplary nuScenes frame. Yellow = Car, Blue = Pedestrian, Orange = Truck, Red = Two-wheeler.*

Chapter 7

Model deployment

Contents

In this chapter, the work introduced in the previous chapters is combined to showcase the deployment of our proposed algorithm on our demonstrator vehicle. To this end, we record a dataset using the AI Sensing test vehicle presented in Chapter 5 and apply the labeling procedure presented in Chapter 6. We then conduct a case study on Deep Neural Network (DNN) deployment using RetinaNet [39] as a well known and representative algorithm to highlight important tools, concepts and considerations that need to be made. Finally, we apply these concepts to the deployment of our proposed RC-BEVFusion presented in Chapter 4 and use the created data to fine-tune the algorithm for domain adaptation.

Some contents of Section 7.2 have been published in [15]. In summary, this chapter contains:

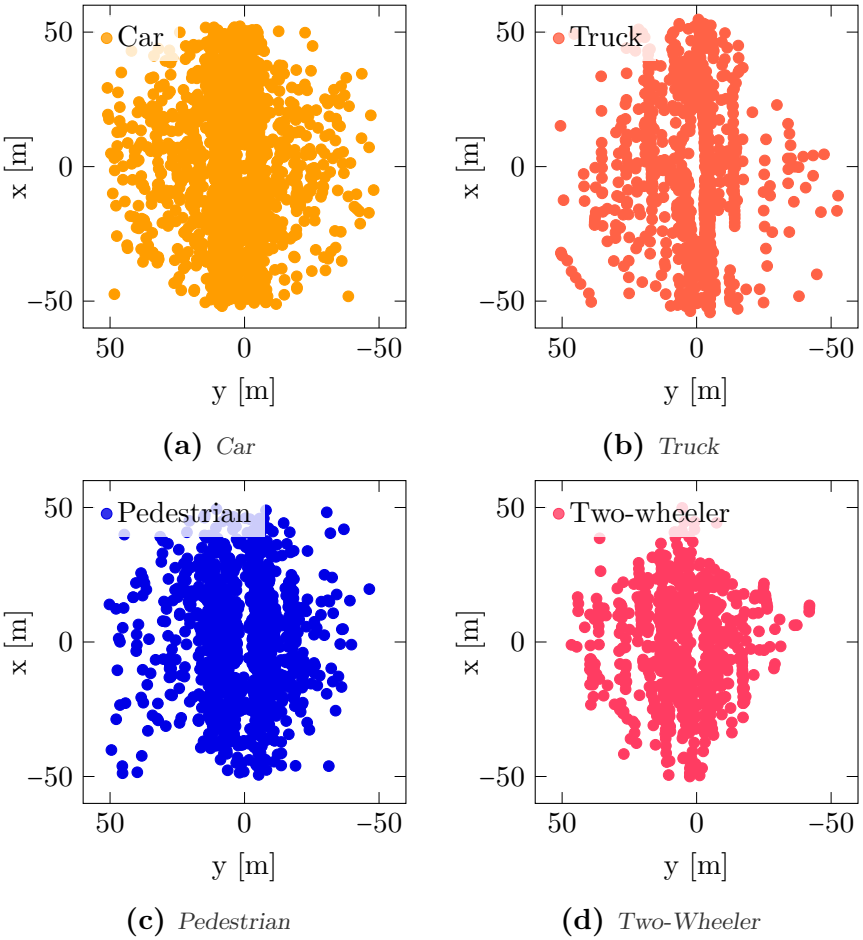
- The proprietary dataset creation and automized labeling.
- A case study on DNN deployment on an edge AI device, highlighting useful tools and technical details.
- Experimental results on the trade-off between detection performance and computational speed when deploying deep neural networks.

- Domain adaptation of the proposed RC-BEVFusion for the proprietary data.
- Deployment of the proposed RC-BEVFusion algorithm on an edge AI device.
- Quantitative and qualitative evaluation of the proposed RC-BEVFusion w.r.t. its camera-only baseline on the proprietary dataset.

7.1. Dataset creation

To be able to perform domain adaptation for our proposed RC-BEVFusion model on the AI Sensing test vehicle presented in Section 5.1, we first record a dataset. We capture several scenes in and around Rüsselsheim, Germany, at different daytime and weather conditions. Using the ROS environment described in Section 5.2, the images are recorded at 10 Hz, the radar and LiDAR point clouds at 20 Hz, the tracked ego vehicle position at 100 Hz and the GPS position at 1 Hz. Initially, the recorded data is stored in ROSBAG files that can be used to simulate the live recording. For more advanced data processing, we extract static files and store them in an organized structure. The images are stored in PNG files, the point clouds in PCD files and the ego position and GPS in CSV files, each file named after the timestamp of the respective ROS message. We record 45 scenes, each between 20 s and 40 s, totaling to 22 minutes driving time. To include many diverse objects, we record mainly on crowded streets and intersections. The dataset covers urban streets, highways, and the drop-off areas of a train station and an airport. We also cover different lighting and weather conditions, including sunny, overcast, rainy and twilight scenes.

Figure 7.1.: *Automated labeling result for exemplary AI Sensing test vehicle data frame.*



(a) *Car* **(b)** *Truck*

(c) *Pedestrian* **(d)** *Two-Wheeler*

Figure 7.2.: *Label distribution for each class on the proprietary dataset.*

Given the frequency of the recorded frames, subsequent frames are highly redundant. To increase the efficiency of the training and evaluation process, we therefore select a set of keyframes for each scene. To allow for sufficient deviations between keyframes, we follow [44] and choose one keyframe about every 500 ms. We further allow for deviations of ± 100 ms to find a timestamp where data from all sensors is well synchronized. For each keyframe, we create automated labels using the procedure proposed in Chapter 6 using the VISTA [146] model pretrained on nuScenes [44].

To verify that the proposed labeling approach also works with data from the AI Sensing test vehicle presented in Section 5.1, we have to rely on visual inspection, since there is no ground truth available. However, a qualitative example is presented in Figure 7.1. The frame is from a particularly challenging intersection with many *Cars* and *Pedestrians*. As can be observed, the approach generalizes well and produces quite accurate labels also for this data from a different domain. Even the distant and partially occluded objects like the *Cars* on the parking lot to the back left, are labeled well. Of course, in some frames there are still failure cases of inaccurate bounding boxes, false

negatives and false positives. The general quality of the labels is still deemed sufficient to train and evaluate models, especially if they are using data from radars and cameras, which typically provide lower 3D detection accuracy.

The automated labeling results in a total of 68000 3D bounding box labels across 2700 keyframes, with an average of 25 objects per frame, confirming that the recordings were mostly done in crowded scenarios. We label four different classes: *Car*, *Truck*, *Pedestrian* and *Two-Wheeler*. In this class mapping, the *Truck* class covers all types of larger vehicles like trucks, buses and construction vehicles, while the *Two-Wheeler* class covers bicycles and motorcycles. The amount of objects per class is naturally imbalanced with 41000 *Cars*, 24000 *Pedestrians* and about 1500 *Trucks* and *Two-Wheelers*, respectively. We show the bounding box distribution in the labeled range of ± 50 m in Figure 7.2 for each class. The area is generally well covered except that *Two-Wheelers* are only detected up to a euclidean distance of 50 m, leaving the corners of the detection area empty.

We split the data into training and validation splits with an 80:20 distribution. The validation split is chosen to be representative in terms of the different driving and lighting conditions as well as the class distribution as shown in Figure 7.3.

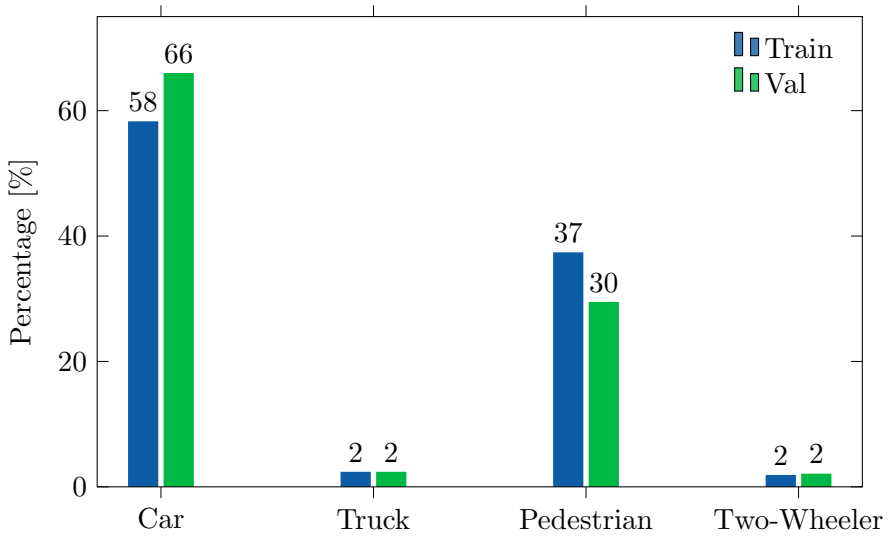


Figure 7.3.: *Class distribution in the train and val splits of the proprietary dataset.*

7.2. Case study on model deployment

Many object detection algorithms are well studied and their performance in development conditions is known in the literature. However, deployment aspects of these DNN models on edge AI devices, and embedded systems in general, are often not addressed in scientific work but only in blog posts in the form of partly very helpful or incomplete notes and hints. Therefore, this chapter intents to present the major findings when deploying a representa-

tive DNN model on a widely used edge AI device, the NVIDIA Jetson AGX Xavier [148]. We conduct this case study on the deployment of the widely used image-based 2D object detection model RetinaNet [39] using data from nuScenes [44]. Combined with the findings from a collaborative work [15] on the deployment of PointPillars [84], these concepts translate well to the deployment of RC-BEVFusion. When deploying DNN models, there are several techniques to optimize the runtime for deployment [149], including (1) the design and manipulation of the DNN model architecture in terms of the model size, depth and width, (2) pruning techniques to remove neurons, groups of neurons, or filters, which have little impact on the output, and (3) quantization to change the numerical representation of data and network weights [150]. These techniques generally result in a trade-off between runtime and performance. In this chapter, the techniques (1) and (3) are considered. First, experiments are performed with varying resolution of the input image for RetinaNet [39]. Then, the effect of quantization to half-precision floating-point format and to fixed-point arithmetic is studied.

7.2.1. Concepts and tools

As it is common in the DNN research community, the algorithms are developed, trained and evaluated in a Python environment, using the PyTorch library [151]. However, when deploying the network on an embedded system for real-time inference, it is beneficial to move to a C++ environment, in which other tools are needed. In this section, a short overview on useful concepts and tools is given.

Network conversion

When deploying a trained DNN on an embedded system in a C++ environment, a conversion becomes necessary. A standardized format to exchange networks within different tools, is the Open Neural Network Exchange (ONNX) [152] format. It can be created using network tracing, which converts the network into a static computational graph based on exemplary input data and allows for efficient hardware acceleration with the NVIDIA TensorRT [153] SDK for runtime optimized inference. However, tracing does not allow data-dependent control-flow operations, so that the original network architecture has to be divided into network graphs and logical operations. These logical operations do not contain network weights and have to be realized with C++ functions. PyTorch offers an alternative built-in framework TorchScript, with the options tracing and scripting. For reasons of comparability and a limited usability of scripting, the focus of this study is on tracing with TorchScript as well.

Quantization

Quantization has the goal to efficiently represent numerical values by a finite number of bits. Widely used formats include single- and half-precision

floating-point formats [154], referred to as Float32 and Float16, respectively, and fixed-point arithmetic with 8 bit integers, referred to as Int8. A reduction of the used quantization format can be beneficial in a deployment setup with limited computational, memory or energy resources. Moving from the default Float32 to Float16 is a straightforward and often considered option. When further moving to Int8 quantization, several tuning and calibration steps have to be taken into account. A recent survey of quantization techniques for DNN inference is given in [149]. In [150], a corresponding practical workflow for Int8 quantization is recommended.

For the sake of simplicity, in this work, the focus is on post-training quantization and do not consider quantization-aware training. In particular, for Int8 quantization, the TensorRT MinMax and entropy calibrator functions [155] are considered. The MinMax calibration measures the maximum absolute activation of each layer and provides an equidistant and symmetric mapping. Likewise, the entropy calibration determines a mapping which minimizes the information loss by saturating the activations above a certain threshold. Since, at the time of writing, TorchScript only supports Int8 quantization for CPU usage, only the corresponding TensorRT variant is considered.

Further tools

ROS [156] is used as the real-time environment in C++. ROS contains helpful tools for sensor data streaming, communication of different nodes and visualization of sensor data and detection results. With Nuscenes2bag [157], data from the public automotive dataset nuScenes [44] can be converted into a ROS compatible format that allows the simulation of a real driving scenario while having access to labeled ground truth. For the pre- and post-processing of the data in C++, efficient implementations from libraries like OpenCV [158] for images and OpenPCDet [159] for point clouds can be used. For many operations, there are also CUDA-based [160] implementations for hardware acceleration in these stages.

7.2.2. Deployment architecture

For this case study, we select the image-based RetinaNet [39] model as a representative and well-known algorithm, that provides a reasonable trade-off between runtime and performance, and is therefore suitable for automotive applications. In collaborative work [15], a similar study was performed on PointPillars [84] to cover deployment aspects for point cloud data. RetinaNet [39] is one of the pioneering one-stage object detectors, which surpassed the preceding two-stage networks in terms of runtime while offering similar detection performance. When deploying RetinaNet, the pipeline shown in Figure 7.4 is divided into the following processing steps:

Pre-processing In the pre-processing stage, the image is resized to a lower resolution of choice and normalized based on the mean and standard deviation of the RGB images in the ImageNet dataset [9], on which the backbone network

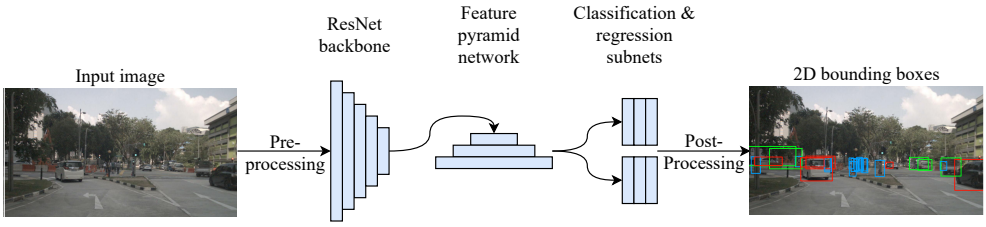


Figure 7.4.: *RetinaNet processing pipeline [15].*

is pretrained. Both of these steps can efficiently be done using the OpenCV CUDA library.

Inference The architecture of RetinaNet [39] is made out of a ResNet [26] backbone network, a subsequent FPN [35] as well as regression and classification heads. It uses the concept of anchor boxes as pre-defined regions in the image, eliminating the need for a Region Proposal Network. All of these steps can be realized with a single neural network graph, that can be created by tracing the computations of an exemplary network input. Assuming that all pre-processed images have the same resolution, the anchor boxes can be included in the graph as a constant tensor. The graph can then be efficiently deployed using either TensorRT or TorchScript. As noted in collaborative work [15], for some models the inference needs to be split into several network graphs, if there are operations that cannot be included in a static computation graph, like the Scatter operation in PointPillars [84].

Post-processing In the post-processing stage, the neural network classification and regression outputs are filtered by a detection threshold and decoded to generate the 2D bounding boxes. Finally, non-maximum suppression (NMS) is applied to avoid multiple boxes per object. Again, this can be efficiently computed using the OpenCV library. Note that these steps can not be included in the network graph, as they require logical operations that depend on the network input but would be treated as constant by the tracer.

7.2.3. Experiments

The models are trained and evaluated in a Python environment using the PyTorch library. As training and validation data, the nuScenes dataset [44] is used. The networks are deployed in a C++ environment with ROS for data streaming and visualization. The algorithm runtime is measured as the average runtime across the validation data. Experiments with different tools for hardware acceleration and quantization on the target platform, TensorRT and TorchScript, are performed. For the inference with TensorRT, trained PyTorch model is first exported to ONNX and then parsed to an optimized TensorRT runtime engine in a C++ environment on the target system. TensorRT allows to select the desired quantization when building the engine, with

Table 7.1.: *Runtime evaluation of RetinaNet with TensorRT and TorchScript using various quantization techniques and batch sizes.*

Batch size	Quant.	TensorRT	TorchScript
1	Float32	104 ms	210 ms
	Float16	38 ms	67 ms
	Int8	25 ms	-
6	Float32	619 ms	1041 ms
	Float16	201 ms	271 ms
	Int8	136 ms	-

the options of Float32, Float16, and Int8. The latter requires a use-case specific calibration, which can be done with the MinMax or entropy technique. With TorchScript, the quantization has to be selected before exporting the PyTorch model. At the time of writing, it is only possible to run Float32 and Float16 calculations with hardware acceleration. In addition, experiments are run with varying input image resolution. Finally, the impact of the available power supply on the runtime of the entire detection pipeline is studied.

Experimental Setup For the experiments, RetinaNet-18 [39], which is based on a small Resnet-18 [26] backbone, is used to account for the real-time requirements. It is trained and evaluated with the nuScenes [44] *train* and *val* dataset, respectively. To adapt the original dataset for the task of 2D object detection, the original 3D bounding boxes are projected onto the image plane and the 27 classes are mapped to a reduced set of *Car*, *Pedestrian*, *Truck*, *Motorcycle* and *Bicycle*. Two evaluation metrics are listed: the mAP and wmAP introduced in Section 2.2.7, with IoU threshold of 0.5.

Framework Analysis In the first experiment, the runtime of RetinaNet [39] is studied when deploying it with TensorRT and TorchScript, using the available quantization techniques. Two different inference batch sizes are considered, corresponding to a single image and a full surround view of six images, as present in autonomous vehicle prototypes or the used nuScenes dataset. Table 7.1 shows the results of this experiment. When running the model with the default Float32 values, the inference with TensorRT is substantially faster than with TorchScript, which underlines its optimization capabilities for convolutional layers. We note that in collaborative work [15], TorchScript has shown superior performance for fully connected layers, suggesting that the framework selection depends on the type of DNN. When using Float16 precision, the inference time significantly reduces across both tools, while TensorRT still allows for faster inference. With Int8 quantization in TensorRT, the runtime again significantly reduces, resulting in over four times faster inference relative to Float32. Some studies have shown that the runtime reduction can have even greater effects when working with larger batch sizes, offering more room for optimization [155]. In the experiment, this trend is confirmed when running inference on images from all six cameras in the nuScenes dataset in a single batch, observing the reduction factors slightly rising for Int8 quantization with TensorRT.

Table 7.2.: *Performance and runtime evaluation of RetinaNet using different image resolutions and quantization techniques on the nuScenes val set.*

Resolution	Quant.	Runtime	mAP	wmAP
Low 416×736	Float32	60 ms	0.298	0.391
	Float16	22 ms	0.299	0.391
	Int8, Entropy	16 ms	0.298	0.391
	Int8, MinMax		0.288	0.381
Mid 576×1024	Float32	104 ms	0.361	0.455
	Float16	38 ms	0.356	0.454
	Int8, Entropy	25 ms	0.355	0.455
	Int8, MinMax		0.355	0.454
High 832×1472	Float32	224 ms	0.395	0.488
	Float16	74 ms	0.395	0.487
	Int8, Entropy	50 ms	0.393	0.485
	Int8, MinMax		0.388	0.483

Runtime-Performance Analysis Of course, not only the runtime reduction should be evaluated but rather the trade-off between runtime and detection performance. Therefore, in the next experiment, the trade-off achievable with quantization versus changing the resolution of the input image is studied. The results of this experiment are shown in Table 7.2. Low, mid and high resolution are compared, which are chosen to have about twice as many pixels as the preceding resolution. Note that the number of anchor boxes increases proportional to the number of pixels. All models have been trained with input data down-scaled to the desired resolution. The mid resolution is the same that was used in the first experiment and TensorRT is used for this experiment due to the lower runtime and availability of Int8 quantization. The runtime reduction factor offered by quantization is similar in all dimensions with almost three times faster inference with Float16 and about four times faster inference with Int8. Interestingly, the impact of Float16 quantization on the performance is minimal, both in terms of mAP and wmAP. This also holds for Int8 quantization, where the performance is slightly reduced for the MinMax calibrator. This is arguably due to the 8-bit RGB images as input data, eliminating the need for higher-precision calculations. The impact of the image resolution is high on both runtime and detection performance, where the runtime decreases proportional to the number of pixels. Therefore, it is argued that the mid resolution model with Int8 quantization offers the best trade-off between runtime and performance. Note that, as discussed in collaborative work [15], these observations do not necessarily hold for point cloud data, where higher precision may be needed.

Power Supply Analysis In a final experiment for RetinaNet [39], the impact of the available power on the runtime of the model is studied using the mid resolution model and Int8 quantization from the previous experiment, due to its good runtime-performance trade-off. All experiments so far have been conducted with the MAXN power mode of the NVIDIA Jetson AGX, which corresponds to roughly 50 W. However, in a practical deployment setup, the

power resources are usually limited. As shown in Table 7.3, reducing the power mode has a high impact on the resulting runtime of the entire detection pipeline. Detailed results of pre-processing, inference and post-processing runtimes are therefore listed. Depending on the use-case, a trade-off between power supply and runtime has to be made.

Table 7.3.: *Detailed runtime of each block in the optimal RetinaNet detection pipeline using different power modes.*

Block	MAXN	30W	15W	10W
Pre-process	3 ms	5 ms	6 ms	7 ms
Inference	24 ms	35 ms	45 ms	94 ms
Post-process	4 ms	4 ms	5 ms	6 ms
Total	31 ms	44 ms	56 ms	107 ms

Summary In conclusion, the experiments show that the TensorRT framework achieves lower inference times for RetinaNet [39] than TorchScript. While quantization should always be considered for deployment, lowering the resolution can help to further reduce the runtime, but at the cost of a decreased detection performance. It is therefore advisable to prefer a mid to high resolution and Int8 quantization over a low resolution and Float32 precision. The runtime is also heavily influenced by the available power supply.

7.3. RC-BEVFusion deployment

From the models presented in Chapter 4, we need to select a variant for the deployment. Since we are aiming for a real-time demonstration, the computational demands are equally important as the detection performance. Therefore, we choose the variant based on MatrixVT [75] presented in Section 4.3.4 as the camera baseline and the proposed BEVFeatureNet radar encoder presented in Section 4.2.2, because it is computationally very efficient with just a slightly worse detection performance. Next, we need to identify necessary alterations to the model for the conversion to ONNX, as described in Section 7.2. First, the depth estimation network used in MatrixVT [75] contains a deformable convolution layer, which is not yet supported by ONNX at the time of writing. We therefore replace it with a regular convolution layer, which induces a slight decrease in detection performance. Further, the BEVFeatureNet includes a scattering operation that re-maps the dense encoded features onto their position in the BEV grid. Similar to how it was described for PointPillars [84] in collaborative work [15], this operation is not compatible with ONNX because it requires logical operations depending on the network input. Therefore, the BEVFeatureNet needs to be split into two ONNX network graphs for inference before and after the scattering operation, and the scattering has to be re-implemented in the C++ environment.

In Table 7.4, we compare the runtime of the main inference block of RC-BEVFusion based on MatrixVT [75] under various configurations and three

quantization techniques. First, we use the original configuration with 5 cameras for full surround view and a ResNet-50 backbone for the camera branch. With the hardware of the NVIDIA Jetson Xavier AGX, we cannot reach real-time performance of less than 100 ms even when using Int8 quantization. For this configuration, we would need to upgrade to a more recent hardware or use several computation platforms in parallel. If we reduce to single camera inference, the runtime decreases significantly, so that real-time performance can be reached. Note that the runtime doesn't decrease by a factor of 5 because the BEV encoder and detection heads of the network remain the same. To further speed up runtime while trading off detection performance, we can replace the ResNet-50 backbone with a ResNet-18 and use Int8 quantization, so that the model can be run in 41 ms.

Table 7.4.: *Runtime evaluation of RC-BEVFusion based on MatrixVT [75] with varying configurations and quantization techniques.*

Images	Image Backbone	Quantization	Runtime
5	ResNet-50	Float32	509 ms
		Float16	301 ms
		Int8	171 ms
1	ResNet-50	Float32	198 ms
		Float16	114 ms
		Int8	72 ms
1	ResNet-18	Float32	142 ms
		Float16	79 ms
		Int8	41 ms

In Table 7.5, we list the runtime of the full detection pipeline, which includes the image and radar data pre-processing, a first small inference block of the radar BEVFeatureNet encoder, the scattering operation, a second large inference block and post-processing to decode the bounding boxes. With our optimized inference block from above, we achieve a total runtime of 61 ms.

Table 7.5.: *Detailed runtime of each block in the optimal RC-BEVFusion detection pipeline.*

Block	Runtime
Image pre-processing	2 ms
Radar pre-processing	6 ms
Inference 1	1 ms
Scatter	1 ms
Inference 2	41 ms
Post-processing	10 ms
Total	61 ms

7.4. Domain adaptation

In this section, we study domain adaptation to improve the performance of RC-BEVFusion for the proprietary dataset. For the experiments in this chapter, we use the camera-only baseline of MatrixVT [75] as well as our radar-camera

fusion model RC-BEVFusion based on MatrixVT and BEVFeatureNet. We perform a set of experiments for each network and evaluate the results using the validation split of the proprietary dataset. For each network, we use three different training configurations: First, we only train the networks using the train split of the proprietary dataset. Then, we use the network pretrained with the train split of the nuScenes dataset [44] for inference. Finally, we apply domain adaptation on the pretrained network, fine-tuning it with the train split of the proprietary dataset. We follow the metrics from nuScenes [44] for evaluation.

Table 7.6.: *Evaluation of the domain adaptation for the proprietary dataset. The models were pretrained using the train split of nuScenes and fine-tuned using the train split of the proprietary dataset. The results are calculated on the validation split of the proprietary dataset.*

	Radar fusion	Pre-train	Fine-tune	mAP ↑	NDS ↑	mATE ↓	mASE ↓	mAOE ↓	mAVE ↓	mAAE ↓
			✓	0.13	0.1953	0.9581	0.2008	1.4467	2.5552	0.5379
[75]		✓		0.1272	0.2158	1.0229	0.2279	0.8823	2.5818	0.3677
		✓	✓	0.2791	0.3751	0.7663	0.1353	0.4717	1.8583	0.2707
	✓		✓	0.2429	0.2714	0.7015	0.1869	1.3786	2.5641	0.6118
Ours	✓	✓		0.2124	0.3364	0.7881	0.2112	0.5243	1.2664	0.1743
	✓	✓	✓	**0.4512**	**0.5143**	**0.4674**	**0.126**	**0.3604**	**1.2087**	**0.1592**

The aggregated metrics of the camera-only and the radar-camera fusion model are shown in Table 7.6. First, we compare the performance of the different versions for each model, either trained only using the proprietary dataset, only pretrained with nuScenes [44], or pretrained with nuScenes and then fine-tuned with the proprietary dataset. The results show that the networks perform best across all metrics when applying the domain adaptation approach with pretraining on the large nuScenes dataset and then fine-tuning on the smaller proprietary dataset. For both models, camera-only and radar-camera fusion, we observe a large increase in detection performance. Interestingly, the models trained on each dataset individually perform similarly well. This indicates that the larger amount of data available in nuScenes is about equally as important as the more domain specific, but smaller scale data of the proprietary dataset. When examining the results in more detail, we observe that the models trained only on the proprietary dataset achieve a higher mAP and a lower translation error, indicating that it is slightly superior at detecting and accurately localizing objects than the model trained only on nuScenes. This may be due to the difficulty of depth estimation across different cameras. However, the model trained only on nuScenes achieves lower errors for scale, orientation, velocity and attribute as well as for the aggregated NDS. For these less camera-specific metrics, the larger variety in nuScenes seems to outweigh the domain-specific knowledge.

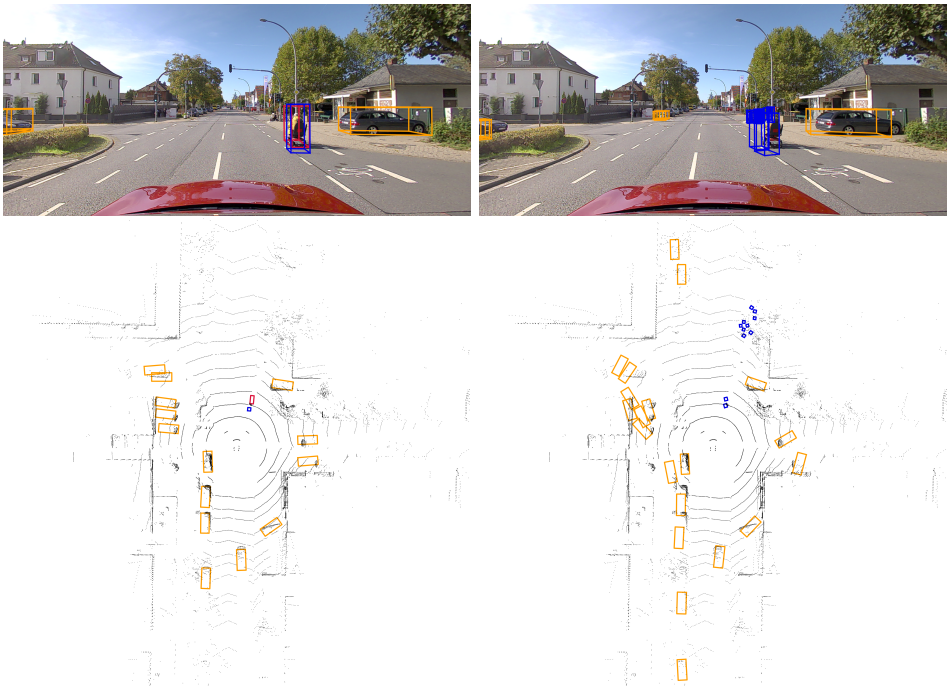
Next, we compare the performance of the radar-camera fusion model with the camera-only model. We can confirm the results achieved in Chapter 4 and observe large performance gains for all metrics across the different training schemes. This indicates that the added radar branch helps in all scenarios.

The model learns more quickly from a small set of training data, it generalizes better across datasets, and it achieves a higher final performance after applying domain adaptation. In the following, we focus on the comparison of the pretrained and fine-tuned models. Most notably, including the BEVFeatureNet increases the mAP by 62 %, the NDS by 37 % and the error metrics decrease by 39 % for translation, 7 % for scale, 24 % for orientation, 35 % for velocity and 41 % for attribute. These results substantiate the effectiveness of the proposed radar-camera fusion model.

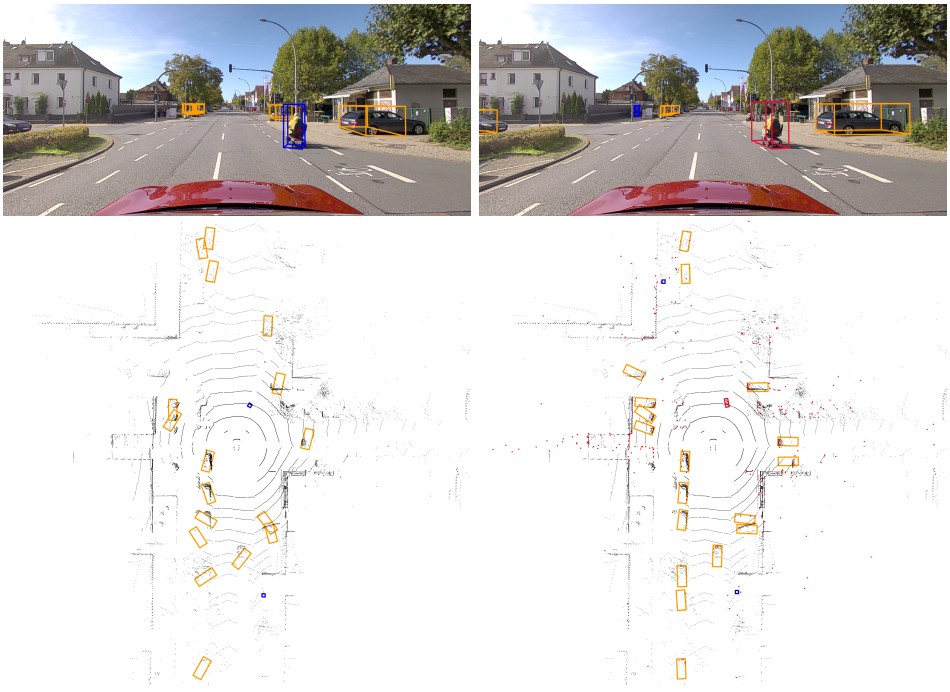
Table 7.7.: *Class-wise evaluation of the domain adaptation for the proprietary dataset. The models were pretrained using the train split of nuScenes and fine-tuned using the train split of the proprietary dataset. The results are calculated on the validation split of the proprietary dataset.*

	Radar fusion	Pre-train	Fine-tune	Class	AP ↑	ATE ↓	ASE ↓	AOE ↓	AVE ↓	AAE ↓
[75]			✓	car	0.252	0.965	0.137	1.505	1.748	0.392
				truck	0.028	1.082	0.271	1.116	4.33	0.635
				pedestrian	0.174	0.971	0.179	1.492	0.538	0.486
				two-wheeler	0.066	0.814	0.217	1.674	3.605	0.637
		✓		car	0.249	0.991	0.125	0.406	1.528	0.322
				truck	0.059	1.12	0.245	0.96	4.097	0.469
				pedestrian	0.148	1.093	0.218	1.458	0.847	0.406
				two-wheeler	0.054	0.888	0.324	0.706	3.855	0.274
		✓	✓	car	0.486	0.645	0.09	**0.195**	1.026	0.228
				truck	0.183	0.954	0.181	**0.127**	3.411	0.326
				pedestrian	0.295	0.803	0.148	1.172	0.601	0.403
				two-wheeler	0.153	0.663	**0.123**	0.392	2.395	0.125
Ours	✓		✓	car	0.439	0.704	0.124	1.558	1.927	0.432
				truck	0.066	1.094	0.265	1.464	4.063	0.618
				pedestrian	0.309	0.59	0.168	1.395	0.63	0.564
				two-wheeler	0.157	0.417	0.19	1.098	3.637	0.834
	✓	✓		car	0.399	0.744	0.129	0.472	0.623	0.21
				truck	0.116	0.952	0.209	0.286	**1.45**	**0.169**
				pedestrian	0.223	0.759	0.229	0.969	0.545	0.21
				two-wheeler	0.112	0.698	0.278	0.37	2.447	0.108
	✓	✓	✓	car	**0.699**	**0.36**	**0.085**	0.212	**0.467**	**0.199**
				truck	**0.284**	**0.695**	**0.163**	0.133	1.739	0.175
				pedestrian	**0.462**	**0.459**	**0.129**	**0.761**	**0.476**	**0.201**
				two-wheeler	**0.361**	**0.356**	0.127	**0.336**	**2.152**	**0.061**

We also list the full class-wise results of the experiments in Table 7.7. We observe that the domain-adapted radar-camera fusion model performs best for almost all metrics across all classes. There are only some isolated cases where other models perform slightly better, like the average orientation error for *Cars* and *Trucks*, which is better for the domain-adapted camera-only model, and the velocity and attribute error for *Trucks*, which is better for the radar-camera fusion model trained only on nuScenes. However, these true positive metrics are measured at significantly lower average precision values, rendering the results less comparable and making the domain-adapted radar-camera fusion model preferable nevertheless.



(a) *Camera-only PT+FT* **(b)** *Radar-camera PT*



(c) *Radar-camera FT* **(d)** *Radar-camera PT+FT*

Figure 7.5.: *Qualitative comparison of RC-BEVFusion variants on an exemplary validation frame of the proprietary dataset. PT = Pretrained with nuScenes [44], FT = Fine-tuned with proprietary dataset.*

We show a qualitative comparison using an exemplary frame from an urban scene in fair weather conditions in Figure 7.5 to showcase the effectiveness of the proposed RC-BEVFusion and the domain adaptation techniques. In Figure 7.5a, we see an inference example from the camera-only variant with domain-adaptation applied. The model is able to detect most objects in the near field pretty well, but it has limited range. In the remaining subfigures, we show inference results from different versions of the radar-camera fusion model. Figure 7.5b is from a model solely trained on nuScenes [44], which needs to generalize to the new domain. While it is able to detect and classify most objects quite well, it is uncertain in terms of depth and rotation estimation, which also leads to many duplicate detections. Next, Figure 7.5c shows the model trained from scratch using the proprietary dataset. It is able to detect most objects and has less duplicate detections, but struggles heavily with accurate rotation estimation. Finally, Figure 7.5d shows the domain adapted radar-camera version of RC-BEVFusion. Apart from slightly inaccurate rotation estimates for the partially occluded vehicles on the left side, it has very accurate predictions, even in far distances. In conclusion, this example clearly shows the superiority of the radar-camera fusion model with respect to its camera-only baseline and demonstrates the effectiveness of the applied domain adaptation.

(a) *False positive Pedestrian due to reflection from gully cover.*

(b) *Inaccurate rotation estimates due to imbalance in training data.*

Figure 7.6.: *Failure cases of the domain adapted RC-BEVFusion model on exemplary validation frames of the proprietary dataset.*

7.5. Failure cases

While the domain adapted RC-BEVFusion model works quite well in many cases, we want to highlight some failure cases in Figure 7.6. A good example for a potentially critical false positive *Pedestrian* is given in 7.6a. While it should be obvious from the camera image that there is no *Pedestrian* in the middle of the road, the network decided to trust the radar detection and created a detection. Upon closer inspection, it can be seen that the radar detection stems from a gully cover. Since this false positive detection does not occur in the previous and subsequent frames, it could have probably been filtered by tracking. However, it does highlight the criticality of trusting the predictions of a neural network, which can be fatally off. In this case, it could have lead to a sudden emergency stop putting the driver and the vehicles behind in danger.

A second, more common but less critical failure case is shown in Figure 7.6b. In this frame, we can observe how the network often tends to predict rotations in multiples of 90°, even though the actual rotations are clearly different. This is due to an imbalance of rotation angles in the training dataset. The vast majority of objects are heading in the same or the opposite direction as the ego vehicle, or are parked in a 90° angle. This error could be addressed by more careful design of the training scenes or by augmentation strategies that mitigate this problem.

7.6. Conclusion

In this chapter, we have presented our proprietary dataset, which was created with data recorded from the AI Sensing test vehicle as well as the use of our proposed automized labeling technique. We have then conducted an experimental study on the neural network deployment of RetinaNet [39] as a representative algorithm on the NVIDIA Jetson AGX Xavier. The necessary modifications and helpful tools are reported. The runtime of TensorRT and TorchScript are studied, indicating that TensorRT should be preferred for convolutional networks. The runtime can be further reduced by utilizing quantization, which has a low impact on the detection performance even with Int8. The available power supply in the embedded environment also has a significant impact on the runtime, which additionally has to be considered when choosing a setup for deployment. Next, we outlined the necessary steps to deploy our proposed RC-BEVFusion algorithm. To enhance the performance of the model on our proprietary data, we showed how to use the proprietary dataset to fine-tune RC-BEVFusion for better domain adaptation. We ran experiments to confirm the effectiveness of the proposed radar-camera fusion technique in comparison with its image-only baselines on the proprietary data, and to prove that the domain adaptation techniques help to improve the generalization capabilities of the algorithms. Finally, we showed qualitative examples confirming the quantitative results and highlighted some critical failure cases that still remain.

Chapter 8

Conclusions

In this thesis, we have explored the field of radar-camera fusion using deep neural networks for automotive object detection, a critical component in ADAS and automated driving systems. We have made significant contributions addressing two primary challenges: the semantic disparity between radar and camera data, and the integration of these heterogeneous data sources inherent in different coordinate systems into a coherent and efficient system. We have made considerable efforts to assure the practical applicability and real-time capability of our contributions using our AI Sensing test vehicle. The presented research has resulted in several novel contributions to the field, each aimed at enhancing the reliability and effectiveness of object detection in varied and challenging automotive environments.

8.1. Summary

One of the main contributions presented in Chapter 3 of this thesis is the development of FPP, a technique designed to intelligently select optimal points for fusing radar and camera data within a neural network architecture. This method addresses the critical question of when and where to merge heterogeneous sensor data for maximal effectiveness, a question that lies at the heart of sensor fusion challenges. The FPP technique, through its iterative pruning of less impactful fusion points, has shown considerable promise in optimizing 2D object detection performance, particularly in scenarios where camera reliability is compromised, such as night scenes. Complementing FPP, we introduced a novel projection technique for radar data, taking into account the uncertainties inherent in radar measurements. By creating a denser radar input, this technique enhances the semantic richness of the radar data, allowing for a more effective fusion with camera data.

In the second major contribution of this thesis, we address the problem which geometric representation should be used to fuse radar and camera data for the task of 3D object detection. In Chapter 4, we present the RC-BEVFusion architecture, a radar-camera fusion method that operates on the

downstream task-friendly BEV plane [130]. This approach leverages the geometric strengths of radar data while retaining the semantic detail provided by cameras. We have proposed two radar encoder branches that can be used to transform the radar point cloud into rich BEV feature grids. The adaptability of the RC-BEVFusion method is further demonstrated by its integration with various camera-based object detection architectures. The empirical results from this approach underscore the effectiveness of BEV plane fusion in terms of the key 3D object detection metrics. In particular, we observed a 28 % increase in terms of the NDS metric, which correlates well with the overall driving task [45]. Our experiments across datasets underscored that while camera-based detection thrives on large datasets with high visual variability, radar-based detection benefits more from high-resolution radars and can operate effectively with smaller datasets.

The practical application of these techniques was rigorously tested using an AI Sensing test vehicle equipped with a comprehensive sensor suite covering 5 radars, 5 cameras and 1 LiDAR sensor for full coverage of the surround view, which we presented in Chapter 5. We highlighted our extensive efforts to ensure data quality, including sensor calibration and synchronization, data streaming and pre-processing in a real-time environment as well as data recording tools.

Furthermore, in Chapter 6, we addressed the challenge of dataset availability considering required efforts for annotation through the development of an automated labeling method leveraging LiDAR data. This method, which involves LiDAR-based object detection followed by tracking and post-processing steps, provides a means to generate high-quality pseudo-labels, thus enabling more efficient training and fine-tuning of neural networks on proprietary datasets.

Finally, we researched the challenges of neural network deployment on an Edge AI in a real-world scenario inside the AI Sensing test vehicle in Chapter 7. To this end, we generated a proprietary dataset using data recorded from the AI Sensing test vehicle annotated using our proposed automized labeling technique. We then conducted a study for neural network deployment on an Edge AI platform. This practical experimentation provided valuable insights, offering guidance on optimizing algorithmic parameters for real-time performance while maintaining detection accuracy. We finally showcased the deployment of the proposed RC-BEVFusion algorithm inside the AI Sensing test vehicle. Our experiments underlined the effectiveness of domain adaptation when using the generated pseudo-labels to fine-tune our algorithm. The results also confirmed the enhanced performance and generalization capabilities of the radar-camera fusion model with respect to image-only and radar-only baselines.

In conclusion, this research has made substantial contributions to the field of automotive object detection through the development of advanced radar-camera fusion techniques using deep neural networks. The novel approaches presented, including Fusion Point Pruning, BEV plane fusion, and automized labeling, address critical challenges in the design and evaluation of radar-camera fusion systems. The experimental results underscore the potential of these methods in real-world applications, setting a foundation for future re-

search and development in the field. We hope that our research contributes to solving the technical challenges in ADAS and automated driving technologies to enable safer vehicles and more autonomous mobility in the future.

8.2. Future work

In future work, our contributions could be extended to more types of sensor data, including raw radar data and data from other sensor modalities like LiDAR or ultra-sound. In particular, we expect that the proposed Fusion Point Pruning technique could generate more valuable insights for the selection of fusion points in multi-sensor systems and be extended to other types of neural network architectures like transformers. We further believe that BEV feature fusion can serve as a flexible framework for multi-sensor fusion with many different modalities. Once larger datasets with high-performance radars and more visual variety are introduced, we think that the proposed RC-BEVFusion can leverage even more synergies between the strengths of cameras and radars. To improve the performance of the radar encoder branches, graph neural networks or cross-attention modules could be used in combination with the encoders suggested in this thesis. Further, the algorithm could be improved by incorporating temporal fusion methods or by adding radar-guided depth estimation to the view transformer. The proposed labeling pipeline may be used for cheap generation of large datasets with pseudo-labels. It can be further improved by combining it with out-of-domain detection to identify edge cases which require human labeling.

List of Abbreviations

2D	Two-Dimensional
3D	Three-Dimensional
4D	Four-Dimensional
AAE	Average Attribute Error
ADAS	Advanced Driver Assistant System
ADC	Analog-Digital-Converter
AI	Artificial Intelligence
AOE	Average Orientation Error
AP	Average Precision
ASE	Average Scale Error
ATE	Average Translation Error
AVE	Average Velocity Error
BB	Bounding Box
BEV	Bird's Eye View
BF	Blurring Filter
CAN	Controller Area Network
CBGS	Class-Balanced Grouping and Sampling
CCD	Charge-Coupled Device
CFAR	Constant False Alarm Rate
CMOS	Complementary Metal-Oxide-Semiconductor
CNN	Convolutional Neural Network
CPU	Central Processing Unit
CSV	Comma-Separated Values
CTRV	Constant Turn Rate and Velocity
CUDA	Compute Unified Device Architecture
DBC	CAN database file
DNN	Deep Neural Network
DS	Doppler Skewing

FFT	Fast Fourier Transform
FMCW	Frequency-Modulated Continous Wave
FN	False Negative
FoV	Field of View
FP	False Positive
FPN	Feature Pyramid Network
FPP	Fusion Point Pruning
FT	Fine-Tuned
GMSL	Gigabit Multimedia Serial Link
GNSS	Global Navigation Satellite System
GPS	Global Positioning System
GPU	Graphical Processing Unit
HD	High Definition
HR	High Resolution Radar
IoU	Intersection over Union
LiDAR	Light Detection and Ranging
LR	Low Resolution Radar
LRR	Long Range Radar
LSS	Lift-Splat-Shoot
mAAE	Mean Average Attribute Error
mAOE	Mean Average Orientation Error
mAP	Mean Average Precision
mASE	Mean Average Scale Error
mATE	Mean Average Translation Error
mAVE	Mean Average Velocity Error
MAXN	Maximum Power
MEMS	Micro-Electro-Mechanical Systems
MIMO	Multiple Input Multiple Output
MLP	Multi-Layer Perceptron
MRR	Mid Range Radar
NDS	NuScenes Detection Score
NMS	Non-Maximum Suppression
NN	Neural Network
ONNX	Open Neural Network Exchange
PC	Point Cloud
PCD	Point Cloud Data
PNG	Portable Network Graphic
PT	Pretrained
RA	Range-Azimuth data

RAD	Range-Azimuth-Doppler data
RAE	Range-Azimuth-Elevation data
RAED	Range-Azimuth-Elevation-Doppler data
RC	Radar Camera
RCS	Radar Cross Section
RD	Range-Doppler data
ReLU	Rectified Linear Unit
RGB	Red-Green-Blue color format
RoI	Region of Interest
ROS	Robot Operating System
RPN	Region Proposal Network
RV	Range View
SAE	Society of Automotive Engineers
SDK	Software Development Kit
SGD	Stochastic Gradient Descent
SP	Spinning Radar
SRR	Short Range Radar
TOPS	Trillion Operations Per Second
TP	True Positive
UC	Uncertainty Channel
UDP	User Datagram Protocol
UwRCS	Uncertainty-weighted Radar Cross Section channel
UYVY	Chrominance-Luminance color format
wmAP	Weighted mean Average Precision

Bibliography

[1] S. Singh, "Critical reasons for crashes investigated in the national motor vehicle crash causation survey," *Traffic Safety Facts Crash Stats*, vol. DOT HS 812 506, 2015.

[2] F. Duarte and C. Ratti, "The impact of autonomous vehicles on cities: A review," *Journal of Urban Technology*, vol. 25, no. 4, pp. 3–18, 2018.

[3] R. Qian, X. Lai, and X. Li, "3D object detection for autonomous driving: A survey," *arXiv preprint arXiv:2106.10823*, 2021.

[4] M. A. Khan, H. E. Sayed, S. Malik, T. Zia, J. Khan, N. Alkaabi, and H. Ignatious, "Level-5 autonomous driving — are we there yet? A review of research literature," *ACM Computing Surveys*, vol. 55, no. 2, pp. 1–38, 2022.

[5] Y. Wang, Q. Mao, H. Zhu, J. Deng, Y. Zhang, J. Ji, H. Li, and Y. Zhang, "Multi-modal 3D object detection in autonomous driving: A survey," *arXiv preprint arXiv:2106.12735*, 2021.

[6] A. Singh, "Vision-radar fusion for robotics BEV detections: A survey," in *IEEE Intelligent Vehicles Symposium*, IEEE, 2023, pp. 1–7.

[7] X. Ma, W. Ouyang, A. Simonelli, and E. Ricci, "3D object detection from images for autonomous driving: A survey," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2023.

[8] Y. Bengio, I. Goodfellow, and A. Courville, *Deep learning*. MIT press Cambridge, MA, USA, 2017, vol. 1.

[9] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," *Advances in Neural Information Processing Systems*, vol. 25, pp. 1097–1105, 2012.

[10] Di Feng, C. Haase-Schütz, L. Rosenbaum, H. Hertlein, C. Glaeser, F. Timm, W. Wiesbeck, and K. Dietmayer, "Deep multi-modal object detection and semantic segmentation for autonomous driving: Datasets, methods, and challenges," *IEEE Transactions on Intelligent Transportation Systems*, 2020, ISSN: 1524-9050.

[11] F. Engels, P. Heidenreich, M. Wintermantel, L. Stäcker, M. Al Kadi, and A. M. Zoubir, "Automotive radar signal processing: Research directions and practical challenges," *IEEE Journal of Selected Topics in Signal Processing*, vol. 15, no. 4, pp. 865–878, 2021.

[12] L. Stäcker, P. Heidenreich, J. Rambach, and D. Stricker, "Fusion point pruning for optimized 2D object detection with radar-camera fusion," in *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, 2022, pp. 3087–3094.

[13] L. Stäcker, S. Mishra, P. Heidenreich, J. Rambach, and D. Stricker, "RC-BEVFusion: A plug-in module for radar-camera bird's eye view feature fusion," in *DAGM German Conference on Pattern Recognition*, Springer, 2023, pp. 178–194.

[14] L. Stäcker, P. Heidenreich, J. Rambach, and D. Stricker, "Cross-dataset experimental study of radar-camera fusion in bird's-eye view," in *31st European Signal Processing Conference*, IEEE, 2023, pp. 810–814.

[15] L. Stäcker, J. Fei, P. Heidenreich, F. Bonarens, J. Rambach, D. Stricker, and C. Stiller, "Deployment of deep neural networks for object detection on edge AI devices with runtime optimization," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2021, pp. 1015–1022.

[16] R. Szeliski, *Computer vision: Algorithms and applications*. Springer Nature, 2022.

[17] *The OpenCV reference manual*, 4.9.0, OpenCV, Dec. 2023.

[18] J. Scheer and W. Holm, "Principles of modern radar: Basic principles," *Institution of Engineering and Technology*, 2010.

[19] S. Sun, A. P. Petropulu, and H. V. Poor, "MIMO radar for advanced driver-assistance systems and autonomous driving: Advantages and challenges," *IEEE Signal Processing Magazine*, vol. 37, no. 4, pp. 98–117, 2020.

[20] M. Toth, P. Meissner, A. Melzer, and K. Witrisal, "Performance comparison of mutual automotive radar interference mitigation algorithms," in *IEEE Radar Conference*, IEEE, 2019, pp. 1–6.

[21] Y. Li and J. Ibanez-Guzman, "LiDAR for autonomous driving: The principles, challenges, and trends for automotive LiDAR and perception systems," *IEEE Signal Processing Magazine*, vol. 37, no. 4, pp. 50–61, 2020.

[22] R. Horaud, M. Hansard, G. Evangelidis, and C. Ménier, "An overview of depth cameras and range scanners based on time-of-flight technologies," *Machine Vision and Applications*, vol. 27, no. 7, pp. 1005–1020, 2016.

[23] H. W. Yoo, N. Druml, D. Brunner, C. Schwarzl, T. Thurner, M. Hennecke, and G. Schitter, "MEMS-based LiDAR for autonomous driving," *e & i Elektrotechnik und Informationstechnik*, 2018.

[24] F. Amzajerdian, V. E. Roback, A. Bulyshev, P. F. Brewster, and G. D. Hines, "Imaging flash LiDAR for autonomous safe landing and spacecraft proximity operation," in *AIAA SPACE 2016*, 2016, p. 5591.

[25] C. V. Poulton, M. J. Byrd, E. Timurdogan, P. Russo, D. Vermeulen, and M. R. Watts, "Optical phased arrays for integrated beam steering," in *IEEE 15th International Conference on Group IV Photonics*, IEEE, 2018, pp. 1–2.

[26] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2016, pp. 770–778.

[27] Z. Liu, Y. Lin, Y. Cao, H. Hu, Y. Wei, Z. Zhang, S. Lin, and B. Guo, "Swin transformer: Hierarchical vision transformer using shifted windows," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2021, pp. 10 012–10 022.

[28] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[29] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "MobileNets: Efficient convolutional neural networks for mobile vision applications," *arXiv preprint arXiv:1704.04861*, 2017.

[30] M. Tan and Q. Le, "EfficientNet: Rethinking model scaling for convolutional neural networks," in *International Conference on Machine Learning*, PMLR, 2019, pp. 6105–6114.

[31] Z. Liu, H. Mao, C.-Y. Wu, C. Feichtenhofer, T. Darrell, and S. Xie, "A ConvNet for the 2020s," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2022, pp. 11 976–11 986.

[32] S. Xie, R. Girshick, P. Dollár, Z. Tu, and K. He, "Aggregated residual transformations for deep neural networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 1492–1500.

[33] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, *et al.*, "An image is worth 16x16 words: Transformers for image recognition at scale," *arXiv preprint arXiv:2010.11929*, 2020.

[34] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in Neural Information Processing Systems*, vol. 30, 2017.

[35] T.-Y. Lin, P. Dollár, R. Girshick, K. He, B. Hariharan, and S. Belongie, "Feature pyramid networks for object detection," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2017, pp. 2117–2125.

[36] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2014, pp. 580–587.

[37] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, "SSD: Single shot multibox detector," in *European Conference on Computer Vision*, Springer, 2016, pp. 21–37.

[38] X. Zhou, D. Wang, and P. Krähenbühl, "Objects as points," *arXiv preprint arXiv:1904.07850*, 2019.

[39] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár, "Focal loss for dense object detection," in *Proceedings of the IEEE international conference on computer vision*, 2017, pp. 2980–2988.

[40] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[41] J. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization," *Journal of Machine Learning Research*, vol. 12, no. 7, 2011.

[42] G. Hinton, N. Srivastava, and K. Swersky, "Lecture 6D–A separate, adaptive learning rate for each connection," *Slides of lecture Neural Networks for Machine Learning*, vol. 1, pp. 1–31, 2012.

[43] I. Loshchilov and F. Hutter, "Decoupled weight decay regularization," *arXiv preprint arXiv:1711.05101*, 2017.

[44] H. Caesar, V. Bankiti, A. H. Lang, S. Vora, V. E. Liong, Q. Xu, A. Krishnan, Y. Pan, G. Baldan, and O. Beijbom, "nuScenes: A multimodal dataset for autonomous driving," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2020, pp. 11 621–11 631.

[45] T. Schreier, K. Renz, A. Geiger, and K. Chitta, "On offline evaluation of 3D object detection for autonomous driving," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2023, pp. 4084–4089.

[46] A. Geiger, P. Lenz, and R. Urtasun, "Are we ready for autonomous driving? The KITTI vision benchmark suite," in *IEEE Conference on Computer Vision and Pattern Recognition*, IEEE, 2012, pp. 3354–3361, ISBN: 1467312282.

[47] A. Palffy, E. Pool, S. Baratam, J. F. P. Kooij, and D. M. Gavrila, "Multi-class road user detection with 3+1D radar in the View-of-Delft dataset," *IEEE Robotics and Automation Letters*, vol. 7, no. 2, pp. 4961–4968, 2022, ISSN: 2377-3766.

[48] W. Luo, J. Xing, A. Milan, X. Zhang, W. Liu, and T.-K. Kim, "Multiple object tracking: A literature review," *Artificial Intelligence*, vol. 293, p. 103 448, 2021.

[49] R. E. Kalman, "A new approach to linear filtering and prediction problems," *Transactions of the ASME–Journal of Basic Engineering*, vol. 82, no. Series D, pp. 35–45, 1960.

[50] S. Ren, K. He, R. Girshick, and J. Sun, "Faster R-CNN: Towards real-time object detection with region proposal networks," in *Advances in Neural Information Processing Systems*, 2015, pp. 91–99.

[51] N. Carion, F. Massa, G. Synnaeve, N. Usunier, A. Kirillov, and S. Zagoruyko, "End-to-end object detection with transformers," in *European Conference on Computer Vision*, Springer, 2020, pp. 213–229.

[52] X. Zhu, W. Su, L. Lu, B. Li, X. Wang, and J. Dai, "Deformable DETR: Deformable transformers for end-to-end object detection," *arXiv preprint arXiv:2010.04159*, 2020.

[53] F. Li, H. Zhang, S. Liu, J. Guo, L. M. Ni, and L. Zhang, "DN-DETR: Accelerate DETR training by introducing query denoising," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022, pp. 13 619–13 627.

[54] H. Zhang, F. Li, S. Liu, L. Zhang, H. Su, J. Zhu, L. M. Ni, and H.-Y. Shum, "DINO: DETR with improved denoising anchor boxes for end-to-end object detection," *arXiv preprint arXiv:2203.03605*, 2022.

[55] A. Mousavian, D. Anguelov, J. Flynn, and J. Kosecka, "3D bounding box estimation using deep learning and geometry," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2017, pp. 7074–7082.

[56] A. Simonelli, S. R. Bulo, L. Porzi, M. López-Antequera, and P. Kontschieder, "Disentangling monocular 3D object detection," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2019, pp. 1991–1999.

[57] T. Wang, X. Zhu, J. Pang, and D. Lin, "FCOS3D: Fully convolutional one-stage monocular 3D object detection," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2021, pp. 913–922.

[58] T. Wang, Z. Xinge, J. Pang, and D. Lin, "Probabilistic and geometric depth: Detecting objects in perspective," in *Conference on Robot Learning*, PMLR, 2022, pp. 1475–1485.

[59] F. Manhardt, W. Kehl, and A. Gaidon, "ROI-10D: Monocular lifting of 2D detection to 6D pose and metric shape," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 2069–2078.

[60] Y. Wang, W.-L. Chao, D. Garg, B. Hariharan, M. Campbell, and K. Q. Weinberger, "Pseudo-LiDAR from visual depth estimation: Bridging the gap in 3D object detection for autonomous driving," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 8445–8453.

[61] Y. Wang, V. C. Guizilini, T. Zhang, Y. Wang, H. Zhao, and J. Solomon, "DETR3D: 3D object detection from multi-view images via 3D-to-2D queries," in *Conference on Robot Learning*, PMLR, 2022, pp. 180–191, ISBN: 2640-3498.

[62] Y. Liu, T. Wang, X. Zhang, and J. Sun, "PETR: Position embedding transformation for multi-view 3D object detection," in *European Conference on Computer Vision*, Springer, 2022, pp. 531–548.

[63] Y. Liu, J. Yan, F. Jia, S. Li, A. Gao, T. Wang, and X. Zhang, "PETRv2: A unified framework for 3D perception from multi-camera images," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2023, pp. 3262–3272.

[64] S. Wang, X. Jiang, and Y. Li, "Focal-PETR: Embracing foreground for efficient multi-camera 3D object detection," *IEEE Transactions on Intelligent Vehicles*, 2023.

[65] T. Roddick, A. Kendall, and R. Cipolla, "Orthographic feature transform for monocular 3D object detection," *arXiv preprint arXiv:1811.08188*, 2018.

[66] J. Philion and S. Fidler, "Lift, splat, shoot: Encoding images from arbitrary camera rigs by implicitly unprojecting to 3D," in *European Conference on Computer Vision*, Springer, 2020, pp. 194–210.

[67] C. Reading, A. Harakeh, J. Chae, and S. L. Waslander, "Categorical depth distribution network for monocular 3D object detection," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021, pp. 8555–8564.

[68] H. Li, C. Sima, J. Dai, W. Wang, L. Lu, H. Wang, J. Zeng, Z. Li, J. Yang, H. Deng, *et al.*, "Delving into the devils of bird's-eye-view perception: A review, evaluation and recipe," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2023.

[69] J. Huang, G. Huang, Z. Zhu, and D. Du, "BEVDet: High-performance multi-camera 3D object detection in bird-eye-view," *arXiv preprint arXiv:2112.11790*, 2021.

[70] J. Huang and G. Huang, "BEVDet4D: Exploit temporal cues in multi-camera 3D object detection," *arXiv preprint arXiv:2203.17054*, 2022.

[71] J. Huang and G. Huang, "BEVPoolv2: A cutting-edge implementation of BEVDet toward deployment," *arXiv preprint arXiv:2211.17111*, 2022.

[72] Y. Li, Z. Ge, G. Yu, J. Yang, Z. Wang, Y. Shi, J. Sun, and Z. Li, "BEVDepth: Acquisition of reliable depth for multi-view 3D object detection," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 37, 2023, pp. 1477–1485.

[73] Y. Li, H. Bao, Z. Ge, J. Yang, J. Sun, and Z. Li, "BEVStereo: Enhancing depth estimation in multi-view 3D object detection with temporal stereo," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 37, 2023, pp. 1486–1494.

[74] J. Park, C. Xu, S. Yang, K. Keutzer, K. M. Kitani, M. Tomizuka, and W. Zhan, "Time will tell: New outlooks and a baseline for temporal multi-view 3D object detection," in *The Eleventh International Conference on Learning Representations*, 2022.

[75] H. Zhou, Z. Ge, Z. Li, and X. Zhang, "MatrixVT: Efficient multi-camera to BEV transformation for 3D perception," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2023, pp. 8548–8557.

[76] Z. Liu, H. Tang, A. Amini, X. Yang, H. Mao, D. L. Rus, and S. Han, "BEVFusion: Multi-task multi-sensor fusion with unified bird's-eye view representation," in *2023 IEEE international conference on robotics and automation (ICRA)*, IEEE, 2023, pp. 2774–2781.

[77] C. Yang, Y. Chen, H. Tian, C. Tao, X. Zhu, Z. Zhang, G. Huang, H. Li, Y. Qiao, L. Lu, *et al.*, "BEVFormer v2: Adapting modern image backbones to bird's-eye-view recognition via perspective supervision," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2023, pp. 17 830–17 839.

[78] Z. Li, Z. Yu, W. Wang, A. Anandkumar, T. Lu, and J. M. Alvarez, "FB-BEV: BEV representation from forward-backward view transformations," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2023, pp. 6919–6928.

[79] O. Schumann, M. Hahn, J. Dickmann, and C. Wöhler, "Semantic segmentation on radar point clouds," in *21st International Conference on Information Fusion*, IEEE, 2018, pp. 2179–2186, ISBN: 0996452761.

[80] B. Major, D. Fontijne, A. Ansari, R. Teja Sukhavasi, R. Gowaikar, M. Hamilton, S. Lee, S. Grzechnik, and S. Subramanian, "Vehicle detection with automotive radar using deep learning on range-azimuth-doppler tensors," in *Proceedings of the IEEE/CVF International Conference on Computer Vision Workshops*, 2019.

[81] A. Palffy, J. Dong, J. F. P. Kooij, and D. M. Gavrila, "CNN based road user detection using the 3D radar cube," *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 1263–1270, 2020, ISSN: 2377-3766.

[82] M. Dreher, E. Erçelik, T. Bänziger, and A. Knoll, "Radar-based 2D car detection using deep neural networks," in *IEEE 23rd International Conference on Intelligent Transportation Systems*, IEEE, 2020, pp. 1–8, ISBN: 1728141494.

[83] N. Scheiner, F. Kraus, N. Appenrodt, J. Dickmann, and B. Sick, "Object detection for automotive radar point clouds - A comparison," *AI Perspectives*, vol. 3, no. 1, pp. 1–23, 2021, ISSN: 2523-398X.

[84] A. H. Lang, S. Vora, H. Caesar, L. Zhou, J. Yang, and O. Beijbom, "PointPillars: Fast encoders for object detection from point clouds," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 12 697–12 705.

[85] A. Danzer, T. Griebel, M. Bach, and K. Dietmayer, "2D car detection in radar data with PointNets," in *IEEE Intelligent Transportation Systems Conference*, IEEE, 2019, pp. 61–66, ISBN: 1538670240.

[86] N. Scheiner, O. Schumann, F. Kraus, N. Appenrodt, J. Dickmann, and B. Sick, "Off-the-shelf sensor vs. experimental radar–How much resolution is necessary in automotive radar classification?" In *IEEE 23rd International Conference on Information Fusion*, IEEE, 2020, pp. 1–8.

[87] M. Ulrich, S. Braun, D. Köhler, D. Niederlöhner, F. Faion, C. Gläser, and H. Blume, "Improved orientation estimation and detection with hybrid object detection networks for automotive radar," in *IEEE 25th International Conference on Intelligent Transportation Systems*, IEEE, 2022, pp. 111–117.

[88] C. R. Qi, W. Liu, C. Wu, H. Su, and L. J. Guibas, "Frustum PointNets for 3D object detection from RGB-D data," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2018, pp. 918–927.

[89] Z. Wang and K. Jia, "Frustum ConvNet: Sliding frustums to aggregate local point-wise features for amodal 3D object detection," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, IEEE, 2019, pp. 1742–1749, ISBN: 1728140048.

[90] S. Vora, A. H. Lang, B. Helou, and O. Beijbom, "PointPainting: Sequential fusion for 3D object detection," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 4604–4612.

[91] C. Wang, C. Ma, M. Zhu, and X. Yang, "PointAugmenting: Cross-modal augmentation for 3D object detection," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021, pp. 11 794–11 803.

[92] J. Fei, W. Chen, P. Heidenreich, S. Wirges, and C. Stiller, "SemanticVoxels: Sequential fusion for 3D pedestrian detection using LiDAR point cloud and semantic segmentation," in *IEEE International Conference on Multisensor Fusion and Integration for Intelligent Systems*, IEEE, 2020, pp. 185–190, ISBN: 1728164222.

[93] T. Liang, H. Xie, K. Yu, Z. Xia, Z. Lin, Y. Wang, T. Tang, B. Wang, and Z. Tang, "BEVFusion: A simple and robust LiDAR-camera fusion framework," *Advances in Neural Information Processing Systems*, vol. 35, pp. 10 421–10 434, 2022.

[94] F. Drews, Di Feng, F. Faion, L. Rosenbaum, M. Ulrich, and C. Gläser, "DeepFusion: A robust and modular 3D object detector for LiDARs, cameras and radars," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, IEEE, 2022, pp. 560–567, ISBN: 1665479272.

[95] S. Yao, R. Guan, X. Huang, Z. Li, X. Sha, Y. Yue, E. G. Lim, H. Seo, K. L. Man, X. Zhu, *et al.*, "Radar-camera fusion for object detection and semantic segmentation in autonomous driving: A comprehensive review," *IEEE Transactions on Intelligent Vehicles*, 2023.

[96] F. Gaisser and P. P. Jonker, "Road user detection with convolutional neural networks: An application to the autonomous shuttle WEpod," in *Fifteenth IAPR International Conference on Machine Vision Applications*, IEEE, 2017, pp. 101–104, ISBN: 4901122169.

[97] R. Nabati and H. Qi, "RRPN: Radar region proposal network for object detection in autonomous vehicles," in *IEEE International Conference on Image Processing*, IEEE, 2019, pp. 3093–3097, ISBN: 1538662493.

[98] R. Nabati and H. Qi, "Radar-camera sensor fusion for joint object detection and distance estimation in autonomous vehicles," *arXiv preprint arXiv:2009.08428*, 2020.

[99] V. John and S. Mita, "RVNet: Deep sensor fusion of monocular camera and radar for image-based obstacle detection in challenging environments," in *Pacific-Rim Symposium on Image and Video Technology*, Springer, 2019, pp. 351–364.

[100] V. John, M. K. Nithilan, S. Mita, H. Tehrani, R. S. Sudheesh, and P. P. Lalu, "SO-Net: Joint semantic segmentation and obstacle detection using deep fusion of monocular camera and radar," in *Pacific-Rim Symposium on Image and Video Technology*, Springer, 2019, pp. 138–148.

[101] S. Chadwick, W. Maddern, and P. Newman, "Distant vehicle detection using radar and vision," in *International Conference on Robotics and Automation*, IEEE, 2019, pp. 8311–8317, ISBN: 153866027X.

[102] F. Nobis, M. Geisslinger, M. Weber, J. Betz, and M. Lienkamp, "A deep learning-based radar and camera sensor fusion architecture for object detection," in *Sensor Data Fusion: Trends, Solutions, Applications*, IEEE, 2019, pp. 1–7, ISBN: 172815085X.

[103] S. Chang, Y. Zhang, F. Zhang, X. Zhao, S. Huang, Z. Feng, and Z. Wei, "Spatial attention fusion for obstacle detection using mmWave radar and vision sensor," *Sensors*, vol. 20, no. 4, 2020.

[104] R. Nabati and H. Qi, "CenterFusion: Center-based radar and camera fusion for 3D object detection," in *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, 2021, pp. 1527–1536.

[105] Y. Long, A. Kumar, D. Morris, X. Liu, M. Castro, and P. Chakravarty, "RADIANT: Radar-image association network for 3d object detection," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 37, 2023, pp. 1808–1816.

[106] Y. Kim, J. W. Choi, and D. Kum, "GRIF net: Gated region of interest fusion network for robust 3D object detection from radar point cloud and monocular image," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, IEEE, 2020, pp. 10 857–10 864, ISBN: 1728162122.

[107] J.-J. Hwang, H. Kretzschmar, J. Manela, S. Rafferty, N. Armstrong-Crews, T. Chen, and D. Anguelov, "CramNet: Camera-radar fusion with ray-constrained cross-attention for robust 3D object detection," in *European Conference on Computer Vision*, Springer, 2022, pp. 388–405.

[108] Y. Kim, S. Kim, J. W. Choi, and D. Kum, "CRAFT: Camera-radar 3D object detection with spatio-contextual fusion transformer," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 37, 2023, pp. 1160–1168.

[109] Z. Wu, G. Chen, Y. Gan, L. Wang, and J. Pu, "MVFusion: Multi-view 3D object detection with semantic-aligned radar and camera fusion," in *IEEE International Conference on Robotics and Automation*, IEEE, 2023, pp. 2766–2773.

[110] S. Pang, D. Morris, and H. Radha, "TransCAR: Transformer-based camera-and-radar fusion for 3D object detection," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, IEEE, 2023, pp. 10 902–10 909.

[111] X. Chen, T. Zhang, Y. Wang, Y. Wang, and H. Zhao, "FUTR3D: A unified sensor fusion framework for 3D detection," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2023, pp. 172–181.

[112] K. Lei, Z. Chen, S. Jia, and X. Zhang, "HvDetFusion: A simple and robust camera-radar fusion framework," *arXiv preprint arXiv:2307.11323*, 2023.

[113] P. Wolters, J. Gilg, T. Teepe, F. Herzog, A. Laouichi, M. Hofmann, and G. Rigoll, "Unleashing HyDRa: Hybrid fusion, depth consistency and radar for unified 3D perception," *arXiv preprint arXiv:2403.07746*, 2024.

[114] Y. Kim, J. Shin, S. Kim, I.-J. Lee, J. W. Choi, and D. Kum, "CRN: Camera radar net for accurate, robust, efficient 3D perception," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2023, pp. 17 615–17 626.

[115] J. Kim, M. Seong, G. Bang, D. Kum, and J. W. Choi, "RCM-Fusion: Radar-camera multi-level fusion for 3d object detection," *arXiv preprint arXiv:2307.10249*, 2023.

[116] Y. Zhou, L. Liu, H. Zhao, M. López-Benítez, L. Yu, and Y. Yue, "Towards deep radar perception for autonomous driving: Datasets, methods, and challenges," *Sensors*, vol. 22, no. 11, p. 4208, 2022.

[117] J.-.-L. Déziel, P. Merriaux, F. Tremblay, D. Lessard, D. Plourde, J. Stanguennec, P. Goulet, and P. Olivier, "PixSet: An opportunity for 3D computer vision to go beyond point clouds with a full-waveform LiDAR dataset," in *IEEE International Intelligent Transportation Systems Conference*, IEEE, 2021, pp. 2987–2993, ISBN: 1728191424.

[118] K. Bansal, K. Rungta, S. Zhu, and D. Bharadia, "Pointillism: Accurate 3D bounding box estimation with multi-radars," in *Proceedings of the 18th Conference on Embedded Networked Sensor Systems*, 2020, pp. 340–353.

[119] T. Matuszka, I. Barton, Á. Butykai, P. Hajas, D. Kiss, D. Kovács, S. Kunsági-Máté, P. Lengyel, G. Németh, L. Pető, *et al.*, "aiMotive dataset: A multimodal dataset for robust autonomous driving with long-range perception," in *International Conference on Learning Representations 2023 Workshop on Scene Representations for Autonomous Driving*, 2023.

[120] A. Ouaknine, A. Newson, J. Rebut, F. Tupin, and P. Pérez, "CARRADA dataset: Camera and automotive radar with range-angle-doppler annotations," in *25th International Conference on Pattern Recognition*, IEEE, 2021, pp. 5068–5075.

[121] A. Zhang, F. E. Nowruzi, and R. Laganiere, "RADDet: Range-azimuth-doppler based radar object detection for dynamic road users," in *18th Conference on Robots and Vision*, IEEE, 2021, pp. 95–102.

[122] T.-Y. Lim, S. A. Markowitz, and M. N. Do, "RaDICaL: A synchronized FMCW radar, depth, IMU and RGB camera data dataset with low-level FMCW radar signals," *IEEE Journal of Selected Topics in Signal Processing*, vol. 15, no. 4, pp. 941–953, 2021.

[123] M. Meyer and G. Kuschk, "Automotive radar dataset for deep learning based 3D object detection," in *16th European Radar Conference*, IEEE, 2019, pp. 129–132, ISBN: 2874870579.

[124] L. Zheng, Z. Ma, X. Zhu, B. Tan, S. Li, K. Long, W. Sun, S. Chen, L. Zhang, M. Wan, *et al.*, "TJ4DRadSet: A 4D radar dataset for autonomous driving," in *IEEE 25th International Conference on Intelligent Transportation Systems*, IEEE, 2022, pp. 493–498.

[125] D.-H. Paek, S.-H. Kong, and K. T. Wijaya, "K-Radar: 4D radar object detection for autonomous driving in various weather conditions," *Advances in Neural Information Processing Systems*, vol. 35, pp. 3819–3829, 2022.

[126] M. Sheeny, E. De Pellegrin, S. Mukherjee, A. Ahrabian, S. Wang, and A. Wallace, "RADIATE: A radar dataset for automotive perception in bad weather," in *IEEE International Conference on Robotics and Automation*, IEEE, 2021, pp. 1–7.

[127] K. Burnett, D. J. Yoon, Y. Wu, A. Z. Li, H. Zhang, S. Lu, J. Qian, W.-K. Tseng, A. Lambert, K. Y. Leung, *et al.*, "Boreas: A multi-season autonomous driving dataset," *The International Journal of Robotics Research*, vol. 42, no. 1-2, pp. 33–42, 2023.

[128] S. A. Janowsky, "Pruning versus clipping in neural networks," *Physical Review A*, vol. 39, no. 12, p. 6600, 1989.

[129] D. Blalock, J. J. Gonzalez Ortiz, J. Frankle, and J. Guttag, "What is the state of neural network pruning?" *Proceedings of Machine Learning and Systems*, vol. 2, pp. 129–146, 2020.

[130] Y. Hu, J. Yang, L. Chen, K. Li, C. Sima, X. Zhu, S. Chai, S. Du, T. Lin, W. Wang, *et al.*, "Planning-oriented autonomous driving," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2023, pp. 17 853–17 862.

[131] C. R. Qi, H. Su, K. Mo, and L. J. Guibas, "PointNet: Deep learning on point sets for 3D classification and segmentation," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2017, pp. 652–660.

[132] T. Yin, X. Zhou, and P. Krahenbuhl, "Center-based 3D object detection and tracking," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021, pp. 11 784–11 793.

[133] J. Hu, L. Shen, and G. Sun, "Squeeze-and-excitation networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 7132–7141.

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

[135] Y. Yao, Z. Luo, S. Li, T. Fang, and L. Quan, "MVsNet: Depth inference for unstructured multi-view stereo," in *Proceedings of the European Conference on Computer Vision*, 2018, pp. 767–783.

[136] H. Law and J. Deng, "CornerNet: Detecting objects as paired keypoints," in *Proceedings of the European Conference on Computer Vision*, 2018, pp. 734–750.

[137] B. Zhu, Z. Jiang, X. Zhou, Z. Li, and G. Yu, "Class-balanced grouping and sampling for point cloud 3D object detection," *arXiv preprint arXiv:1908.09492*, 2019.

[138] J. Beck, "Camera calibration with non-central local camera models," Ph.D. dissertation, Karlsruher Institut für Technologie, 2021.

[139] J. Kümmerle, T. Kühner, and M. Lauer, "Automatic calibration of multiple cameras and depth sensors with a spherical target," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, IEEE, 2018, pp. 1–8.

[140] R. Schubert, E. Richter, and G. Wanielik, "Comparison and evaluation of advanced motion models for vehicle tracking," in *11th International Conference on Information Fusion*, IEEE, 2008, pp. 1–6.

[141] Y. Zhou and O. Tuzel, "VoxelNet: End-to-end learning for point cloud based 3D object detection," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 4490–4499.

[142] B. Li, "3D fully convolutional network for vehicle detection in point cloud," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, IEEE, 2017, pp. 1513–1518.

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

[144] K. Chen, R. Oldja, N. Smolyanskiy, S. Birchfield, A. Popov, D. Wehr, I. Eden, and J. Pehserl, "MVLiDARNet: Real-time multi-class scene understanding for autonomous driving using multiple views," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, IEEE, 2020, pp. 2288–2294.

[145] A. Laddha, S. Gautam, S. Palombo, S. Pandey, and C. Vallespi-Gonzalez, "MVFuseNet: Improving end-to-end object detection and motion forecasting through multi-view fusion of LiDAR data," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021, pp. 2865–2874.

[146] S. Deng, Z. Liang, L. Sun, and K. Jia, "VISTA: Boosting 3D object detection via dual cross-view spatial attention," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022, pp. 8448–8457.

[147] O. Ronneberger, P. Fischer, and T. Brox, "U-Net: Convolutional networks for biomedical image segmentation," in *Medical Image Computing and Computer-Assisted Intervention–MICCAI 2015: 18th International Conference, Munich, Germany, October 5-9, 2015, Proceedings, Part III 18*, Springer, 2015, pp. 234–241.

[148] NVIDIA Corporation, *Jetson AGX Xavier*, `https://www.nvidia.com/de-de/autonomous-machines/embedded-systems/jetson-agx-xavier/`, [Hardware; accessed 09-July-2021], 2018.

[149] A. Gholami, S. Kim, Z. Dong, Z. Yao, M. W. Mahoney, and K. Keutzer, "A survey of quantization methods for efficient neural network inference," in *Low-Power Computer Vision*, Chapman and Hall/CRC, 2022, pp. 291–326.

[150] H. Wu, P. Judd, X. Zhang, M. Isaev, and P. Micikevicius, "Integer quantization for deep learning inference: Principles and empirical evaluation," *arXiv preprint arXiv:2004.09602*, 2020.

[151] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, *et al.*, "PyTorch: An imperative style, high-performance deep learning library," *Advances in neural information processing systems*, vol. 32, 2019.

[152] J. Bai, F. Lu, K. Zhang, *et al.*, *ONNX: Open neural network exchange*, https://github.com/onnx/onnx, [GitHub repository; accessed 09-July-2021], 2019.

[153] H. Vanholder, "Efficient inference with TensorRT," in *GPU Technology Conference*, vol. 1, 2016.

[154] "IEEE standard for binary floating-point arithmetic," *ANSI/IEEE Std 754-1985*, pp. 1–20, 1985. DOI: 10.1109/IEEESTD.1985.82928.

[155] S. Migacz, "8-bit inference with TensorRT," in *GPU Technology Conference*, vol. 2, 2017, p. 5.

[156] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, A. Y. Ng, *et al.*, "ROS: An open-source robot operating system," in *ICRA workshop on open source software*, Kobe, Japan, vol. 3, 2009, p. 5.

[157] V. Comito, *Nuscenes2bag*, https://github.com/clynamen/nuscenes2bag, [GitHub repository; accessed 09-July-2021], 2019.

[158] G. Bradski, "The OpenCV library," *Dr. Dobb's Journal of Software Tools*, 2000.

[159] OpenPCDet Development Team, *OpenPCDet: An open-source toolbox for 3D object detection from point clouds*, https://github.com/open-mmlab/OpenPCDet, 2020.

[160] D. Luebke, "CUDA: Scalable parallel programming for high-performance scientific computing," in *5th IEEE International Symposium on Biomedical Imaging: From Nano to Macro*, 2008, pp. 836–838. DOI: 10.1109/ISBI.2008.4541126.

Appendix A

Curriculum Vitae

Lukas Stefan Stäcker

Contact via LinkedIn

Professional Experience

2024–Present	**Heraeus** Hanau, Germany *Data Scientist*
2020–2024	**Stellantis** Rüsselsheim am Main, Germany *Researcher*

Education

2020–Present	**RPTU Kaiserslautern-Landau** Kaiserslautern, Germany *Computer Science, PhD*
2018–2019	**TU Darmstadt** Darmstadt, Germany *Computational Engineering, M. Sc.*
2015–2019	**TU Darmstadt** Darmstadt, Germany *Industrial Engineering, M. Sc.*
2012–2015	**TU Darmstadt** Darmstadt, Germany *Industrial Engineering, B. Sc.*
2008–2011	**Internatsschule Schloss Hansenberg** Geisenheim, Germany *Abitur*

Appendix B

Publication List

B.1. Journal articles

1. F. Engels, P. Heidenreich, M. Wintermantel, **L. Stäcker**, M. Al Kadi, and A. M. Zoubir, "Automotive Radar Signal Processing: Research Directions and Practical Challenges," *IEEE Journal of Selected Topics in Signal Processing*, vol. 15, no. 4, 2021, pp. 865–878.

B.2. Conference papers

1. **L. Stäcker**, P. Heidenreich, J. Rambach, and D. Stricker, "Fusion Point Pruning for Optimized 2D Object Detection with Radar-Camera Fusion," in *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, 2022, pp. 3087–3094.
2. **L. Stäcker**, S. Mishra, P. Heidenreich, J. Rambach, and D. Stricker, "RC-BEVFusion: A Plug-In Module for Radar-Camera Bird's Eye View Feature Fusion," in *DAGM German Conference on Pattern Recognition*, Springer, 2023, pp. 178–194.
3. **L. Stäcker**, P. Heidenreich, J. Rambach, and D. Stricker, "Cross-Dataset Experimental Study of Radar-Camera Fusion in Bird's-Eye View," in *31st European Signal Processing Conference*, IEEE, 2023, pp. 810–814.
4. **L. Stäcker**, J. Fei, P. Heidenreich, F. Bonarens, J. Rambach, D. Stricker, and C. Stiller, "Deployment of Deep Neural Networks for Object Detection on Edge AI Devices with Runtime Optimization," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2021, pp. 1015–1022.