Machine Learning Applied to Radar Data: Classification and Semantic Instance Segmentation of Moving Road Users

Von der Fakultät für
Elektrotechnik und Informationstechnik
der Technischen Universität Dortmund
genehmigte

Dissertation

zur Erlangung des Grades
Doktor der Naturwissenschaften
(Dr. rer. nat.)

eingereicht von

Ole Schumann

Dortmund, 2021

Tag der mündlichen Prüfung:	16.03.2021
Hauptreferent:	Prof. Dr. rer. nat. Christian Wöhler
Korreferent:	Prof. Dr.-Ing. Klaus Dietmayer

Ole Schumann, M. Sc.: *Machine Learning Applied to Radar Data: Classification and Semantic Instance Segmentation of Moving Road Users,* Arbeitsgebiet Bildsignalverarbeitung, Technische Universität Dortmund, © March 2021.

ABSTRACT

Classification and semantic instance segmentation applications are rarely considered for automotive radar sensors. In current implementations, objects have to be tracked over time before some semantic information can be extracted. In this thesis, data from a network of 77 GHz automotive radar sensors is used to construct, train and evaluate machine learning algorithms for the classification of moving road users. The classification step is deliberately performed early in the process chain so that a subsequent tracking algorithm can benefit from this extra information. For this purpose, a large data set with real-world scenarios from about 5 h of driving was recorded and annotated.

Given that the point clouds measured by the radar sensors are both sparse and noisy, the proposed methods have to be sensitive to those features that discern the individual classes from each other and at the same time, they have to be robust to outliers and measurement errors. Two groups of applications are considered: classification of clustered data and semantic (instance) segmentation of whole scenes. In the first category, specifically designed density-based clustering algorithms are used to group individual measurements to objects. These objects are then used either as input to a manual feature extraction step or as input to a neural network, which operates directly on the bare input points. Different classifiers are trained and evaluated on these input data.

For the algorithms of the second category, the measurements of a whole scene are used as input, so that the clustering step becomes obsolete. A newly designed recurrent neural network for instance segmentation of point clouds is utilized. This approach outperforms all of the other proposed methods and exceeds the baseline score by about ten percentage points.

In additional experiments, the performance of human test candidates on the same task is analyzed. This study shows that temporal correlations in the data are of great use for the test candidates, who are nevertheless outrun by the recurrent network.

Them bats is smart. They use radar!

— David Letterman

ACKNOWLEDGMENTS

Writing an external PhD thesis requires that a common understanding about its contents is shared among all parties concerned. I am extremely grateful that my two supervisors, Prof. Christian Wöhler at TU Dortmund and Dr. Markus Hahn at Daimler, not only quickly agreed on the basic topic but also gave me all the freedom one could wish for to develop my own ideas and to go after my research interests.

I want to thank Prof. Christian Wöhler for his valuable feedback, his constructive criticism and his openness in every situation. From our first phone call onward, I enjoyed his down-to-earth views, his positive support as well as his words of caution, which turned out to be valuable in the following years. He cushioned the impact of drastic modifications in the organizational structure at Daimler, and his encouragement to pursue one of the emerging possibilities made the subsequent adaptions much easier for me.

I am also grateful for his support in administrative tasks at TU Dortmund, especially for his commitment related to changes in the examination regulations. In this context, I would like to thank Prof. Rehtanz for the kind communication and his willingness to liberalize the rules for PhD candidates from other disciplines.

I want to thank Dr. Markus Hahn for his innovative ideas, his contagious positive attitude and his amazing support from the first minute onward. Without him, this thesis would not have been possible in this form. His feedback and advice always helped me to do the next step and to overcome the many challenges and hurdles that presented themselves. Having him as a supervisor at Daimler was of inestimable value for me.

I thank all “Radar Doks” for the inspiring discussions we had, the honest feedback we shared and the mutual support we gave each other. Working with you really eased the whole process and made it fun to come to work. I thank the Radar team at Daimler – including the Ulm colleagues who left the company – for helping me out whenever needed and giving me valuable tips about the life in a major corporation.

I want to thank everybody who helped me counterchecking my thesis and giving me ideas how to visualize some data.

I am grateful to my family for everything they have done for me and for always supporting me in any possible way. Last, but definitely not least, I want to thank my wife for her endless support, understanding and love.

Thank you!

PUBLICATIONS

The following list contains all publications related to this thesis that were released beforehand in workshops, conference proceedings or journals.

1. O. Schumann, M. Hahn, J. Dickmann, C. Wöhler, "Comparison of random forest and long short-term memory network performances in classification tasks using radar", in *2017 Sensor Data Fusion: Trends, Solutions, Applications (SDF)*, https://doi.org/10.1109/SDF.2017.8126350, 2017.
2. O. Schumann, M. Hahn, J. Dickmann, C. Wöhler, "Supervised Clustering for Radar Applications: On the Way to Radar Instance Segmentation", in *2018 IEEE MTT-S International Conference on Microwaves for Intelligent Mobility, ICMIM 2018*, https://doi.org/10.1109/ICMIM.2018.8443534, 2018.
3. M. Horn, O. Schumann, M. Hahn, J. Dickmann, K. Dietmayer, "Motion Classification and Height Estimation of Pedestrians Using Sparse Radar Data", in *2018 Sensor Data Fusion: Trends, Solutions, Applications (SDF)*, https://doi.org/10.1109/SDF.2018.8547092, 2018.
4. O. Schumann, M. Hahn, J. Dickmann, C. Wöhler, "Semantic Segmentation on Radar Point Clouds", in *2018 21st International Conference on Information Fusion (FUSION)*, https://doi.org/10.23919/ICIF.2018.8455344, 2018.
5. N. Scheiner, O. Schumann, "Machine Learning Concepts For Multiclass Classification Of Vulnerable Road Users", in *16th European Radar Conference (EuRAD 2019)*, Presentation in a workshop, 2019.
6. O. Schumann, J. Lombacher, M. Hahn, C. Wöhler, J. Dickmann, "Scene Understanding with Automotive Radar", in *IEEE Transactions on Intelligent Vehicles*, vol 5, no. 2, https://doi.org/10.1109/TIV.2019.2955853, 2020.

The next list contains publications which are not directly linked to the contents of this thesis.

1. J. Tilly, F. Weishaupt, O. Schumann, J. Klappstein, J. Dickmann, G. Wanielik, "Polarimetric Signatures of a Passenger Car", in *IEEE 2019 Kleinheubach Conference*, https://ieeexplore.ieee.org/document/8890117, 2019.

2. O. Schumann, J. Dickmann, J. Lombacher, N. Scheiner, "AI in automotive Radar", in *22nd European Microwave Week (EuMW 2019) – Automotive Forum*, Presentation in a workshop, https://doi.org/10.13140/RG.2.2.10544.38407, 2019.

3. N. Scheiner, O. Schumann, F. Kraus, N. Appenrodt, J. Dickmann, B. Sick, "Off-the-shelf sensor vs. experimental radar – How much resolution is necessary in automotive radar classification?", in *2020 23rd International Conference on Information Fusion (FUSION)*, 2020.

4. J. Tilly, S. Haag, O. Schumann, F. Weishaupt, B. Duraisamy, J. Dickmann, M. Fritzsche, "Detection and Tracking on Automotive Radar Data with Deep Learning", in *2020 23rd International Conference on Information Fusion (FUSION)*, 2020.

The following master's thesis was supervised:

1. M. Horn, "Motion Classification and Height Estimation of Pedestrians Using Sparse Radar Data". Ulm University, 2018.

CONTENTS

1

INTRODUCTION

Radar sensors are frequently used in the automotive field, e.g. for driver assistance systems. Their robustness to adverse weather conditions, the possibility to measure not only the position of an object but also its radial velocity and their low price make radar sensors a viable solution for various applications. Often, however, these sensors are only used for their eponymous tasks – detection and ranging. Semantic information about a detected object is either not considered at all or only available after an object has been tracked over a certain period. Especially due to the low angular resolution of most automotive radars, little information is available for each object so that classification algorithms have to deal with sparse and noisy data. Usually, cameras and lidar sensors are used for the *understanding* of the environment due to their high angular resolution and dense data representation. Given the increasing requirements on the perception side of autonomous driving systems, it has to be re-evaluated if additional class information from a radar sensor becomes relevant.

In the last years, machine learning algorithms gained popularity. The widespread use of these type of algorithms has three causes: availability of data, potent computer hardware and convincingly high performance. Especially for the image community, many publicly available data sets make it easy for researchers to develop new algorithms and to compare them to existing ones using standardized benchmarks. With modern GPUs, both training and evaluation of large neural networks becomes feasible and the accuracy of many machine learning based perception systems is a convincing argument against more traditional approaches.

At the most basic level, the distinction between truly moving road users and noisy measurements is relevant for almost all driving tasks. In order to estimate how other road users currently move and to predict where they will be in the next few seconds, tracking algorithms are needed. Semantic information can help to choose the correct motion model for a tracking algorithm and therefore allow for a more reliable tracking and prediction. Functional safety considerations suggest that multiple redundant sensors should provide the needed information so that it is reasonable to maximize each sensor's capabilities.

In this thesis, different machine learning based methods are proposed, extended and analyzed which provide semantic information for *moving* road users based on data measured by off-the-shelf automotive radar sensors. Further information about the static environment is not provided. The goal is to provide semantic information as early as possible, so that subsequent processing steps like a tracking algorithm can profit from it. Hence, the approach followed in this work inverts the usual order in which reliable tracking is needed before any classification can be done.

This thesis is structured as follows. In the next two chapters, basic information about the machine learning algorithms used in this work is given and the fundamental measurement principle of radar sensors is introduced. These two chapters summarize currently available knowledge and contain no own contribution, except for bringing this material together. Excluding the "State of the Art" sections, the following five chapters contain own material that was partly published beforehand in [105], [232]–[235].

In Chapter 4, the data set that was recorded for this thesis is described and the annotation process is explained. Furthermore, radar cross sections of different road users are presented and it is analyzed how different bicycle types look like for a radar sensor.

Grouping of individual radar measurements to objects is discussed in Chapter 5, where clustering methods are introduced and a supervised component is added to a usually unsupervised density based clustering algorithm. Adding domain knowledge to the clustering step proofs to be beneficial in the subsequent classification steps, which are discussed in detail in Chapter 6. Three different classifiers are confronted with the problem of assigning the correct class label to the extracted clusters. Methods relying on a manual feature extraction step are compared to a setting where a neural network generates the relevant features on its own. To insert temporal information into the classification process, recurrent neural networks are used as well.

To eliminate the difficult clustering step, semantic (instance) segmentation networks are considered in Chapter 7. A novel recurrent network structure is introduced that provides not only semantic segmentation for each input point but also groups measurements of the same instance together.

For the classification task as well as for the semantic segmentation, experiments with human annotators are performed to assess how well humans perform on the same exercise.

The last sections of Chapters 5, 6 and 7 summarize the main findings of each chapter and a global recap is given in Chapter 8.

2

FUNDAMENTALS OF MACHINE LEARNING

In 1988, the Austrian scientist Hans Moravec wrote in his book "Mind Children" that

> "[...] it is comparatively easy to make computers exhibit adult-level performance [...] on intelligence tests or playing checkers, and difficult or impossible to give them the skills of a one-year-old when it comes to perception and mobility." [168]

Similar statements were previously made by himself, Rodney Brooks and Marvin Minsky and are today summarized under the term "Moravec's paradox". The core statement is that in contrast to logical and abstract reasoning, sensorimotor skills are exceptionally difficult to implement in robots. For example, a computer can easily integrate a system of (complex-valued) differential equations – a task that is extremely difficult and time consuming for most humans – but the decision whether a photograph contains a bird proofs to be exceptionally more complex for a computer [170]. For the integration of differential equations, strict rules can be formulated so that after repetitive application of said rules a solution to the problem can be obtained within some numerical bounds. In contrast, there are no simple rules to determine if a picture contains a bird or not. One could try to describe a bird as having wings, feathers and a beak and look for these key features in the image. However, wings are not necessarily visible when a bird sits still, a beak cannot be seen if the bird is photographed from behind and individual feathers are only visible at close distances. Nevertheless, we as humans have no problem to identify a bird in a picture even if one or even all of the mentioned properties are not captured. Of course, one could try to find more elaborate rules, e.g. use color information, gradients and contrast to nearby objects or infer that objects in the sky have a higher probability to be a bird. History proofed, however, that no matter how hard one tries to find more and more elaborate rules for such *classification* problems, one does not reach the performance a human shows on this task.

Therefore, rule-based algorithms seem insufficient for some tasks humans do intuitively correct. So instead of giving a full set of instructions to the computer

that describe the algorithm completely from input to output, the idea of machine learning algorithms is to provide large amounts of data and let the computer infer itself the rules it needs to solve the given task. To stay with the example of the detection of birds in a picture, one would collect large a amount of images that either do or do not contain a bird, present them to the machine learning algorithm in a suitable form and let the algorithm create a set of rules that discriminate images with birds from images without birds. That is, the computer *learns* the decision function from provided data instead of applying fixed rules to obtain a result. The scientific effort now shifts from designing appropriate algorithms that detect a bird in a picture to the task of designing algorithms that allow the computer to learn these rules by themselves.

Of course, there is no strict border between traditional rule based systems and machine learning algorithms. In traditional machine learning, the researcher might use his domain knowledge to calculate input features from the actual raw input data, which are then provided to the learning part of the algorithm. That is, rule based feature extraction steps are combined with a learning part in which a decision function is inferred from the "classically" created features. The feature extraction part is often interpreted as providing a more suitable *representation* of the input data to the learning algorithm. At the other end of the spectrum of machine learning, even the suitable representation is learned from the input data. That is, the algorithm learns which representation of the input data is useful for the task as well as the rules that map the representation to a decision. In popular science, the term *deep learning* is often used for these kind of algorithms.

This chapter aims to introduce basic terminology that is needed for later reference. The presentation of each topic will be brief to avoid unnecessary long repetition of textbook knowledge.

Machine learning algorithms can be sorted into categories depending on the amount of human supervision the respective algorithm needs to fulfill its task. Commonly, three categories are used:

1. Unsupervised Learning,
2. Semi-Supervised Learning,
3. Supervised Learning.

The term "supervised" describes that annotated data (e.g. annotated by humans) is needed by an algorithm in a training phase. Annotation (or labeling) of data means that some knowledgeable entity attaches a label to each input datum. For example, for the classification task of birds, the labeling process would require that either the label "contains a bird" or the label "does not contain a bird" be attached to each input picture. The concrete label task depends of course on the application and has

to match the output of the target algorithm since for example an algorithm that decides about the pure existence of a bird in a picture requires simpler annotation than an algorithm which should mark all pixels in an image that belong to birds. Annotation of data is an often-overlooked difficulty when machine learning topics are discussed. Details on the labeling of radar data for this work are presented later in Chapter 4.

For each of the three categories, this chapter contains a separate section in which algorithms are introduced that are relevant for the remainder of this work. It should be noted, however, that there is a fluid transition between the three categories and algorithms from one category can often be extended or simplified so that they shift more to the (un-)supervised direction.

2.1 SUPERVISED LEARNING

The currently most prominently used machine learning algorithms stem from the field of supervised learning. At the core, all supervised learning schemes need the following five ingredients:

1. A data set $\mathcal{X} = \{x_0, x_1, \dots x_N\}$
2. Ground truth labels $\mathcal{Y} = \{y_0, y_1, \dots y_N\}$
3. An algorithm that takes one x_i as input, has trainable parameters θ_j, $j = 1, \dots, N_{\text{param}}$, and outputs a prediction $\hat{y}_i$
4. A loss function $L(\hat{y}, y)$, sometimes also called cost function or error function
5. An optimizer

For a given task, all five elements have to be chosen carefully to construct a well performing system. A famous quote by Tom Mitchell lists three instead of five different ingredients to describe a machine learning system [165]:

> "A computer program is said to **learn** from experience E with respect to some class of tasks T and some performance measure P, if its performance at tasks in T, as measured by P, improves with experience E."

In [79], this statement is used to define the three individual pieces experience E, task T and performance measure P. Here, however, a more fine-grained view is taken with the five elements listed above.

The general working procedure of a supervised machine learning algorithm is that an input datum x is presented to the program which then computes an output $\hat{y}$. This predicted output is compared to the desired output y and an error $L(\hat{y}, y)$ is computed. It is then the optimizer's task to apply updates to the internal weights θ_j based on the current error L. These steps are repeated for a fixed number of times, until the error function L converges or after any other user defined stop criterion. The goal is that the error function decreases so that the predicted output $\hat{y}$ equals the ground truth output y for as many training samples x as possible. The important aspect that separates supervised learning algorithms from pure optimization is that one does not only want high performance on the already collected data $\mathcal{X}$ but also on new and unseen data $\mathcal{X}'$. That is, the algorithm should learn a *generalization* and not learn the presented examples by heart. Therefore, the *training phase* of an algorithm is always followed by a *test phase* during which the outputs of the algorithm are again compared to the ground truth labels to calculate performance metrics. In this phase, however, the parameters θ_j remain constant since only the performance of the algorithm is evaluated and no further refinement of the weights is done. In Section A.1 in the appendix, more detailed information is given about each of the five ingredients and some examples are presented.

Since many researchers in the field of machine learning use the same terminology but with varying definitions, it is useful to define some common phrases. The term **classification** is often used for two slightly different things: a) as a headline for all tasks in which an algorithm returns semantic information about the input data or b) specifically for tasks in which a single label (often an integer $y, \hat{y} \in \{1, \ldots, N_{\text{cls}}\}$) is returned for each input datum. In this work, solely the second definition of the term *classification* is used. More specifically, exclusively *multi-class* classification tasks are considered, i.e. $N_{\text{cls}} > 2$. In *multi-label* classification tasks, each input x_i can have multiple ground truth values. For example, finding genres a movie or a newspaper article belongs to is an ambiguous task so that multiple categories fit for a given movie or article. In this work, *multi-label* tasks are *not* considered, i.e. each input x_i has exactly one single ground truth value y_i. With images as input, a typical classification task is to assign the largest object shown in the image to one of N_{cls} categories, e.g. an image showing a cat would get the label `cat` and an image with a dog would get the label `dog`. For point cloud data, e.g. data from a lidar or a radar sensor, a classification task would use as input either the raw measurement points that belong to one object or some higher-level representation in the form of extracted feature vectors.

Semantic segmentation is in general a more complex task than mere classification. Segmentation of an image means that lines, shapes and connected regions that are defined by color, shape or gradients are returned so that the input image is cut into multiple segments. In *semantic* segmentation, the segmentation is performed in

such a way that pixels from the same semantic class are grouped together and the respective label is attached to each pixel. In contrast to classification, not a single label $\hat{y}$ for an input image x is returned, but a whole matrix with one label (or even one probability vector) for each pixel of the image is produced by the machine learning algorithm. For example, an image which shows two cats and one dog would be segmented into three parts. The first part contains all pixels that belong to the two cats, the second part contains only pixels with the label `dog` and the third part consists of all `background` pixels. Note that in semantic segmentation different objects of the same semantic class are not discerned but grouped together. That is, the number of objects of the same semantic class cannot be retrieved from the semantic segmentation result. Semantic segmentation on point clouds just means that for each pixel a class label is computed.

Finally, in **semantic instance segmentation** not only a semantic segmentation is performed but also bounding boxes or even pixel-masks are returned for each individual object in the input image. In the example given before, three bounding boxes with the labels `cat`, `cat` and `dog` would be returned along with the pixel-wise semantic segmentation. On point cloud data, either the instances can be represented with bounding boxes – just as it is done for image data – or the individual points that belong to one object are grouped together. The latter method corresponds to a pixel-wise mask in image processing.

Slightly different from the three previously named tasks is supervised **regression**. In this task – in contrast to classification and segmentation – a continuous output variable $\hat{y} \in \mathbb{R}$ is estimated for each input x. Popular examples in the machine learning community with accompanied data set are the estimation of house prices in Boston [45] or the prediction of the burned area of forest fires in Portugal [90].

In the following subsections, two algorithms that are used throughout this thesis will be introduced: random forest classifiers and artificial neural networks.

2.1.1 *The Random Forest Classifier*

The random forest algorithm is usually grouped into the "traditional" machine learning category, because the representation of the input features remains unchanged during training, i.e. no further feature extraction of the input data is done. The method was developed by Leo Breiman in 2001 [26] and therefore the idea was published about 10 years before the current machine learning hype cycle started and before deep neural networks started to dominate the field.

Random forests are a clever way to combine weak learners – in this case *decision trees* – to one strong learner. To understand the mechanics of a random forest it is

therefore necessary to introduce decision trees first. Even though random forests can be used for both classification and regression, only the classification algorithm is discussed here. Detailed information about regression forests can be found in [47], [104], [149].

2.1.1.1 *Decision Trees*

A decision tree for classification tasks takes as input a feature vector $x \in \mathbb{R}^{N_{\text{feat}}}$ and outputs a vector of "probabilities" $p(y|x) \in \mathbb{R}^{N_{\text{cls}}}$ so that the predicted class label of input x is given by $\hat{y} = \arg\max_y p(y|x)$. The tree consists of several nodes at which the incoming data is split into two sub-nodes. The split condition of each node is derived during training and is always based on exactly one of the N_{feat} available features. A split S is of the form

$$f_S : \mathbb{R}^{N_{\text{feat}}} \to \{0,1\}, \quad x \mapsto \begin{cases} 0, & \text{for } x_i \leq a_S \\ 1 & \text{otherwise,} \end{cases} \tag{2.1}$$

where x_i is one of the N_{feat} features of the feature vector x and a_S is a threshold for this feature. Both the index i of the feature dimension and a_S are determined during training and are part of the split definition. At test time an input vector is passed down the tree by checking the split criterion of each node and depending on whether x passes the test ($f_S(x) = 1$ or $f_S(x) = 0$), the data are directed to one or the other child node. This is repeated until a *leaf node* is found, i.e. a node without any child nodes. The distribution of the training data in this leaf node is stored during training and returned as $p(y|x)$. This is the reason why the term *probabilities* was put into parentheses previously: $p(y|x)$ reflects merely the distribution of the training data in the leaf nodes of the tree and is not an independent certainty measure. If the tree is grown until only samples of one class remain in the leaf nodes, the reported probability for the chosen class will always be one.

In Fig. 2.1, a decision tree for a simple three-class classification problem is displayed. Each data point is characterized by a feature vector $x \in \mathbb{R}^7$. At the first node n_1 the third feature f_3 is used for the split. If for an input vector $x_3 \leq a_1$, where a_1 is the threshold that was fixed during training, then the feature vector is passed down to the left child n_2 and if $x_3 > a_1$ the sample is passed to the right child which is in this case already a leaf node n_3. If a feature vector ends up in n_3, one always obtains $p(\text{blue}|x) = 1$ since during training only samples of the "blue" class were found in this region. If some x passes the first test, i.e. $x_3 \leq a_1$ holds true, then at node n_2 a second split is performed, but this time the seventh feature is tested against threshold a_2. The leaf nodes n_4 and n_5 are not pure since each node contains samples from multiple classes. Part b) of the figure shows the data distribution

along the two selected feature dimensions three and seven. It should be noted here that a decision tree can only devise split rules that are axis parallel. Every other non-linear split rule is approximated by cutting the feature space in small axis-aligned regions.

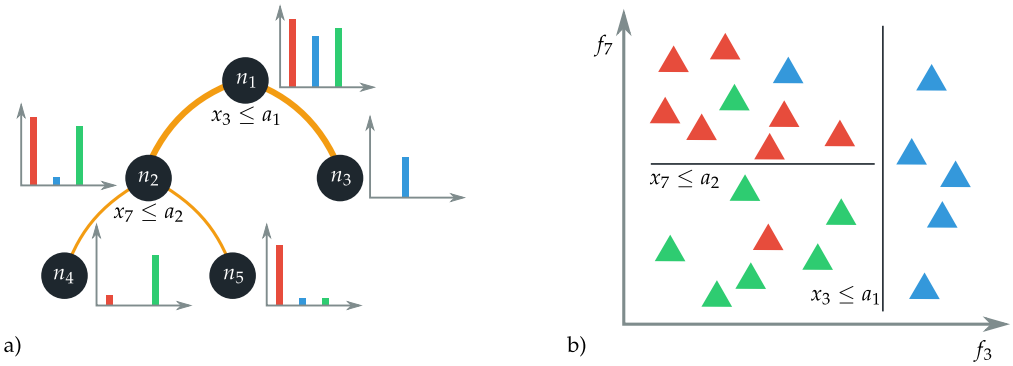


Figure 2.1: Decision tree for three different classes (red, blue and green). In a) the tree with its three leaf nodes (n_3, n_4, n_5) along with the class distribution in each node is displayed. The first split criterion checks whether the third feature in the input x is smaller than a_1 and the second criterion at node n_2 check whether the seventh feature in x is smaller than a_2. If input x passes a test, it is redirected to the left child node. In b) the data distribution in the two-dimensional subspace spanned by feature f_3 and feature f_7 is displayed along with the two decision boundaries that separate the space in three regions.

During training the tree has to "grow", i.e. the split criterion for the current node has to be fixed (selection of feature index i and threshold a_S) and child nodes for the two possible decisions have to be created. Construction of the trees is often done in a greedy fashion, i.e. the best possible split is performed on the first level and thereafter the split criteria for the two emerging child nodes are identified. That means that once a node was created, it is not modified again. The best split S is found when the *information gain* obtained through splitting the data under this criterion is maximal. Information gain IG is defined in this context as the difference between an impurity measure $I(T)$ of the data T in the parent node and the "conditional impurity" $I(T|S)$:

$$IG = I(T) - I(T|S). \tag{2.2}$$

The "conditional impurity" $I(T|S)$ is given by the weighted sum of the impurities that emerge when the split criterion S is applied to the current node:

$$I(T|S) = \sum_{i \in \{L,R\}} \frac{|T^i|}{|T|} I(T^i). \tag{2.3}$$

The split causes that the parent data T is divided into T^L and T^R (left and right branch of the tree). The expression $|T^i|$ stands for the number of elements in the respective branch after the split node.

For the impurity measure I itself two popular definitions exist. The first definition assumes that the best split S is found when the entropy is reduced. In this case, the impurity measure is given by

$$I_H(T) = H(T) = -\sum_{i=1}^{N_{\text{cls}}} p_i \log p_i. \tag{2.4}$$

The probabilities $p_1, \ldots, p_{N_{\text{cls}}}$ are given by the fraction of data in the current node of the respective class. The "conditional impurity" is defined in this case as the more commonly used *conditional entropy* $H(T|S)$. In fact, the term "conditional impurity" seems not to be used in the literature as a generalization of the concept of conditional entropy. In [49], the term is defined in a different way than it is used here, namely as a weighted impurity measure with the weights stemming from some predictability index. Nevertheless, in this context the wording fits in the sense that different impurity measures I can be inserted in the calculation of $I(T|S)$ and $H(T|S)$ being a special case of this. Loosely speaking, the conditional entropy $H(Y|X)$ describes the uncertainty about the random variable Y given information about the random variable X. In the case of a decision tree, $H(T|S)$ quantifies the uncertainty in the data after split criterion S was applied. It is therefore reasonable to seek a split S that minimizes this uncertainty, i.e. a split that maximizes $H(T) - H(T|S)$.

The second impurity measure is defined by the so-called *Gini impurity* I_G. In contrast to the entropy, a more probabilistic description of impurity is chosen here: The Gini impurity equals the probability that a randomly chosen sample would be incorrectly labeled, if both the sample and the label were chosen according to the label distribution in the current node. The probability for choosing an element of class i is per definition given by p_i and the probability of assigning a wrong label is simply $1 - p_i$. Hence the Gini impurity reads

$$I_G = \sum_{i=1}^{N_{\text{cls}}} p_i(1 - p_i) = 1 - \sum_{i=1}^{N_{\text{cls}}} p_i^2. \tag{2.5}$$

Comparing Eq. (2.4) and Eq. (2.5) shows that entropy requires a computationally expensive logarithm for the estimation of the impurity whereas the Gini impurity simply uses the squares of the p_i. In practice both measures yield similar or identical results, so that often the Gini impurity is favored [201], [202].

Given an impurity measure I, the information gain IG has to be computed for each possible split of the N_{feat} different features. To illustrate this process, consider the

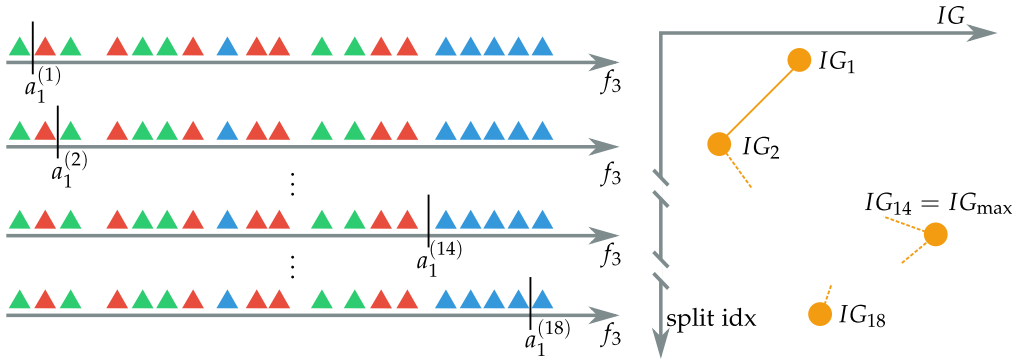


Figure 2.2: Different splits of the training data along the dimension of feature f_3. On the right hand side a plot of the information gain at each split index is shown. At split index 14, the maximum information gain is found so that this split would be chosen.

situation in which a tree was built up to a certain depth and the optimal split for the current node is sought. For each of the N_{feat} feature dimensions a sorted list of training samples in the current node is needed. Often, the whole training set is sorted once prior to training so that no additional sorting in the child nodes is needed. The information gain of one feature dimension is computed by starting with a cut that separates the sample with the smallest value from the remaining samples. Then the cut is shifted one position further towards larger values so that the second cut separates two samples. This is repeated until the final cut between the two largest values in this feature dimension has been performed. In Fig. 2.2, the split procedure along the dimension of feature f_3 is shown. The best split is found at index 14, when the information gain is maximal and most "blue" data points are separated from the others. This procedure has the great benefit that the information gain at each new cut can be computed in $\mathcal{O}(1)$ time. This follows from the fact that no recursion in the computation of IG is needed if only a single data point is added or removed from a previous configuration. Sorting the data along each feature dimension beforehand ensures this and hence allows for a fast computation of the information gain in each feature dimension. It can be shown that a decision tree with N training samples can be constructed in $\mathcal{O}(N_{\text{feat}} \cdot N \log N)$ time, see chapter 5 in [149].

2.1.1.2 *Boosting and Bagging: From Decision Trees to a Random Forest*

Fully-grown decision trees have a small bias but a large variance. The large variance is a direct result of how decision trees are constructed: the deeper one goes down a tree, the more the resulting region in feature space is confined by the split criteria of the previously passed nodes. Each split is created by looking for the optimal

split at the current node. Therefore, often rather noisy decision boundaries are created which take the many fluctuations in the training data set into account. A single decision tree hence tends to fit the resulting decision function too close to the training distribution, i.e. decision trees tend to overfit to the data. Enforcing bounds on the depth of a tree (sometimes called *pruning*) lowers the variance and is hence a well-established method to reduce overfitting. This happens at the cost of an increase in the bias so that the overall accuracy of the decision tree classifier decreases.

The bias–variance tradeoff is a well-researched problem in statistics and machine learning [73]. It states that predictive models with low bias often have a high variance and vice versa. Ensemble methods aim to reduce the variance (or bias) of a group of classifiers while keeping the low bias (variance) of each single classifier. This is done by combining the high variance (high bias) "weak learners" into one classifier so that the variances (biases) reduce by taking averages over the individual classifier outputs.

Multiple popular ways exist to combine the weak-learns "decision tree" into one strong learner. Two of them are *random forests* [26] and *(gradient) boosted trees*. The former uses fully-grown decision trees which have a low bias but a high variance whereas the latter method is an example of the more general *AdaBoost* algorithm [68] which is designed to reduce the bias of low-variance base-classifiers. That is, for the boosted trees algorithm, decision trees with a very small depth are used which have a high bias and a small variance in contrast to the fully-grown decision trees used in random forests.

In practice, both random forests and gradient boosted trees yield similar results, although with careful tuning the boosted trees algorithm can provide more accurate results with less trees. However, as so often in machine learning, general statements are hard to make since the performance of classification algorithms varies strongly with the data set under investigation so that for one data set gradient boosted trees show higher performance [33], [174] and for other data sets random forests are better or at least on par [172], [262], [287]. In contrast to random forests, which can be built in parallel as each decision tree is independent of the other trees in the forest, boosted trees are built sequentially. The idea is to add trees that perform well on those samples that were incorrectly classified by the already present trees. Details can be found in [68], [69].

Random forests are one example of a *bagging* method. The term *bagging* is a portmanteau of **b**ootstrap and **agg**regat**ing** [25]. Bagging works by selecting for each of the k base-classifiers a different training data set from the N available training samples. Just like the original training data set, each of the k newly generated training sets consists of N samples. These samples are drawn *with replacement* from

the original training data so that one specific sample may occur multiple times in one generated training set or not at all [25]. The probability p that one sample makes it into one specific training set is the complementary probability to the event that after N draws only the other $N-1$ samples were chosen:

$$p = 1 - \left(\frac{N-1}{N}\right)^N \tag{2.6}$$

$$= 1 - e^{-1} \approx 0.63 \quad \text{for} \quad N \to \infty. \tag{2.7}$$

This implies that on average 63 % of each generated training set are unique samples so that about 37 % of the original training samples are not present in one bootstrapped set. Each of the k base-classifiers (decision trees) is then trained on one of the k generated training sets, the results of the classifiers are aggregated and by averaging the scores, the most probable class can be obtained. The samples that are not used for training are called *out of bag* samples. They can be directly used to estimate the performance of the random forest by passing these samples down the trees and comparing the predicted output with the ground truth label.

In addition to randomness in the data selection step, random forests introduce also random selection of the features that are considered at each split during the creation of the individual decision trees. To be more precise, at each level of each decision tree, m of the originally present N_{feat} features are chosen and only those are considered during the search for the best possible split. Popular choices for the hyper-parameter m are $m = \lfloor \sqrt{N_{\text{feat}}} \rfloor$ and $m = \lfloor \log_2 N_{\text{feat}} \rfloor$. Historically, this second randomness was introduced a few years after experiments with combining decision trees solely via bagging were performed. The term "random forest" was initially used for a many different methods that combine decision trees to one strong learner. Today, however, "Random Forests" is a registered trademark of the company Minitab LLC [279] and used almost exclusively to describe the algorithm that uses both bagging and random feature selection.

Two schemes exist to merge the individual results of the decision trees in the random forest: hard-voting and soft-voting. In the hard-voting scheme, it is simply counted how many of the k trees in the forest voted for each class and the class with the maximum number of votes wins. Soft-voting differs in so far that the class-probabilities obtained from each tree are taken into account. If $p_j(y|x)$ is the probability vector obtained from the jth tree, then the combined probability vector $p(y|x)$ is the average taken over all trees in the forest:

$$p(y|x) = \frac{1}{k} \sum_{j=1}^{k} p_j(y|x). \tag{2.8}$$

The predicted class is then again simply the class with the highest probability value in $p(y|x)$.

The number of features m considered in each split is often quoted to be the most influential hyper-parameter of a random forest [10]. Although $m = \lfloor \sqrt{N_{\text{feat}}} \rfloor$ is a reasonable starting point, careful tuning of m can often increase the performance of a random forest since for example small values of m lead to less correlated trees which is in general beneficial for a random forest [10]. At the same time, trees created with a small value for m may perform worse on average since the optimal split variable might not be available. Another hyper-parameter is the maximum depth of a tree, which can be controlled by the minimum number of samples needed to split a node or by the minimum number of samples required to define a node as leaf node. In [191] a detailed analysis of parameter tuning for random forests is given. The number of trees in a random forest k is usually not considered as a hyper-parameter. Even though [190] shows that under some circumstances a larger value of k results in a weaker performance, they argue that k should be set as large as computationally feasible. Often the number of trees can be used as a tunable parameter for the tradeoff between performance and computational power.

In addition to obvious ways to parallelize training as well as execution of a random forest and the scalability with the number of trees, random forests have the pleasant property that they can report how important each of the input features is for the classification task. The importance of a feature can be directly obtained for example from impurity measures that are calculated during training. Loosely speaking, the more often a feature was selected in a node of a decision tree for a split, the more important it is for the given task. A feature is chosen more often if it is effective in reducing uncertainty, so that important features are those that have a large mean decrease in the impurity measure. In [250], it is argued that this form of measuring the feature importance is biased. This view is supported by [179] and as an alternative and unbiased importance measure, the so-called *permutation importance* is recommended. The more robust importance measure was initially described by Breiman in 2001 [26] but today in most random forest libraries the default setting for variable importance computation is still such that the biased but faster and easier to calculate impurity reduction measure is used. The permutation importance measure works as follows: First, a random forest is trained normally and a baseline performance on a validation set is estimated. Within this data set, the values of the nth feature are permuted and the data is passed again through the random forest. The increase in miss-classification rate is then a measure of how important the nth feature is for the task. This method can also be applied to other machine learning algorithms since this approach makes no use of any specific features of a random forest – in contrast to importance measures, which make use of the reduction in impurity. Additional information on the different ways to compute the feature importance and possible biases of these methods can be found in [27], [179], [249].

2.1.2 *Artificial Neural Networks*

The currently most popular and most successful machine learning algorithms stem from the category of *Artificial Neural Networks (ANNs)*. With the phrase "artificial neural network", many different algorithms are subsumed which have in common that input data is passed through one or multiple layers, where each layer maps the input to a different output representation.

In addition to terms containing the phrase "neural network", a new description for these kind of algorithms emerged since about 2013: "deep learning". As the *deep* in *deep learning* is neither well defined nor is every type of neural network automatically deep in some sense, the currently more popular term is not used here.

Interestingly, almost all building blocks of modern neural networks were already known many years before the current hype-cycle started around 2010. In [79], it is well explained why neural networks currently gain so much attention. Firstly, large amounts of labeled data are nowadays either directly available or easy to collect. Secondly, the computational power accessible for machine learning algorithms increased tremendously with the development of modern GPUs. The third reason is that many companies expect disruptive changes in their market with the introduction of "intelligent" systems so that they invest large amounts of money in research and acquisition of start-ups that work in this field. Especially the first two points highlight why already well-known algorithms are studied again by a large community and why interest in neural networks decreased before the current hype-cycle started. With the data sets and computational resources available at that time it was simply not possible to reach the desired performance levels.

From the many types and layers of different neural networks that exist today, only the *Convolutional Neural Network (CNN)* and the *Long Short-Term Memory (LSTM)* network will be discussed here briefly.

2.1.2.1 *Convolutional Neural Networks*

The current interest of many scientific fields in machine learning and the popularity of neural networks in general is tightly linked to the performance boost granted by convolution layers within neural networks. Although the first steps towards convolutional neural networks were made in the 1960s with the works by Hubel and Wiesel [106], [107], it took about 30 more years until the full potential of convolution operations in neural networks became apparent. Hubel and Wiesel found out that different biological neurons in the visual cortex of a cat's brain fired only for certain simple structures like vertical or horizontal bars. That is, some

neurons are specialized in the detection of vertical bars whereas other neurons only respond to horizontal edges or bars. In other words, neurons in the visual cortex are orientation detectors [16], [17]. In addition to the specialization of individual neurons, they also discovered a column-like structure in which the neurons produce the full visual perception from a cascade of these highly specialized neurons. What is more, each neuron is only responsive for a certain *receptive field* [106], [108].

These ideas inspired the so-called convolution layers of modern neural networks used in computer vision tasks. They are designed so that each convolution kernel is specialized to one task (e.g. edge detection) and they are applied only to a certain region of the previous layer (receptive field). For some time, convolution kernels were designed by human experts and tailored to one specific task. In 1989, the research group around Yann LeCun showed that convolution kernels could be efficiently learned during training of the network using back propagation [133]. In contrast to the hand-made kernels, the network was then able to learn the relevant kernels itself. This finding – combined with efficient implementations of convolution operations on modern GPUs – are the foundation of today's neural networks. It is therefore worthwhile to look at the exact definition of a convolution operation in modern neural networks.

Historically, one of the first times the convolution operation appeared in mathematics was in 1768 in Leonhard Euler's book *Institutionum Calculi Integralis, vol. 2* when Euler studied the solution of certain partial differential equations [57]. Even though this operation was not called *convolution* at that time, its properties were analyzed in detail in the following years and centuries. The modern definition of a convolution between two functions f and g is given by

$$(f * g)(t) := \int_{-\infty}^{\infty} f(\tau) g(t - \tau) \mathrm{d}\tau \quad \text{for} \quad f, g : \mathbb{R} \to \mathbb{C} \tag{2.9}$$

$$(f * g)(n) := \sum_{k \in \mathbb{D}} f(k) g(n - k) \qquad \text{for} \quad f, g : \mathbb{D} \to \mathbb{C},\ \mathbb{D} \subset \mathbb{Z}. \tag{2.10}$$

One possible interpretation of the convolution operation is that a running mean of the function f is calculated where the weighting is defined by the function g. That is, a weighted sum of "echoes" (previous function values of f) is computed and the weight of each past value is defined by g.

For digital image processing, a formulation of the convolution operation on discrete and finite regions is needed. For a two-dimensional data structure with input I, *convolution kernel* K (sometimes also called *convolution filter*) and output S (sometimes also called *feature map*), the convolution operation is defined as [237], [245]

$$S_{ij} = (I * K)_{ij} := \sum_n \sum_m I(m, n) K(i - m, j - n). \tag{2.11}$$

The two sums run over the size of the kernel and border cases have to be treated separately, e.g. by padding, see also chapter 2 of [244] or chapter 5 of [12]. In non-border cases, the convolution operation on image like data can be described as follows. The kernel is placed with its center onto one input image pixel, each value of an input pixel within the receptive field of the kernel is multiplied with the appropriate kernel value, the result of all products is summed together and finally the kernel is shifted to the next position.

In machine learning usually batches of input data are considered and input images as well as the intermediate representations have not one but multiple *channels*. For example, typical color images consist of three channels: one channel for each of the base colors red, green and blue. Additionally, a stride parameter can be selected that defines how many pixels the convolution kernel moves forward after one output value was computed.

Combining all these extra parameters into the definition of the convolution operation, the value at position (h_o, w_o) of output b of the data batch in channel c_o is given by

$$S(b, h_o, w_o, c_o) = \sum_{h_k=0}^{H_k-1} \sum_{w_k=0}^{W_k-1} \sum_{c=0}^{C-1} I(b, s_h \cdot h_o + h_k, s_w \cdot w_o + w_k, c) K(h_k, w_k, c, c_o), \tag{2.12}$$

if the input I has in total C channels, the convolution kernel has size $H_k \times W_k$ and the strides of magnitude s_h and s_w, respectively, are used. This means that for one output channel all input channels of I along with all channels of the convolution kernel K are considered. The number of parameters of such a convolution kernel is given by

$$N_{\text{param}}(K) = H_k \cdot W_k \cdot C \cdot C_{\text{out}}, \tag{2.13}$$

where C_{out} is the number of output channels. A different wording would be that there is not one single convolution kernel with C_{out} channels but rather that there are C_{out} different convolution kernels (with one channel each).

Despite the fact that the use of convolution kernels can be biologically motivated, this kind of operation also has other upsides. To show this, a comparison with a *fully connected layer* (sometimes also called a *dense layer*) is useful. For an input $X \in \mathbb{R}^{N_{\text{feat}}}$, a fully connected layer produces an output $X' \in \mathbb{R}^m$ via a simple matrix product with weight matrix $W \in \mathbb{R}^{m \times N_{\text{feat}}}$ and addition of a bias vector $b \in \mathbb{R}^m$:

$$X' = W \cdot X + b. \tag{2.14}$$

The number of parameters involved in this operation is given by $N_{\text{feat}} \times m + m$. For an image with size 800×800 pixels and 3 RGB channels, the input vector

already has a size of $N_{\text{feat}} = 1\,920\,000$. So even for small values of m, the number of parameters in this layer becomes prohibitively large. In contrast, a convolution kernel of size $H_k = W_k = 3$ and $C_{\text{out}} = 64$ output channels has only $3 \times 3 \times 3 \times 64 = 1728$ parameters. Therefore, convolution layers require per definition much less parameters for the transformation of the input image to the next deeper layer. The number of parameters in an operation is not a valuable quantity on its own but rather the resulting performance in a given task has to be evaluated and a trade-off between accuracy and computational demand has to be made. Fortunately, convolution layers are extremely well suited for computer vision tasks and often accuracy even *increases* so that there is no trade-off but rather a win-win situation.

The reduced number of parameters stems of course from the fact that the same kernel is used for all different positions on the input. This implies two things. Firstly, there is a massive weight sharing (or put differently: a sparsity of connections) since input features at all positions are exposed to the same weights in the convolution kernel. This contrasts fully connected layers where each input is connected to each output by a unique weight. Secondly, translation invariance is automatically built into the network so that a certain feature can be detected anywhere (and even multiple times) in an input. This is a useful property since for example a classifier should be able to detect features and objects anywhere in the input irrespective of the exact location and a feature learned from one position in a training image might be useful at a different position in another image. Another advantage is that neighborhood relations are trivially captured so that for example a network has the chance to infer information about one pixel from the surrounding pixels.

Especially sharing of weights proofed to be a successful property of neural network layers. It is illustrative to make this weight sharing more explicit on one simple example. Consider an input $X \in \mathbb{R}^3$ that shall be mapped to an output $X' \in \mathbb{R}^2$. With a fully connected layer, the output is defined by the six parameters of the weight matrix W (ignoring the bias vector b):

$$X' = W \cdot X \quad \Leftrightarrow \quad \begin{pmatrix} x'_1 \\ x'_2 \end{pmatrix} = \begin{pmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix}. \tag{2.15}$$

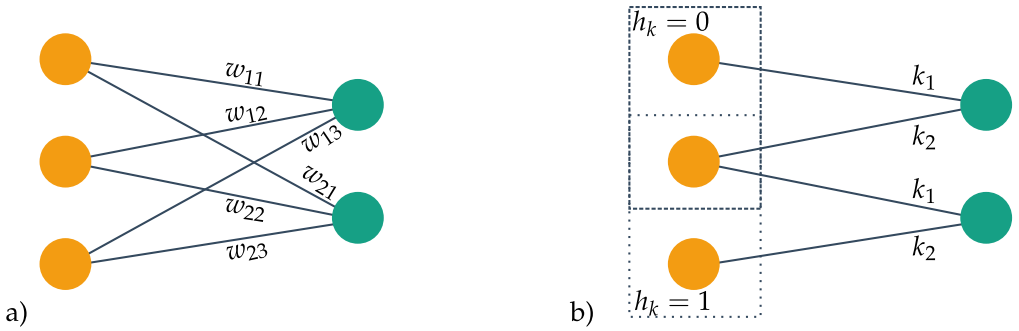


Figure 2.3: Comparison between a fully connected layer and a convolution layer for an example input feature vector with three entries. In a) a fully connected layer is used to map the input to a two-dimensional output. In b) a convolution kernel of height $H_k = 2$ is used and the two positions of the kernel at $h_k = 0$ and $h_k = 1$ are displayed (dashed and dotted rectangles).

Application of a convolution kernel with size $H_k = 2$, $W_k = 1$ and $C_{\text{out}} = 1$ output channel would result in (cf. Eq. (2.12))

$$\begin{aligned} X' = X * K \quad &\Leftrightarrow \quad X'(h_o) = \sum_{h_k=0}^{H_k-1} X(h_o + h_k) K(h_k, w_k = 1, c = 1, c_o = 1) \\ &\Leftrightarrow \quad \begin{pmatrix} x'_1 \\ x'_2 \end{pmatrix} = \begin{pmatrix} k_1 & k_2 & 0 \\ 0 & k_1 & k_2 \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} . \end{aligned} \tag{2.16}$$

Notice that the (empty) sums over the width of the convolution kernel and the input channels were left out for brevity. By writing both operations as a matrix-vector product, the two properties *weight sharing* and *sparsity in the connections* become apparent: Whereas in the fully connected layer there are six different parameters and each entry in the matrix is filled, the convolution kernel has only two different parameters and connections between the first and third input feature x_1 and x_3 are not modeled. In Fig. 2.3, the two layers are visualized again where the orange circles stand for the input features and the teal circles symbolize the two output features. The dashed and dotted rectangles in part b) of the figure indicate the two positions of the convolution kernel as it is shifted over the input vector.

To show that convolution layers are not only useful for image-like data, an example from the PointNet++ network [197], [198] is presented here. As the name of the network already suggests, PointNet++ is tailored for the usage of point cloud data as input. Point clouds are often stored as a list of points p_i where each point has d spatial coordinates and f extra features. The extra features depend on the sensor

that recorded the data. For radar sensors, the extra features could be for example the Doppler velocity or the measurement time. The list of points is usually not sorted and sorting of a one-dimensional list of d-dimensional points would just yield sub-optimal neighborhood relations anyway. Hence, convolution kernels with a size larger than 1×1 are not used. To illustrate how 1×1 convolution kernels can be employed, consider Fig. 2.4.

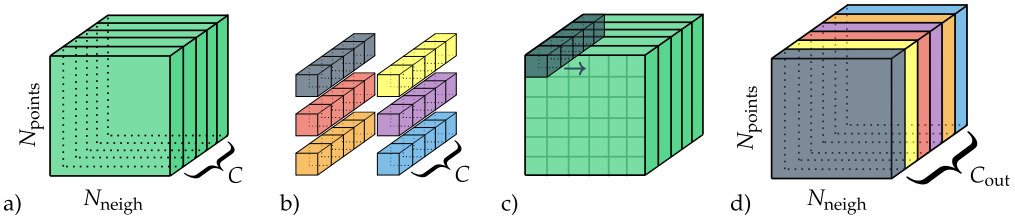


Figure 2.4: Illustration of a convolution. a) Input tensor of size $N_{\text{points}} \times N_{\text{neigh}} \times C$. b) The C_{out} different 1×1 convolution kernels. c) All convolution kernels are shifted individually over the input tensor, each creating one $N_{\text{points}} \times N_{\text{neigh}}$ output. d) Combination of all C_{out} convolution outputs results in the final $N_{\text{points}} \times N_{\text{neigh}} \times C_{\text{out}}$ tensor.

A data cube with dimensions $N_{\text{points}} \times N_{\text{neigh}} \times C$ is displayed there along with a hint how the convolution kernels are applied. The parameter N_{points} stands for the number of points in the point cloud and N_{neigh} is the number of neighborhood points which were collected by a previous function for each of the N_{points} points. Each of the in total $N_{\text{points}} \times N_{\text{neigh}}$ points is described by a C dimensional feature vector x_{mn}. Application of a convolution kernel of size 1×1 with C_{out} output channels results in a tensor of size $N_{\text{points}} \times N_{\text{neigh}} \times C_{\text{out}}$, where the same C_{out} convolutions are applied to each input point. Let $(x_{mn})_i$ be the ith entry of the C dimensional feature vector x_{mn} of the point at position (m, n) in the input matrix and let $(x'_{mn})_j$ be the jth entry of the generated output feature vector. Then

$$(x'_{mn})_j = \sum_{i=0}^{C-1} (x_{mn})_i \cdot k_{ji}, \tag{2.17}$$

where k_{ji} can be interpreted both as the ith entry of convolution kernel K_j ($j = 0, \ldots, C_{\text{out}} - 1$) or alternatively as entry (j, i) of a kernel matrix K. The former interpretation is aligned with the usual way how convolution operations are defined. The latter interpretation suggests that the 1×1 convolution can in fact be

interpreted as the application of a fully connected layer on the feature dimension so that each of the $N_{\text{points}} \times N_{\text{neigh}}$ points is transformed by the same matrix:

$$\begin{pmatrix} (x'_{mn})_0 \\ (x'_{mn})_1 \\ \vdots \\ (x'_{mn})_N \end{pmatrix} = \begin{pmatrix} k_{00} & k_{01} & \dots & k_{0M} \\ k_{10} & k_{11} & \dots & k_{1M} \\ \vdots & \vdots & \ddots & \vdots \\ k_{N0} & k_{N1} & \dots & k_{NM} \end{pmatrix} \cdot \begin{pmatrix} (x_{mn})_0 \\ (x_{mn})_1 \\ \vdots \\ (x_{mn})_M \end{pmatrix}. \tag{2.18}$$

The short hands $N := C_{\text{out}} - 1$ and $M := C - 1$ are used here for brevity. A 1×1 convolution can therefore be used as an efficient implementation of a *shared* weight matrix of a fully connected layer. This will become useful again later when semantic segmentation on radar data is discussed.

2.1.2.2 *Recurrent Neural Networks*

In the previous sections, *convolutional* and *fully connected* layers were mentioned as building blocks of neural networks. Networks built solely from these blocks (and of course built from activation functions, pooling layers, transposed convolution operations etc.) are summarized in the category of *feed-forward* neural networks. Defining property of these kind of neural networks is that there are no circular connections so that once an input passed a layer, no feedback is given to any other layer except for the following. In other words, feed-forward networks can be described as acyclic graphs [227]. The category of *Recurrent Neural Networks (RNNs)* differs from feed-forward networks in this aspect: Feedback loops are implemented so that the first element of a sequence of inputs alters the state of the network. Subsequent elements are then treated differently compared to the situation in which there was no preceding data.

Recurrent neural networks possess an internal state, which is updated with each new input and the output depends on the current internal state. Therefore, RNNs are popular in those fields of research where *sequential* data is used. Especially in tasks from the area of *Natural Language Processing (NLP)*, recurrent connections are inevitable, because for example preceding words highly influence both the likelihood of the following words and the semantic meaning of an expression.

In earlier days, temporal connections were not modeled via recurrent connections but for example, *Time Delay Neural Networks (TDNNs)* [261], [273]–[277] or hidden Markov models were used. The former were constructed with the goal in mind that relationships between events in time can be represented and that the network is invariant under translations in time [261]. In a TDNN, not only the input of the

current time t is considered, but also past data from times $t-1, t-2, \ldots, t-N$. Input data from the different time steps are put next to each other in a matrix-like structure and fixed size windows are shifted over this matrix to produce features from the respective time window. For example, in the original paper by Waibel [261], 15 different time steps are considered in the input where each input vector has 16 scalar features. Windows of size three are moved over this 15×16 input matrix and the same weight matrix of size 16×3 is used in each window to compute an output signal. This is then repeated eight times so that a new 13×8 intermediate matrix with 13 different time steps is created. The number of time steps is reduced from 15 to 13 in this case because a window of size three can only fit into 13 different positions in a frame with 15 places. The hidden layer is then transformed in the same way again (but now with a larger time window of five steps) before the final output is calculated. The structure of a basic TDNN is very similar to a CNN and in fact, in modern deep learning frameworks, TDNNs are actually implemented as convolution layers. Just as in CNNs, each unit in a TDNN has connections to a receptive field in the previous layer, where the receptive field now extends over time. The weight sharing that is used in TDNNs by using the same weight matrix for each of the different time windows in one layer just corresponds to using the same convolution kernel, which is shifted over the input.

One downside of TDNNs is that the "delays" in the time dimension are *manually* selected prior to the actual training and are held constant afterwards. The network is therefore restricted to only the presented snapshots in time and cannot learn on its own which points in time are useful for the task. Recurrent neural networks heal this shortcoming by introducing internal memory states, which are learned during training.

To make this more explicit, the equations for a vanilla RNN are presented in the following. To this end, let $\{x_1, x_2, \ldots, x_N\}$ be a sequence of N inputs with $x_i \in \mathbb{R}^n$ being some feature vector. Further, let $W_{hx} \in \mathbb{R}^{n \times m}$ and $W_{hh} \in \mathbb{R}^{m \times m}$ be trainable weight matrices and let $\{\hat{y}_1, \hat{y}_2, \ldots, \hat{y}_N\}$ with $\hat{y}_i \in \mathbb{R}^k$ be the output vector. Then the input and output are linked via the following set of equations:

$$h_t = \phi\left(W_{hx} \cdot x_t + W_{hh} \cdot h_{t-1}\right) \tag{2.19}$$

$$\hat{y}_t = h_t, \tag{2.20}$$

where ϕ is a non-linear activation function, e.g. the hyperbolic tangent function, a sigmoid or a *Rectified Linear Unit (ReLu)* [171]. A possible additional bias vector is neglected here for brevity. In Fig. 2.5a the cell is sketched. The internal state h_{t-1} is modified by the current input x_t so that in the next step $t+1$ the input x_{t+1} is altered by the then updated h_t. The hidden state is often initialized as a zero vector, so that the first input of a sequence is treated like in a simple feed-forward neural network. In this most simple RNN cell, the output is identical to the current hidden

state. An obvious extension of this recurrent layer would be to introduce another weight matrix W_{oh} which connects the hidden state h_t and the output $\hat{y}_t$.

Gradients for the training of usual feed-forward neural networks are computed using backpropagation, i.e. by repetitive application of the chain rule. In general, the same is done for RNNs, except that now care has to be taken when the derivatives for the weight matrices that appear in the recursive step are calculated. For example, when calculating the derivative of the error function with respect to the weights in W_{hh}, then it has to be considered that h_t does not only depend directly on W_{hh} but it also depends indirectly on W_{hh} through h_{t-1}. In practice, RNNs are often *unrolled* so that multiple copies of the same network are created and these are treated as normal feed-forward networks. Details about backpropagation in time can be found in [169], [209], [269].

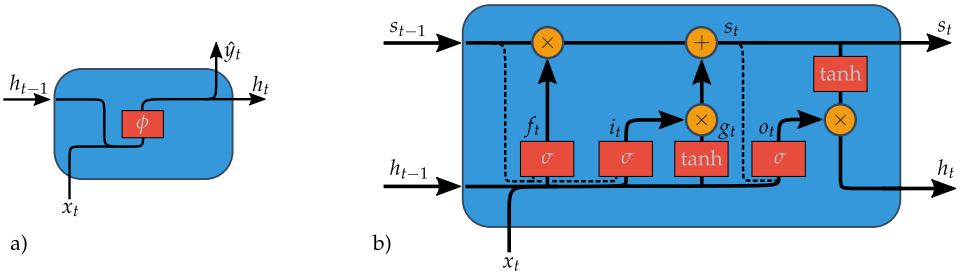


Figure 2.5: Connections within a recurrent neural network. In a) the most simple RNN cell is sketched, see Eq. (2.19). In b) an LSTM cell is drawn and peephole connections are illustrated by dashed lines. Based on Fig. 6 in [175].

For long input sequences – and hence long temporal dependencies of the gradients – the contributions from the different time steps are multiplied together. For contributions that are greater than one, the gradients become quickly unreasonably large and for derivatives smaller than one, the total gradient may vanish. Both effects are undesirable and stop the neural network from learning. The first scenario is often described as *exploding gradients* whereas the latter scenario is referred to as *vanishing gradients* [180]. Both effects were described the first time in detail in the diploma thesis of Hochreiter [100]. He showed that the gradients of past inputs are exponentially weighted in the calculation of the error function causing either too large values or values close to zero. Four years later, Hochreiter and his former professor Schmidhuber proposed in 1995 the *Long Short-Term Memory (LSTM)* network, which solves the problem of vanishing or exploding gradients [101]. The first version of LSTMs was trained with truncated gradients so that not the full backpropagation trough time scheme was used [102]. In the following years, different extensions were made to the original formulation of the LSTM, for example *forget gates* were introduced, peephole connections integrated and the output activation function was removed. Introduction of the forget gate allowed training with full

backpropagation trough time because they create an extra path in the gradient calculation which is not defined by the product of many small or many large values [85].

The recurrent units of an LSTM contain far more internal structure than simple RNNs. One widely used way to illustrate an LSTM cell is shown in Fig. 2.5b. In contrast to a simple RNN cell, an LSTM cell contains not only the hidden state h_t that is updated in each time step t, but also a cell state s_t that is updated via the already mentioned *input* and *forget* gates i_t and f_t. The nowadays most commonly used form of an LSTM is defined by the following set of equations [82]:

$$f_t = \sigma\left(W_{fx} \cdot x_t + W_{fh} \cdot h_{t-1} + V_{fs} \cdot s_{t-1}\right), \tag{2.21}$$
$$i_t = \sigma\left(W_{ix} \cdot x_t + W_{ih} \cdot h_{t-1} + V_{is} \cdot s_{t-1}\right), \tag{2.22}$$
$$g_t = \tanh\left(W_{gi} \cdot x_t + W_{gh} \cdot h_{t-1}\right), \tag{2.23}$$
$$s_t = f_t \odot s_{t-1} + i_t \odot g_t, \tag{2.24}$$
$$o_t = \sigma\left(W_{ox} \cdot x_t + W_{oh} \cdot h_{t-1} + V_{os} \cdot s_t\right), \tag{2.25}$$
$$h_t = o_t \odot \tanh(s_t). \tag{2.26}$$

The weight matrices W_{ab} follow the naming convention that the index a signals to which gate the matrix belongs (i for input, f for forget and o for output) and the index $b \in \{x, h, s\}$ describes to which vector this matrix is applied to. For example, the matrix W_{ih} is the weight matrix of the input gate, which acts on the hidden state vector h_{t-1}. The weight matrices V_{ab} follow the same naming convention but are called V instead of W to indicate that these matrices form the optional peephole connections (dashed lines in Fig. 2.5). For an input $x_t \in \mathbb{R}^N$ and a user-defined size H of the hidden state h_t, the matrices W_{ax} have size $H \times N$ and the matrices including the hidden state or the cell state W_{ah} or V_{as} are square with size $H \times H$.

The intuition behind the input and forget gates is that the network should decide on its own which information are carried over to the next time step and which information should be removed from the cell state to free up space. Both the input and forget gate take the current element of the sequence x_t as well as the previous hidden state h_{t-1} as input (and optionally also via peephole connections the cell state s_{t-1}) to a fully-connected layer with sigmoid activation function σ defined as

$$\sigma : \mathbb{R} \to \mathbb{R}, \quad y \mapsto \frac{1}{1 + \exp(-y)}. \tag{2.27}$$

As $\sigma(y) \in (0, 1)$, the resulting values of f_t and i_t can be interpreted as masks, indicating which values of another vector z are kept when the Hadamard products $f_t \odot z$ or $i_t \odot z$ are calculated. Therefore, $f_t \odot s_{t-1}$ describes which parts of the previous cell state are carried over to the next time step and which parts of s_{t-1} will be "forgotten". The output of the input gate i_t is not directly applied to the cell

state but rather multiplied with an intermediate representation g_t, which is again just the output of a fully-connected layer with the current x_t and previous hidden state h_{t-1} as input. The reasoning behind this extra step is that g_t should represent a set of new features, which are then masked by the input gate before they are added to the cell state. The output gate o_t controls the update of the hidden state $h_{t-1} \rightarrow h_t$ where h_t is also the output of the cell. Again, a peephole connection from the already updated cell state s_t can be included in the computation of o_t. With the same reasoning as before, the Hadamard product of this "output mask" is computed, but this time the cell state s_t is used as the underlying feature vector, which is masked out by o_t.

Multiple LSTM cells can be stacked behind each other so that similar to CNNs, features of different complexity can be extracted. Networks including or consisting solely of LSTM cells proofed to be very successful in NLP tasks and are widely used in commercial products like Amazon's speech assistant Alexa [160], Apple's QuickType keyboard [3] or in Google translate [280], [288].

A large zoo of LSTM variants was developed in the last years, including bi-directional LSTMs [83], [84], [236], the simpler *Gated Recurrent Unit (GRU)* cells [39], dropout regularization [70] or convolutional LSTMs [239]. An overview article about different (recurrent) deep learning architectures can be found in [227]. More recently, attention based CNNs gained popularity in NLP tasks [71], [258], since they tend to be easier trainable than LSTMs and highly optimized libraries for convolution operations can be borrowed from the field of computer vision so that attention based CNNs are more resource friendly.

2.2 UNSUPERVISED LEARNING

Machine learning algorithms from this category require as input only the data $\mathcal{X}$, i.e. no supervision in form of annotations $\mathcal{Y}$ is needed. However, it is rarely the case that for the same application algorithms from both the supervised and the unsupervised category are suitable so that the intrinsic feature that no labeled data are necessary for these classes of algorithms cannot be counted as an advantage over supervised algorithms.

Among others, unsupervised learning tackles the problems of clustering, anomaly detection and dimensionality reduction [151]. Clustering describes the process of finding frequent patterns in data and grouping elements with the same (or similar) pattern together. The grouping criteria can be very different but at the root, some distant measure in a metric space is used to describe neighborhood relations

between the data points. This neighborhood relation is then exploited to group nearby points and split far distant points from each other [74], [150].

The large variety of different clustering algorithms makes it difficult to sort the methods unambiguously into categories. A common attempt for the grouping of clustering algorithms is by looking at the decision criteria applied during clustering: *hierarchical* models use cuts in tree-like structures to group the data (e.g. the SLINK algorithm [240]), *distribution* models try to fit statistical distributions and obtain thereby a measure for "nearness" (e.g. Gaussian mixture models [218]), *subspace* models analyze data in different subspaces of the original feature space (e.g. sparse subspace clustering [63]) and *density* models build clusters by computing the density of different regions in feature space and grouping points in high density regions together (e.g. DBSCAN [65]). The whole Chapter 5 is devoted to describe clustering methods for radar data, so that no further details are presented here.

Anomaly detection is a complementary task to clustering in the sense that no grouping of common and reappearing patterns is sought but rather reporting of the rare cases that deviate from the usual data point is desired. The term “rare” does not necessarily refer to sparsity in the data because high activity in a bank account or high network traffic may also indicate anomalies from the average status. Depending on the application, density based methods like Isolation Forests [254], subspace projections [127] or autoencoders [19] can be used.

The aim of dimensionality reduction algorithms is to keep the expressiveness of the input data while the amount of information that describe the data is reduced. Popular techniques include *Principal Component Analysis (PCA)* [183], random projection (based on the Johnson–Lindenstrauss lemma [115]) and a common choice for visualization purposes is *t-SNE (t-Distributed Stochastic Neighbor Embedding)* [257]. The t-SNE method is a rather new algorithm to project high dimensional data down into two or three dimensions so that neighborhood relations can be visualized. The actual way in which this is accomplished is nicely explained in detail in the original publication [257]. Since t-SNE is used later on to visualize high dimensional feature vectors that appear in the classification tasks, a basic outline of the algorithm is presented here.

The idea is to compute an affinity measure p_{ij} in the high dimensional feature space and an affinity measure q_{ij} in the low dimensional feature space for all points i and j and to minimize a cost function that describes the similarity between the two resulting distributions. Because if the distributions of p_{ij} and q_{ij} are similar, then neighborhood relations in the high dimensional space are resembled appropriately in the low dimensional space. The affinity measure between the high dimensional points x_i and x_j is given by a symmetrized version of the conditional probabilities $p_{i|j}$ and $p_{j|i}$. The term $p_{j|i}$ describes the probability that point x_i would choose x_j as

its neighbor, "if neighbors were picked in proportion to their probability density under a Gaussian centered at x_i" [257]. The standard deviation of the Gaussian distribution is determined by a user-defined value for the *perplexity* $\mathrm{Perp}(\cdot)$ which in turn is defined via the Shannon-entropy H:

$$\mathrm{Perp}(P_i) = 2^{H(P_i)} \quad \text{with} \quad H(P_i) = -\sum_j p_{j|i} \log_2 p_{j|i}. \tag{2.28}$$

The standard deviation σ_i of the Gaussian around x_i is found by a binary search so that $\mathrm{Perp}(P_i)$ equals the user-defined value for the perplexity. The similarity measure in the low dimensional space (which is usually two- or three-dimensional) is defined via the neighborhood relations of the corresponding low dimensional points y_i and y_j. In contrast to the high dimensional space where the conditional probabilities are modeled via a Gaussian, a Student t-distribution is used to eliminate the crowding problem [257]. The cost function is defined by the Kullback-Leibler divergence of the probability distributions in the high dimensional space and the low dimensional space:

$$C = \sum_i \sum_j p_{ij} \log \frac{p_{ij}}{q_{ij}}. \tag{2.29}$$

The algorithm then starts with a single calculation of the probabilities p_{ij} and random initialization of the positions of the points in the low dimensional space y_i. For a user defined number of times the affinities in the low dimensional space q_{ij} are calculated, the gradient of the cost function with respect to the y_i is determined and finally the positions y_i are updated using gradient descent with momentum. In this way, the distributions of the affinities in the high and low dimensional space approach each other so that in the final configuration the probability that points x_i and x_j are neighbors in the high dimensional space is close to the probability that the corresponding low dimensional points y_i and y_j are neighbors.

Extensive discussion about various unsupervised learning algorithms – especially clustering methods – can be found in the three books [74], [150], [151].

2.3 SEMI-SUPERVISED LEARNING

Now that supervised and unsupervised learning methods were introduced, some motivation for the intermediate category *semi-supervised learning* is given in this section. Semi-supervised algorithms can be viewed from both ends of the spectrum. The first perspective is that semi-supervised learning algorithms use *un*labeled data to leverage the performance of a supervised learning algorithm. That is, the supervised model is the basic component and unlabeled data is only used as

an addition. The second perspective is that semi-supervised learning methods enrich unsupervised learning algorithms by a supervised component. The second perspective can be found later on in this work: In section 5.1.3, an unsupervised clustering algorithm will be extended by a supervised component.

The first perspective is backed by the fact that it is often much easier to collect (unlabeled) data than it is to manually annotate these samples. If information from the unlabeled data can be used to increase the performance of an initially purely supervised learning algorithm, then both time and money can be saved.

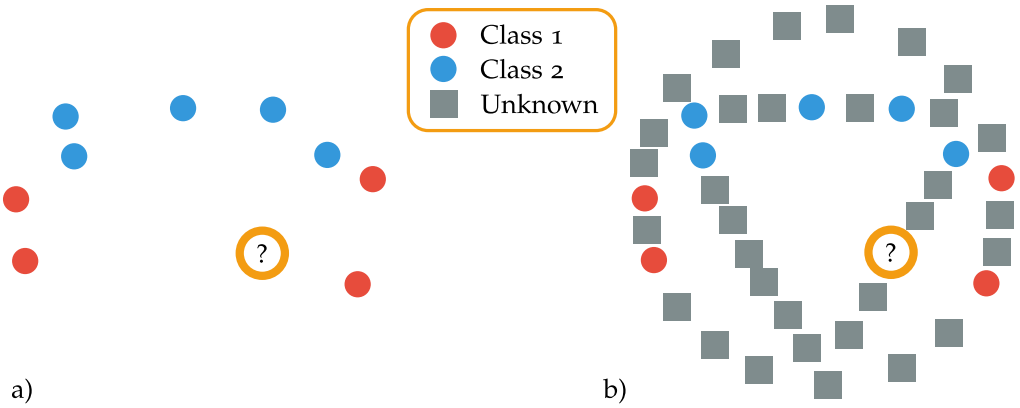


Figure 2.6: Example situation in which unlabeled data helps to identify the underlying structure in a data set. In a) only the labeled data are drawn. With the unlabeled data shown in b), more structure of the underlying distribution becomes visible. Inspired by Fig. 2.1 in [78].

A simple illustration why unlabeled data can increase the performance of a supervised learning algorithm is shown in Fig. 2.6. In part a) of the figure, only the labeled data are plotted. One could draw a simple linear decision boundary between the two classes (red and blue circles) which separates all samples from one class perfectly from samples of the other class. With such a decision function the point in question (orange circle) would be ascribed to class one (red). If, however, unlabeled data is considered as well, a different structure of the data distribution emerges. The labeled samples of class one lie all on a circle and the labeled samples of class two lie all on a triangular shape that is located inside the circle. With this additional information, the point in question is more likely to belong to the second class as the point lies on the triangle. Even though this is an artificially created example, it conveys the idea that the distribution of unlabeled data around labeled data can leverage the performance of a supervised learning algorithm if the underlying structure of the unlabeled data is correctly identified.

3

FUNDAMENTALS OF AUTOMOTIVE RADAR

The aim of this chapter is to give a general introduction to radar signal processing as it is done in the automotive field. Since this is a rather broad topic and as it is discussed in various theses, publications and books, only the necessary terminology is introduced as well as radar specific properties are presented. The focus lies on topics that reappear later in this thesis when algorithms are defined and evaluated based on how radar sensors perceive the world.

The term *radar* is an acronym for *ra*dio *d*etection *a*nd *r*anging although in earlier times it stood for the more application centered phrase *r*adio *a*ircraft *d*etection *a*nd *r*anging [22], [153], [186].

A basic description of a radar could be given as follows: A radar sensor is a device that emits electromagnetic waves as a primary signal and listens for returned echoes (the secondary signal) to detect objects as well as to infer properties about them. The kind of information that can be collected about a detected object depends highly on the used radar sensor and on the object itself. For example, some radar sensors can measure the position of an object in the plane of the radar sensor (radial distance and azimuth angle) and additionally its radial velocity whereas other sensors are well equipped to measure radial velocities but can hardly measure the position of an object [242].

The general working principle of a radar sensor can be found again in the mathematical description of the received energy P_r that a radar sensor measures after the transmit antennas emitted radiation with power P_t [242]:

$$P_r = \frac{P_t G_t A_e \sigma}{(4\pi)^2 R^4} = \underbrace{\frac{P_t}{4\pi R^2}}_{1} \cdot \underbrace{G_t}_{2} \cdot \underbrace{\frac{\sigma}{4\pi R^2}}_{3} \cdot \underbrace{A_e}_{4}. \tag{3.1}$$

This equation is often referred to as the *radar equation*. The first term describes the power density at a distance R from an isotropic antenna. The denominator $4\pi R^2$ accounts for the fact that the power of an isotropic emitter is uniformly distributed over the surface of sphere with radius R centered at the position of the emitter. Since radar antennas are usually directed, i.e. they do not emit the radiation

isotropically but focus the beam in some direction, the second term G_t accounts for this so-called *antenna gain* of the transmit antenna. The combination of terms one and two therefore describe the power density that arrives at an object located at a distance R from the sensor. The third term describes that the illuminated object reflects some of the incident power isotropically back. The *RCS (Radar Cross Section)* σ is a measure for the apparent size of the object. The RCS value is measured in m^2 and often reported in a logarithmic scale as dBsm. The RCS value is not directly linked to the physical size of an object but rather to the objects shape and material [206], [242]. An extended discussion about the RCS values of the semantic classes considered in this work will be given later in Chapter 4. The fourth and final part of the radar equation accounts for the additional gain of the receiver antenna. In this formulation of the radar equation, it is described by the effective area A_e of the receiving antenna.

Two important conclusions can be drawn from this equation:

1. The received power decreases with the fourth power of the distance. It is therefore an inherent property of a radar sensor that the detection performance decreases with increasing distance. This already suggests that also the classification accuracy of the algorithms discussed later in this work may decrease with the distance of the objects under consideration.

2. The radar cross section σ is a distance independent parameter that describes shape and material properties of the detected object. The most prominent example where geometry highly influences the detectability of an object are stealth aircraft. The design goal of these aircraft is to reduce the reflection of electromagnetic radiation in a specific frequency region, so that an incoming radar signal is not returned to the emitter. The RCS value can be used as extra information for the classification and segmentation algorithms and its explanatory power has to be analyzed.

Radar sensors are used in astronomy, meteorology, geology, in the defense industry, for aircraft tracking, as a level-sensor, for navigation and in the automotive industry. Depending on the application, different design choices have to be made. One important factor is the structure of the emitted signal for which – on the most basic level – two choices can be made. Pulse radars emit wave packages for a short period and listen afterwards for the echo signal while pausing the emission of further pulses. In contrast, continuous wave radars constantly emit radiation and mix the returned signal with the emitted signal.

For current automotive applications, *Frequency Modulated Continuous Wave (FMCW)* radars are used most frequently because they can measure range, angle and Doppler velocity at the same time, are inexpensive and can be manufactured small enough

which allows for a seamless integration in current car designs [20], [21], [43], [55], [56], [91], [242].

Experiments with radar for automotive applications started in the 1970s with the goal of reducing the number of traffic accidents and fatalities. Different frequency ranges were tested on various installation points on test vehicles: The companies VDO and SEL experimented with 10 GHz sensors mounted on the roof and 16 GHz sensors mounted on the front bumper of sedans, respectively [158]. With the advent of 35 GHz technology, smaller sensors could be constructed allowing for a more visually appealing integration in the vehicles. First blind spot detection sensors, i.e. short range radar sensor operating at 16 GHz were already presented in the 1970s and with the introduction of 77 GHz technology for long range radars, applications like radar based *Adaptive Cruise Control (ACC)* could be developed [156]–[158]. Mercedes-Benz introduced their radar-only ACC-system "distronic" in 1999 in the W220 S-class after other car companies like Toyota and Mitsubishi previously experimented with lidar based cruise control systems. Nowadays, most ACC-systems are radar based [4], [50], [189], [259] – even though they are sometimes supported by a front facing camera [5], [259] – and also emergency braking systems with pedestrian recognition rely on radar sensors [51].

Future trends in automotive radar concerning autonomous driving are extensively discussed in [263]. On the functional side, improved imaging capabilities, automotive *SAR (Synthetic Aperture Radar)* and better resolved Doppler signatures are mentioned by the authors. Imaging capabilities can be increased by exploiting *MIMO (Multiple Input Multiple Output)* technology, i.e. by using multiple antennas for transmitting and receiving signals. Multiple antennas leverage the angular resolution and at the same time reduce the ambiguities in the measured angle.

For SAR to work, the antenna has to be moved perpendicular to the direction of the beam, i.e. in the so-called *cross range* direction, and the precise position of the antenna has to be known at each point in time. Only static scenes can be properly resolved so that parking lot detection is a reasonable field of application for this technique [113], [132], [263].

Increased resolution of Doppler values makes it possible to detect *micro-Doppler signatures* of objects and hence allow for easier classification at an early stage in the signal processing chain. Micro-Doppler signatures are modulations of the base Doppler signal of an object caused by smaller moving parts [37]. For example, a walking pedestrian creates one base Doppler signal that results from the mean speed of the pedestrian's torso, and multiple periodic signals with smaller amplitude and different frequencies that stem from the movement of the pedestrian's arms and legs [38], [166], [253], [263].

Additionally, the authors in [263] state that better shape resolution is needed, especially for classification tasks. This statement will be explored later in more detail, when the impact of spatial properties on the classification results are explored.

3.1 INTRODUCTION TO RADAR SIGNAL PROCESSING

Modern automotive radar sensors use the FMCW technique to measure range, azimuth angle and radial velocity simultaneously. In this section, signal processing methods used to obtain these three features for multiple objects are introduced. Again, only the most relevant steps are presented here since this work does not focus on the improvement of signal processing in automotive radar but rather the output of the industry standard signal processing is used. For the interested reader, multiple references are given for further reading on the presented topics.

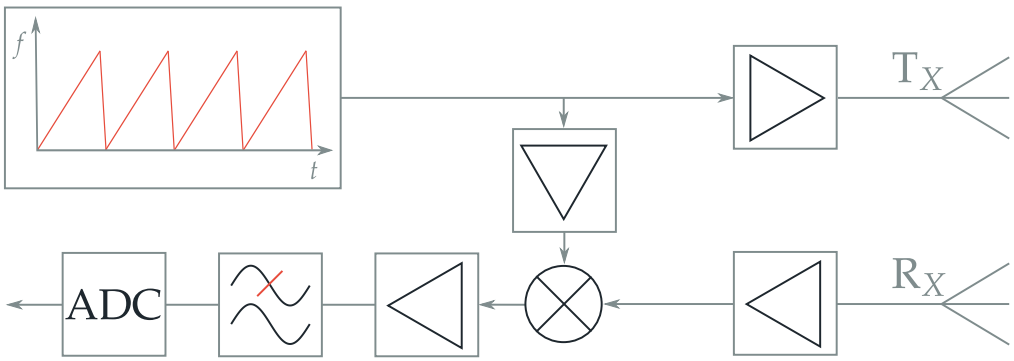


Figure 3.1: Simplified system diagram of an FMCW radar. Based on Fig. 2 in [270].

In Fig. 3.1, a system diagram of an FMCW radar with one transmit antenna and one receiver antenna is shown. In a MIMO setting, multiple transmit and receiver antennas are present. A chirp generator creates a signal that is amplified and directed to the transmitter antenna T_X. The receiver antenna listens for the returned echo signal and after amplification, this signal is mixed with the current signal. The newly created mixture signal is amplified and the high frequency parts are filtered out by a low-pass filter. An *ADC (Analog-to-Digital Converter)* transforms the filtered signal, which is then subject to further signal processing.

As the term FMCW radar already hints, the transmitted signal $s_{T_X}(t)$ is a continuous signal whose frequency is modulated periodically. Since the modulation time is usually much larger than the round trip time, the signal appears as a "continuous wave" for the illuminated objects. Linear frequency modulations of the form $f(t) \propto t$ are commonly used for automotive radar sensors although many approaches exist to overlay the signal with more complex patterns in order to detect or even avoid

interference with other radar sensors [136], [278], [284]. In this section, however, a simple linearly increasing frequency modulation of the form

$$f(t) = f_c + \frac{B}{T_c}t \tag{3.2}$$

is assumed because the relevant steps can be discussed quite as well with such a simple signal modulation. The carrier frequency f_c is the lowest frequency in each chirp of length T_c. The parameter B controls the bandwidth of the signal so that the highest frequency during the chirp is $f_c + B$. With the instantaneous phase [14], [187]

$$\Phi(t) = 2\pi \int_0^t f(\tilde{t})d\tilde{t} = 2\pi \left(f_c t + \frac{B}{2T_c}t^2 \right) + \phi_0, \tag{3.3}$$

the transmitted signal with amplitude A can be written as

$$s_{T_X}(t) = A\cos(\Phi(t)). \tag{3.4}$$

The echo signal obtained at the receiver antenna is identical to the emitted signal, except for the amplitude and a temporal shift:

$$s_{R_X}(t) = \tilde{A}\cos(\Phi(t-\tau)). \tag{3.5}$$

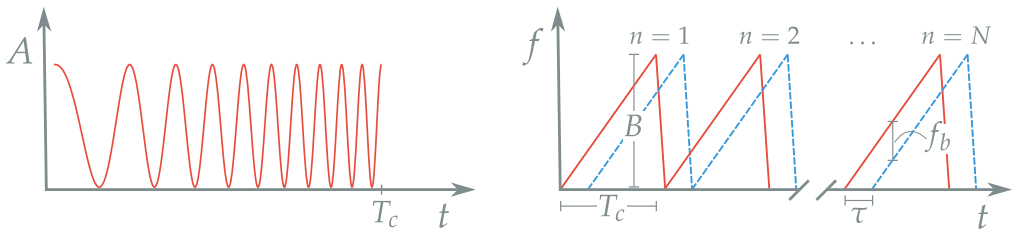


Figure 3.2: Left: Amplitude vs. time plot of the transmitted signal. Right: Frequency vs. time plot for the transmitted as well as for the received signal. The transmitted signal is drawn in red and the received signal is drawn in blue with dashed lines. Symbols are defined in the text. Based on Fig. 3 in [182].

Fig. 3.2 shows the transmitted signal in an amplitude – time plot, and in a frequency – time plot the transmitted (red) as well as the received signal (blue, dashed lines) are displayed. The index n enumerates the N chirps that make up one *scan*. In this work, a *scan* is defined as one complete measurement cycle during which radial distance, azimuth angle and radial velocity are measured for the objects in the field of view of the sensor.

3.1.1 *Range and Doppler Estimation*

The time delay τ between the transmitted and received signal is proportional to the distance of the object that caused the echo signal:

$$\tau = \frac{2(R + v_r t)}{c}. \tag{3.6}$$

The factor 2 stems from the fact that the signal has to cover twice the distance to the object: once towards the object and once back to the sensor. The term containing the radial velocity v_r takes the change of position of the object during measurement into account. In the special case of a static object, the radial velocity v_r is zero and hence the time delay reads

$$\tau' = \frac{2R}{c}. \tag{3.7}$$

The parameter c stands for the velocity of the emitted signal. For automotive radar, this is simply the speed of light in air. For radar applications in which the signal traverses the atmosphere, the propagation speed cannot be assumed constant but atmospheric effects have to be taken into account, especially when the signal traverses the ionosphere [126].

Mixing of the transmitted and received signal leads to

$$s_{IF}(t) = s_{T_X}(t) \times s_{R_X}(t) \propto \cos(\Phi(t)) \cdot \cos(\Phi(t - \tau)), \tag{3.8}$$

where the index IF stands for *intermediate frequency*.

Using $\cos(x) \cdot \cos(y) = \frac{1}{2}(\cos(x + y) + \cos(x - y))$, it becomes apparent that the mixed signal has two components. The first component is a high-frequency oscillation that is removed by the low-pass filter and is not further processed. The frequency of the second component is the so-called *beat* frequency f_b, i.e. it is the difference of the transmitted and received frequency.

In the academic case of one stationary object that causes a single echo signal, the radial distance to the object can be calculated using similarity statements about triangles: The ratio of the bandwidth B to the chirp duration T_c is the same as the ratio of the beat frequency f_b to the time delay $\tau = \tau'$:

$$\frac{B}{T_c} = \frac{f_b}{\tau} \quad \Leftrightarrow \quad \tau = \frac{f_b T_c}{B} \quad \Leftrightarrow \quad R = \frac{f_b}{B}\frac{cT_c}{2}. \tag{3.9}$$

Hence, by measuring the beat frequency f_b of the mixed signal, the distance to an object can be directly calculated from the known parameters of the radar sensor.

For real world applications with multiple moving objects, the processing steps are more complex. Inserting the value for the instantaneous phase from Eq. (3.3) into the expression of the mixed signal, Eq. (3.8), results in

$$s_{IF}(t) \propto \cos\left(2\pi\left[f_c\tau + \frac{B\tau}{T_c}t - \frac{B}{2T_c}\tau^2\right]\right) \tag{3.10}$$

$$\approx \cos\left(2\pi\left[f_c\tau' + \left(\frac{2f_c v_r}{c} + \frac{B\tau'}{T_c}\right)t\right]\right). \tag{3.11}$$

In the second line, small terms were neglected, for example terms with c^2 in the denominator. The mixed signal contains a phase term $\varphi = f_c\tau' = 2f_cR/c$ and a frequency f_{IF} that depends on both, the range R and the Doppler velocity v_r, and is given by $f_{IF} = 2f_cv_r/c + 2RB/(cT_c)$. At first sight it might seem tempting to use the intermediate frequency to extract information about the Doppler velocity. However, since f_{IF} depends also on the range R, the so-called range-Doppler coupling would have to be resolved. This can be done either by using a saw-tooth modulation of the frequency or by changing the steepness of the individual ramps during one scan. A different route is to work only with the phase term φ and to collect information about the change in the phase over N chirps. The intermediate frequency is then used to extract the range information, which later has to be compensated for the Doppler term. This last approach is most commonly used in automotive radar sensors, because for example the saw-tooth modulation pattern can only resolve the range-Doppler coupling in situations with a single return signal.

Two different time scales are considered in the following steps: the *fast-time* and the *slow-time*. The fast-time is the time scale of one chirp, i.e. of the order of T_c and the slow-time is the time scale of one scan with N chirps. Performing a Fourier transformation on $s_{IF}(t)$ yields the frequency spectrum of the mixed signal. If only a single echo from one object creates the mixed signal, the spectrum will contain a single peak around the frequency that corresponds to the distance of the object to the sensor (and due to the range-Doppler coupling also on its velocity). For multiple illuminated objects, the returned signal becomes more complex as each object contributes to the mixed signal with its own range dependent frequency. The Fourier transform then contains one peak for each object, and the frequencies at which the peaks appear can be directly converted to range information. Often, a *Fast Fourier Transform (FFT)* is used as an efficient algorithm to perform the Fourier transform on the digital signal. As this FFT results in information about the distance of the detected objects, it is often called *range-FFT*.

A second FFT is performed over the slow-time dimension to obtain the rate of change of the phase term over N chirps. For a stationary object, the phase remains constant since the distance R to the object stays the same. For a moving object, however, R changes over time causing a change in the phase φ. The peaks in the

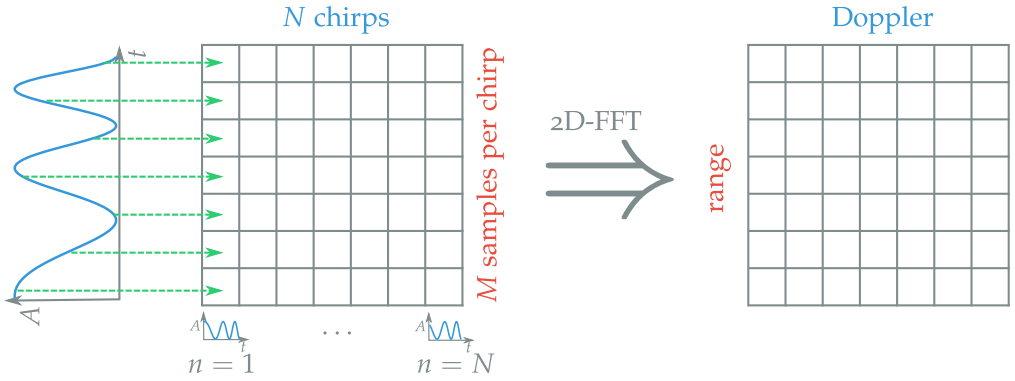


Figure 3.3: Creation of the range-Doppler-matrix by application of a 2D-FFT on the rows and columns of the input matrix. The input matrix consists of M sampled values from each of the N chirps.

frequency spectrum can therefore be used to calculate the radial velocity of the objects and hence this second FFT is often called *Doppler-FFT*.

If each chirp is sampled at M positions, an $M \times N$ matrix with the sampled values of all N chirps can be created. The range-FFT corresponds to an FFT along the M rows of the matrix, whereas the Doppler-FFT corresponds to an FFT along the N columns. The resulting matrix is called the *range-Doppler-matrix*. The generation of this matrix is sketched in Fig. 3.3.

A more detailed mathematical analysis of the presented processing steps can be found in [241], [270].

3.1.2 *Azimuth Angle Estimation*

Up to now, only a single antenna was considered and range as well as Doppler velocity information were extracted from the mixed signal. However, to obtain the *direction* of the objects that caused the return signals, a single stationary antenna is not sufficient. A mechanically rotating antenna as it is used in different radar applications (for example at a larger scale in airport surveillance) is not the optimal choice for automotive applications. For a good angular accuracy and resolution, the rotation has to be fast and constant in time. As a moving car causes various vibrations and disturbances, such a mechanical system is bound to be difficult to implement. Nevertheless, earlier automotive radar sensors relied on mechanically rotating antennas [42], [231], [271].

Instead, modern automotive FMCW radars use an array of antennas to estimate the direction of the returned signal. The basic idea is sketched in Fig. 3.4. The primary

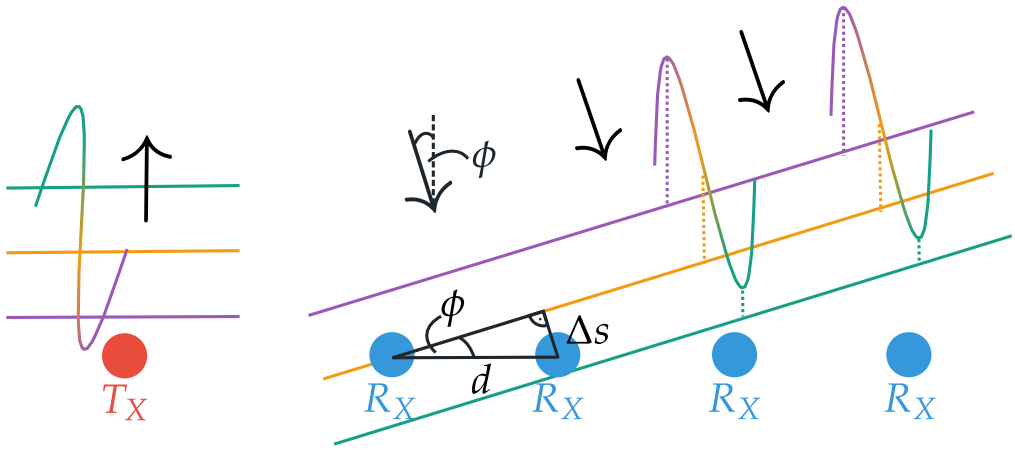


Figure 3.4: Sketch of the phase difference that occurs on the receiver antennas (R_X, blue circles) if an object is detected under an angle ϕ. Colored lines indicate areas with the same phase (wave fronts). Similar figures are in [120], [241].

signal is reflected off an object whose position in polar coordinates is given by (r, ϕ) relative to the sensor. For automotive radar, the distance to the object r is usually so large that the echo signal can be modeled as a plane wave. The path lengths to the receiver antennas is now a function of the angle ϕ. A wave front that hits the first antenna element is still a distance $\Delta s = d \sin(\phi)$ away from the second antenna. The parameter d describes the distance between two receiver antennas. Therefore, neighboring receiver antennas measure a phase difference of $\Delta\varphi = 2\pi\Delta s/\lambda$, where λ is the wavelength of the returned signal. Inserting the expression for Δs yields

$$\Delta\varphi = \frac{2\pi}{\lambda} d \sin(\phi). \tag{3.12}$$

For N_{R_X} receiver antennas, the relative phase differences to the first antenna are given by $[0, \Delta\varphi, 2\Delta\varphi, \ldots, (N_{R_X} - 1)\Delta\varphi]$. Just as for the range and Doppler information, a Fourier transformation can be used to extract the value of $\Delta\varphi$ and hence the angle information. In this case, however, the Fourier transform is taken over the N_{R_X} values of the receiver antennas. With four to six receiver antennas in current series automotive radars, this number is rather limited [20], [43]. As the angle resolution is proportional to the inverse number of sample points $1/N_{R_X}$, performing only a FFT would not yield desirable resolutions of about 1°.

Multiple methods exist to increase the angle resolution. In [120], [155] an extrapolation scheme of the received phase-difference signal is described. Linear combinations of measured values are created to artificially increase the number of antenna elements and hence increase the number of data points over which the FFT is performed. The increase in resolution is directly proportional to the number of predicted values and can therefore be scaled almost freely depending

on the available resources. However, this method assumes that the true signal is a sum of sin terms with additional white noise. If this assumption is violated, the performance of the algorithm degrades and alternative methods are needed.

Since the angular resolution scales with the number of antenna elements N_{R_X}, it is desirable to increase this number. Naïvely, one could use twice the number of receiver antennas to double the angular resolution. On the hardware side, the MIMO technique follows a more sophisticated route by using not only multiple receiver antennas but also multiple transmit antennas with orthogonal signals. Thereby, virtual antenna elements are created causing a better sampling of the return signal. In Fig. 3.5, a sketch of a MIMO setup is shown. Two transmit antennas (red circles) are separated by a distance $4d$ from each other and emit orthogonal signals that can be distinguished at the receiver side (blue filled circles). As discussed before, the phase differences at the four receiver antennas from the signal of the first transmit antenna is given by $[0, \Delta\varphi, 2\Delta\varphi, 3\Delta\varphi]$. The signal of the second transmit antenna has a relative phase difference of $4\Delta\varphi$ to the signal of the first antenna because of the spatial separation of $4d$. When these signals arrive at the receiver, they still have this relative phase difference to the first transmit antenna, so that the phase differences at the four receiver antennas are given by $[4\Delta\varphi, 5\Delta\varphi, 6\Delta\varphi, 7\Delta\varphi]$. Therefore, eight different values can be sampled in total, i.e. by the introduction of one additional transmit antenna four additional virtual antennas were created (blue unfilled circles in Fig. 3.5). In general, a combination of N_{T_X} transmit antennas and N_{R_X} receiver antennas can yield up to $N_{T_X} \cdot N_{R_X}$ sample points, provided that the signals of the transmit antennas are orthogonal. For the generation of orthogonal signals, *TDM (Time Division Multiplexing)*, *CDM (Code Division Multiplexing)* or other variants can be used. One of the downsides of using MIMO radars is that a more complex signal processing is needed. More elaborate details on MIMO techniques can be found in [36] and references therein.

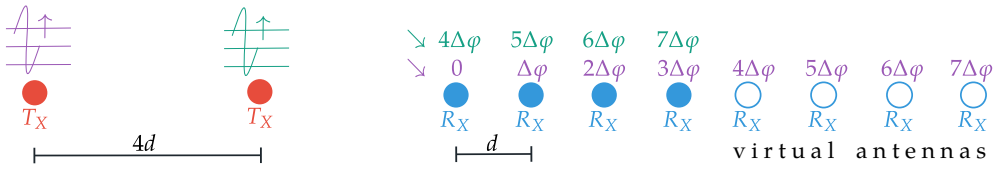


Figure 3.5: Sketch of a MIMO setup. Two transmit antennas in combination with four receiver antennas can emulate the same behavior as one transmit antenna and eight receiver antennas. Based on Fig. 5 in [200].

A different way to increase the angular resolution is the application of super-resolution algorithms like *MUSIC (Multiple Signal Classification)* and *ESPRIT (Estimation of Signal Parameter via Rotational Invariance Technique)* [59], [67], [214], [228], [263]. As these algorithms are computationally expensive, experiments have been

conducted to perform basic signal processing first and then apply high-resolution algorithms for selected regions of interest [53].

3.1.3 *Ambiguities and Resolution Limits in Range, Doppler and Angle*

Just as every other measuring device, also a radar sensor can resolve received signals only up to a certain precision. In addition to resolution limits, *ambiguities* have to be accounted for during signal processing and later on in the development of clustering and classification algorithms as well. These ambiguities stem from the inherent cyclic properties that come along with wave phenomena and processing steps that include FFTs. In this section, the resolution limits of an FMCW automotive radar are stated and examples for possible ambiguity effects are given. For each of the three measurement dimensions range, Doppler velocity and angle, the maximum resolution and the maximum ambiguous free value will be derived in a similar way. Two general statements are used repetitively:

1. The bin resolution of an FFT is given by the ratio of the sampling frequency f_{sample} of the signal and the number of sampling points N_{sample}.
2. Nyquist–Shannon sampling theorem: "If a function contains no frequencies higher than W, it is completely determined by giving its ordinates at a series of points spaced $1/(2W)$ seconds apart" [238]. In other words, a sampling rate of at least $2W$ is needed to capture all parts of a signal with maximum frequency W.

The range-FFT is performed with M samples per chirp, i.e. $f_{\text{sample}} = M/T_c$. Hence, the bin resolution is $1/T_c$. In Eq. (3.10) it was shown that the intermediate frequency is given by $f_{IF} \approx B\tau'/T_c$, where the range-Doppler coupling is neglected due to its small effect and $\tau' = 2R/c$. If two objects are separated by ΔR, they are separated in frequency by

$$\Delta f_{IF} = \frac{2B}{cT_c}\Delta R. \tag{3.13}$$

In order to resolve the two objects, the frequency difference Δf_{IF} has to be resolvable, i.e. the frequency difference has to be greater than one bin width: $\Delta f_{IF} > 1/T_c$. Combining these information yields

$$\Delta f_{IF} = \frac{2B}{cT_c}\Delta R > \frac{1}{T_c} \quad \Leftrightarrow \quad \Delta R > \frac{c}{2B}. \tag{3.14}$$

That being said, the larger the bandwidth B, the better the range resolution. To determine the maximum range of a radar sensor, the ADC sampling rate F_s has to be

considered. The Nyquist–Shannon sampling theorem demands that the sampling rate F_s has to be larger than twice the maximum frequency in the spectrum $f_{IF}^{\max}$. This leads to

$$F_s > 2f_{IF}^{\max} \quad \Leftrightarrow \quad F_s > 2\frac{2B}{cT_c}R_{\max} \quad \Leftrightarrow \quad R_{\max} < \frac{cT_c}{4B}F_s. \tag{3.15}$$

Increasing the ADC sampling rate or the duration of one chirp therefore allows for a greater detection range.

To estimate the Doppler velocity, an FFT over the slow-time is performed, i.e. the change in phase $\varphi = f_c\tau'$ over N chirps is considered (see previous section). In this case, the sample frequency is given by $1/T_c$ and hence the bin resolution is $1/(NT_c)$. The change in phase $\Delta\varphi$ is caused by a change in distance ΔR which in turn is caused by the movement of the object with radial velocity v_r. Hence, the phase changes during the time T_c by

$$\Delta\varphi = \frac{2f_c}{c}\Delta R = \frac{2f_c}{c}v_r T_c. \tag{3.16}$$

If two objects are separated by Δv_r, they can again only be resolved if the corresponding change in frequency is larger than the bin width. That is, a velocity difference of Δv_r can be resolved if

$$\Delta v_r > \frac{c}{f_c}\frac{1}{2NT_c}. \tag{3.17}$$

Therefore, the more chirps being performed (larger N), the higher the carrier frequency f_c or the longer the chirp duration T_c, the higher gets the Doppler resolution of the radar. This equation displays why systems with $f_c = 76\,\text{GHz}$ became prevalent instead of the previous 24 GHz radars. Just as it was done for the maximum range of a radar sensor (cf. Eq. (3.15)), the maximum unambiguous Doppler value of an FMCW radar is considered here as well. Again, the Nyquist-Shannon sampling theorem can be used to derive the expression for the maximum Doppler value $v_r^{\max}$. As stated before, the sample frequency is given by $1/T_c$ so the highest frequency in the spectrum

$$f_v^{\max} = 2f_c\frac{v_r^{\max}}{c} \tag{3.18}$$

is limited by $2f_v^{\max} < 1/T_c$. Plugging in the value of $f_v^{\max}$ and rearranging yields

$$v_r^{\max} < \frac{c}{4f_cT_c}. \tag{3.19}$$

A different (although related) way to arrive at this result is by noting that the change in phase between two consecutive samples has to be smaller than π in

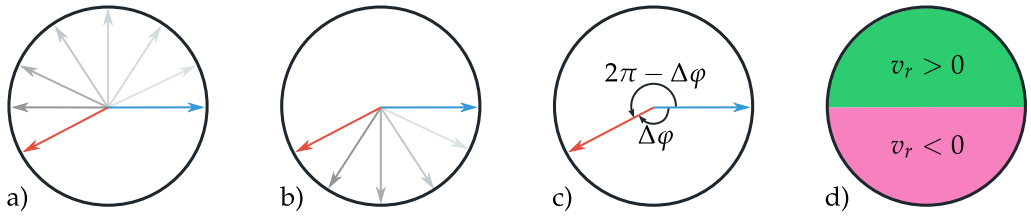


Figure 3.6: Blue arrow: time t_0, red arrow: time t_1. In a) and b), two different possible paths the phasor could have taken between t_0 and t_1 are displayed. In c), the two phase differences are plotted and d) shows regions with positive and negative Doppler velocity.

order to unambiguously resolve the velocity. In Fig. 3.6, the phasors (complex amplitudes) of the signal are depicted by rotating arrows. The blue and red arrows show the phasor at times t_0 and $t_1 > t_0$. In Fig. 3.6a and Fig. 3.6b, two different paths are depicted the phasor could have taken between t_0 and t_1. In this example, a positive radial velocity causes a counter-clockwise rotation of the phasor and a negative radial velocity results in a clockwise rotation. The clockwise path in Fig. 3.6b corresponds to a change in phase of $\Delta\varphi$, whereas the counter-clockwise path in Fig. 3.6a corresponds to a change in phase of $2\pi - \Delta\varphi$. If the change in phase is now larger than π between two samples, there is no way to know that this change in phase was not caused by a movement in the opposite direction. Therefore, even though the true path was the counter-clockwise one from Fig. 3.6a, a clockwise rotation of the phasor would be incorrectly assumed and hence a *negative* velocity reported for an object with a high *positive* velocity. This graphical argument is of course just a rephrasing of the Nyquist-Shannon sampling theorem. In Fig. 3.6d, areas that correspond to positive and negative Doppler velocities are highlighted. Mathematically, the result can be derived with this line of reasoning as follows

$$\begin{aligned} & \Delta\varphi && < \pi \\ \Leftrightarrow \quad & 2\pi T_c \cdot 2 f_c \frac{v_r^{\max}}{c} && < \pi \\ \Leftrightarrow \quad & v_r^{\max} && < \frac{c}{4 f_c T_c}, \end{aligned} \tag{3.20}$$

leading naturally to the same result as before. The important finding is that if the relative radial velocity between radar sensor and object is larger than $v_r^{\max}$, a velocity with the opposite sign is reported. In situations where the radar sensor and one detected object move in opposite directions, these limits can be reached quite easily so that especially oncoming traffic is affected by this. Without any counter measures, the unambiguous Doppler range of modern automotive FMCW radars lies in intervals of about $[-15; 15]$ m/s. One counter measure to reduce this unwanted effect is to integrate a mini-tracking algorithm into the sensor so that

from the possible values $v_{r,i} = v_r + i \cdot v_r^{\max}$, with $i \in \mathbb{Z}$, the correct ambiguity index i is chosen, i.e. that the correct number of times the maximum Doppler velocity is added or subtracted from the initially reported value. Even without an integrated mini-tracking such an ambiguity index (or a list of possible Doppler values) can be reported so that the correct Doppler value can be resolved in later processing steps.

With the same line of reasoning, statements for the maximum unambiguous angle can be made as well. From Eq. (3.12) and $\Delta\varphi < \pi$ it follows

$$\phi_{\max} < \arcsin\left(\frac{\lambda}{2d}\right). \tag{3.21}$$

This expression directly shows that an antenna spacing of $d = \lambda/2$ leads to the largest unambiguous angle of $\phi_{\max} = \pm 90°$.

For the angle resolution the bin resolution of the angle-FFT has to be considered, which is given by $1/N_{R_X}$ since one sample per receiver antenna can be obtained. The phase difference between two objects, separated by an angle $\Delta\phi$, is according to Eq. (3.12) given by

$$\Delta\varphi = \frac{2\pi}{\lambda} d(\sin(\phi + \Delta\phi) - sin(\phi)) \tag{3.22}$$

$$\approx \frac{2\pi}{\lambda} d \cos(\phi)\Delta\phi. \tag{3.23}$$

Again using the statement that the change in frequency has to be greater than one bin width in order to be resolvable, one arrives at

$$\Delta\phi > \frac{\lambda}{N_{R_X} \cdot d} \frac{1}{\cos\phi}. \tag{3.24}$$

Two statements are in order here:

1. In contrast to the range and Doppler resolution which are independent of the respective range or Doppler value, the angle resolution depends on the actual angle and is maximal for $\phi = 0$.
2. For an optimal antenna spacing of $d = \lambda/2$, the resolution is solely determined by the number of receiver antennas N_{R_X}. Doubling the number of receiver antennas (by using twice as many physical antennas or MIMO techniques) therefore doubles the angle resolution.

The statements about ambiguities and maximum resolution values are only true if basic signal processing as discussed here is applied. More advanced algorithms like MUSIC [228] or ESPRIT [214] of course yield a better resolution so that these basic formulas cannot be directly applied. However, especially the Doppler ambiguities

resulting possibly in sign changes of the reported values will be discussed again when the radar data set is introduced.

One further ambiguity in the angle is discussed here, because its manifestations become also apparent in the data set. If only one object causes a reflection of the transmitted signal, the mixed signal is a constant during the times t_0 and t_1 when both the transmitted and received signal are non-zero, see Fig. 3.7. The Fourier transformation of such a rectangle signal is given by the si-function $\mathrm{si}(x) = \sin(x)/x$. The maximum of the transformed signal lies at the beat frequency f_b. However, in addition to the central maximum, there are multiple smaller maxima located symmetrically around the central peak. Ideally, the range-bin b_k corresponding to the beat frequency f_b has its center directly at the maximum so that the value at the center of the adjacent range bins $b_{k\pm1}$ is zero. In this case, the sidelobes have no negative effect and only one target would be reported. If however the bins are not ideally distributed, a non-zero value is reported in neighboring bins. In cases where the original object is a strong reflector, the amplitude of the sidelobes could raise above the surrounding noise level so that in addition to the original object other objects are reported. This discussion is not limited to the range-FFT but applies generally. In our data set, this effect is most pronounced for the angle-FFT. In this case, a "ghost" object is created a few degrees away from the original object. These ghost objects are well visible in situation where the ego-vehicle follows a truck or other dynamic objects with high RCS value.

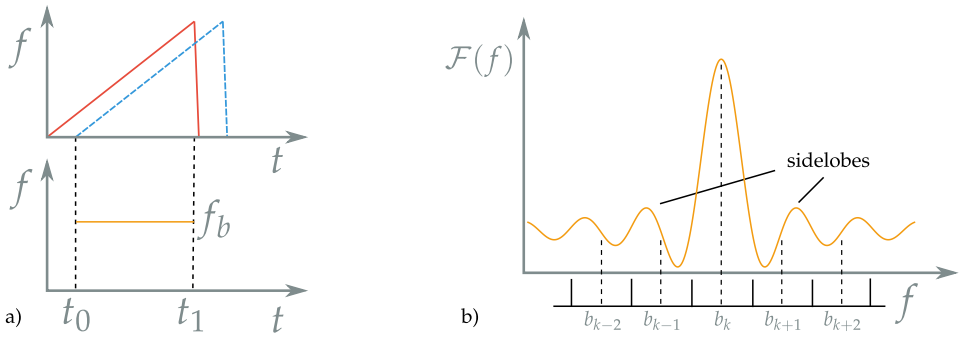


Figure 3.7: In a), the transmitted and received signals are plotted, similar to Fig. 3.2. The resulting mixed signal with constant frequency is drawn for the times where receive and transmit signal both exist. In b) the Fourier transformation of the mixed signal is plotted along with ideally located bins.

3.1.4 *Target Extraction*

Up to now, a transformation of the measured signals into a three-dimensional data cube was described. This procedure is sketched again in Fig. 3.8. However, not

every cell in this data cube is necessarily needed for further applications, so it is desirable to reduce the data stream that has to be transferred from the sensor to the control unit. Most of the cells in the data cube contain only noise so that an algorithm is needed to extract the signals that stem from real objects. A simple peak-finding algorithm is not sufficient because of varying noise levels and hence different criteria are needed.

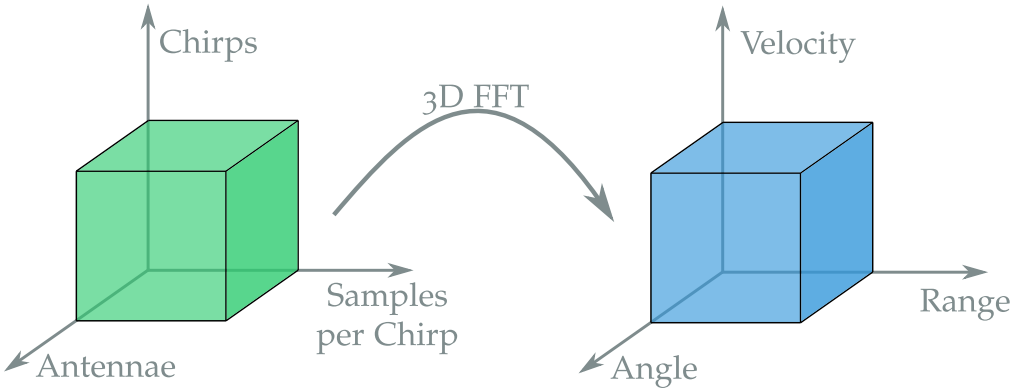


Figure 3.8: Transformation of the measured return signals to the three dimensions range, Doppler velocity and angle. The fast-time (samples on one chirp) corresponds to the range dimension, the slow-time (the number of chirps) corresponds to the Doppler velocity dimension and finally the number of antenna elements determines the angle. Based on figure in [7].

Most commonly, a *CFAR (Constant False Alarm Rate)* algorithm is used to identify only those peaks that belong to real objects [206]. The criteria for selecting these peaks is already hinted by the algorithm's name: the resulting output has a constant false alarm rate. A false alarm in this context means that a peak was identified as a true measurement even though it is only a result of noise.

Multiple variations of the CFAR algorithm exist that differ in computational complexity and accuracy. One basic implementation is the so-called *CA-CFAR (Cell Averaging Constant False Alarm Rate)* algorithm [210]. Here, one *Cell Under Test (CUT)* is picked and the noise level around this cell is estimated by the average of the surrounding cells. To improve the noise level estimation around broad peaks, so-called guard cells are introduced around the CUT. These guard cells do not contribute to the average so that the predicted noise level is not incorrectly increased around peaks. The algorithm declares a *target* in one CUT, if its value is both higher than the estimated noise level and higher than the neighboring cells.

The term *target* is used in this work to describe one single CFAR detection. In the radar community, the term *target* has often been used to describe one whole object, e.g. one car. This wording stems from earlier times, when radars could only resolve

one target on one object so that a distinction between the terms was not necessary. However, since modern automotive radars can measure several targets on one object, the wording has to be chosen more carefully. Synonyms for the word *target* (in the sense used in this work) encountered in other publications are *location*, *detection* or *measurement*.

The target extraction with CA-CFAR is sketched in Fig. 3.9 for a one-dimensional signal. The noise level around the current CUT is estimated over five surrounding cells, where two guard cells from each side around the CUT are excluded from the calculation. Since the value of the CUT is higher than the neighboring values and also higher than the noise level, a target is extracted from this cell.

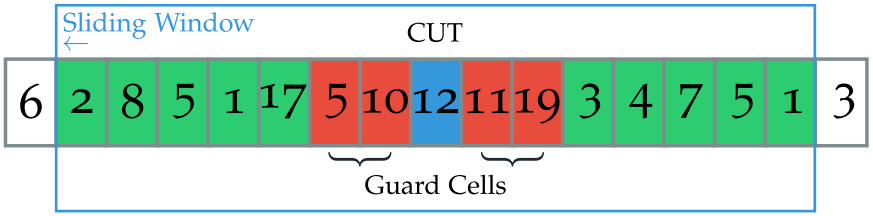


Figure 3.9: Sketch of the cell-averaging CFAR algorithm. The two guard cells (red) at each side of the cell under test (blue) are not considered during estimation of noise level (green cells). Based on Fig. 7.3 in [206].

A more advanced target extraction algorithm is called *OS-CFAR (Ordered Statistics Constant False Alarm Rate)* [15], [211]. This approach fixes one shortcoming of the simpler CA-CFAR method, namely that two nearby signals can no longer mask each other. To achieve this, all values in the current sliding window have to be sorted and the kth value in the sorted array is used for the estimation of the noise level. The performance gain of OS-CFAR in comparison with CA-CFAR comes at the cost of increased computational complexity, caused by the need to sort the values in each sliding window.

Multiple other variations like *Cell Averaging Statistic Hofele* CFAR [103] or *Trimmed Mean* CFAR exist [62], which all try to yield as many true positive targets as possible while keeping the computational costs low.

The extracted CFAR targets form the basis of all algorithms discussed in this work. Each target has the following four basic properties:

1. range r (in m),
2. azimuth angle ϕ (in rad),
3. Doppler velocity v_r (in m/s),
4. RCS value σ (in dBsm).

Additionally, each radar sensor reports the time of the scan that contains the target as well as variances for all measured quantities. The measured targets of one or multiple radar sensors form a spatio-temporal point cloud with two spatial dimensions. The spatial properties are reported relative to the sensor, i.e. in a *sensor coordinate system*. In the following sections, the transformation into other coordinate systems is discussed as well as the ego-motion compensation of the measured Doppler velocity.

3.2 COORDINATE SYSTEMS

In this thesis, data from a network of radar sensors is used for the development of classification algorithms. The data from each sensor is reported in the sensor's own polar coordinate system with the sensor located at $r = 0$. Common coordinate systems are needed into which the data from all sensors can be transformed so that following algorithms can work with the merged data. In addition to the sensor coordinate system (abbreviated as *sc*), a vehicle-fixed car coordinate system (*cc*) and a global coordinate system (*gc*) are used in this work.

The origin of the car coordinate system is fixed at the center of the rear axle. In accordance with ISO 8855, the x-axis of the car coordinate system points in the forward direction and is "substantially horizontal" [114]. The y-axis points to the left of the driving direction so that in a right-handed coordinate system the z-axis points upwards. While the test vehicle is moving, also the car coordinate system moves through the global coordinate system, whose origin is aligned with the car coordinate system's origin when the vehicle is started for the first time in a measurement cycle. Therefore, the global coordinate system cannot be used to identify the position of the vehicle uniquely on the globe, but it is rather a frame in which all measurements made during one single drive can be collected and uniquely positioned.

The position of the ego-vehicle in the global coordinate system is obtained from either the vehicle's odometry sensors or a *DGPS (Differential Global Positioning System)* that is built into the vehicle. Usually, data from a DGPS system is more accurate as less numerical integration has to be performed and hence drifts in the trajectory are smaller. However, high costs of a DGPS system prohibit the widespread usage in a large fleet of vehicles. For parts of the data set used in this work, a DGPS system was available. The rest of the data set was recorded with only odometry information. Both systems have in common that they yield at least the following information about the ego-vehicle's position and motion:

- Position in the global coordinate system: $x_{\text{car}}^{(gc)}, y_{\text{car}}^{(gc)}$

- Speed in driving direction: $v_{\text{car},x}$
- Yaw angle of the vehicle: γ. It is measured from the x-axis of the global coordinate system to the x-axis of the car coordinate system.
- Yaw rate: $\dot{\gamma}$. The center of rotation is assumed to be the center of the rear axle.

It is further assumed that the ego-vehicle does not drift so that $v_{\text{car},y} \equiv 0$. With this definition, the point $(x_{\text{car}}^{(gc)}, y_{\text{car}}^{(gc)})$ defines the origin of the car coordinate system as seen from the global coordinate system. This origin is of course time dependent due to the car's motion.

The positions of the radar sensors are defined by two coordinates $x_{\text{sens},i}^{(cc)}$ and $y_{\text{sens},i}^{(cc)}$. The orientation of the sensors is given by the yaw, pitch and roll angles $\phi_{\text{sens},i}$, $\theta_{\text{sens},i}$ and $\psi_{\text{sens},i}$. The index i enumerates the N_{sens} different radar sensors of a vehicle. In this work $N_{\text{sens}} \in \{4, 8\}$, i.e. either four or eight sensors are used on the test vehicles. A superscript in parentheses like (cc), (sc) or (gc) indicates the coordinate system the respective quantity belongs to.

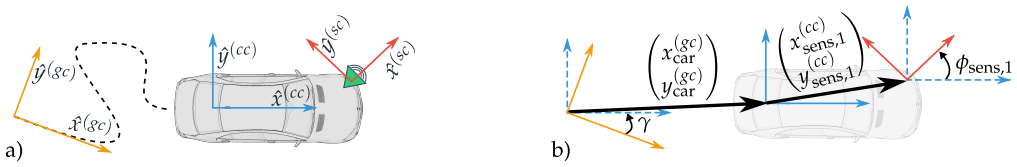


Figure 3.10: Location and orientation of the three coordinate systems. The black dashed line in a) symbolizes the trajectory of the test vehicle and the green triangle stands for one radar sensor. In b) the vectors connecting the origins of the coordinate systems as well as the angles γ and $\phi_{\text{sens},1}$ are displayed.

In Fig. 3.10, the relations of the three coordinate systems are depicted. The global coordinate system (orange) with its basis vectors $\hat{x}^{(gc)}$ and $\hat{y}^{(gc)}$ has its origin at the beginning of the trajectory (black dashed line). A hat over a symbol as in $\hat{x}$ indicates a unit vector. The angle γ increases in mathematical positive direction, i.e. from $\hat{x}^{(gc)}$ to $\hat{y}^{(gc)}$, so that in the displayed case $\gamma > 0$. Similarly, the angles $\phi_{\text{sens},i}$ are measured from $\hat{x}^{(cc)}$ towards $\hat{y}^{(cc)}$ so that in the example in Fig. 3.10b $\phi_{\text{sens},i} > 0$. The pitch and roll angles of the sensor alignment are not displayed here but are defined analogously.

Transformation of a measured target t with distance r and azimuth angle ϕ from polar coordinates to the Cartesian sensor coordinate system is done as follows:

$$\begin{aligned} x^{(sc)} &= r\cos(\phi) \\ y^{(sc)} &= r\sin(\phi). \end{aligned} \tag{3.25}$$

In general, the transformation from one coordinate system A into another coordinate system B can either be done by applying a rotation matrix R followed by the addition of a translation vector $\vec{d}$ or by applying one homogeneous transformation matrix ${}_BT_A$. The matrix T is a combination of R and $\vec{d}$:

$$ {}_BT_A = \begin{pmatrix} & & & d_x \\ & R & & d_y \\ & & & d_z \\ 0 & 0 & 0 & 1 \end{pmatrix}. \tag{3.26} $$

The rotation matrix R is the product of rotation matrices that describe rotations around the x-axis (roll angle δ), the y-axis (pitch angle β) and the z-axis (yaw angle α):

$$ R = R_z(\alpha)R_y(\beta)R_x(\delta) \qquad \text{with} \tag{3.27} $$

$$ R_x(\delta) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\delta & -\sin\delta \\ 0 & \sin\delta & \cos\delta \end{pmatrix}, \qquad R_y(\beta) = \begin{pmatrix} \cos\beta & 0 & \sin\beta \\ 0 & 1 & 0 \\ -\sin\beta & 0 & \cos\beta \end{pmatrix} $$

$$ R_z(\alpha) = \begin{pmatrix} \cos\alpha & -\sin\alpha & 0 \\ \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{3.28} $$

Using only one homogeneous transformation matrix has the upside that for multiple consecutive transformations only the product of two matrices has to be calculated which in turn is just another homogeneous transformation matrix.

For the concrete example of a transformation from sensor coordinates (of sensor i) to car coordinates, the angles α, β and δ are given by $\phi_{\text{sens},i}$, $\theta_{\text{sens},i}$ and $\psi_{\text{sens},i}$ and the translation vector $\vec{d}$ is built from the two entries $x^{(cc)}_{\text{sens},i}$ and $y^{(cc)}_{\text{sens},i}$ as the z displacement can be neglected in all our use cases:

$$ {}_{cc}T_{sc} = \begin{pmatrix} & & & x^{(cc)}_{\text{sens},i} \\ & R(\phi_{\text{sens},i}, \theta_{\text{sens},i}, \psi_{\text{sens},i}) & & y^{(cc)}_{\text{sens},i} \\ & & & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}. \tag{3.29} $$

Therefore, the coordinates of the target t in the car coordinate system are given by

$$\begin{pmatrix} x^{(cc)} & y^{(cc)} & z^{(cc)} & 1 \end{pmatrix}^T = {}_{cc}T_{sc} \begin{pmatrix} x^{(sc)} & y^{(sc)} & 0 & 1 \end{pmatrix}^T. \tag{3.30}$$

For the transformation from car coordinates to the global coordinate system, only the yaw angle γ between the global coordinate's x-axis and the x-axis of the car coordinate system is relevant for the rotation matrix, since pitch and roll motions of the vehicle are expected to be negligible. Hence, the rotation matrix simplifies to $R = R_z(\gamma)$ and the full homogeneous transformation matrix is given by

$${}_{gc}T_{cc} = \begin{pmatrix} & & & x_{\text{car}}^{(gc)} \\ & R_z(\gamma) & & y_{\text{car}}^{(gc)} \\ & & & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}. \tag{3.31}$$

To directly transform measurements from the sensor coordinate system to the global coordinate system, the product of the two intermediate transformations has to be considered:

$${}_{gc}T_{sc} = {}_{gc}T_{cc} \cdot {}_{cc}T_{sc}. \tag{3.32}$$

The inverse transformation, e.g. from car coordinates to sensor coordinates, is given by the inverse of the respective transformation matrix:

$${}_{sc}T_{cc} = {}_{cc}T_{sc}^{-1}, \qquad {}_{cc}T_{gc} = {}_{gc}T_{cc}^{-1} \quad \text{and} \qquad {}_{sc}T_{gc} = {}_{gc}T_{sc}^{-1}. \tag{3.33}$$

To display the need of transforming the measured targets into the different coordinate systems, Fig. 3.11 shows measurements obtained from four radar sensors in two coordinate systems. Measurements from a time window of 500 ms are shown. The test vehicle on which the sensors were mounted performs a right turn in this scenario. Colors indicate from which sensor the respective targets come from. In the left panel, the locations are shown in the car coordinate system and in the right panel, the targets are plotted in the global coordinate system. The visualization of all targets in a single sensor coordinate system is of course not useful, bears little information and is hence not displayed here. If the targets are plotted in the car coordinate system of their respective recording time, the structure of the scene becomes already visible. Since the ego vehicle is moving, the origin of the car coordinate system is not fixed and hence targets of the same static object are not positioned at the same coordinates. Therefore, the whole scene looks "smeared out". This is only patched if the global coordinate system is considered, in which

points from the same physical position are plotted at the same point, irrespective of the vehicle movement. Notice that the orientation of the global coordinate system differs in general from the current driving direction so that the point clouds shown in the right panel of Fig. 3.11 are rotated compared to the points in the left panel.

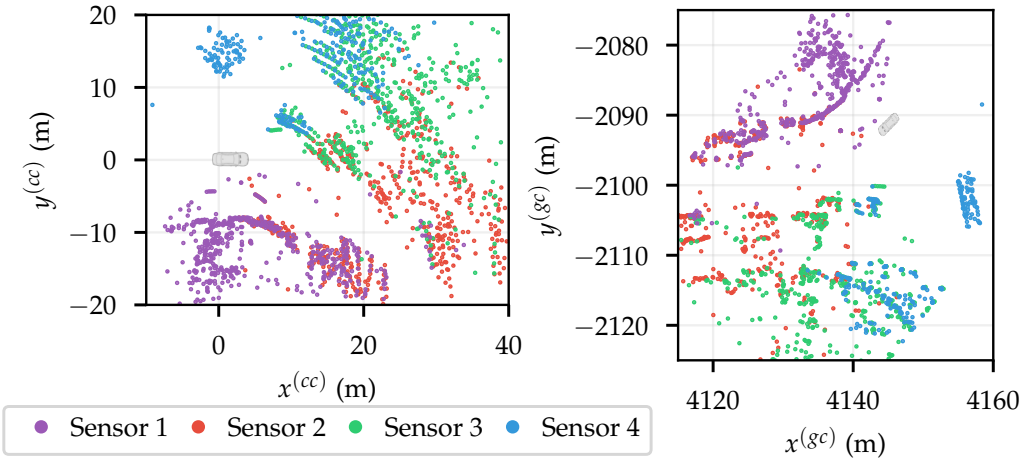


Figure 3.11: Radar targets from the four sensors are displayed in two different coordinate systems. Left: Car coordinate system. Right: Global coordinate system. Data accumulated over 500 ms is shown.

The downside of displaying the targets in a global coordinate system is that the numerical values of the coordinates are usually not very meaningful for a human observer and that the coordinate system can be arbitrarily rotated. A position in car coordinates is easier to understand, since with the ego vehicle at the origin, there is a clear fix point and the numerical values are in the order of 10 m to 100 m. For example, a coordinate tuple in world coordinates like $(4157, -2105)$ is harder to grasp and to put into context than the same position in the vehicle coordinate system: $(3, 15)$. The latter can be directly understood as "three meters in front of the vehicle and 15 meters to the left". Good orientation in the radar point cloud is important for the labeling task, which will be discussed later in more detail. Since the origin of the global coordinate system is chosen rather arbitrarily, it can also be translated to the current position of the ego vehicle. Thereby a new global coordinate system is created which has the pleasant feature that the positions of all targets are now relative to the ego vehicle and the absolute values of the coordinates are smaller and hence easier to grasp. Another way to think about this coordinate system is that the positions of the targets are transformed to a car coordinate system of a different point in time.

If not explicitly stated otherwise, the term car coordinates will in this work always mean that a car coordinate system of one point in time is chosen and all points

from different time steps are projected into this frame of reference. That is, the position of the ego vehicle at t_0 is selected and all targets that were measured at this time are inserted into this frame. Targets from time $t_1 \neq t_0$ are projected into this selected frame of reference and are *not* positioned at the coordinates of the car coordinate system of time t_1.

In such a car coordinate system the targets of a static object lie at one fixed spatial area whereas the measurements of a dynamic object show up as a trail, which is solely defined by the object's size and motion but is independent of the motion of the ego vehicle.

Measurements collected during 1 s of one example scene are displayed in the car coordinate frame of the earliest time stamp in Fig. 3.12. The extended trail of targets of the oncoming car (red points) is clearly visible as well as the fact that targets that belong to the two parked cars on the right are all located at the same spatial area even though the ego vehicle moves during the measurement time.

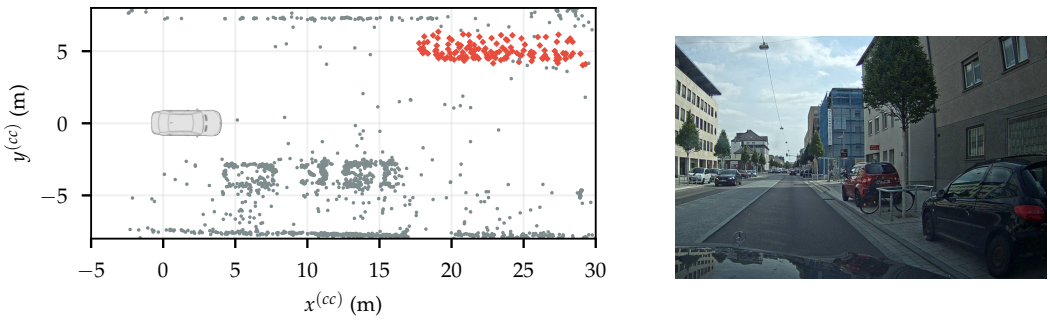


Figure 3.12: Left: Radar targets of an oncoming car (red) and the static infrastructure (grey). Right: Cropped camera image of the same scene. Data accumulated over 1 s is shown.

3.3 EGO-MOTION COMPENSATION

As discussed in section 3.1.1, the Doppler effect is used to estimate the relative radial velocity between radar sensor and object. In automotive applications, the radar sensor is usually not stationary so that the Doppler values reported from the sensor have to be compensated for the ego-motion. Only then the *Doppler over ground* is obtained, i.e. the Doppler velocity of the detected object relative to the stationary environment.

The Doppler over ground (the ego-motion compensated Doppler velocity) $\hat{v}_r$ can be calculated from the measured Doppler velocity v_r by subtracting the apparent radial velocity of static objects v_r^{static}:

$$\hat{v}_r = v_r - v_r^{\text{static}}. \tag{3.34}$$

The measured radial velocity of static objects is the projection of the velocity of the radar sensor in the radial direction:

$$v_r^{\text{static}} = -\Big(v_x^{\text{sens}} \cos(\phi + \phi_{\text{sens}}) + v_y^{\text{sens}} \sin(\phi + \phi_{\text{sens}})\Big). \tag{3.35}$$

The minus sign accounts for the fact that if a radar sensor is moved towards a static object, a Doppler velocity is measured that indicates a movement of the object *towards the sensor*, i.e. a movement in the *negative* radial direction. The quantities ϕ and ϕ_{sens} are just as before the azimuth angle of the target (in sensor coordinates) and the rotation of the sensor relative to the ego vehicle, respectively. Since the azimuth angle of the target ϕ enters the calculation and this is the measurement dimension that is usually worst resolved by a radar sensor, the error in ϕ propagates to the ego-motion compensated Doppler velocity so that the accuracy of the usually well resolved Doppler value degrades.

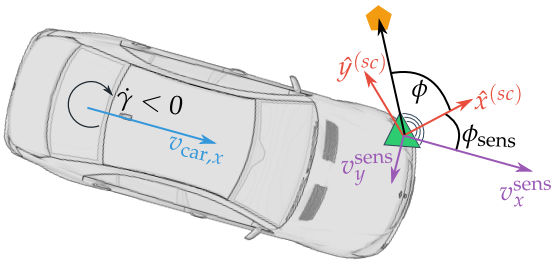


Figure 3.13: Sketch of the ego vehicle during a right turn where $\dot{\gamma} < 0$. Quantities used for the ego-motion compensation of the Doppler velocity of one measured target (orange pentagon) are shown.

Finally, the velocity components of the sensor v_x^{sens} and v_y^{sens} are determined by the speed of the ego vehicle, its yaw rate and the sensor position:

$$v_x^{\text{sens}} = v_{\text{car},x} - y_{\text{sens}}^{(cc)} \cdot \dot{\gamma} \tag{3.36}$$

$$v_y^{\text{sens}} = x_{\text{sens}}^{(cc)} \cdot \dot{\gamma}. \tag{3.37}$$

Again, the assumption is made that the car does not slip or drift so that $v_{\text{car},y} \equiv 0$. In Fig. 3.13, the relevant quantities are sketched. It should be noted that v_x^{sens} and

v_y^{sens} are components corresponding to the basis vectors $\hat{x}^{(cc)}$ and $\hat{y}^{(cc)}$, i.e. the components are relative to the car coordinate system:

$$v^{\text{sens}} = \begin{pmatrix} v_x^{\text{sens}} \\ v_y^{\text{sens}} \end{pmatrix} = v_x^{\text{sens}} \hat{x}^{(cc)} + v_y^{\text{sens}} \hat{y}^{(cc)}. \tag{3.38}$$

In the remainder of this work, the term *Doppler velocity* will usually be used as a synonym for the *ego-motion compensated Doppler velocity*, as the pure Doppler value is often not meaningful. However, in formulas there will always be a distinction between the measured Doppler v_r and the ego-motion compensated Doppler velocity $\hat{v}_r$ to avoid any confusion.

3.4 CLUTTER

Depending on the application, the term *clutter* is used in the radar community for very different effects or objects [242]. The common basis is that clutter measurements are undesired information for the current application that can either result in false positives or overlay and thereby alter the true signal. For example, in aircraft detection, precipitation, ground reflections or even birds are considered as clutter. In maritime applications, sea waves are a large source of clutter measurements, which can be hardly accounted for due to their variable sizes and velocities. In contrast, a weather surveillance radar would certainly not consider precipitation as clutter and in the context of driver assistance systems, ground reflections are not per se clutter as they can be used e.g. for road course estimation [75], [220] and larger birds or other animals might be relevant for precrash systems.

As this work deals with the semantic classification of moving objects, one necessary condition for a target to be declared as clutter is that it has a non-zero Doppler velocity $\hat{v}_r$. Three different groups of this type of clutter can be identified:

1. **Mirrored objects.** Just as there are mirrors for electromagnetic waves in the visible range, there are also objects that work as a mirror for electromagnetic waves around 77 GHz. Metallic surfaces of fences, gates or guardrails can create a mirrored object of a truly existing object that appears at a different range or angle in the radar point cloud. Two features make it difficult to detect mirrored objects even with the help of a documentary camera. Firstly, mirrors for radar waves look very different from commonly known mirrors that work in the visible range of electromagnetic radiation and secondly, multi path propagation of the returned signal makes it difficult to track the exact way the wave took. Since mirrored objects have very similar properties in all

measurement dimensions (r, ϕ, v_r and σ), their distinction from real objects is a challenging task. Information about the polarization of the returned signals may help to distinguish mirrored objects from true objects, since for example a circular polarized wave changes its polarization direction each time it is reflected.

2. **Ambiguities.** As discussed in section 3.1.3, ambiguities in angle can cause targets to be reported at a wrong value of ϕ so that either parts of an object are displayed at the wrong position or the whole object is moved to a different angle. This type of clutter objects appear at a fixed angle relative to a real object if they stem from sidelobe measurements or if the *TDM (Time Division Multiplexing)* *MIMO (Multiple Input Multiple Output)* corrections are incorrectly calibrated. Ambiguities in Doppler may result in attributing a non-zero Doppler velocity to in fact static targets.

3. **Noise.** Each measurement device suffers from some amount of noise in each of the measurement dimensions. A radar sensor is of course no exception to this so that random fluctuation in the noise may lift one CUT over the adaptive noise threshold of the CFAR algorithm. Processing errors in the sensor may be another source of falsely created targets. Thirdly, misalignment in time of odometry information and sensor data or simply erroneous odometry information can result in an incorrect ego-motion compensation which in turn causes that static targets are attributed with a non-zero Doppler velocity.

Clutter measurements have to be dealt with when classification tasks with radar are performed. Firstly, because radar sensors are in contrast to cameras and lidars rather noisy so that clutter appears rather frequently and hence classification algorithms should not confuse clutter with a real object. Secondly, algorithms further down in the processing chain of a driver assistance system, e.g. tracking algorithms, can benefit enormously if they can sort out clutter measurements from the start. For example, a nearby clutter object with apparent velocity towards the ego vehicle would get unnecessary much attention from the system if it were not known that the targets stem in fact from a ghost object.

Therefore, clutter targets are considered explicitly in this work from the labeling stage onward.

4

DATA RECORDING, ANNOTATION AND MEASUREMENT STATISTICS

One often overlooked difficulty when working with machine learning algorithms is the whole area of data management. This broad topic includes raw data acquisition, development of a labeling strategy, a label guide as well as suitable tooling, the actual labeling, quality control and finally refinement steps in the complete work process.

For various machine learning tasks, publicly available data sets exists so that researches can start right away with the design and implementation of their own algorithms. For image data, large data sets are for example ImageNet with over 14 million annotated images [54], CIFAR-10 and CIFAR-100 which consist both of the same 60 000 images split into either ten or 100 classes [128], or Microsoft's COCO with 2.5 million annotated images [141]. Especially for automotive applications, the cityscapes data set with camera data from 50 different cities [44], the KITTI vision benchmark data set with recordings of both a stereo camera and a lidar sensor [72] or the nuScenes data set [31] proofed to be valuable starting points for the development of new machine learning algorithms. More and more companies working on autonomous driving start to release their own labeled data sets to enable the community to contribute to the field [34], [122], [268]. Unfortunately, currently no publicly available data set exists with which classification of dynamic objects based solely on radar sensors would be possible. In all available automotive data sets, radar sensors were either not included in the sensor set up [34], [44], [122], [268] or as it is the case in the nuScenes data set, the radar data are too sparse to allow usage in semantic tasks. A more detailed analysis of currently published data sets can be found in the overview article [117].

It is not surprising that radar data were often neglected in these data sets and that therefore only few works deal with classification of radar data. The first reason is that radar data is both sparser and noisier than for example data from a lidar sensor and hence semantic information is much more difficult to extract. Historically, radar sensors were used for tasks in which semantic information was only necessary at the basic decision *noise* vs. *not noise* and due to the sparseness in the data, extraction of more semantic information was just impossible. Secondly, publicly available

radar sensors often return the measured data on an *object* level, i.e. the whole object formation from the individual targets is performed on the sensor and the researcher has no means of getting access to deeper data levels. Manufacturers often do not want to make deeper processing layers available for research so that especially universities struggle to include radar sensors in their work [255]. Thirdly, labeling of radar data on target level is a difficult and time consuming task with which only very few companies have experience so that both a lot of money and time have to be invested to obtain annotated data.

For this thesis, a new data set was recorded and annotated so that machine learning methods could be developed for the task of classification and semantic (instance) segmentation of dynamic objects based solely on radar data. Even though most of the data presented here were recorded for this thesis, the data were internally made available for other publications [87], [222]–[225].

This chapter is devoted to the whole data management topic and detailed information is given on how the data were recorded, how the labeling took place, which artifacts and labeling errors occurred and finally statistical information about the collected data is presented. The last point includes an in-depth analysis of the measured RCS values of various semantic classes.

The contributions of this chapter are hence as follows:

- description of the sensor setup and the recording method,
- a detailed description of the collected data used in the remainder of this work,
- analysis of the RCS values of different classes of road users,
- information on how radar data can be annotated so that the labels are useful for classification and semantic segmentation tasks,
- a comparison of radar characteristics of three different bicycle types.

4.1 SENSOR SETUP AND RECORDING PROCEDURE

The data used in this work stem from a single vehicle that is equipped with four industry standard automotive radar sensors, which operate at a frequency of 77 GHz. The sensors are mounted on the two front corners as well as on the left and right side of the car. The two sensors at the front corner are tilted 25° outwards while the sensors on the two sides are oriented perpendicular to the driving direction. The radar sensors operate in two different modes which are triggered alternately: the far-range mode and the near-range mode. In the far-range mode, targets can be detected up to a distance of $r_{\text{max}} = 200\,\text{m}$ whereas in the

near-range mode $r_{\max} = 100\,\text{m}$. However, the far-range mode is restricted to a much narrower field of view than the near-range mode. In this work, only data from the near-range mode is considered since data from the far-range mode often showed inexplicable artifacts that degrade the overall quality. The accuracy in radial direction is given by $\Delta_r = 0.15\,\text{m}$ and the accuracy in the Doppler velocity is $\Delta_v = 0.1\,\text{km/h}$. Depending on the azimuth angle ϕ, the angular accuracy lies between $\Delta_\phi = 0.5°$ and $\Delta_\phi = 2°$ with higher accuracy at smaller angles.

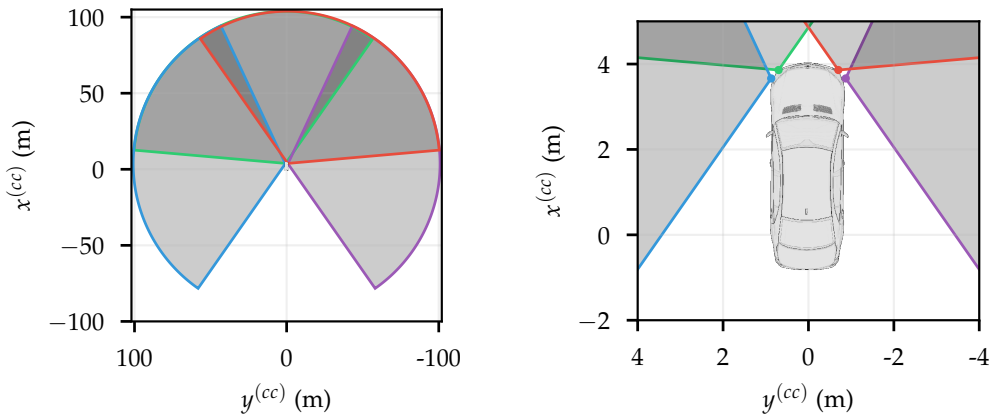


Figure 4.1: Field of view and positions of the sensors on the test vehicle.

In Fig. 4.1, the positions as well the field of view of the radar sensors are displayed. Only the field of view in the near-range mode of the sensors is depicted here. The field of view of each sensor is represented by a unique color of the arc's outline. The arcs representing the sensor's field of view are filled with 80 % transparent black color so that regions covered by multiple sensors are displayed in a darker color (overlapping field of view). A radar sensor does not work equally well throughout its whole field of view since for example the angular accuracy decreases with increasing angle and targets with small RCS values are more likely to be detected around $\phi \approx 0$ and less likely at the borders of the field of view. Therefore, the arcs shown here are simplifications but since the figure is meant to give an impression from which area around the car data can be measured, this simplification is justifiable here. The data sheet lists the field of view of the sensor as $\pm 60°$. Interestingly, the sensors frequently report measurements from angles $\phi > 60°$. If the sensor field of view had been estimated from the actual recorded data by choosing those two boundary angles for which still targets are measured throughout the whole range region, then a value of about $\pm 72°$ would have to be reported. The exact reason why some targets are reported with azimuth angles larger than 60° even though the sensor is not specified for this region is not clear. One possibility is that for

these targets a wrong azimuth angle hypothesis was chosen so that the target lies in fact at a different angular position, see Section 3.1.3.

In addition to the radar data, also images from a documentary camera installed behind the windscreen were recorded. For the calculation of the ego-motion compensated Doppler velocity and for the transformation of the measured targets into a global coordinate system (see also Section 3.2), the vehicle's current position and velocity at each point in time has to be known. For some recordings, the odometry information was obtained from on-board sensors like a speedometer and a gyroscope, and for other recordings, a DGPS system was used to estimate a precise pose of the vehicle. The odometry data were stored along with the radar data and the camera images where each odometry measurement consists of a timestamp, the x and y position of the vehicle in the global coordinate system, the yaw-angle γ, the vehicle's velocity in x-direction $v_{\text{car},x}$ and its yaw rate $\dot{\gamma}$.

Most data were recorded in Ulm, Germany, and the surrounding area. To ease data handling and labeling, the recordings were split into sequences with lengths between two to five minutes. Almost all recordings were made in everyday traffic except for a few sequences that were recorded at the Daimler research campus in Ulm to collect more data from pedestrians.

The four radar sensors of the test vehicle fire independently of each other with a non-constant time interval between two consecutive scans. Each near-range scan is followed by a far-range scan and vice versa. The sensor cycle time is not constant and a histogram of the cycle times is drawn in Fig. 4.2 for one sensor, where the sensor cycle time is used as the time difference between two near-range scans. The distributions for the other three sensors are very similar and hence omitted here for brevity. The figure shows that no more than $\Delta t = 80\,\text{ms}$ lie between two consecutive near-range scans with an average cycle time of $\overline{\Delta t} = 59.2\,\text{s}$. The fact that the radar sensors fire asynchronously of each other with a non-constant cycle will be important in the labeling stage and will be discussed in more detail in the next section.

In Fig. 4.3, two histograms from two different sensors are displayed which show the number of targets measured per scan. In each figure, two different states of the ego-vehicle were considered: a moving ego-vehicle and a stationary ego-vehicle. First, consider the data obtained from the sensor mounted at the front left corner in Fig. 4.3a. It can be clearly seen that the motion of the ego-vehicle – and hence the motion of the sensor – influences the number of measured targets. If the sensor is stationary, on average fewer targets are measured.

The distributions of the number of measured targets resemble in both cases a Gaussian. To underline this, normal distributions were fitted to the empirical data resulting in the blue and dark-blue lines in the figure. Notice that the y-axis is

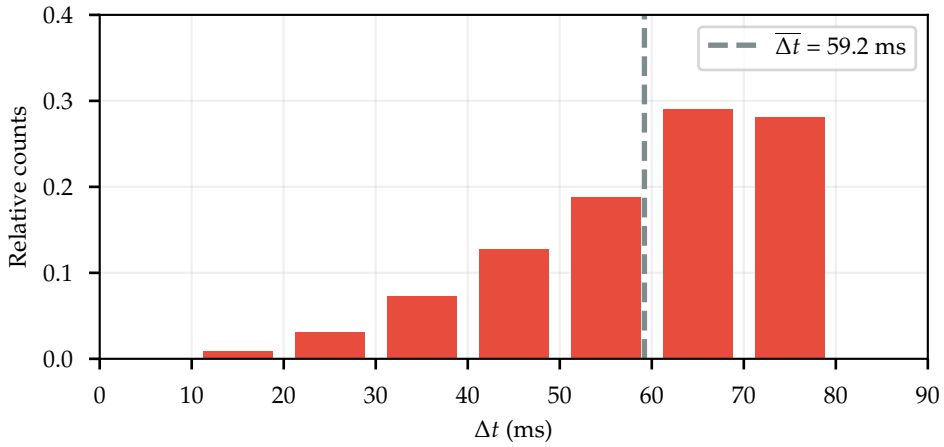
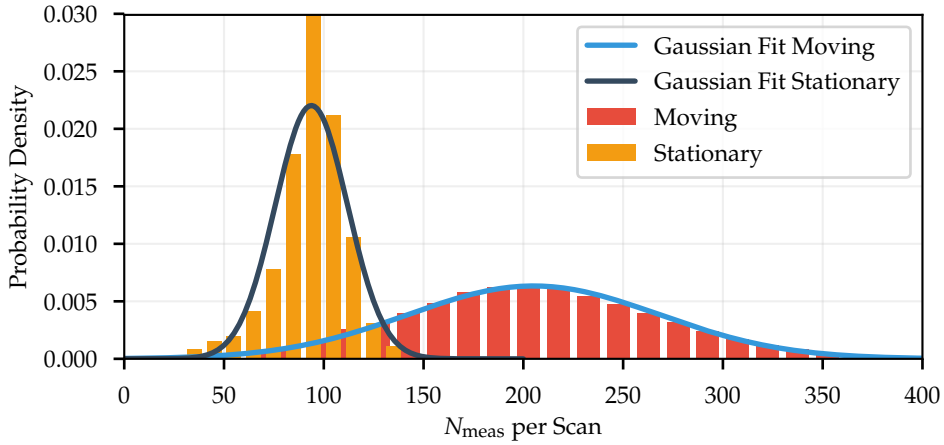


Figure 4.2: Distribution of the sensor cycle times of one radar sensor. The cycle time is here defined as time difference between two near-range scans. The grey dashed line shows the average cycle time.

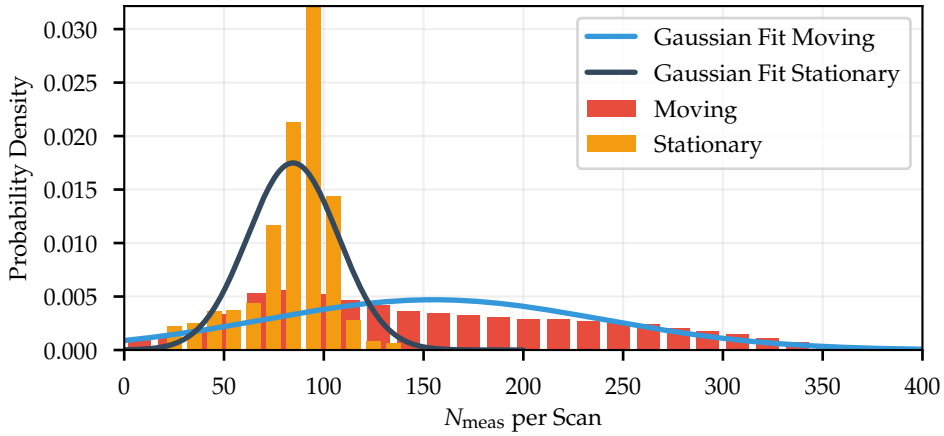
scaled such that the integral over each of the Gaussians as well as the integral over each of the two histograms is equal to one. The probability densities for the moving and the stationary case were calculated separately from each other so that for each scenario a separate Gaussian distribution can be fitted. The distribution of the measurements obtained from a moving sensor is much broader than the one of the stationary sensor measurements. On average, the moving sensor delivers about twice as many points as the stationary sensor.

Now consider 4.3b, i.e. the sensor mounted at the left side of the vehicle, facing 90° to the left of the driving direction. In this case, neither of the two distributions is well described by a Gaussian distribution. The distribution of the data obtained from the moving sensor is shifted to smaller values where the maximum value now approximately coincides with the maximum of the distribution obtained from the stationary sensor. However, the distribution has still a long and slowly decaying tail with values ranging up to the same maximum values as the distribution of the front sensor.

A definite explanation for the two different distributions in the cases of a moving or a stationary ego-vehicle is hard to obtain. One possible explanation could be the following reasoning. A moving sensor (ego-vehicle) causes that stationary targets are measured with an angle-dependent and non-zero Doppler value. If now a CFAR algorithm is applied on individual range–angle slices of the data cube (see Fig. 3.8 and Fig. 3.9), then spreading the data along the Doppler dimension results in a higher number of targets because the noise level is reduced around one cell. This hypothesis is backed by the observation that the sensor facing to the front



a) Sensor front left.



b) Sensor side left.

Figure 4.3: Number of measured targets per scan for two different sensors. Measurements with a moving or stationary ego-vehicle are displayed separately from each other.

is more affected than the sensor facing to the left of the vehicle since this sensor moves perpendicular to stationary targets (at least for small azimuth angles) so that no additional radial velocity of these targets is measured and hence no spread in the Doppler dimension takes place. Since the manufacturer did not provide a documentation of the radar sensor up to this level of detail, no final explanation for this behavior can be given.

4.2 LABELING

Large portions of this work make use of supervised learning approaches for classification or semantic (instance) segmentation applications. The foundation of every supervised machine learning algorithm is a sufficiently large data set with annotations of high quality. In this section, information on how the labeling procedure took place are presented.

Labeling of radar data can happen on different computation levels: early in the computation chain on the range-Doppler matrix [1], [154], [181], [192], [193], on CFAR target level (this work), on tracked objects (e.g. data for ACC-systems) or in occupancy grid maps [144]–[147], [272]. The level on which the data is labeled depends strongly on the application since for example for an ACC-system there is no need to annotate individual CFAR targets because the final application works with tracked *objects*. Labeling radar data on occupancy grid maps proved to be very successful for classification tasks of *static* objects. However, accumulating data over multiple seconds is not useful if *dynamic* objects are considered which by definition move throughout the integration time. Moving objects create a velocity-depended trail in a world-fixed coordinate system if accumulated data is considered. These trails may overlap in situations where the trajectories of multiple objects cross in space but are separated in time. As it is impossible to ascribe intersecting regions uniquely to one object, labeling of dynamic objects in grid maps cannot be done in all situations and is hence not considered here.

Labeling on object level has the upside of being relatively cheap since (ideally) for each object only one single decision has to be made, namely to which class this object belongs. The downside of this approach is that object formation has to be done at some point and that the label quality depends on the algorithm that created the object. This is not an issue for systems which directly take the formed objects as input (from the same algorithm which created the objects for the labeling stage) and predict one single object label. In this work, however, the goal is to predict a class label on target level early in the process chain so that during object formation and tracking these class labels can be used as input. If now tracking algorithms are used to create the objects for the labeling stage, the quality of these labels will be bounded by the currently used tracking algorithm and will not be a valid "ground truth".

Labeling on the range-Doppler matrix or on the full 3D data cube can be desirable if the algorithms used later on also work on this computation level or if one wants to be independent of the applied CFAR algorithm. The latter might be interesting for example if high-resolution algorithms are applied to regions of interest, which are then resolved more accurately resulting in more CFAR targets. In this case, it

could be useful to have ground truth labels attached already to the data cube so that irrespective of the applied target extraction algorithm ground truth labels can be propagated from the data cube to the individual targets. The downside, however, is that these deeper signal processing stages are often not available for series automotive radar sensors because target extraction from the data cube is done directly on the sensor to save bandwidth. Additionally, humans are not experienced in visually understanding the contents of the 3D data cube so that labeling of this data is a great challenge and way more difficult than annotating more processed data.

Having considered the different possible label stages, it appears reasonable to select the CFAR target level as the input for the labeling: The targets can be extracted directly from the sensors and stored in an appropriate file format without getting too close to bandwidth problems if multiple sensors are used. The algorithms discussed in this work use as input either the targets themselves or clusters created from these targets. It is therefore necessary and sufficient to have one ground truth label attached to each target in addition to the information to which object each target belongs to.

Each recorded sequence with length of a few minutes is the basis for one labeling task and treated independently from all other recordings. Since data from multiple radar sensors has to be annotated, all measured targets from all sensors are integrated into one single spatio-temporal point cloud. This point cloud is then cut into frames of length 100 ms so that each of these frames contains all targets that any of the radar sensor measured in this time range. The points from the different sensors are transformed into the current car coordinate system of the respective frame, cf. Section 3.2. In each of these frames, the labeler (i.e. the person who labels the data) compares the images from the documentary camera with the radar point cloud and groups all points that belong to the same object together to one cluster. Every cluster is identifiable by a *Universally Unique Identifier (UUID)* which works as an *object label*. In addition, the labeler adds a *class label* to each cluster which is then automatically propagated to each point within the cluster. In this work, the following semantic classes of true objects are considered

- `Passenger Car` (e.g. everyday sedans, SUVs, ...),
- `Pedestrian` (with or without smaller objects like suitcases, canes, ...),
- `Pedestrian Group` (used when individual pedestrians cannot be discerned in the radar point cloud),
- `Two-Wheeler` (e.g. bicycles, motorbikes, ...),
- `Large Vehicle` (e.g. buses, trucks, construction vehicles, ...).

These classes were chosen because on the one hand they cover most of the moving objects one encounters in every day traffic and on the other hand because they are not too fine grained so that a separation between the classes based on radar data would become impossible. In Section 6.5.1, it is analyzed if other motion types of pedestrian can be classified if longer observation windows are available. These motion types contain for example jumping, walking with crutches or skateboarding.

Radar sensors deliver very noisy data compared to a lidar sensor. Therefore it happens that points which are in fact static are reported with a non-zero ego-motion compensated Doppler velocity $\hat{v}_r$. Ambiguities in Doppler or azimuth as described in Section 3.1.3 are one possible origin of points with wrong Doppler values. Other origins of non-zero $\hat{v}_r$ values of in fact static points are interference effects caused by other sensors, incorrectly reported (or delayed) ego-motion values of the vehicle, errors in the calibration or misalignment. Mirror effects are another source of apparently moving points that have no match to an object in the real world. To consider these noise points, they are ascribed to the `Clutter` class during labeling. Notice that such a `Clutter` cluster is only annotated if clutter is visible for multiple time steps so that not every single point with $|\hat{v}_r| > 0$ has to be labeled. Only those `Clutter` "objects" are of interested which for example would also be considered in tracking algorithms. Moving points for which it cannot be decided whether they belong to a real object or are in fact just clutter, are marked as `Unknown`. This label is used for example for occluded objects or for objects outside the field of view of the documentary camera. Targets marked as `Unknown` are excluded in the classification algorithms. Targets that are not ascribed to any of the aforementioned classes are automatically labeled as `Static`.

In the annotation software, the velocity $\hat{v}_r$ is visualized for each target by an arrow pointing towards or away from the radar sensor it was measured by. The length of the arrow indicates the absolute value of $\hat{v}_r$. The object label (i.e. the cluster UUID) is used for one object as long as it is visible in the radar point cloud so that the same UUID is used across multiple frames. One object gets only a new UUID if it was not visible in the radar point cloud for more than five consecutive frames, i.e. for more than 500 ms. Association of targets to one real world object happens only within one sequence. If for example one recording ends with a pedestrian walking past the ego vehicle and the next sequence starts with this same pedestrian, then different object UUIDs are used.

As an additional support for the labelers, radar points of future or past time steps can be projected into the current view, i.e. they are transformed into the same car coordinate system as the points from the actual selected frame. This has the effect that targets from different time steps that belong to one static object remain at the same spatial region whereas moving objects create a velocity-dependent trail – just as they would do in a grid map. In situations with only a few well separated

moving objects, the labeler can increase the time window so that points from multiple time steps can be annotated in one move and hence not each individual frame has to be considered. In situations with many objects, the time window can be decreased again to allow for a more fine-grained labeling.

Evaluation of the labeling tasks revealed that annotation of one frame with length 100 ms takes about two minutes. This value varies of course with the complexity of the scene: in situation with only one car in front of the ego-vehicle, labeling can be done much faster due to the possibility to label multiple frames at once. However, recordings made in the inner city with many road users take naturally more time to annotate.

In Fig. 4.5, one example scene is displayed. The annotated dynamic objects are highlighted in different colors for each class and a convex hull around targets displays that these points belong to the same object. Notice that the Doppler velocities of the pedestrian group in front of the ego vehicle is close to zero since they move mostly orthogonal to the sensors so that the radial component of the velocity vector is very small. The `Clutter` cluster at position $(10, 10)$ is presumably a mirrored object of the left-turning `Passenger Car` where the fences from the construction site work as a mirror. An image of the documentary camera of this scene can be found in Fig. 4.4.

Figure 4.4: Camera image of the scene shown in Fig. 4.5.

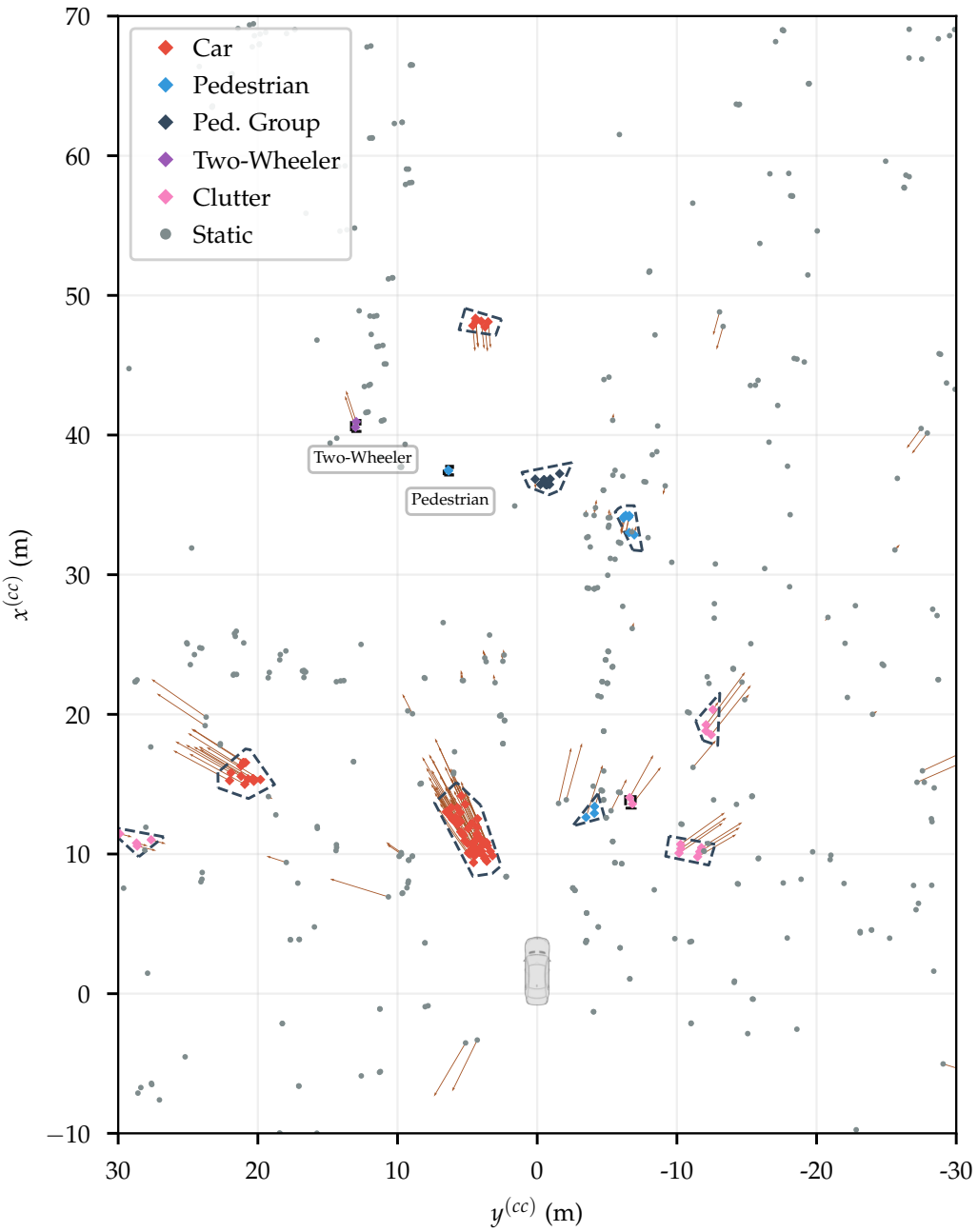


Figure 4.5: Example scene with labeled objects. The point cloud shows all annotated radar targets as well as the targets of the static environment. Data accumulated over 100 ms are shown.

4.3 ARTIFACTS AND LABELING CHALLENGES

In this section, examples of different radar specific artifacts will be presented and difficulties that occur during labeling are discussed. The intention is to give more insight into how the annotated data actually looks like and what challenges in the classification task arise due to the structure of the data itself. In Section 3.1.3, the origin of ambiguities in azimuth angle and Doppler velocity were already discussed. In Section 3.4 different types of clutter were defined. Examples for ambiguities and clutter objects are now presented here.

In Fig. 4.6, one example for a Doppler ambiguity is shown. The ego-motion compensated Doppler velocities of the individual targets are indicated by arrows where the length of each arrow symbolizes the magnitude of the velocity and the direction of the arrow encodes the sign. For the oncoming car on the opposite lane, one expects the Doppler velocities to be negative (arrows pointing towards the sensor) since the car approaches the ego-vehicle. However, in this scene some velocities are reported to be positive (arrows pointing away from the sensor) indicating that the relative velocity between sensor and measured object is larger than the unambiguous Doppler range and the sensor was not able to resolve this internally.

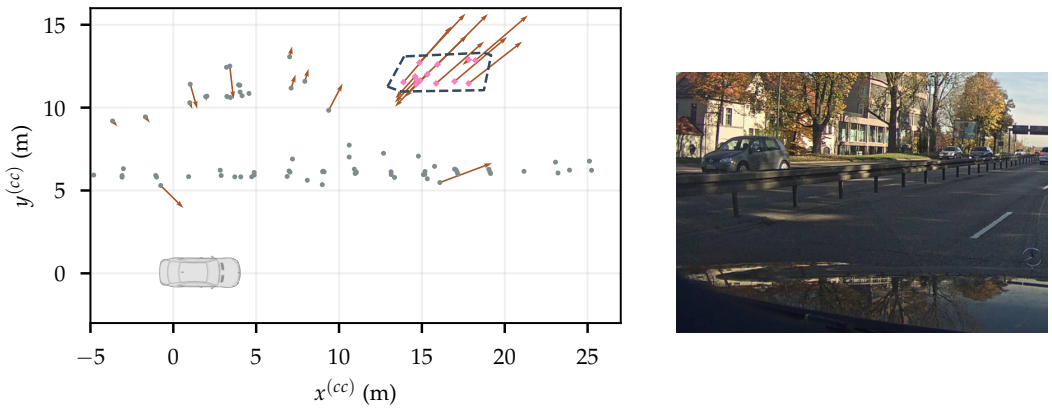


Figure 4.6: Left: Radar targets of an oncoming vehicle. Right: Cropped camera image of the same scene. The targets originating from the oncoming car are highlighted in pink. Targets between the ego-vehicle and the other car stem from the guardrail. Arrows indicate the radial velocity over ground. Data accumulated over 100 ms are shown.

Figure 4.7 shows one example scene in which the azimuth angle was not properly resolved by the sensor. The truck driving in front of the ego-vehicle is not only visible at its true position (green points) but also about 40° to the left (pink points). It appears as if the truck also drives “inside” the building next to the street. In

this situation the truck is visible at both positions for multiple seconds so that also tracking algorithms would spawn and track an object at this second position. Since the RCS values of the targets at the wrong position as well the shape of the emerging object are plausible, these situations are a great challenge for each classification algorithm.

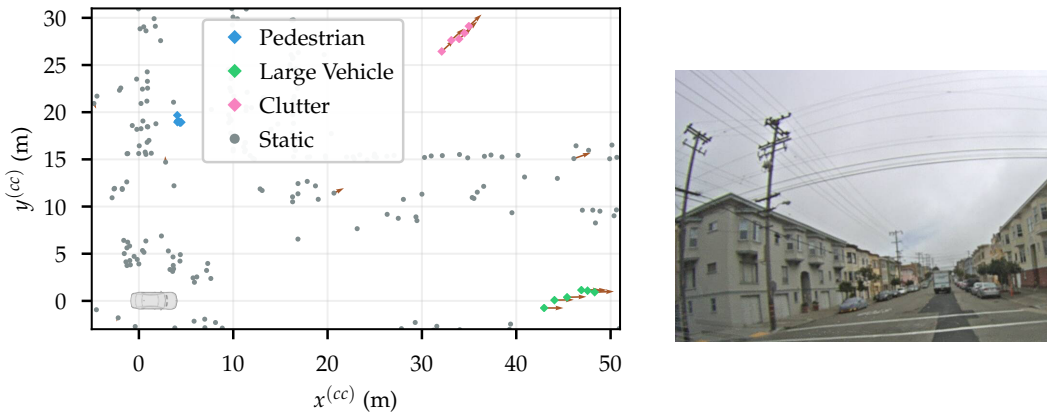


Figure 4.7: Left: Radar targets of a truck (green), the static infrastructure (grey) and a clutter object (pink) caused by an angle ambiguity. Right: Cropped camera image of the same scene. The pedestrian (blue points) is not visible in the camera image. Data accumulated over 100 ms are shown.

One further challenge is that sometimes the reported ego-motion values are not perfectly synchronized to the radar data. This becomes obvious especially in situations with large accelerations, e.g. during a turn of the ego-vehicle, since the yaw-rate of the vehicle changes quickly here which has a great influence on the ego-motion compensation, cf. also Eq. (3.36). This causes that truly static objects are erroneously reported with non-zero $\hat{v}_r$ which might mislead clustering as well as classification algorithms. In Fig. 4.8, an example scene is displayed in which the ego-vehicle does a left turn. The actually static targets from the surrounding are reported with a non-zero ego-motion compensated Doppler velocity since the yaw-rate obtained from the vehicle's odometry sensors does not correspond well enough to the actual motion of the vehicle at the measurement time of the radar sensors. This is well visible on the targets obtained from the curbstones, especially on targets from the lowered curbstone to the left of the vehicle. The magnitude of the erroneously reported velocities ranges around 0.5 m/s so that the longest visible arrows in the graphic correspond to about 1 m/s. It should be noted that the camera image was extracted two seconds before the turn was done on the intersection to better show the scenery.

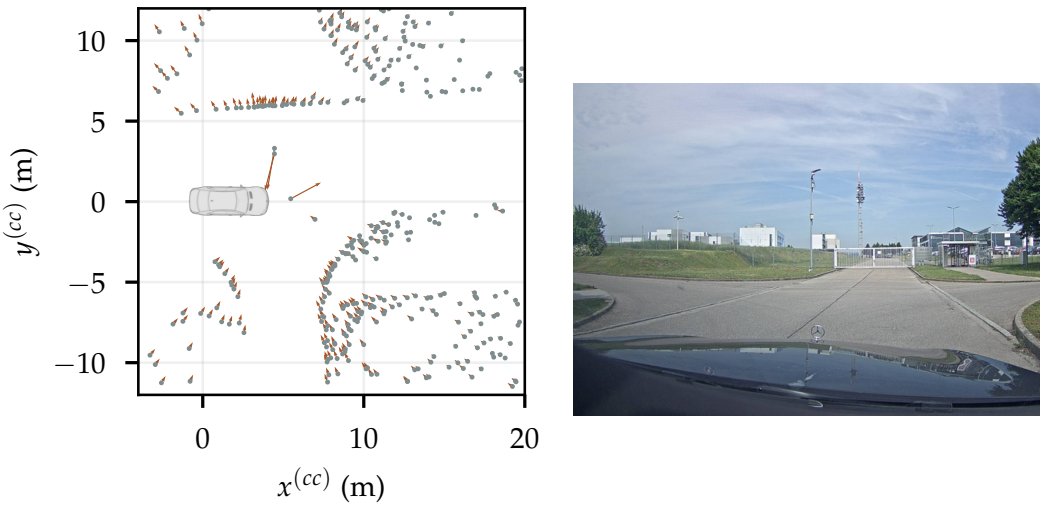


Figure 4.8: Left: Radar targets of the static infrastructure (grey) with ego-motion compensated Doppler velocity $\hat{v}_r$ depicted as arrows. Data accumulated over 100 ms are shown. Right: Cropped camera image of the scene. The ego-vehicle performs a left turn and the odometry sensors yield inaccurate data.

As described in Section 3.4, one type of radar clutter is formed by mirrored objects. An example scene in which the ego-vehicle is mirrored in a metallic gate is displayed in Fig. 4.9, where data from a time range of 500 ms is shown to pronounce the effect. The camera image of this scene can be found in Fig. 4.8. The gate is located about 37 m away from the ego-vehicle (center of rear-axle) and the mirrored object appears at about twice the distance from the front of the ego-vehicle to the gate, i.e. at about 70 m. Notice that the ego-vehicle moves at a slight angle with respect to the gate so that the mirrored object is not at the same $y^{(cc)}$ coordinates but slightly shifted.

These radar specific artifacts are one reason why annotation of radar data is extremely difficult and hence both expensive and time consuming. One other complication is that due to the lack of height information of the detected targets and the sub-optimal azimuth angle resolution at larger detection angles, association of the radar targets with the objects in the camera image is quite challenging. In combination with the rather high amount of clutter, it is often difficult to decide whether a certain target still belongs to one object or should rather be regarded as clutter. In Fig. 4.10 a scene is displayed in which many clutter points between two oncoming trucks make it almost impossible to associate the targets to either one of the two trucks or to a clutter cluster. Only with the help of the camera image and by looking at previous and following frames, a labeler can estimate the

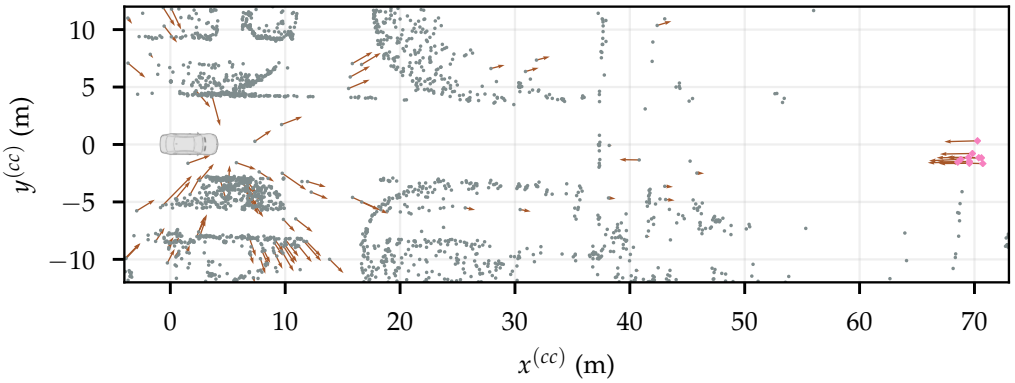


Figure 4.9: Radar targets of the static surrounding (grey) and the mirrored ego-vehicle (pink). Data is accumulated over 500 ms to emphasize the effect. Camera image of this scene can be found in Fig. 4.8.

dimensions of the objects and afterwards infer a reasonable segmentation of the scene. Additionally, radar sensors are often able to detect objects which are not visible in the documentation camera either because the field of view of the radar sensors are larger than the camera's field of view, because illumination conditions degrade the image's quality (direct sunlight, shadows etc.), or because occlusions hide objects from the camera but not from the radar sensor. In the latter case, the radar detects an object due to multipath propagation of the radar signal while the direct line of side is occluded. At larger distances from about 80 m onward, the image resolution of the documentary camera becomes a limiting factor for labeling since especially partly occluded pedestrians are difficult to identify in the camera image and hence it is also difficult to annotate the radar targets correctly. This shortcoming can only be healed in situations where the ego-vehicle approaches the far-distant object so that the labeler can use the information from later time steps to infer the true semantic class of an object.

Leaving the radar specific challenges during labeling aside, there are corner cases for which it is difficult to assign a class label. There is no strict border between a `Large Vehicle` and a `Passenger Car` but rather a smooth transition between the two classes: Figure 4.11 shows two vehicles for which it is not obvious to which of the two classes they belong. In this thesis, the rule for the distinction between these two classes is that vehicles with length of more than 5.5 m should belong to the `Large Vehicle` class. However, measuring the length of a moving object is not directly possible from the detected radar targets since the contour information from one radar scan is often insufficient. If data from multiple scans were used, precise knowledge of the object's motion state would be needed to compensate for its dynamics. Only then the measurements from the different time steps could

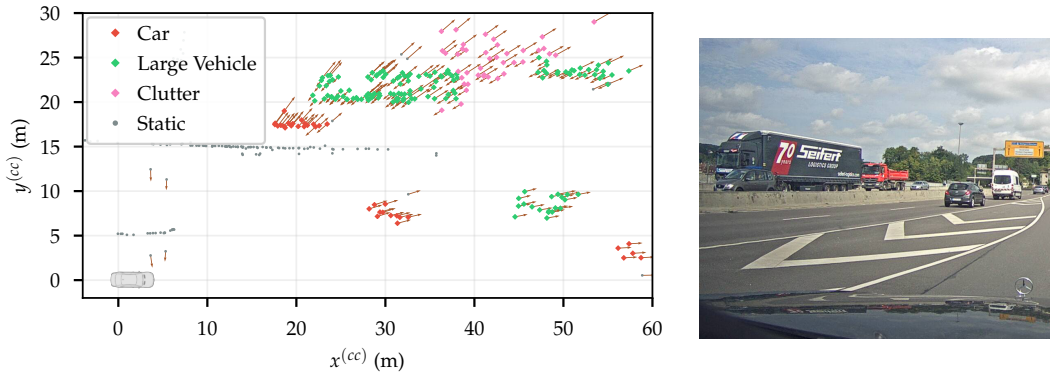


Figure 4.10: Left: Radar targets of other vehicles (red and green), the static environment (grey) and clutter (pink). Many clutter points between the two trucks make it difficult to annotate the real objects. Data accumulated over 100 ms are shown. Right: Cropped camera image of the scene.

be mapped to the correct Cartesian positions so that the contours become more pronounced. Since such a precise motion state estimation and tracking algorithm does not yet exist, the objective condition "object length is $\leq 5.5\,\mathrm{m}$" cannot be followed in all cases and the labeler has some freedom to assign these corner cases to either class. Human errors like overlooking objects, assigning wrong attributes or class labels to an object, merging multiple objects into the same cluster or adding points from the environment to a dynamic object cluster are further common sources of errors.

Figure 4.11: Two examples of vehicles for which a mapping to the two classes `Passenger Car` and `Large Vehicle` is non-trivial.

4.4 DESCRIPTION OF THE DATA SET

In the following subsections, details about the recorded data are presented. General properties such as the hours of driving, the number of driven kilometers, the

Property	Value	Description
Recording Time	4.95 h	Accumulated time in hours of all recordings
Driven Distance	104.11 km	Accumulated distance in kilometers covered during all recordings
Speed Range	0 m/s – 25 m/s	Minimum and maximum speed of ego-vehicle during the recordings
Number of Targets	1.35×10^8	Total number of measured radar targets from all four sensors
Labeled Targets	5.81×10^6 (4.3 %)	Total and relative number of annotated radar targets
Objects	7614	Total number of annotated objects in the data set (without clutter)

Table 4.1: General properties of the recorded data.

number of recorded as well as labeled radar targets and the total number of labeled objects will be presented. Afterwards the spatial distribution of the labeled data within the field of view of the radar sensors is discussed, the Doppler velocity distribution of the dynamic objects is presented and the RCS values of the different object categories are analyzed. Then, three different bicycle types are compared concerning their appearance in an automotive radar sensor.

4.4.1 *General Properties*

Data were recorded between December 2016 and May 2018 in Ulm and the surrounding area. The most general properties of the data set are summarized in Table 4.1. From this table, it can be seen that only a small fraction (4.3 %) of the recorded data was annotated. This is simply a result from the fact that dynamic objects are much rarer than static objects during a normal drive so that the majority of the measured targets belong to static infrastructure. The ego-vehicle moved with a maximum speed of 25 m/s during the recordings with an average velocity of a bit under 6 m/s. Some recordings were even done with the ego-vehicle being stationary.

In Table 4.2 more details about the class distribution of the annotated objects and targets is given. In the first part of the table, the number of labeled targets per

	Passenger Car	Pedestrian	Pedestrian Group	Two-Wheeler	Large Vehicle	Clutter	Static
# labeled targets	1 922 385	424 047	934 994	237 258	913 047	1 379 757	128 828 260
in %	1.43	0.31	0.69	0.18	0.68	1.02	95.68
# labeled objects	3815	1701	1216	379	503	49 236	–
in %	6.71	2.99	2.14	0.67	0.88	86.61	–
total observ. times (h)	7.28	3.68	4.42	1.10	1.39	6.71	–
in %	29.60	14.98	17.99	4.48	5.63	27.31	–

Table 4.2: Distribution of the annotated targets and objects among the classes.

class are listed and in the second part, the number of unique objects per class is stated. There is no distinction between objects that were visible only for a few seconds and objects which were visible for multiple minutes. To quantify how long dynamic objects were recorded, the total observation duration of object instances is calculated as

$$T_{\text{obj.}} = \sum_{i=1}^{N_{\text{obj.}}} \left(t_f^i - t_s^i \right), \tag{4.1}$$

with t_s^i and t_f^i being the start and final time at which the ith object was seen. This number is listed for each object class except for the `Static` class since no object formation was done for any static object. The observation time can be also interpreted as the time one single object from each semantic class would have to move in front of the ego-vehicle with no other moving objects present in order to generate the same amount of data as it is present in this data set. This is a useful metric for comparison with other publications, e.g. [192], [193] in which statements are derived based on data sets consisting of only one or just a few members of each class. The numbers in the table now list for how long one class representative would have to be recorded to make a data set comparable to this one (irrespective of the smaller variety of objects and traffic scenarios if only one object from each class is used at one single location).

To give an impression of how many targets N_t are measured during one scan of one radar sensor on objects of the five classes (`Clutter` and `Static` are not considered), in Fig. 4.12 average values of this quantity are displayed. As N_t depends strongly on the distance of the object to the sensor, one mean value along with the corresponding standard deviation σ is calculated for each distance bin

of width 1 m in the range 0 m to 100 m. That is, for each annotated object of one semantic class, the measured targets per individual scan are counted and the average position of the object during the measurement is calculated. The counts are then sorted into the respective range bins that are in the end used for the calculation of the mean and standard deviation. The solid lines in Fig. 4.12 display the mean values and the slightly transparent regions around the solid lines mark the $\pm\sigma$ area around the mean.

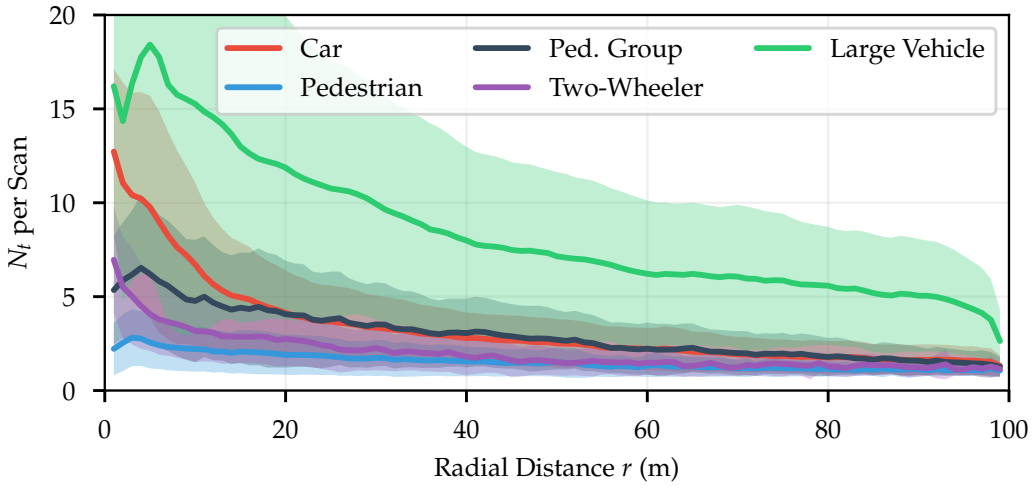


Figure 4.12: Number of targets per object measured by one radar in one scan versus radial distance to the object. The colored areas indicate $\pm\sigma$ regions around the mean value.

Unsurprisingly, objects of the class `Large Vehicle` contain on average the most points and objects from the class `Pedestrian` the least points. Even though the average value for the `Large Vehicle` class is still far above the values from the other classes, the large standard deviation already hints that there are large fluctuations, which are mostly induced by the orientation of the objects. For example, if a truck drives in front of the ego-vehicle at a distance of about 90 m, it cannot be expected that many more targets are measured than from a normal car, since only the back of the truck is well visible. Measurements from the truck's underbody are rare at these distances. If, however, the truck is observed at the same distance at an intersection so that the full length of the truck is visible, more targets will be measured and the difference to a normal car becomes more pronounced.

The reason why the number of detected targets per object decreases with increasing distance is twofold. Firstly, the greater the distance to an object the higher the chances that occlusions perturb the measurement of the object. For example, a smaller number of targets will be measured from a car at 80 m distance if two other

cars are in between the ego-vehicle and the considered car compared to the case when there is a free line of sight to the car. Secondly, the noise level increases with increasing distance so that in order to maintain a constant false alarm rate during target extraction, the detection thresholds are increased so that on average less targets are reported per object. This point will be discussed in more detail in Section 4.4.4, when the distance dependency of the RCS values is presented. Notice that for the annotated objects no difference in the N_t distributions between a stationary and moving radar sensor were detected. This hints that the difference observed in Fig. 4.3 only affects measurements of the static environment and hence no further distinction is needed in this thesis.

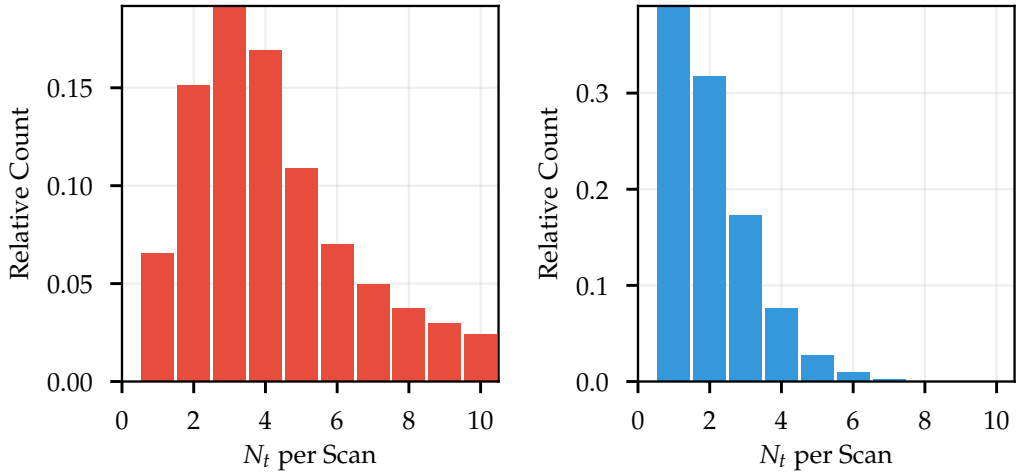


Figure 4.13: Histogram with relative counts of the number of measured targets per scan. Only targets from the range 0 m to 30 m are considered. Left: Histogram for the `Passenger Car` class. Right: Histogram for the `Pedestrian` class.

Finally, in Fig. 4.13 histograms of the distribution of N_t with targets from the range 0 m to 30 m are displayed. The histogram on the left shows the relative counts for the `Passenger Car` class, whereas the histogram on the right shows data for the `Pedestrian` class. Since in Fig. 4.12 only the mean values and the standard deviations are displayed, the two histograms give more detailed information about the actual distribution of N_t. Interestingly, the data of the `Passenger Car` class resemble a Poisson distribution with $\lambda \approx 3.5$. However, a rigorous χ^2 test against a Poisson distribution failed so that only a visual resemblance can be stated. The histogram for the `Pedestrian` class illustrates that most often only one or two targets are returned from one pedestrian so that a single-shot distinction between a clutter point and a pedestrian is almost impossible. Only if data from more than one sensor or from multiple scans is considered, classification seems feasible.

4.4.2 *Spatial Distribution of Labeled Objects*

The field of view displayed in Fig. 4.1 indicates the regions in which each radar sensor can possibly detect objects. However, it does not indicate how the recorded data are distributed in both azimuth angle and range. Therefore, in Fig. 4.14 four polar plots are shown in which the positions of the recorded targets of the semantic classes `Passenger Car`, `Pedestrian`, `Pedestrian Group`, `Two-Wheeler` and `Large Vehicle` are displayed. Each plot shows the targets recorded by one sensor. The plot is rotated by the sensor's installation angle so that the vehicle's heading direction (x-axis of the vehicle coordinate system) always points upwards and the vehicle can be imagined as being at the center of the four plots. The azimuth angles are, however, relative to the sensor coordinate system so that an angle of 0° corresponds to the position directly in front of the sensor. In Appendix A.2, these kind of plots are repeated with each class displayed in a separate figure.

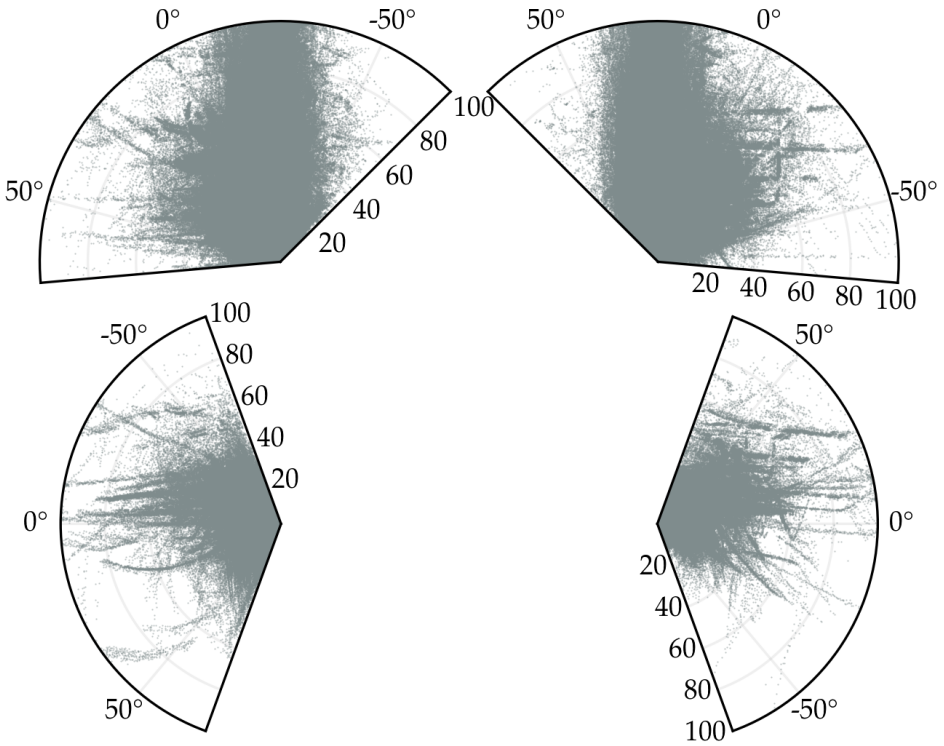


Figure 4.14: Target distribution in a polar plot. The measured targets of each sensor are displayed in a separate diagram. The diagrams are rotated by the mounting position of the respective sensor.

Since traffic scenarios usually do not have rotational invariance, a homogeneous distribution of the measured dynamic objects around the ego-vehicle cannot be

expected. Rather, objects are much more likely to appear in front of the ego-vehicle so that the distribution is biased toward the forward direction. Streets in the city are likely to be surrounded by buildings, rural streets are often neighbored by fields and motorways contain guardrails next to the road. In all three cases, dynamic objects cannot be expected to appear at large distances to the left or to the right side of the ego-vehicle either because the field of view of the radar sensors is blocked or because it is unlikely for road users to be at these places. Intersections, side roads, squares or roundabouts are of course exceptions and measurements from dynamic objects in these scenarios will show up in other locations than the forward direction.

The plot in Fig. 4.14 supports this reasoning since indeed most dynamic objects were measured in the forward direction and the two side radars measured way less dynamic objects at larger distances than the front facing radars did. What is more, some individual tracks become visible in this representation.

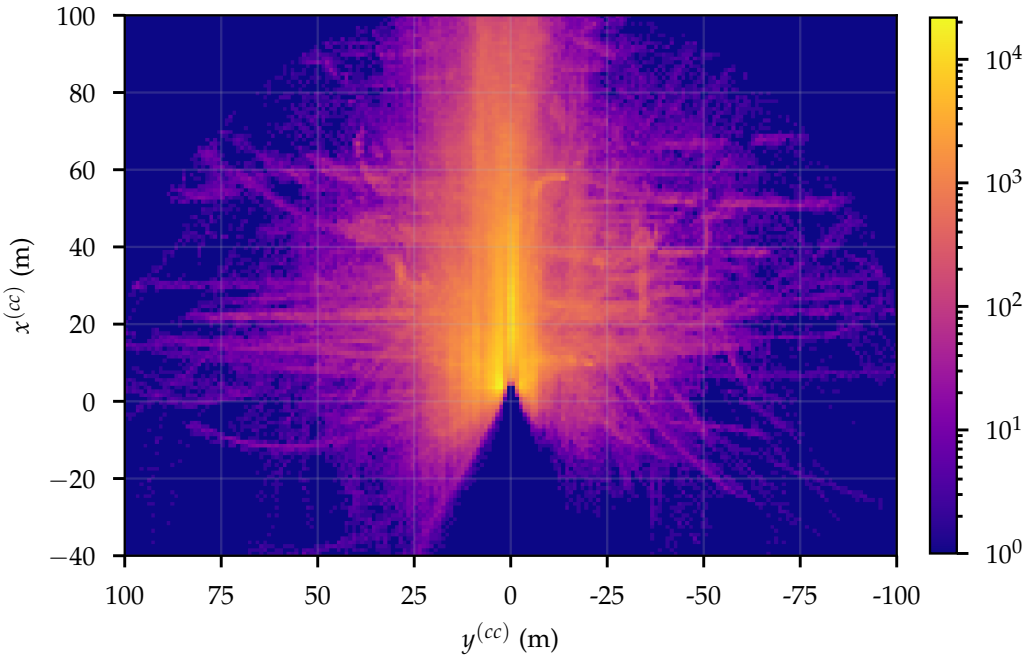


Figure 4.15: Heat map of the target distribution around the ego-vehicle. Colors indicate how many annotated targets were measured in each cell. Log-scale is used to make areas with low occupancy visible.

Another way to visualize the same data is shown in Fig. 4.15, where a heat map displays how many targets were measured around the ego-vehicle. In this grid, a cell size of $1\,\mathrm{m} \times 1\,\mathrm{m}$ is chosen and a log scale is used so that also areas with just a

few measurements are visible. Again, the analysis made before is backed by this plot.

However, even though a homogeneous coverage of each region around the ego-vehicle with objects from the five classes cannot be expected due to the above reasoning, these plots (especially the per-class plots in Appendix A.2) display that the data set is still not large enough to make broad and general statements about each situation. Especially members of the `Two-Wheeler` class are not equally distributed over the sensors' field of view so that more data from this class would be desirable. Statements made in this work should therefore always be understood with this limitation in mind.

4.4.3 *Distribution of Doppler Values*

After the spatial distribution of the annotated targets was discussed in the last section, now the distribution of the measured Doppler velocities is considered here.

To this end, Fig. 4.16 shows one histogram for each semantic class in which the velocities $\hat{v}_r$ are displayed on the horizontal axis and the counts of each bin are plotted on the vertical axis. The histogram was normalized so that the sum of all bins equals one. In Section 4.3, an example was shown for a situation in which the Doppler velocities of an oncoming vehicle were incorrectly reported due to the incapability of the radar sensor to resolve the ambiguous Doppler value correctly. As this problem arises frequently in the recorded data whenever the relative velocities between ego-vehicle and detected object are large, also the displayed histograms are influenced by this. It can therefore be expected that the whole histogram is shifted towards smaller absolute velocities so that the real object velocities are larger (in magnitude) than the figure suggests. Again, it should be kept in mind that only the radial velocity is considered here since the true Cartesian velocity is not accessible.

Nevertheless, it can be seen from these histograms that objects from all classes were measured in a velocity range that one would ascribe to such an object so that the data set appears diverse enough in this dimension. It is interesting to see that the maximum of all distributions lies close to $0\,\mathrm{m/s}$. Targets with small $\hat{v}_r$ can result from slowly moving objects (e.g. slowing down and finally stopping at intersections or traffic lights), by tangential motion with respect to the radar sensor, stationary parts of a moving object (e.g. the standing leg of walking pedestrian or the bottom of a rotating car wheel) or by annotation errors (e.g. assigning targets from a curbstone to the cluster of a nearby walking pedestrian). From the mentioned possibilities, the greatest influence stems probably from tangentially

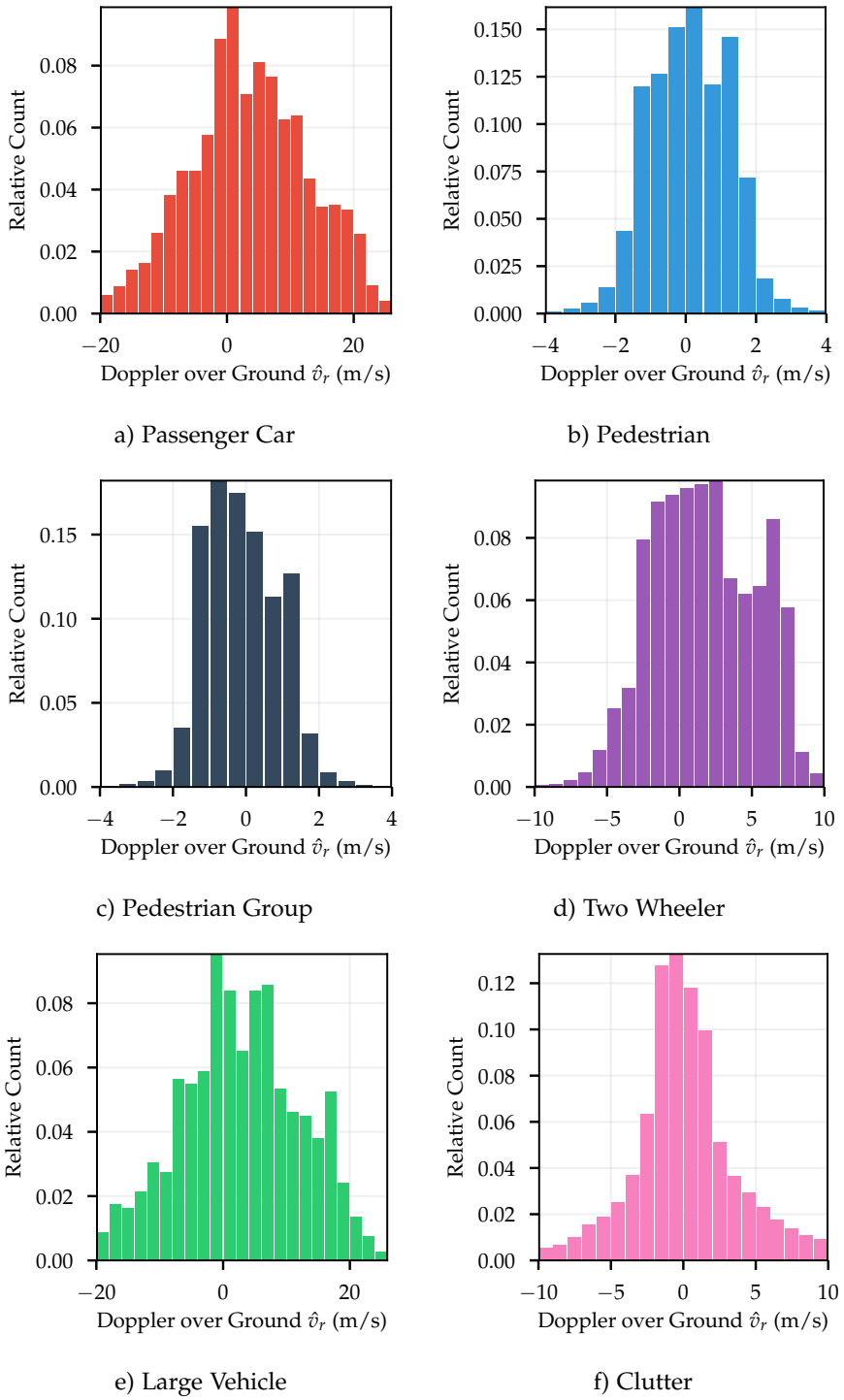


Figure 4.16: Histograms of ego-motion compensated Doppler velocities $\hat{v}_r$ for each semantic class. The histograms are normalized so that the sum of all bins equals one.

moving objects, followed by slowly moving objects. With the help of a sufficiently well performing multi-target tracking algorithm, the true Cartesian velocities could be estimated and hence the proportion of tangentially moving objects estimated. As the obtained insight of this analysis is expected to be of minor relevance for this thesis, it is omitted here.

Another interesting result from these histograms is that the data set does not contain any *running* pedestrians (at least not running in the radial direction) since almost no $\hat{v}_r > 4\,\mathrm{m/s}$ were measured. For future analysis, it would be worthwhile to analyze if the later on trained classification algorithms generalize to faster moving pedestrians.

4.4.4 *Radar Cross Sections of Different Road Users*

The RCS value is a measure of how large an object appears to the radar and has units of an area. Since the RCS value of objects differs in orders of magnitude, it is often given in logarithmic units, expressed as dBsm (*decibel square meter*). If x is the apparent size of the object to the radar measured in m^2, then the value σ in units of dBsm is obtained as follows:

$$\sigma = 10 \cdot \log_{10} \frac{x}{1\,\mathrm{m}^2}. \tag{4.2}$$

It should be noted that for a rigorous RCS analysis a special measuring chamber should be used which absorbs all signals stemming not from the object under consideration. In real world driving scenarios the RCS value is strongly affected by the path the electromagnetic wave took since for example fading due to multi-path reflections changes the return signal and hence also the RCS calculation. The data presented in this section should therefore be understood as real world measurements with all the side-effects included.

In Fig. 4.17, the distributions of the RCS values of the six different classes is displayed. Except for the `Clutter` class, the distributions resemble Gaussians, even though they appear slightly skewed for the `Passenger Car` and `Two-Wheeler` classes. Notice that the RCS values are given in logarithmic units, so that the linearly scaled values measured in m^2 would resemble log-normal distributions.

Not many publications intensively analyze the RCS values of different road users as measured by an automotive radar in the 76 GHz band. In [286], the RCS value of pedestrians is measured and the influence of different clothes is analyzed. They conclude that the RCS value of a pedestrian lies at $-8\,\mathrm{dBsm}$. In [242], the RCS values of a pedestrian, a bike, a car and a truck are given as $1\,\mathrm{m}^2$ (0 dBsm), $2\,\mathrm{m}^2$ (3 dBsm), $100\,\mathrm{m}^2$ (20 dBsm) and $200\,\mathrm{m}^2$ (23 dBsm), respectively. In [118], an

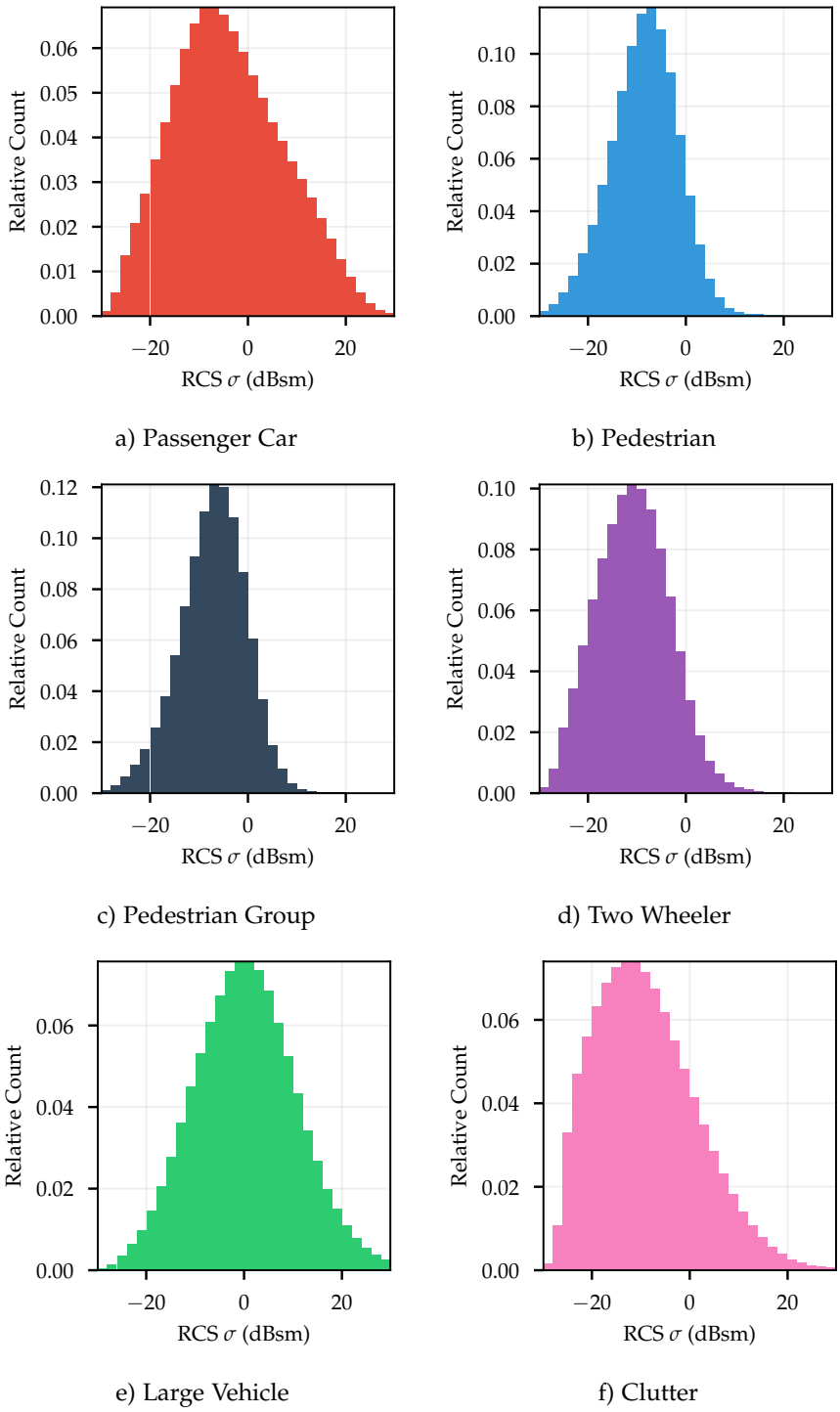


Figure 4.17: Histograms of RCS values σ for each semantic class. The histograms are normalized so that the sum of all bins equals one.

empirical model for the RCS values of different road users is created for a radar simulation platform. For pedestrians and motorcycles, fixed values of $-10\,\mathrm{dBsm}$ and $7\,\mathrm{dBsm}$, respectively, are used. For cars and trucks, the RCS value is scaled with the logarithm of the radial distance r to the object, so that $\sigma_{\mathrm{car}} = \min\{10\log_{10}(r) + 5;\ 20\}$ and $\sigma_{\mathrm{truck}} = \min\{20\log_{10}(r) + 5;\ 45\}$.

In [229] the distance dependence of the RCS value is again highlighted as for the two distances 5 m and 25 m the RCS value of a car is reported to change from 2 dBsm to 16 dBsm.

The data collected for this thesis support the general trend that the RCS value of an object increases with increasing distance. In Fig. 4.18, the RCS values for the two classes `Passenger Car` and `Pedestrian` are displayed with respect to the distance at which the targets were measured. The average value was computed on the linearly scaled RCS values and afterwards converted back to the log-scale. In Fig. A.6, the range-dependent RCS plots are repeated for each class. Notice that the averaging is done on a target level, i.e. all targets within one range bin are used for the calculation. In contrast, in Fig. A.7 the RCS values are plotted for the individual *objects*. That is, the RCS values of all targets measured during one scan of one object are summed together and this value is entered in the range bin corresponding to the mean radial distance of the targets to the sensor. In comparison with Fig. A.6, the curves in Fig. A.7 are shifted towards higher RCS values since now all targets of one object contribute to the displayed value and not the RCS value of the individual targets are displayed.

The reason why the RCS value apparently increases with increasing distance is that targets with small RCS value cannot be resolved at larger distances because the signal-to-noise ratio gets smaller the larger the distance. Only those targets of an object that have a high RCS value can still be resolved by the sensor so that only these targets enter the statistics. At smaller distances, however, also targets with small RCS value can be measured so that not only more targets per object are detected but also on average targets with a smaller RCS value. An additional effect is that at small distances not the whole object can be illuminated by the radar so that not every part of the object contributes to the total RCS value. In general, the RCS value is not only distance dependent but additionally has a strong dependence on the angle under which an object is detected. This influence, however, cannot be extracted from our data set since the orientation of an object to the sensor was not annotated by the labelers and no ground truth data exists for this quantity. Therefore, the angular dependency is averaged over in these plots and cannot be made more explicit.

The spikes in some of the plots in Fig. A.6 (and in Fig. 4.18) are probably caused either by errors in the measurements or by annotation mistakes. Since especially

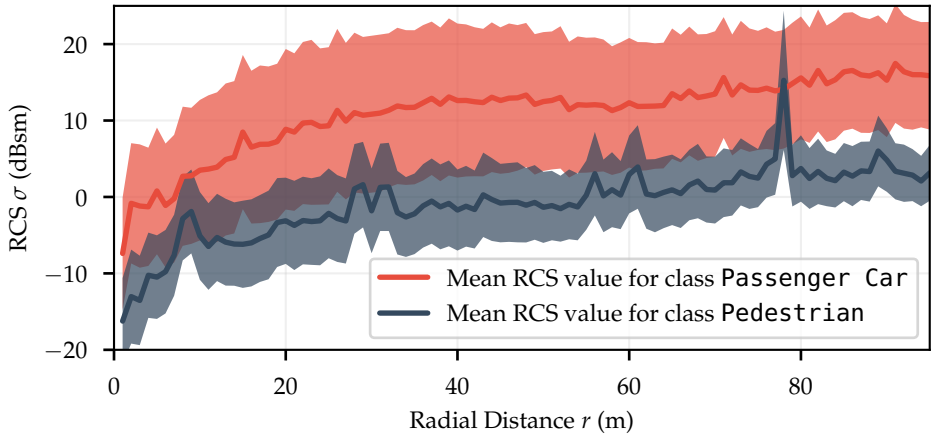


Figure 4.18: Measured RCS values as a function of the radial distance at which the respective targets were measured. Comparison between the two classes `Passenger Car` and `Pedestrian`. The shaded region mark areas within one standard deviation around the mean.

data from the `Two-Wheeler` class is rather rare compared to the `Passenger Car` class, a small number of labeling mistakes can cause large deviations in one range bin. For example, if a `Two-Wheeler` drives close to a `Large Vehicle` at a large distance and a labeler erroneously adds one target of the `Large Vehicle` to the `Two-Wheeler` cluster, then the RCS value in this range bin increases considerably since the RCS value of a `Large Vehicle` is usually much larger than the one from a `Two-Wheeler`. Because there are often not enough data points within one range bin to compensate entirely for one outlier, these mistakes show up easily. This fact can also be used to detect labeling errors automatically or to bring them to the attention of the labeler: if the average RCS value of a created cluster differs too much from an ensemble average, then this cluster can be marked as potentially defective and either the labeler or a quality control instance should check this cluster again.

4.4.5 *Measurements of Three Different Bicycles*

The results shown in this section were presented together with N. Scheiner in the workshop "Automotive Radar Systems and Signal Processing" at the European Microwave Week 2019 [226].

In the previous sections, inter-class variability was analyzed. Now, differences between objects of the same class, namely the `Two-Wheeler` class are inspected in more detail. To be more precise, in this section the question is answered whether

the bicycle types *carbon bike*, *folding bike* and *city bike* appear significantly different to an automotive radar sensor.

To analyze this question, recordings with the three different bicycle types were performed, recorded and annotated, resulting in about 110 000 labeled targets from the three bikes, i.e. about 20 min non-stop driving. Three different people rode each of the three bikes in turns to diminish the effect the rider has on the measurements.

The hypothesis prior to the measurements was that less targets from the carbon bike will be measured and that the RCS value of this bicycle type will be smaller than that of the other two bikes since carbon fibers are often used as absorber materials for electromagnetic waves, e.g. in stealth aircraft [119], [178], [207].

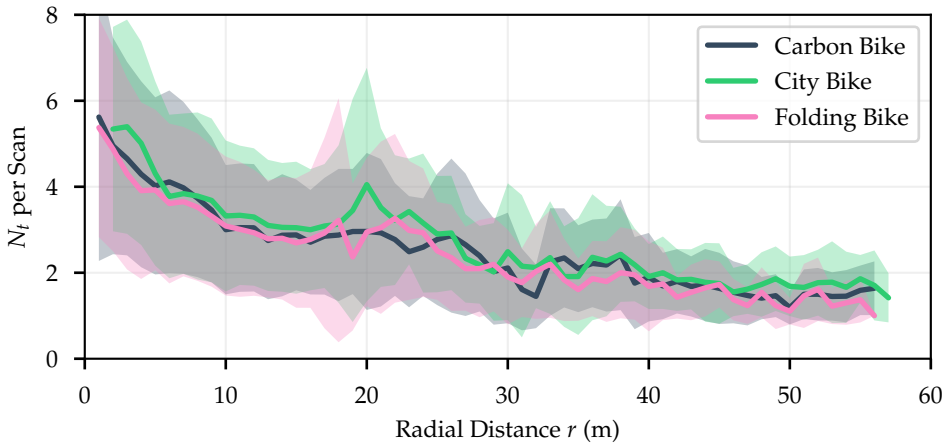


Figure 4.19: Comparison of the measured number of targets N_t per scan between the three different bicycle types.

In Fig. 4.19, the number of targets N_t measured per scan is plotted against the radial distance for each of the three bikes. The figure was created in the same way as Fig. 4.12, namely by counting the number of targets which were measured per object in each scan, sorting this number into the range bin which corresponds to the distance of the object to the sensor and finally averaging the values in each bin. This plot displays no visible difference between the three bike types. The number of measured targets decreases with distance at about the same rate for all three bikes and the deviations around the mean are very similar.

For further analysis, the RCS values of the three bicycle types are plotted in Fig. 4.20 versus the mean range of the respective object. Just as discussed in the previous section, the RCS value increases with increasing distance of the object. However, again no real difference between the different bicycle types can be found. The carbon bike was expected to have a smaller RCS value than the other two bikes due

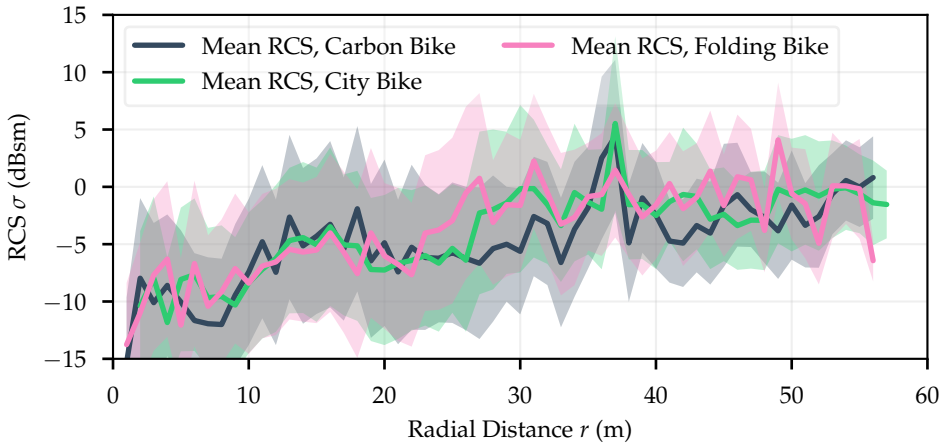
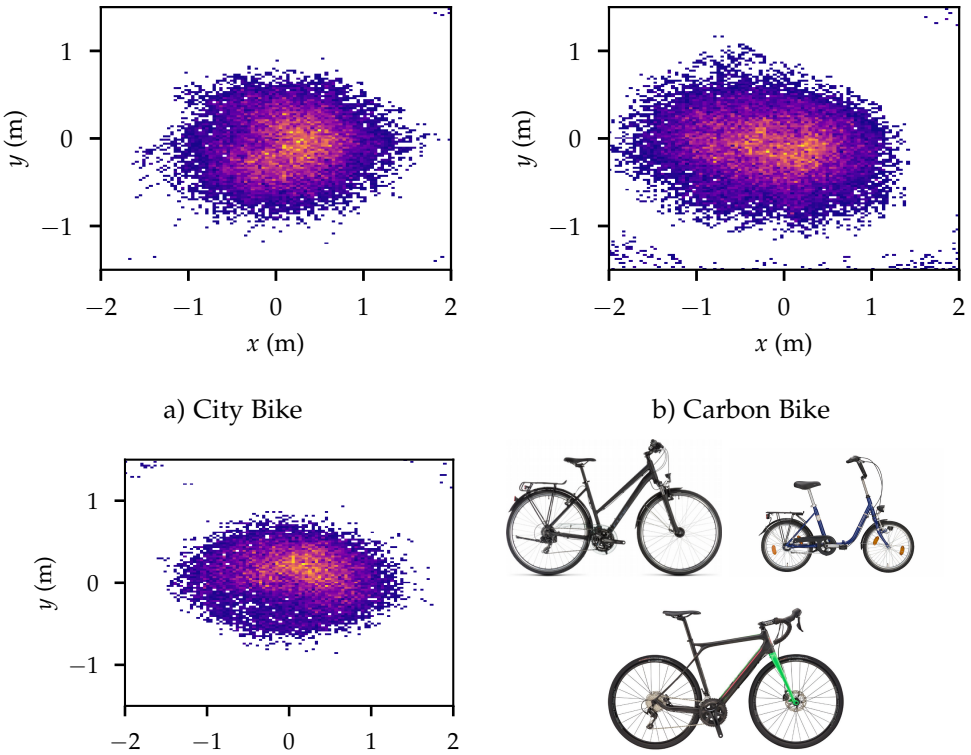


Figure 4.20: Comparison of the range dependency of the RCS values σ between the three different bicycle types.

to the absorption properties of carbon. Two possible explanations emerge from this. Either most of the measured targets stem from the *rider* of the bike and not from the bike itself or the material properties of the carbon used in racing bikes are different from the carbon fibers used for electromagnetic absorption.

To assess from which positions of the bicycles the measured targets originate, the precise ground truth position of the bikes during driving is needed. To obtain these values, a new measurement campaign was started during which the riders of the bike were equipped with DGPS systems stored in a backpack which were then synced with the DGPS system of the ego-vehicle. The targets obtained from the bikes can then be entered in a coordinate system that has its origin at the position of the DGPS system. Details about this procedure can be found in [225]. The results of these measurements are shown in Fig. 4.21, where one heatmap for each bicycle type is shown. Color indicates the relative frequency with which targets were measured in the respective cell. These plots show again no significant difference between the three bicycles. What is more, the distribution of the targets resulting from the carbon bike seem to have a *larger* spread than the distributions of the other two bikes. Hence, the hypothesis that most reflections stem from the rider and therefore no difference between the bikes becomes visible cannot be backed by these data. Nevertheless, all heatmaps show that the highest measurement frequency is at the center of the bikes, so that one can assume that the rider does have an influence on the measured targets, even though the overall spread of the data is not altered by the bicycle type.

Since still no difference between the bicycle types is visible, the initial hypothesis has to be rejected. One possible explanation for the similar radar characteristics of



a) City Bike

b) Carbon Bike

c) Folding Bike

d) Pictures of the three bicycles [48], [94], [184].

Figure 4.21: Spatial distribution of the measured targets of the three bicycles. Color indicates the relative count in each cell.

the bikes might be that for radar absorber material short carbon fibers with random orientation are used [221], whereas in other applications long and axis oriented carbon fibers are preferred for mechanical stability [134]. In [207], the influence of orientation of a carbon fiber plate on the RCS value is analyzed. The authors state that under some circumstances the RCS value of the carbon fiber plate is almost as high as the RCS value of an Aluminum panel of comparable size despite the great difference in electric conductivity. The results of this section are therefore in accordance with these articles and no significant differences between the bicycle types can be measured.

5

CLUSTERING OF RADAR DATA

For many radar applications, grouping of targets that belong to the same object is necessary. For example, to estimate the dimensions of an object in extended target tracking algorithms, a method is needed to decide whether a measured reflection belongs to one object. This holds also true for classification algorithms that take as input either the raw targets of one object or some feature vector generated from these targets. Clustering algorithms provide means to group individual items with similar properties together. The way the grouping is done depends on the clustering algorithm itself and the choice of the algorithm's parameters. Since the performance of the clustering algorithm has direct influence on all subsequent processing steps, tuning this step is vital to obtain high performance, for example in classification tasks.

In this chapter, clustering of radar data is discussed and a new method is presented to incorporate domain knowledge into an unsupervised clustering algorithm. First, a review of state of the art clustering methods on radar data is given. This includes the introduction of the general clustering algorithm DBSCAN *(Density-Based Spatial Clustering of Applications with Noise)* which is often used in radar applications. Since the proposed new clustering method relies on DBSCAN as well, it is worthwhile and necessary to first introduce the mechanics of this algorithm before its extensions are discussed.

In the second part of this chapter, the following own contributions are made:

- Introduction of a new metric to evaluate a clustering result by comparison with ground truth data
- Extension of the DBSCAN algorithm to work with multi-dimensional data with different scaling
- Extension of the unsupervised algorithm with a supervised component to learn different parameter sets for different range/velocity regions

This second part is based on the publication "Supervised Clustering for Radar Applications: On the Way to Radar Instance Segmentation" which was published beforehand [234].

5.1 STATE OF THE ART

This section is split into four parts. First, the DBSCAN algorithm is introduced before in the second part of this section some of the more popular variations and extensions of DBSCAN are introduced. Then, different approaches for clustering radar data are presented which most often rely on density-based clustering algorithms. Finally, the optimization scheme *simulated annealing* is explained as it is used later on in the supervised clustering algorithm.

5.1.1 *DBSCAN*

DBSCAN and its variants are popular density-based clustering algorithms. The algorithm was proposed in 1996 by Ester et al. [65] and specifically designed for working on data sets with spatial features with minimum requirements on domain knowledge. The algorithm takes as input a list of points $x = \{x_1, x_2, \ldots, x_n\}$ and outputs for each point a cluster id $y_i \in \{0, 1, \ldots, N_{\text{clst}}\}$ with $i \in \{1, \ldots, n\}$. The points x_i can stem from any metric space M with distance function $d : M \times M \to \mathbb{R}$. If two points share the same cluster id they belong to the same cluster. One cluster id is singled out (for example $y = 0$) to mark all the *noise* points, i.e. points that do not belong to any of the created clusters. The fundamental idea is that each cluster has a higher density of points within the cluster than outside and that the density around each noise point is smaller than inside any cluster. The existence of *noise* points is one of the aspects that discerns DBSCAN from other clustering algorithms like k-means, which assign each of the points x_i to one of the clusters and no point is left out.

To formalize the idea of density-based clustering, the following terms are defined in accordance with [65]. An ϵ-region around a point x_j is defined as the set of all points for which the distance to x_j is smaller than or equal to ϵ, that is $N_\epsilon(x_j) = \{x_i \in x \,|\, d(x_i, x_j) \leq \epsilon\}$. A *core* point is then defined as a point x_j for which $|N_\epsilon(x_j)| \geq N_{\text{min}}$, where $N_{\text{min}} \in \mathbb{N}$ is in addition to $\epsilon \in \mathbb{R}$ the second hyper-parameter of the DBSCAN algorithm. Since border points of a cluster usually have less neighbors in an ϵ-region than a core point, not every point of a cluster has to be a core point. However, a border point x_j of a cluster can be defined as a point which lies in an ϵ-region of a core point but for which $|N_\epsilon(x_j)| < N_{\text{min}}$. In [65], two points $p, q \in x$ are said to be *density-reachable*, if there is a chain $x_1, \ldots, x_n$ with $x_1 = p$ and $x_n = q$ such that x_{i+1} lies in the ϵ-region of the core point x_i. Furthermore, the points $p, q \in x$ are *density-connected* if there is a third point $s \in x$ such that p and q are both density-reachable from s. A cluster $C \subset x$ is then defined via two

conditions: Firstly, if $x_i \in C$ and x_j is density-reachable from x_i, then also $x_j \in C$, and secondly, if $x_i, x_j \in C$, then the two points are density-connected.

These definitions directly imply that given values for ϵ and N_{min}, a cluster is uniquely defined by any of its core points. This allows to construct clusters by selecting an arbitrary core point $x_i \in x$ and collecting all density-reachable points from this seed point by querying the points in the respective ϵ-regions until no more density-reachable points can be found. In [65], a pseudo-code for one possible implementation of DBSCAN is listed.

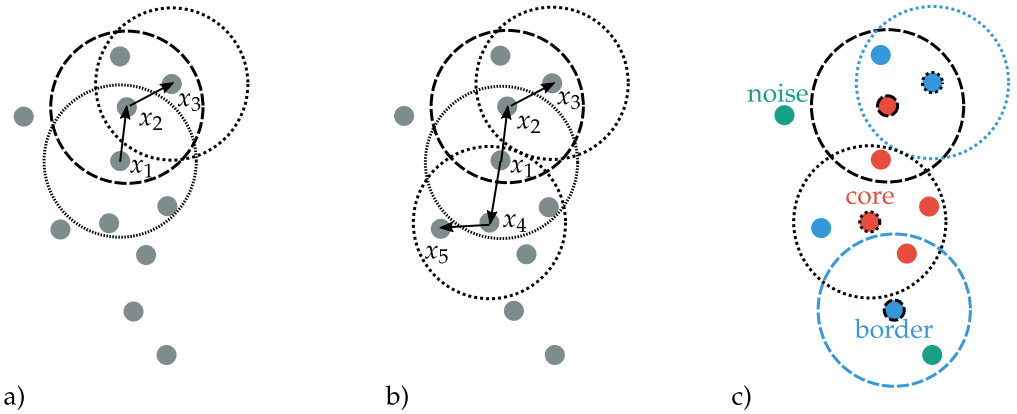


Figure 5.1: Examples for a) density-reachable points, b) density connected points and the final clustering result in c). See text for details. Inspired by Fig. 3 in [65].

In Fig. 5.1, one set of points x is drawn. The radius of the circles indicates the choice of ϵ and N_{min} is in this example set to $N_{\text{min}} = 4$. Notice that each point has at least one point in its ϵ-neighborhood, namely itself. In part a) of the figure, one example for two density-reachable points is shown. The point x_3 is density-reachable from x_1, but not vice versa since x_3 is not a core point. The points x_3 and x_5 are density-connected to each other by x_1 as shown in part b) of Fig. 5.1. The full clustering result is shown in c), where core points are drawn in red, border points of the cluster in blue and noise points are colored in teal. Even though the noise point on the bottom right of the data set lies in one ϵ-neighborhood of a point from the cluster, it is not part of the cluster since it is not density reachable from any of the cluster's points. This would only be the case if the point were in the ϵ-neighborhood of a core point and not only in the ϵ neighborhood of a border point as it is illustrated here.

The role of the two hyper-parameters ϵ and N_{min} is not completely disjoint. The threshold ϵ defines the search radius around a given point $x_i \in x$ in which neighboring points are sought. Core points are created if more than N_{min} neighbors are found in the search region. Since increasing ϵ also increases the possibility of

finding more neighbors, more core points are found and on average larger clusters are created the larger ϵ gets. The same holds true if N_{min} is decreased since then less neighboring points in a fixed ϵ-region are needed for the creation of a core point. Suitable values for these coupled hyper-parameters have to be determined for each clustering problem individually.

The run time complexity of DBSCAN depends on the method used for the neighborhood searches. If a fast indexing structure like an *R*-tree is used for the region queries, one neighborhood lookup can be done in $\mathcal{O}(\log n)$. This cost has to be paid at most once for each of the n points in x so that the total complexity of the algorithm is $\mathcal{O}(n \log n)$. If no indexing structure can be used, the run time complexity degrades to $\mathcal{O}(n^2)$.

The reasons why DBSCAN is so frequently used are manifold. Firstly, the algorithm can identify a variable number of clusters with arbitrary shape – in contrast to e.g. the *k*-means algorithm that constructs exactly k clusters and this number has to be specified by the user beforehand. Since it is often unclear how many clusters exist in a data set, this property of DBSCAN is a large advantage over many other approaches. Secondly, not every point of the data set has to be added to one of the clusters. Noise is very common in real world data sets but often not desired to be part of a cluster so that additional filtering would have to be done if it was included. Thirdly, using density information to create clusters is often a natural choice if spatial data is used. The downsides of DBSCAN are that proper choices for the values of ϵ and N_{min} are sometimes difficult to make and that data sets with varying density cannot be clustered properly since global parameters for ϵ and N_{min} are used.

5.1.2 *DBSCAN Variants*

In the years following the publication of DBSCAN, multiple variants were proposed. The authors of the DBSCAN algorithm proposed GDBSCAN [219], where the *G* stands for *generalized*. In their work, the classical DBSCAN is extended by allowing any neighborhood definition as long as this relation is both symmetric and reflexive and the number of neighbors is no longer defined to be the sheer number of objects in a neighborhood but rather non-spatial features of objects can be taken into account. The claimed advantage is that GDBSCAN can cluster not only points but also extended objects and can take non-spatial features into account. In practical applications, however, GDBSCAN is much less used than the "special case" DBSCAN.

A parameter free variant is called *Distribution Based Clustering of Large Spatial Databases (DBCLASD)* and aims to create clusters based on nearest-neighbor dis-

tances and the assumption that points within a cluster are uniformly distributed [282]. Loosely speaking, a point is added to a cluster if the distribution of its nearest neighbor distances aligns well enough with the assumed uniform distribution of a cluster. A χ^2 test is used to estimate whether the distance-distribution of a candidate point fits with the expected distribution and if so, the point is added to a candidate cluster. In terms of run time efficiency, DBCLASD performs slightly worse than DBSCAN but indexing structures can again help reducing the time-consuming neighborhood searches.

A popular improvement over DBSCAN is the *OPTICS (Ordering Points To Identify the Clustering Structure)* algorithm [2]. OPTICS heals the shortcoming of DBSCAN that only clusters with similar density can be created and at the same time the influence of the parameter ϵ is reduced. In contrast to DBSCAN, a point is not defined as a core point if more than N_{min} neighbors are in an ϵ-region around the point but rather distances are calculated which define how large ϵ would have to be chosen to make the point a core point. In this sense, the parameter ϵ can be understood as a maximum radius that should be checked. A so-called *core-distance* and a *reachability-distance* are defined which in addition to an ordering of the data set form the basis for the main loop of the algorithm. The output of the algorithm is an ordering of the input points along with corresponding reachability values. If the points are displayed in the proposed order on the x-axis and the reachability values are drawn on the y-axis, then clusters can be identified as the "dents" in this reachability-plot. Cutting this graph at a fixed y-value corresponds to the application of DBSCAN with this value set as DBSCAN's ϵ. In OPTICS, however, not a fixed value for cutting is used, but rather the steepness of the decrease or the increase in reachability values is considered and cuts are made based on regions with a steep decrease followed by a steep increase in the reachability plot.

The algorithm *HDBSCAN (Hierarchical Density-Based Spatial Clustering of Applications with Noise)* is one further improvement over the OPTICS algorithm and tries to heal the downside of DBSCAN that the two parameters ϵ and N_{min} have to be defined manually beforehand [32]. Hierarchical DBSCAN uses only the parameter N_{min} that it is now used as a threshold for the minimum size of the resulting clusters. The ϵ parameter used in DBSCAN or OPTICS is no longer required. Similar to OPTICS, a tree-like structure is created from which in principle all DBSCAN clustering results with arbitrary ϵ could be extracted. In HDBSCAN, however, only the most stable clusters are extracted which do not necessarily have to have the same density. This is done by condensing a generated cluster hierarchy so that all cluster candidates with less than N_{min} points are removed. Details can be found in [32].

The algorithm IDBSCAN (Improved DBSCAN) [18] does not alter the way that clusters are created but rather introduces a sampling method so that run time and memory consumption of the algorithm are reduced.

5.1.3 *Clustering Applied to Radar Data*

With the increase in resolution of modern radar sensors, the need for grouping targets originating from the same object has risen. For example, in [137] a density-based algorithm is proposed for clustering targets of an airport surveillance radar. The algorithm was designed to work with different density thresholds in each feature dimension but density variations within one feature dimension are not considered. In [290], an implementation of the k-means clustering algorithm on the GPU is presented with focus on clustering radar reflectivity data used for weather prediction. Their implementation uses the massive parallelism options GPUs offer to increase the execution speed of k-means. However, no radar specific adaptions were made to the algorithm so that in contrast to the publications title a rather generic clustering algorithm was implemented.

In [121], however, the DBSCAN clustering algorithm is modified specifically for the application on automotive radar data. It is argued that since the resolution of a radar sensor is (approximately) constant in radial and azimuth direction (i.e. in a polar grid), it is not uniform in a Cartesian grid. Therefore objects close to the sensor can create more targets than far distant objects which cover only very little range-azimuth bins. This effect is of course also visible in the data set used in this thesis, cf. Fig. 4.12. With a fixed value of ϵ in the DBSCAN algorithm, far distant objects cannot be clusters since less targets are measured in the given neighborhood region. Additionally, objects close to the sensor can be hardly separated for larger ϵ values since then the nearby targets of the different objects would be grouped together. Position depended values for ϵ and N_{min} are proposed to take the varying sampling densities into account. These are implemented implicitly by working on a grid with scaled range and azimuth cells. The scaling factors of each cell then depend on the range and angle resolution as well as on the range and angle bin itself. Since these scaling factors can be computed beforehand and stored in a lookup-table, the run time complexity of DBSCAN is maintained.

Basic DBSCAN is applied to radar data in [230], where different ϵ values are used for the spatial direction and the space of Doppler velocities. Additionally, stationary objects are excluded from the clustering step by using a threshold for the Doppler velocity. Similarly, also in [173] and [140] basic DBSCAN is used for clustering of automotive radar data without tuning it for this specific application.

In [260], small changes to the OPTICS algorithms are made to leverage its performance on radar data. More specifically, OPTICS is used to cluster regions in the range-Doppler matrix so that in contrast to the other methods discussed in this section, no individual targets are considered. The changes in the algorithm include for example that the valley detection in the reachability plot is modified to allow

for clusters separated by one target and that a bounding box check is implemented to add all points of the interior of a cluster. Additionally, the distance metric is scaled by the amplitudes of the return signal.

A border following method on 2D binary images of radar targets is presented in [247]. For the three different feature pairs range-azimuth, range-Doppler velocity and azimuth-Doppler velocity, binary images with a fixed grid size are created. Then, a slightly modified version of the border following algorithm presented in [285] is applied to the image to detect connected groups of targets. The advantages and disadvantages of each of the three image representations is discussed with the conclusion that the range-azimuth representation works best in the presented scene.

Based on the work presented in this thesis and the article previously published in [234], in [223] a two-stage clustering approach is presented. Since only dynamic objects are considered for clustering, the static background is filtered in a first step by taking both the density of the targets and their Doppler velocity into account. Afterwards, a first clustering step in a low dimensional feature space is performed. In a second step, the cluster candidates are merged together based for example on a velocity estimation of the clustered objects. Similarly as in the work presented below, an optimization framework is used to learn the free parameters of the filtering step and the two clustering steps from a data set. Additionally, the authors use also the idea presented below that only points with a sufficiently larger Doppler velocity can become core points the DBSCAN algorithm. Access to the data set recorded for this thesis was granted to the author.

5.1.4 *Simulated Annealing*

The optimization scheme *simulated annealing* [124] displays a deep connection between statistical mechanics and traditional optimization problems. Kirkpatrick et al. noticed that the Metropolis algorithm [161] can be applied to various combinatorial problems, as long as a few basic ingredients are provided. The original Metropolis algorithm was designed to calculate equilibrium properties for a set of atoms at a specific temperature. In statistical mechanics, the Boltzmann factor $\exp(-E/k_B T)$, in which k_B is the Boltzmann constant, weighs the probability that a state with energy E is encountered at temperature T. This means that the higher the temperature T of a system, the larger the probability to detect the system in a state with energy E, and the lower the energy E of a state, the more likely it is to encounter it. In the Metropolis algorithm, a starting configuration of the atoms is chosen before one atom is selected from the set, randomly displaced and the change in energy ΔE of the system is calculated. If $\Delta E \leq 0$, then the move

is accepted since a more favorable energy state is found. If on the other hand $\Delta E > 0$, then the value $p_1 = \exp(-\Delta E / k_B T)$ is compared to a random number p_2 generated uniformly over the range $(0, 1)$ and the move is only accepted if $p_1 > p_2$. This implies that moves which increase the energy by a large positive ΔE are only likely at higher temperatures and become more unlikely the lower the temperature gets. The temperature T of the system is slowly decreased while the presented basic steps are repeated. In this way, the system can explore states with different energies at higher temperatures and settle into a global minimum the lower the temperature gets. Local optima can be left at non-zero temperatures since then states with $\Delta E > 0$ can be accepted.

The ingredients needed to apply simulated annealing to an optimization problem are rather easy to collect: a description of the current configuration of the system, a generator of moves through the configuration space, a score/energy function and an annealing schedule. The latter describes how fast the "temperature" of the system decreases, i.e. how long the configuration space can be explored before the system no longer (or only very unlikely) accepts moves with $\Delta E > 0$.

The use of a Boltzmann distribution for modeling the acceptance probabilities is motivated by physical applications. For other optimization schemes, however, there is no real need to utilize this specific distribution. One downside of the Boltzmann distribution is that a logarithmic annealing scheme of the form

$$T_k = T_0 \frac{\log k_0}{\log k} \quad \Rightarrow \quad T_{k+1} = T_k - T_0 \frac{\log k_0}{k(\log k)^2} \quad \text{for} \quad k \gg 1 \tag{5.1}$$

has to be chosen in order to guarantee a proper search through the parameter space. Here, k denotes the current step in the algorithm an T_k the temperature at step k.

In *adaptive simulated annealing* [110], [111], a different probability distribution is chosen such that *exponential* decrease of temperature is possible:

$$T_k = T_0 \exp(-ck). \tag{5.2}$$

For more details on the derivation of this distribution, see [111]. Additionally, adaptive simulated annealing introduces the concept of *reannealing*, i.e. increasing the temperature from time to time to sample parameter regions with little impact more roughly and to escape local minima.

The score function that will be introduced in the next section in Eq. (5.5) is in general neither differentiable nor continuous. Gradient-based optimization schemes can therefore not be used and other methods are needed to find optimal parameters of the clustering algorithm.

Simulated annealing fulfills this requirement since multi-dimensional non-continuous score functions can be handled. Other non-gradient based optimization

schemes like differential evolution [248] could be chosen as well for the task at hand but since no further requirements are made for the optimization scheme and no better solutions are expected from more complicated algorithms, (adaptive) simulated annealing is chosen due to its simplicity.

5.2 SUPERVISED CLUSTERING

This section is based on the previously published work [234]. In the literature, the term "supervised clustering" is used in different ways than it is used here. In this work, "supervised clustering" means that an unsupervised clustering algorithm is extended by a supervised component, which fixes the free parameters of the unsupervised method. Thereby, domain knowledge can be incorporated into the previously purely unsupervised algorithm, resulting in a "supervised clustering algorithm". In [61], [89], the term is used to describe algorithms that create dense clusters in some feature space but with the additional constraint that these clusters should contain only members of the same class. The class label is known to the algorithm and can be taken into account during clustering. The second way the term "supervised clustering" is used in the literature is that examples for clusters are provided and the algorithm learns from these examples how the clustering should be done. In [66], a support vector machine is used to learn a similarity measure between two objects. Approaches in which "must-link" and "cannot-link" information are provided are strongly related to this approach. After a similarity value for each pair of items is calculated, correlation clustering is applied on the resulting similarity matrix. A similarity neural network is used in [159] for the computation of the similarity values which are then used in k-means clustering step during the distance calculation.

In this work, the goal of clustering radar data is to identify all targets that belong to the same dynamic object. This cluster should be valid over multiple time steps so that not only the targets of one object obtained from one single scan are grouped together but rather data from all sensors obtained in a certain time window is considered. Targets that do not belong to any dynamic object should not be included in any cluster and only one cluster per true object should be created. Difficulties that arise during clustering of radar data are manifold. For example, while the number of targets measured on an object decreases with increasing distance, the nearest-neighbor distance between targets increases. Micro Doppler effects caused by moving wheels, arms, legs or any other small moving parts of an object create an inhomogeneous velocity profile so that fluctuations in the Doppler velocity value of spatially nearby points can be expected. Additionally, ambiguities in the Doppler value (see also 3.1.3) can cause large deviations of the Doppler velocity of

neighboring targets. Other sensor artifacts and mirror effects can also create targets with non-zero Doppler velocity, which ideally should not be part of any cluster.

In Fig. 5.2a and Fig. 5.2b, the spatial difference Δr between nearest neighbor targets of the same moving object are plotted against the average radial distance r the object is located at as well as against the average Doppler velocity $\hat{v}_r$ of the object. Data from all sensors during time windows of length 100 ms were considered for the nearest neighbor searches. The nearest neighbor difference Δr increases with both r and $\hat{v}_r$. Because data from multiple time steps is considered, the motion of the objects under consideration influences the distribution of the measured targets. The faster an object moves, the further apart are the targets measured at two different points in time and hence the nearest neighbor distance increases on average. With less targets being detected from far distant objects, it becomes more likely that the nearest neighbor stems from a measurement cycle of a different sensor so that again the motion of the object has an influence here. Notice that it is necessary to choose a finite time window for this analysis and not separate radar scans, since in the clustering and classification step also data from a certain time range from multiple sensors are considered. In the bottom row of Fig. 5.2, the difference in Doppler velocity $\Delta\hat{v}_r$ of the same nearest neighbor targets as calculated before are displayed versus the average radial distance of the respective objects (Fig. 5.2c) and versus their average Doppler velocity (Fig. 5.2d). The large standard deviations (shaded area) around the mean value (solid line) already indicate that trends are less reliable and interpretations need to be done more carefully. One striking feature in Fig. 5.2d is that for large Doppler velocities the value of $\Delta\hat{v}_r$ drastically increases again. This might be explained by the fact that especially for fast moving objects the sensor cannot resolve the ambiguous Doppler velocity correctly so that nearby targets are erroneously reported with very different Doppler velocities. For slow moving objects the values of $\Delta\hat{v}_r$ are on average larger than for faster moving objects. Two effects can cause this: Micro-Doppler effects of moving pedestrians – with one stationary leg and one moving leg – create a large difference in the Doppler velocities of nearby targets and objects moving perpendicular to the sensor (e.g. a car moving from left to right in front of the ego-vehicle) have a strongly varying Doppler velocity pattern including a sign change of the measured Doppler velocities. The almost constant non-zero value of $\Delta\hat{v}_r$ in Fig. 5.2c is probably a geometric result. An extended moving object seen by one radar sensor will create multiple targets whose Doppler velocities differ due to the difference in angle between the measured target and the sensor. The same holds true if multiple sensors measure the same object so that by design of the measurement a non-zero difference in the Doppler values can be expected.

The idea is now to *learn* suitable parameters ϵ and N_{min} of the DBSCAN algorithm for different regions of the 2D feature space spanned by radial distance and Doppler

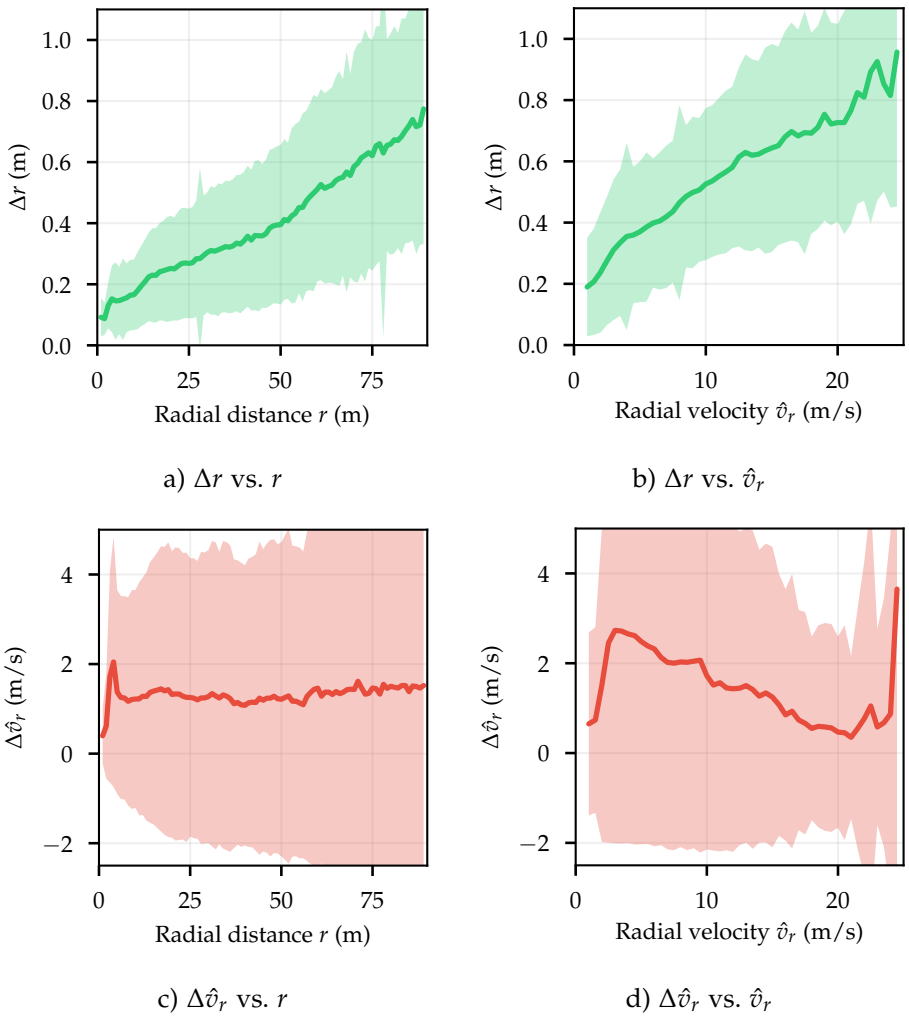


Figure 5.2: Statistics about nearest neighbor differences in space and velocity. The solid lines mark the average value of all measurements that fell into the respective range or velocity bin and the shaded areas are regions with plus/minus one standard deviation. Only the absolute value of the Doppler velocity $\hat{v}_r$ is considered.

velocity. Targets close to the radar sensor will therefore be clustered with a different set of parameters than targets far from the sensor and fast moving objects with high Doppler velocities will be treated differently than targets with Doppler values close to zero. Learning these values directly from the data set has the upside that no "magic-thresholds" have to be introduced as a best guess for parametrizing DBSCAN. A performance metric can be directly used to evaluate the clustering results and thereby quantify the performance of the algorithm. In order to find suitable parameters for the clustering of a specific region of the 2D feature space,

the already introduced optimization scheme *simulated annealing* is applied so that different parameter sets are explored and gradually the best performing set is obtained.

For the rest of this chapter, the following definitions are used. A cluster $X = \{q_1, \dots, q_n\} \subset \mathcal{P}$ is a subset of the whole set of radar targets $\mathcal{P} = \{p_1, \dots, p_N\}$ that were measured by all sensors in a given time range. From each target p, four properties are used: the measurement time p_t, the Cartesian position in car coordinates p_x and p_y (see also Section 3.2) and the ego-motion compensated Doppler velocity p_v. A ground truth cluster of object i is represented as $Y_i = \{t_1, \dots, t_m\} \subset \mathcal{P}$, where the information which target belongs to which cluster is obtained from the labeling stage discussed in Section 4.2. Application of a clustering algorithm results in a set $\chi_i = \{X_1, \dots, X_{n_{clst}}\}$ of predicted clusters for ground truth object i with single ground truth cluster Y_i.

5.2.1 *Score Function to Assess Clustering Results*

For evaluating how well a specific clustering algorithm performs on the radar data set, a score function is needed. This score function is then also used during the optimization step as a cost function. Let s be the score function which takes as arguments a set of created clusters χ_i for one object i as well as the ground truth cluster Y_i and maps this onto a real number $s(\chi_i, Y_i) \in [0, 1]$. A perfect clustering results in a score of one, whereas the score function returns the value zero if no clusters at all were created for a given object.

Let $\chi = \bigcup_{j=1}^{n_{clst}} X_j$ be the union of all clusters created for one ground truth cluster Y_i. Then the *true positives* are defined as $TP = |\{p \in \mathcal{P} \mid p \in Y_i \wedge p \in \chi\}|$, the *false positives* are defined as $FP = |\{p \in \mathcal{P} \mid p \notin Y_i \wedge p \in \chi\}|$ and the *false negatives* are given by $FN = |\{p \in \mathcal{P} \mid p \in Y_i \wedge p \notin \chi\}|$. Then the F_1 score is defined as the harmonic mean between precision $Pr = TP/(TP + FP)$ and recall $Re = TP/(TP + FN)$:

$$F_1 = \frac{2}{1/Pr + 1/Re} = 2\frac{Pr \cdot Re}{Pr + Re}. \tag{5.3}$$

The F_1 score is a suitable measure for how well the clustering algorithm managed to assign targets to the correct ground truth object. It takes into account how many of the object's targets are part of the cluster (TP), how many targets were erroneously not included in any of the generated clusters (FN) and how many targets of different objects or of the static environment were added (FP). The precision Pr then gives the ratio of the correctly identified targets to the predicted

targets and the recall value describes the ratio of the correctly clustered targets to all true targets of the object.

However, the F_1 score alone is not sufficient to describe the clustering result since a clustering algorithm that creates many clusters for one object should be considered worse than an algorithm that creates only one cluster. Therefore, the *variety* measure V is introduced, which penalizes the creation of many small clusters:

$$V = 1 - \eta \cdot \tanh(\alpha[n_{clst} - 1]), \qquad \eta = 1 - \max_j \frac{|x_j \cap Y_i|}{|Y_i|}. \tag{5.4}$$

The variety measure is defined for one ground truth object i and the factor η takes into account the size of the created clusters. If for example one large cluster and many smaller clusters were created for one dynamic object, then this should be ranked higher than a clustering result with equally many clusters, which all have the same size. The intuition behind this is that if one large cluster is created for the object, then the main structure of the object is covered and any following algorithm can work e.g. with the emerging shape information more reliably than if the object was fragmented into equally many but on average smaller clusters. The parameter α controls how fast the variety measure decreases with increasing number of created clusters n_{clst}. In this work, the parameter is set to $\alpha = 0.3$.

The total score function is then composed of the harmonic mean of the F_1 score and the variety measure V:

$$s(\chi_i, Y_i) = 2\frac{F_1(\chi_i, Y_i) \cdot V(\chi_i, Y_i)}{F_1(\chi_i, Y_i) + V(\chi_i, Y_i)}. \tag{5.5}$$

The V-measure introduced in [212] has some similarities with the metric defined in this work. The reason why the V-measure is not used directly here is due to the presence of static targets in the measured radar point cloud. The goal of the clustering algorithm is to group all targets of dynamic objects together and leave targets of static objects out of any cluster. During the calculation of the V-measure, the conditional entropy over all clusters is calculated and every created cluster is treated equally. It is unclear how static targets could be integrated into this scheme. If static targets were treated as if they were all part of the same static cluster, then the V-measure could be directly calculated but the total score would depend on the number of static targets in the scene. This can be made more explicit with an example: Consider the situation in which only one dynamic object is present in a scene with some static environment and a clustering algorithm correctly identifies the dynamic object but erroneously adds some of the static targets to the cluster of the dynamic object. Then the score for the clustering result reported by the V-measure gets higher the more static targets are in a scene even though the overlap

between static targets and the dynamic cluster remains constant. This property is not desired since clustering of dynamic objects should only depend on how well the dynamic objects are clustered and not how many static clusters are in a scene. If on the other hand static targets were not treated as belonging all to the same cluster, then false positive static points in a dynamic cluster would not be considered at all.

5.2.2 *Method and Training*

In vanilla DBSCAN only one distance threshold ϵ is used for the definition of the neighborhood regions around a point p, namely $N_\epsilon(p) = \{q \in \mathcal{P} \mid d(p,q) \leq \epsilon\}$. For the clustering of radar data, however, one combined distance function that takes all four features of a target into account is not reasonable. This would require a weighting for example between the spatial dimensions and the Doppler velocities. Using different thresholds for each feature dimension seems therefore a reasonable extension. With the three thresholds ϵ_r, ϵ_v and ϵ_t, the ϵ-region around point $p \in \mathcal{P}$ is defined as

$$N_\epsilon(p) = \{q \in \mathcal{P} \mid |p_x - q_x| \leq \epsilon_r \wedge |p_y - q_y| \leq \epsilon_r \\ \wedge |p_v - q_v| \leq \epsilon_v \wedge |p_t - q_t| \leq \epsilon_t\}. \tag{5.6}$$

The two spatial dimensions were treated equally because the clustering should be rotationally invariant. The condition can be reformulated and shortened, if the Chebyshev distance

$$d_{\text{cheby}}(\vec{p}, \vec{q}) = \max_i |p_i - q_i| \tag{5.7}$$

is used as a metric and the vector $\vec{p} = (p_x, p_y, p_v, p_t)$ is scaled with the inverse ϵ-threshold prior to the distance calculation:

$$\vec{p}' = (p_x/\epsilon_r, p_y/\epsilon_r, p_v/\epsilon_v, p_t/\epsilon_t). \tag{5.8}$$

Then the neighborhood condition can be reformulated as

$$|p_x - q_x| \leq \epsilon_r \wedge |p_y - q_y| \leq \epsilon_r \wedge |p_v - q_v| \leq \epsilon_v \wedge |p_t - q_t| \leq \epsilon_t \tag{5.9}$$
$$\Leftrightarrow |p'_x - q'_x| \leq 1 \wedge |p'_y - q'_y| \leq 1 \wedge |p'_v - q'_v| \leq 1 \wedge |p'_t - q'_t| \leq 1 \tag{5.10}$$
$$\Leftrightarrow \max_i |p'_i - q'_i| \leq 1 \qquad \Leftrightarrow \qquad d_{\text{cheby}}(\vec{p}', \vec{q}') \leq 1. \tag{5.11}$$

This results in the simple term $N_\epsilon(p) = \{q \in \mathcal{P} \mid d_{\text{cheby}}(\vec{p}', \vec{q}') \leq 1\}$, where the different ϵ thresholds are now implicitly incorporated in the scaled feature vectors of the targets.

Now the goal is to find suitable values for the three ϵ-thresholds as well as for the N_{min} parameter. As it cannot be expected to find one suitable set of parameters that works for all distances and object velocities, the whole point cloud $\mathcal{P}$ is divided into six different range regions and five Doppler velocity regions. For each of these 30 individual subspaces, an individual set of parameters should be found.

To extract training data from the labeled point clouds, so-called *sectors* are created for each individual object. A *sector* is defined as a collection of the object's reflections and surrounding measurements up to 6m away from the object borders. If an object leaves a specific sector during its observation (for example an approaching car), the measured targets are split into multiple regions. This process results in a list of sectors for each individual range-velocity region and is repeated for training and test data sets.

One further extension made to DBSCAN is that the algorithm is only allowed to consider those targets as core points of a clusters if their Doppler-velocity exceeds a threshold of 0.4 m/s. Thereby, static or slowly moving targets can still be added to a cluster if they are density-reachable but they cannot work as seed points for further expansion of the cluster.

Adaptive simulated annealing with the score function defined in Eq. (5.5) is applied to each range-Doppler region so that 30 different optimal parameter sets $\epsilon_r, \epsilon_v, \epsilon_t$ and N_{min} are iteratively obtained. These parameters are then used for clustering the test data and comparison with ground truth clusters and the score function is done. This process allows incorporating domain knowledge into the DBSCAN clustering since now the radar specific distance and Doppler velocity dependencies are represented by the *learned* thresholds.

To compare the clustering results obtained by this method, also standard DBSCAN with fixed parameters as well as HDBSCAN are applied to the test set. From now on, the standard DBSCAN algorithm will be referenced as DBSCAN$^-$ and the new variant with learned parameters will be referenced as DBSCAN$^+$. For DBSCAN$^-$, the parameters $\epsilon_r = 1\,\text{m}$, $\epsilon_v = 5\,\text{m/s}$, $\epsilon_t = 0.2\,\text{s}$ and $N_{\text{min}} = 1$ are used as expert guesses for reasonable values. For HDBSCAN a metric has to be defined. For example, if an Euclidean metric is used, the distance between two points is calculated as $d(p,q)^2 = \sum_i (p_i - q_i)^2$, $i \in \{x, y, v, t\}$. This expression is only meaningful and useful, if each dimension is scaled by a characteristic length ℓ_i, so that the dimensions have the same impact on the distance. Two approaches to find suitable values for the ℓ_i are shown here: i) Nearest neighbor distances are calculated from the ground truth clusters in space, time and velocity for different range-velocity regions and thereby a set of ℓ_i is obtained for each region. ii) The learned parameters ϵ_r, ϵ_v and ϵ_t from DBSCAN$^+$ are used for the ℓ_i. The two approaches are referenced as HDBSCAN$^{(i)}$ and HDBSCAN$^{(ii)}$, respectively.

Algorithm	$\bar{s} \pm \sigma_s$	$\hat{s}$	$\overline{V} \pm \sigma_v$	$\overline{Pr} \pm \sigma_{Pr}$	$\overline{Re} \pm \sigma_{Re}$
HDBSCAN$^{(i)}$	0.60 ± 0.35	0.72	0.90 ± 0.16	0.61 ± 0.37	0.88 ± 0.17
HDBSCAN$^{(ii)}$	0.66 ± 0.35	0.82	$\mathbf{0.92 \pm 0.14}$	0.65 ± 0.35	0.92 ± 0.14
DBSCAN^{-}	0.81 ± 0.21	0.88	0.80 ± 0.24	$\mathbf{0.88 \pm 0.20}$	0.98 ± 0.08
DBSCAN^{+}	$\mathbf{0.87 \pm 0.17}$	**0.94**	0.90 ± 0.16	0.86 ± 0.21	$\mathbf{0.98 \pm 0.07}$

Table 5.1: Overview of the clustering scores.

5.2.3 *Evaluation*

The extracted sectors from all annotated recordings were used for five-fold cross validation. The results shown in this section are summarized over all five test folds. For each object in the test folds, the total score $s(\chi_i, Y_i)$ and its components $V(\chi_i, Y_i)$, $Pr(\chi_i, Y_i)$, $Re(\chi_i, Y_i)$ were calculated and stored. From these bare values, arithmetic means $\bar{s}, \overline{V}, \overline{Pr}, \overline{Re}$ along with their standard deviations $\sigma_s, \sigma_V, \sigma_{Pr}, \sigma_{Re}$ were computed and the median score $\hat{s}$ was calculated. In Table 5.1, these values are presented for the four different clustering approaches.

The DBSCAN variant with learned parameters outperforms the other methods with regard to the mean and median values of the total scores $s(\chi_i, Y_i)$. Especially the variety measure V and the recall value Re are large for this method. Interestingly, DBSCAN^{-} with fixed parameters has a higher precision than the variant with learned parameters, indicating that less false positives are inserted into the proposed clusters. The variety scores show that DBSCAN^{+} creates less clusters for one object with more true positive points per cluster compared with DBSCAN^{-}. Combining these two observations allows the conclusion that DBSCAN^{+} captures an object with less created clusters at the cost of including some false positive points. The number of true positive points left out of a cluster is almost the same for the two methods as indicated by the similar recall values.

The scores of the two HDBSCAN approaches are even below the scores of DBSCAN^{-}. This might be explained as follows: For DBSCAN we could restrict the core points to those with some minimal Doppler velocity. By design, this is not possible for HDBSCAN. To stop HDBSCAN from adding too many static points from the surrounding to a cluster, one would have to increase the distance between moving and static points manually. This, however, would also hinder HDBSCAN to add static points of a moving object to a cluster, for example the bottom part of a turning

wheel. An unmodified version of HDBSCAN therefore does not seem to be a good choice for our task.

The high standard deviations in all the computed scores demand some extra consideration. In Fig. 5.3, the distribution of the score values obtained from DBSCAN$^-$ and DBSCAN$^+$ are plotted and the count differences of each bin are visualized. This representation shows that the score distributions are highly skewed towards higher scores. The high standard deviation in the scores stems from the fact that in both methods the full range of scores is covered and the score counts decrease only slowly in the direction of lower scores. The visualization of the count differences shows that the greatest difference exists in the region of scores greater than 0.85: Significantly more sectors obtained a high score when clustered with DBSCAN$^+$. On the other end of the score range, there is almost no difference in the counts because both algorithms have problems with a few rare situations in which the measured reflections have low velocities or only a few reflections were measured. If no cluster was created for an object, a score of zero was assigned. In the difference plot in Fig. 5.3 it is visible that for DBSCAN$^+$ the smallest score bin has a higher count so that DBSCAN$^+$ failed more often to create at least one cluster for a true object than DBSCAN$^-$.

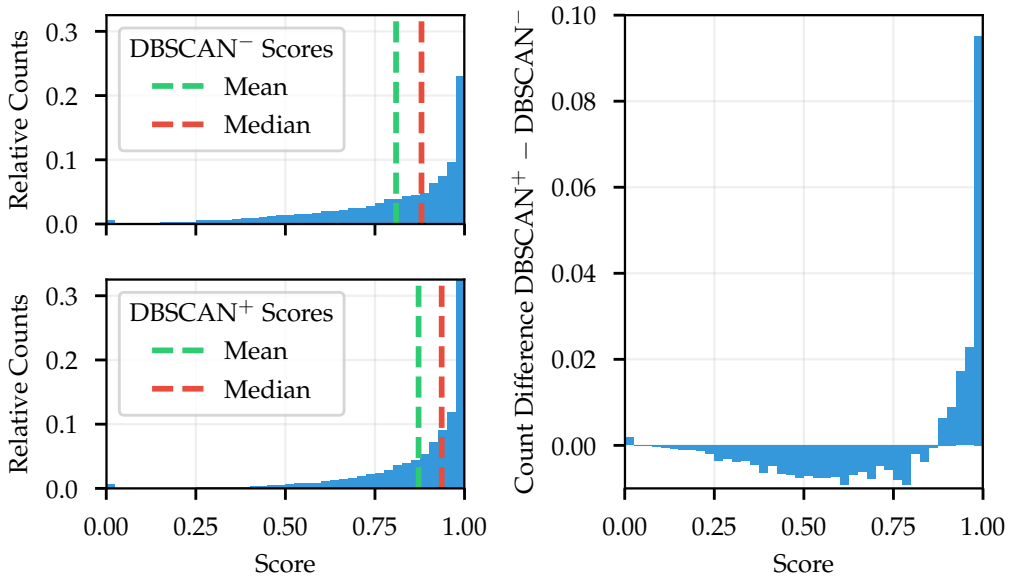


Figure 5.3: Histograms displaying the score distribution of DBSCAN$^-$ and DBSCAN$^+$ as well as the distribution of the bin count differences. The histograms are normalized so that the bins sum to one.

To justify the claim that despite the large standard deviations DBSCAN$^+$ performs better than DBSCAN$^-$, the 95 % bias corrected and accelerated bootstrap confidence

interval [60] for the mean difference of the scores $s(\chi_i, Y_i)^+ - s(\chi_i, Y_i)^-$ is calculated. The symbols $s(\chi_i, Y_i)^+$ and $s(\chi_i, Y_i)^-$ indicate the score of the i-th sample obtained by using DBSCAN$^+$ and DBSCAN$^-$, respectively. This interval describes that in 95 % of the cases the interval encloses the true difference between the scores of the two methods. The interval was calculated to be $[0.060, 0.068]$ so that in 95 % of the cases the DBSCAN$^+$ method has a higher score by at least 0.06.

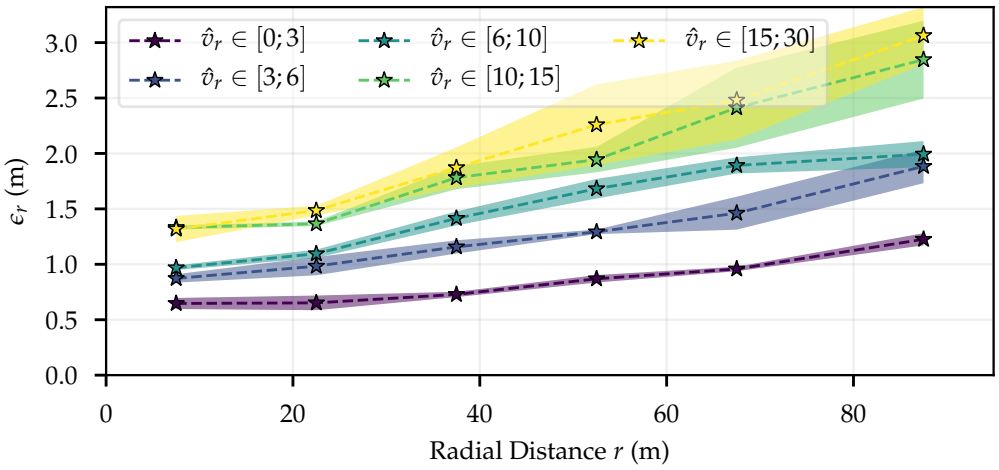


Figure 5.4: Averaged values for ϵ_r plotted against radial distance for each of the five different velocity intervals. Shaded areas indicate regions with plus/minus one standard deviation around the mean value.

The final parameters obtained from adaptive simulated annealing proofed to be relatively invariant of the initial values. For ϵ_r, the search space was limited to 0.4 m to 5 m, for ϵ_v the interval 0.2 m/s to 8 m/s was used and for ϵ_t values from 0.1 s until 0.8 s were allowed to be selected during the annealing process. The optimized parameter ϵ_r for the different range and velocity regions is depicted in Fig. 5.4 and Fig. 5.5. From the five test folds, average values and standard deviations were computed for each of the range/velocity regions. The mean values are represented by the $\star$ symbols in the plot and the shaded areas indicate the standard deviation. Similar plots for ϵ_v and ϵ_t can be found in Appendix A.4.

In Fig. 5.4, it can be seen that the threshold ϵ_r increases with increasing radial distance to the object – just as it is also indicated by the nearest neighbor differences Δr in Fig. 5.2a. This plot also hints that there is a clear structure in the velocity domain since ϵ_r in one range bin is almost always smaller for lower Doppler velocities. This is again illustrated in Fig. 5.5 where now ϵ_r is explicitly plotted against the Doppler velocities.

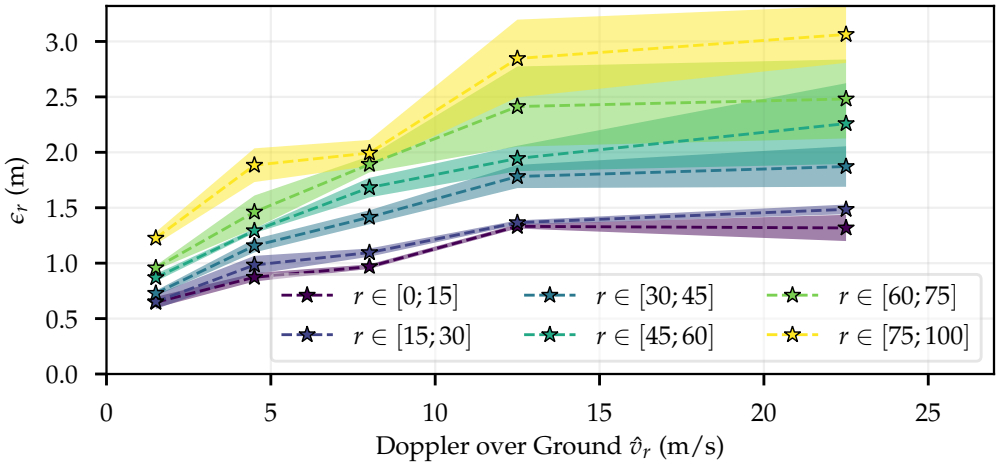


Figure 5.5: Averaged values for ϵ_r plotted against Doppler velocity for each of the six range regions. Shaded areas indicate regions with plus/minus one standard deviation around the mean value.

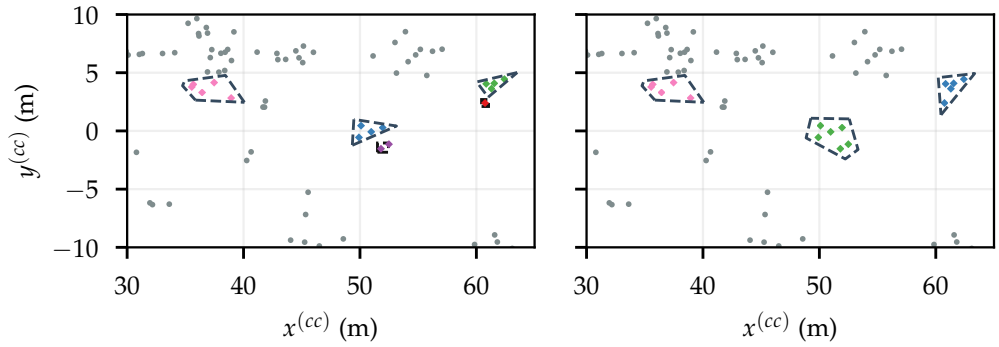


Figure 5.6: Example clustering result of a scene with two approaching cars and one car in front of the ego-vehicle. Left: Clustering result of the DBSCAN$^-$ algorithm. Right: Result of the DBSCAN$^+$ method with learned parameters.

Finally, in Fig. 5.6 an example scene is displayed with the clustering results of DBSCAN$^-$ and DBSCAN$^+$. Both algorithms managed to cluster the first oncoming car perfectly, whereas only DBSCAN$^+$ was able to create exactly one cluster for each of the other two vehicles. Especially at larger distances, the difference between the two algorithms becomes visible. In this concrete situation, the difference in Doppler velocity caused DBSCAN$^-$ to create multiple clusters. However, in the cases of Doppler ambiguities in which the sign of the Doppler value switches, also DBSCAN$^+$ was only rarely able to include these points. Some post processing steps

that include all points of the interior of the convex hull of the cluster could solve this problem partially.

The parameters obtained here are possibly biased towards the majority classes of the data set. Further improvements could be made if balancing is introduced so that semantic classes which are underrepresented in the data set are weighted higher in the score function than majority classes so that the learned parameters work not only well on e.g. the `Passenger Car` class.

6

CLASSIFICATION OF CLUSTERED RADAR DATA

After discussing methods for clustering of radar data in the last chapter, this chapter presents approaches for classification of these formed clusters. This means that the algorithms discussed here will use clusters – or time slices of these clusters – as input and a vector of class probabilities will be the output. Since the clustering performance has direct influence on the classification, evaluations are performed on ground truth clusters as well as on predicted clusters. For some algorithms a *feature extraction* step precedes the actual classification. Manual feature extraction as well as automatic feature extraction will be demonstrated, evaluated and discussed. In Fig. 6.1, the process pipeline is illustrated.

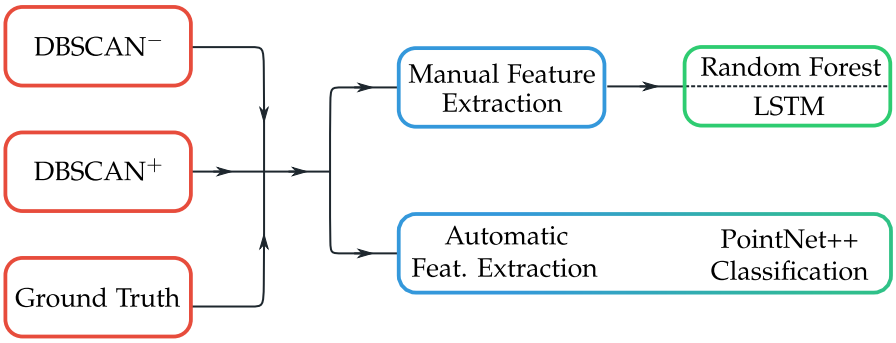


Figure 6.1: Process pipeline for classification based on clustered data. From the created clusters (red boxes) either feature vectors are extracted manually (blue box) and then classified (green box) or feature extraction and classification is performed in a combined step (bottom part).

In the first part of this chapter, related work and state of the art is discussed. Afterwards, approaches are introduced which rely on a manual feature extraction step and then classify the generated feature vectors. Two different classifiers with different feature sets are used: a random forest and an LSTM (see Fig. 6.1 top part). It is discussed how changes over time can be incorporated into the classification scheme so that predictions are not only made on a single feature vector but rather on a sequence of feature vectors. In the third part of the chapter, the bare clusters are used as input so that the manual feature extraction step becomes obsolete

(bottom part of Fig. 6.1). For all of these methods various scores are presented so that a performance comparison is possible. Thereafter, the human performance on the classification task is evaluated and compared to the outputs of the algorithms introduced before.

Finally, situations are considered in which pedestrians are tracked over a longer period so that features from a greater time window can be extracted. Classification of more fine-grained movement patterns as well as a height classification of pedestrians will be presented. This last part was done together with a master student and culminated in his master's thesis [104] and a publication [105].

To summarize, the following own contributions are made

- Comparison of different feature-based classification algorithms on both ground truth clusters and automatically predicted clusters
- Consideration of time information in the classification procedure
- Classification based on bare point clouds with automatic feature extraction
- Assessment of human performance on the classification task
- Body height estimation and movement classification of pedestrians using solely 2D radar data

The first item mentioned here is based on the publication "Comparison of random forest and long short-term memory network performances in classification tasks using radar" which was published beforehand [232]. The numbers presented in that paper differ from the ones in this thesis since the data set size has increased considerably in the meantime.

6.1 STATE OF THE ART

Compared to the image community where a sheer endless body of publications related to classification approaches exists, extracting semantic information from automotive radar sensors is a rather novel topic and not too many published articles cover this area. Often, the term "classification" is understood in the radar community as the task of deciding whether a detected target is *noise* or a real measurement. That is, a binary decision for the detection is made and no semantic information is added to a target. Literature covering this kind of classification is not discussed here since the *detection* stage is not part of this work. However, it is interesting to note that machine learning also starts to enter in these processing steps, which are usually performed without any learned parameters [28].

Before machine learning gained popularity, pedestrian classification was done with hand-made features extracted from range-profiles, Doppler spectrograms and RCS values [95]. In their work, only simulated data of a 24 GHz radar was used to discern between pedestrians and cars. Later, the same authors added a subsequent tracking stage to improve the performance [96], developed more expressive features [97], and analyzed the impact of different wave forms on the classification performance [98]. Generating features which work equally well for radar sensors operating on 24 GHz or 77 GHz is discussed in [167]. A small set of recordings from a 24 GHz radar and simulated data from a 77 GHz radar is used to train a *Support Vector Machine (SVM)* which discerns between pedestrians and vehicles. The main observation used in their approach is that Doppler spectra measured at different frequencies can be considered as scaled versions of each other. In [109] the task of distinguishing between pedestrians and vehicles is tackled again. As before, the Doppler spectrum of a 24 GHz radar is used as a basis for feature extraction. Handcrafted thresholds for the extent of the spectra and their change over time are used for the classification. An SVM with features based on handcrafted RCS thresholds is used by the authors in [135]. Similarly, in [1] the Doppler spectra are used as input to a CNN. The three classes vehicle, pedestrian and bicycle are distinguished and a small data set with staged scenes is used as input for training and evaluation. With a similar approach, but with entirely simulated data, human activities are classified based on micro Doppler variations on the Doppler spectrograms [93]. Another CNN approach is shown in [129], where only the distinction between the two classes "walking" and "noise" is made. Similarly, [176] analyzes how multiple radar sensors influence the classification performance of an SVM if different motion types are to be classified. All of the listed publications do not explicitly mention how the relevant parts of the full spectrum are extracted. Often, the starting point of these works is a single Doppler spectrogram or a series of spectrograms over time from one moving object from which features are calculated. The difficult task of identifying the parts of the complete measurement that belong to one object is not discussed. How the approaches can be extended to real-life scenarios with multiple moving objects remains open. However, this step is mentioned in [192], where DBSCAN is used in a first stage for clustering, followed by manual feature calculation, and finally classification by different tree-based algorithms. Again, a Doppler spectrogram is used as basis for feature extraction.

A shared feature of the presented works is that they work either on simulated data or on a minuscule data set with staged scenarios. Naturally, the performance metrics listed in these publications are quite high since a controlled environment is used. Additionally, a variety of radar sensors with different measurement capabilities is used which makes a comparison of the approaches almost impossible.

A second group of publications works on CFAR-level data, i.e. on the same data level as used in this work. Those articles which are based on the data set (or a smaller subset of it) that was collected for this thesis [222]–[224] are of special interest. Access to the data was granted to the authors for evaluation of their algorithms. In [222], a set of one-versus-one and one-versus-all LSTM-classifiers are trained and their performance is evaluated. A large handcrafted set of features is used and their importance is ranked. As input to the feature extraction process, ground truth clusters, which are the ones created by the labelers, are used. The impact on the total performance of using either ground truth clusters or automatically created clusters is analyzed later in this chapter. In [223], a multi-stage clustering framework is introduced which is supposed to be beneficial for the following feature extraction and classification steps. Similarly, in [224] the aforementioned approaches are extended by a novelty-detection algorithm so that semantic classes which were not seen before can be automatically detected. Three different variants of a novelty detection are tested which are all based on the uncertainty the one-versus-one and one-versus-all classifiers show on a sample. According to the authors, simply assigning a sample to the *novelty* class if all one-versus-all classifiers report a probability of less than some threshold performed best.

Another notable set of publications based on a large real world data set deals with the classification of static objects [144]–[146]. The authors of these publications also work on CFAR-level data but accumulate the measurements in a grid-mapping process. Clustering and classification algorithms are then applied to the created multi-layer grid maps. Only measurements with an ego-motion compensated Doppler velocity below $0.5\,\mathrm{m/s}$ are entered in the grid maps so that dynamic objects are entirely excluded. Different classifiers like random forests and CNNs, as well as ensembles of these algorithms are used to evaluate the performance of this approach [144].

In [185], a lidar sensor is used to extract regions of interest. These regions are then cut out of the range-Doppler map and passed through a neural network for classification. That is, a lidar sensor is used as an object detector and semantic information is added by the radar sensor. The data set used in their work contains real work data but makes up less than about 5 % of the data used in this thesis. Application of classification algorithms after a tracking stage to eliminate ghost objects is analyzed in [217]. A binary classifier is trained which discerns between real objects and false positive tracks. Handcrafted features like the lifetime of the track are used to train a simple neural network with fully-connected layers. Evaluation is done on 28 min of real world data recorded at one intersection.

A promising approach using both CFAR-level data and low-level information from the 3D data cube is presented in [177]. For each of the measured targets, the corresponding region of the 3D data cube is extracted and passed through a set of

convolution layers to generate feature vectors that describe the target's surrounding. Along with the measured position, Doppler velocity and RCS value of the target, the created feature vector is passed through another neural network for classification. After each measured target is ascribed with a class label, DBSCAN is used for clustering during which the predicted class label is taken into account.

6.2 COMPARISON OF DIFFERENT FEATURE BASED APPROACHES

In this section, two classification approaches are evaluated on the task of classifying the manually extracted feature vectors. First, classification is performed with a random forest and in order to incorporate time information, an LSTM is used as a second classifier.

The way the features are extracted from the formed clusters is explained in the next section. Thereafter, results from the two classifiers are presented and discussed. A complete comparison is done in Section 6.6 after also the approaches with automatic feature extraction have been evaluated.

6.2.1 *Feature Extraction*

As displayed in Fig. 6.1, the input to the random forest and the LSTM are manually extracted feature vectors. These feature vectors can stem either from ground truth clusters or from a re-clustering step with either DBSCAN$^-$ or DBSCAN$^+$. Irrespective of the origin of the clusters, the same set of features are calculated.

Since the LSTM requires as input a *sequence* of feature vectors rather than a single feature vector, the methods to calculate the respective classifier's input data are slightly different. For the random forest, a sliding window of length $T = 150\,\text{ms}$ is moved over a cluster and all targets measured from any sensor within this time window are collected.

After one feature vector with temporal length T has been computed, the starting point of the time window is moved to the next time stamp at which a sensor has measured points of the cluster. Hence, the extracted feature vectors of a cluster have a temporal overlap and are therefore strongly correlated. It should be noted that even though a fixed window size of $T = 150\,\text{ms}$ is used, the difference between the latest and the earliest time stamp of the selected targets will be non-constant

and will be always less than T because the sensors provide data only at discrete points in time, fire asynchronously and at non-equidistant intervals, see Section 4.1.

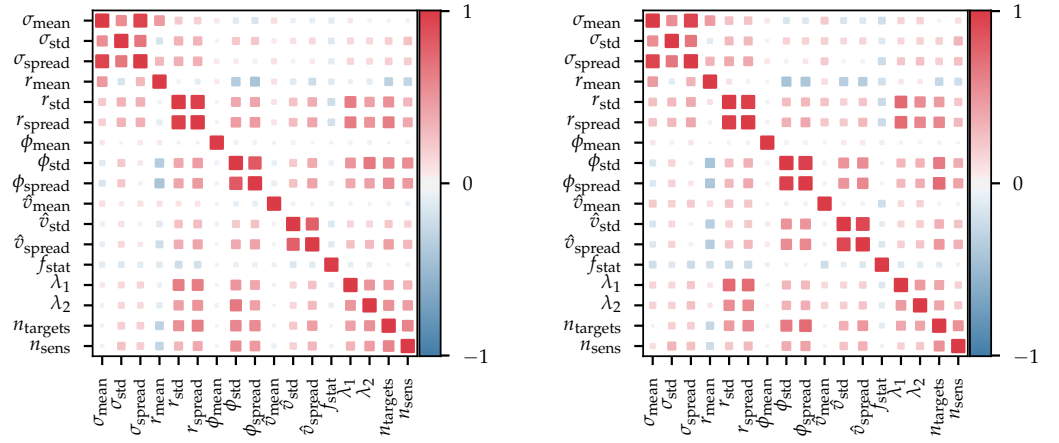
Two different approaches are used to extract *sequences* of feature vectors for the LSTM. The first method is simply that the feature vectors extracted for the random forest are stacked on top of each other so that all feature vectors have the same length T (with the limitations described previously) and are separated by the varying number Δt. The value Δt is the time difference between data from the last sensor scan in the current time window and the first sensor scan in the following time window. If an object is only seen by one sensor, the values of Δt are distributed according to Fig. 4.2. The second method used here to create sequences of feature vectors adds all previously seen data to the new feature vector so that the ith feature vector in the sequence contains all data up to the ith unique time stamp. That is, the first feature vector contains data only from the measurements made at t_0, the second feature vector contains data from t_0 until $t_0 + \Delta t_1$, the third one from t_0 until $t_0 + \Delta t_1 + \Delta t_2$ and so on.

With both methods, sequences with a maximum length of eight feature vectors are created. If the end of a cluster was reached before all eight feature vectors could be computed, the remaining time steps are filled with zeros. Feature vectors are only computed if more than three targets were measured within T, because otherwise classification seems impracticable.

6.2.1.1 *Basic Feature Set*

Since the targets originate from different time stamps within the selected time window and in general the ego-vehicle moves during this time, the data are transformed in a first step to the same car coordinate system (cf. also Section 3.2). Then, from the radial distances $r_i^{(cc)}$, the azimuth angles $\phi_i^{(cc)}$, the ego-motion compensated Doppler velocities $\hat{v}_{r,i}$ and the RCS values σ_i, the average value, the standard deviation and the spread (difference between maximum and minimum value) are computed. This results in the first 12 features σ_{mean}, σ_{std}, σ_{spread}, r_{mean}, r_{std}, r_{spread}, ϕ_{mean}, ϕ_{std}, ϕ_{spread}, $\hat{v}_{\text{mean}}$, $\hat{v}_{\text{std}}$ and $\hat{v}_{\text{spread}}$. Additionally, the bare number of targets n_{targets} within this time window as well as the fraction of "stationary" targets f_{stat} for which $|\hat{v}_r| < 0.3\,\text{m/s}$ are added as features. The threshold of $0.3\,\text{m/s}$ is chosen not to be zero since imperfect ego-motion compensation and noise enforces a non-zero threshold in order to obtain a reasonable feature. From the spatial positions $x_i^{(cc)}$ and $y_i^{(cc)}$ of the targets, the covariance matrix is generated and the two eigenvalues λ_1 and λ_2 are computed. The eigenvalues of this matrix are a measure for the magnitude of the spread in the directions of the two principal components. In this application, they describe the extension of the object along its

two major directions, which are expected to be the width and the length of the cluster. Along with the number of sensors n_{sens} which contributed targets to this cluster, the two eigenvalues λ_1 and λ_2 complete the now 17-dimensional feature vector.



a) Feature vectors are calculated on ground truth clusters.

b) Feature vectors are calculated on re-clustered data.

Figure 6.2: Correlations between the features of the basic feature set.

In Fig. 6.2, mutual correlations between the 17 features are shown for feature vectors extracted from ground truth clusters and from re-clustered data. Positive correlations are marked with red color and negative correlations are marked in blue. The size of the small squares increases proportionally with the correlation. Not surprisingly, the *spread* and *std* features show a strong positive correlation. Another reasonable finding of this evaluation is that the closer objects get to the sensor (smaller r_{mean}), the larger their spread in angle ϕ_{spread} becomes because nearby objects cover a greater angular range than more distant objects. The positive correlation between n_{targets} and both ϕ_{spread} and r_{spread} simply shows the fact that clusters with a larger extent contain more targets. Interestingly, there is also a high correlation between σ_{mean} and σ_{spread}, indicating that the larger the average RCS value of a cluster gets, the larger the difference between the maximum and minimum value becomes. One explanation for this behavior could be that targets obtained via multipath reflections with the street have a smaller amplitude due to fading effects. The larger the RCS value of an object is, the larger the difference to these targets gets so that the spread increases. This explanation assumes that a large portion of the annotated objects contains targets measured via multipath effects.

6.2.1.2 *Extended Feature Set*

In addition to this basic set of features, also an extended feature set is computed via the same process. The idea of this extended feature set is that not only the statistical quantities "mean", "standard deviation" and "spread" are useful for the classification task but that the whole *distribution* of the values within the cluster can be beneficial. Especially for the RCS value and the Doppler velocity this seems reasonable since both of these properties can be highly affected by outliers and sensor artifacts, e.g. Doppler ambiguities. Different options exist to encode a histogram of the respective distributions into a feature vector. One option is that the position and the number of bins is fixed to some reasonable scales so that the whole spectrum of possible values is well covered and then only the relative number of points within each bin is used as a feature. This method has the advantage of being simple and that the individual feature vectors are directly comparable to each other. The downside is that since each cluster will likely cover only a fraction of the possible feature range, a cluster will fall almost exclusively into one bin. This could be corrected by increasing the number of bins but then many bins would be simply zero so that the classifiers would have to learn to deal with sparsely filled feature vectors. A different method of encoding the histogram is fixing only the number of bins to a constant value but choosing the position of the bins dynamically for the current cluster. Since the bare counts of a bin are then no longer meaningful, the position of the bin must also be encoded along with the bin count into the single number describing the feature. Mapping the vector $\vec{w} \in \mathbb{R}^2$ of these two numbers injectively to a single scalar $p \in \mathbb{R}$ can be done as follows. Let b_i denote the position of the left edge of the bin i and let δ_i be a small variation in the sense that $\delta_i / (b_{i+1} - b_i) \ll 1$. The meaning of the histograms should be barely altered if each bin experiences such a small disturbance δ_i and the emerging classifier should be robust enough to these changes. Hence, an approximation of the relative bin counts and the position of the bin with a few digits precision should suffice for the task at hand. If the bin counts c_i are normalized so that $c_i \in [0, 1)$, then the first two decimals are enough for the description of the bin's content and the c_i can be rounded to two decimals. Furthermore, let $b_i^{(0)} = \lfloor b_i \rfloor$ be the integer part of b_i and let $b_i^{(j)} = \lfloor (b_i \cdot 10^j) \bmod 10 \rfloor$ be the jth decimal of b_i. Then a single scalar $f_i = b_i^{(0)} + c_i + b_i^{(1)}/10^3 + b_i^{(2)}/10^4$ can be formed which encodes both the bin position b_i up to two digits precision and the relative bin count c_i. The integer part of f_i is defined via the integer part of b_i and the first two decimals of f_i are the value of c_i. The following digits in f_i are then the further decimals of b_i. This second approach of encoding the histogram proved to be more beneficial to the classifiers so that only this method is used from here on. The extended feature set then contains the 17 basic features and $n_{\text{bins}} = 8$ additional features for each of the two distributions of the RCS values and the Doppler velocities $\hat{v}_r$.

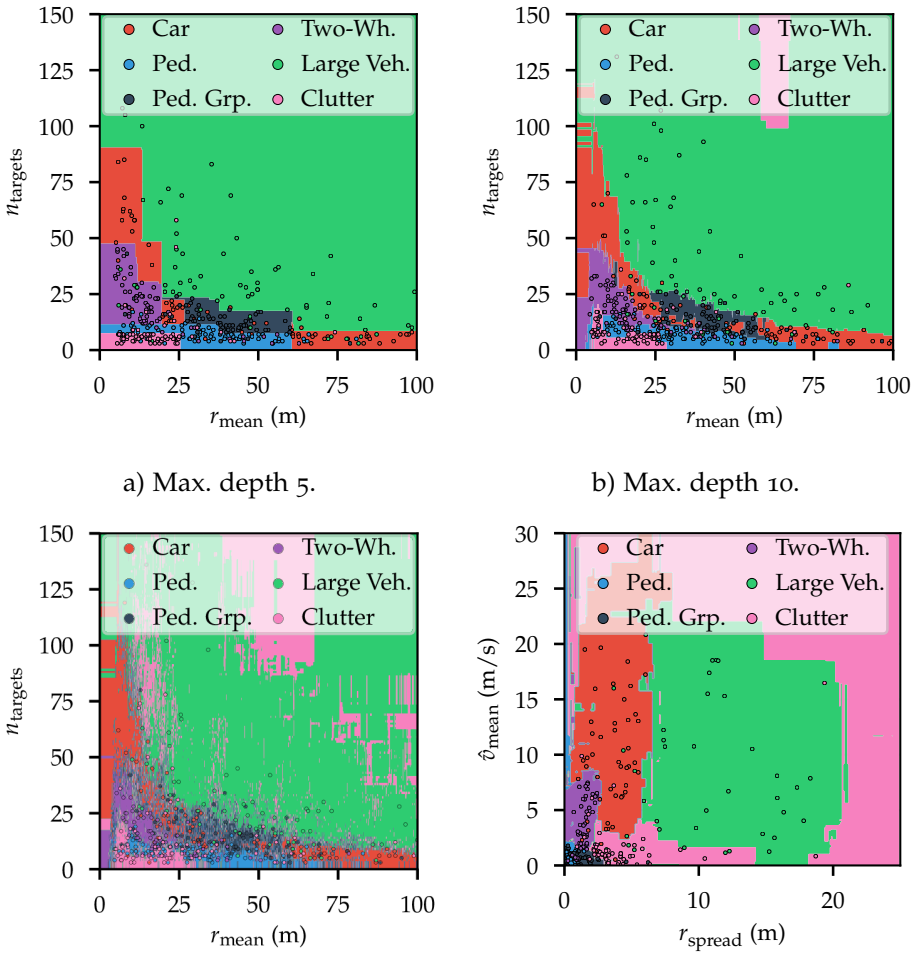
6.2.2 *Random Forest*

In this section, a random forest is used as a classifier and trained on the extracted feature vectors. Relevant hyper-parameters are tuned and their effects are discussed. Decision surfaces, feature importances and the resulting performance of the classifier are illustrated.

6.2.2.1 *Decision Surfaces*

To get a feeling how the random forest makes its decisions, it is illustrative to plot 2D decision surfaces of a reduced feature set. To this end, a random forest was trained on only two features and evaluated on a wide range of values of these two features. A class label can be obtained for each point of this 2D feature space and the decision boundaries become apparent. In Fig. 6.3, this is illustrated for the two feature pairs n_{targets} and r_{mean} as well as for $\hat{v}_{\text{mean}}$ and r_{spread}. The colored background areas indicate to which class a feature vector would be assigned to if it lay in the respective region. Some sample points are drawn from the training data set and displayed. If the color of a point coincides with the background color, the correct class label would be assigned. Otherwise, the background color shows which class would be erroneously predicted. For the first pair of features, the maximum depth of each tree in the forest is altered and the resulting decision surfaces are drawn. The first observation is that the decision surface between the number of targets in a cluster and its average radial distance has great similarity with Fig. 4.12 where the actual measurements are displayed in a comparable way. This shows that the random forest learns from the data set that for all classes the number of points in a cluster decreases with increasing distance and that e.g. the number of points in a `Passenger Car` cluster is larger than the number of points in clusters of class `Pedestrian`. With increasing depth of the trees, the decision surface becomes more complex and the block structure in Fig. 6.3a transforms first into a more continuous decision boundary before the decision surface begins to look more noisy and possibly signs of overfitting can be observed.

The decision surface for the two features $\hat{v}_{\text{mean}}$ and r_{spread} displays the expected boundaries between the classes. Objects with a large spread and high velocities are likely members of the class `Large Vehicle` and with decreasing spread the probability for the `Passenger Car` class increases. Clusters with small spread and small velocity are likely to be either from the classes `Pedestrian` or `Pedestrian Group` and if $\hat{v}_{\text{mean}}$ increases the class `Two-Wheeler` is chosen. A `Clutter` cluster is obtained e.g. for objects which have small Doppler velocities and at the same time a large spread or in general if the spread becomes larger than about 20 m.



a) Max. depth 5.

b) Max. depth 10.

c) Max. depth 20.

d) r_{spread} vs. $\hat{v}_r$.

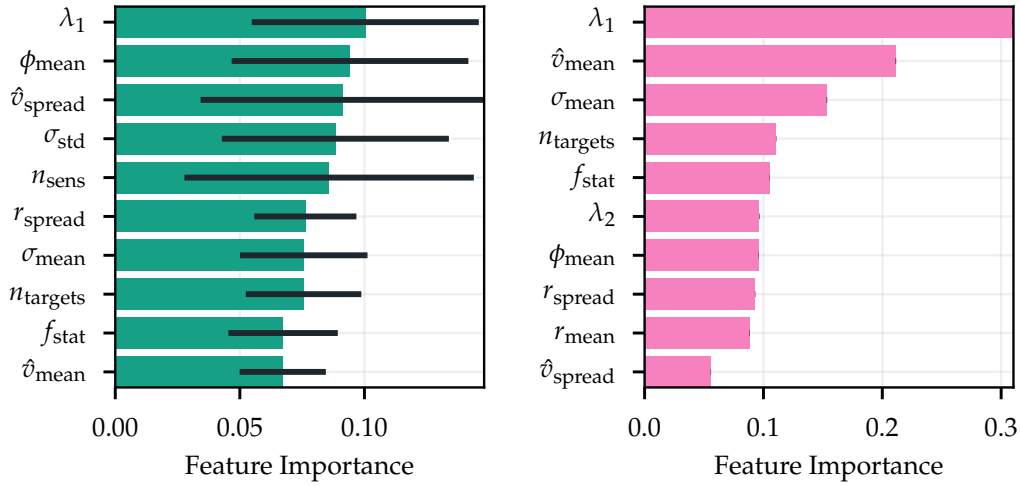
Figure 6.3: Decision surfaces for different features and classifiers. In parts a), b) and c) of the figure, the features r_{mean} and n_{targets} are considered with varying depths of the trees and in part d) the features r_{spread} and $\hat{v}_{\text{mean}}$ are used with unrestricted tree depths.

These plots show nicely that reasonable features were provided for the classification task and that the random forest successfully learned the expected decision boundaries. It should be noted, however, that only two features were used for training and evaluation and that a random forest with all 17 or 33 features has a far more complicated decision surface, which cannot be displayed directly.

6.2.2.2 *Feature Importances*

Random forests have the property that they can directly report which input features were important for the classification task. As noted in Section 2.1.1.2, the standard way to compute the feature importances via the mean decrease in impurity can be biased and the more general method of permutation importance calculation is recommended in the literature [27], [179], [249].

In Fig. 6.4, the importances of the ten highest ranked features of the basic feature set are displayed for both calculation methods. Training is repeated ten times for the creation of the diagrams and the average values of these ten runs are displayed along with the standard deviation indicated by the error bars. The values of the two importance measures cannot be compared between the two methods, but the ranking of the features and the relative distances between them can be analyzed. The permutation importance can be directly read as the decrease in the F_1 macro averaged score if a given feature is shuffled randomly.



a) Importance via mean decrease in impurity.

b) Permutation importance.

Figure 6.4: Feature importances of the random forest classifier, trained on the basic feature set.

One striking difference between the two methods is that the mean decrease in impurity calculation shows way larger fluctuations between the individual runs than the permutation importance, which has almost no noticeable deviations from the mean. Both methods rank λ_1 as their most important feature, where in Fig. 6.4b the distance from the first place to the second place is much larger than for the impurity based estimation. The feature $\hat{v}_{\text{mean}}$ is ranked quite differently by the two

methods: the first method ranks it at place ten whereas the permutation importance puts it in second place with a mean decrease in the F_1 score by about 0.2. It appears reasonable that $\hat{v}_{\text{mean}}$ is ranked quite high since the average Doppler velocity is clearly a good feature to discern e.g. between fast moving cars and pedestrians. Since the impurity based feature importance assessment is biased towards features with large numerical values, it is not surprising that ϕ_{mean} (measured in degrees) is ranked quite high. It is noteworthy that the permutation importance ranks features from the three measurement dimensions space, velocity and RCS on the first three places and that therefore all feature dimensions a radar can measure are well represented and apparently useful for classification. The fact that mostly spatial features dominate the top ten list hints that radar sensors with higher spatial resolution and more measured targets per object can greatly increase the classification result.

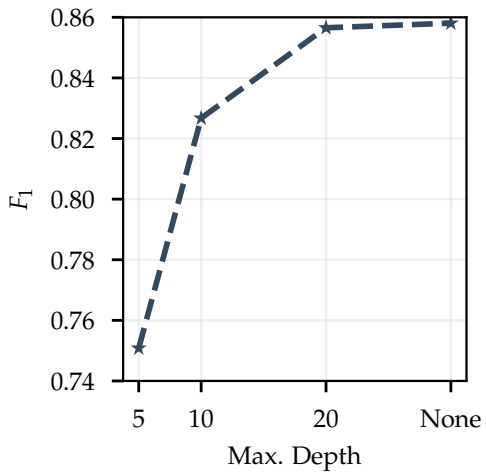
For the permutation importance method, data from the training set was used. Comparison with data from the test set showed that no big differences between the feature importances exist so that none of the top ten features can be identified as causing overfitting. A feature would be suspected to cause overfitting if it has a high importance during training but shows a low importance during testing. The impurity based importance measure can only be evaluated on training data.

Overall, it can be summarized that the results obtained from the permutation importance method are more reasonable and give better insight into the true relevance of the input features.

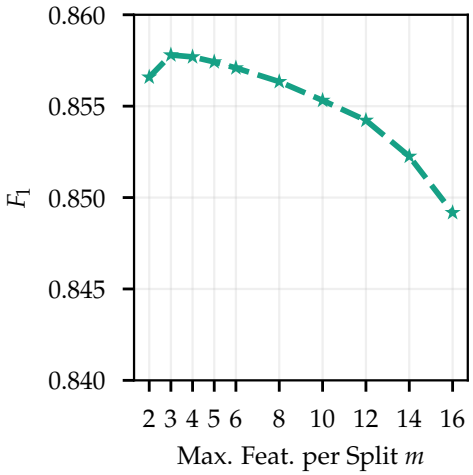
6.2.2.3 *Parameter Sweep*

To assess which hyper-parameter set works best for the classification task, the following settings are tuned: the maximum depth of the individual trees, the number of features considered in each split, the number of trees in the forest and balancing methods to increase the performance on minority classes. The basic feature set is used for this evaluation but no significant changes are expected if the extended set is used. Since only the classification task itself should be evaluated, influences from the preceding clustering step are ignored and only features from the ground truth clusters are used for training and evaluation. The scores presented here were obtained after five-fold cross validation.

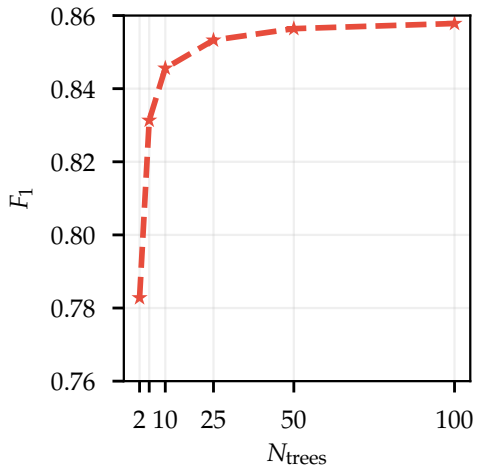
In Fig. 6.5a, the maximum depth of the decision trees are tuned and the resulting F_1 macro averaged scores are computed. A forest with 100 trees was used and a maximum of $m = 3$ features per split were considered. The score continuously increases with increasing maximum depth of the trees. Pruning of the trees does therefore not help and overfitting does not occur, in accordance with [26]. However,



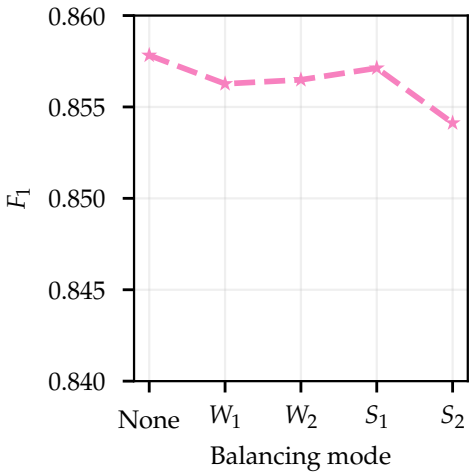
a) Variation of the maximum depth of the trees.



b) Variation of the max. number of features considered in each split.



c) Variation of the number of trees.



d) Class balancing options.

Figure 6.5: Classifier performance as a function of different random forest parameter settings.

the difference between pruning the trees at a depth of 20 and letting the trees grow to their full extent shows no great difference in performance, although the fully-grown trees have an average depth of 50 with in total twice as many leave nodes as the pruned forest. On systems were memory resources are sparse, pruning the trees can therefore be a valid method to find a good trade-off between performance and memory footprint.

Varying the maximum number of features m that are considered per split is often cited as the most influential parameter for tuning a random forest, see also Section 2.1.1.2. In Fig. 6.5b, this number m is varied and again the F_1 score is reported. Notice that the scales in Fig. 6.5a and Fig. 6.5b are different since the variations are in this case much smaller. The 100 trees in the forest were allowed to grow to their full extent. Often the square root of the total number of features is used as a default setting, which in this case corresponds to $m = 4$. The figure shows that for $m = 3$ the score is maximal, but since the difference to the case with $m = 4$ is minuscule, the rule of thumb with setting m to the square root of the total number of features is confirmed here. However, for this task the parameter has only very little impact compared to the variation of the maximum depth of the trees.

Generally, the number of trees in a forest should be chosen as high as possible since the more trees are used, the better the overfitting of individual trees can be compensated for. Figure 6.5c supports this view and shows that the performance saturates if about 100 trees are used. The trees were not pruned and $m = 3$ features per split were considered. Adding more trees beyond this value just adds extra computation time with no real improvement in the classification performance.

Finally, it is evaluated how different balancing methods influence the classifier's score. In total, four different balancing and weighting methods are compared with the basic setting where no balancing is done and all feature vectors are weighted equally. In general, two different categories exist to increase the performance of a classifier on minority classes. Weighting methods influence the way the loss function of a classifier is computed by increasing the loss if an example from a minority class is misclassified. Balancing approaches change the number of training samples presented during training by oversampling minority classes, undersampling majority classes or by combining these two methods. The two weighting methods used here differ on how the weights for the respective class are calculated. In the first case W_1, the weight of a class is given by the total number of samples of this class in the training set divided by the number of all training samples. In the second case W_2, not the whole training data set is considered but rather only the samples that were chosen for a given tree. For balancing the data set, *Synthetic Minority Over-sampling Technique (SMOTE)* [35] is used as a first method S_1, and as a second balancing method, SMOTE oversampling is combined with *edited nearest neighbor* [8] undersampling. The F_1 scores for the different methods are depicted in Fig. 6.5d. The results indicate that neither balancing nor weighting increases the overall performance. Since the differences in the overall score are rather small, it is worthwhile to look at how the scores of the minority classes change with the different methods. For all four methods, the true positives of each of the two minority classes `Two-Wheeler` and `Large Vehicle` increased, so that the recall values are higher than for the unbalanced and un-weighted random

forest. This, however, happened at the cost of a slight decrease in true positives of the `Passenger Car` class and at the cost of predicting far less true positives of the `Clutter` class. The latter has a great impact on the precision of the classifiers so that the increased recall values are compensated by the decrease in precision. One therefore has to decide in the broader context of the application whether one would rather classify more instances of a minority class correctly at the cost of identifying fewer `Clutter` objects and thereby creating "ghost" objects or if the opposite scenario is more desirable.

For the remainder of this chapter, random forests are always trained with 100 trees which can grow to their full extent, a maximum of $m = 3$ features are considered per split and no balancing or weighting takes place.

6.2.2.4 *Performance Evaluation Based on Feature Vectors*

To assess how well the random forest performs on the different feature sets and clustering methods, scores and confusion matrices are now presented on a feature vectors basis. This means that neither individual points nor complete clusters are considered but only the extracted feature vectors are taken into account.

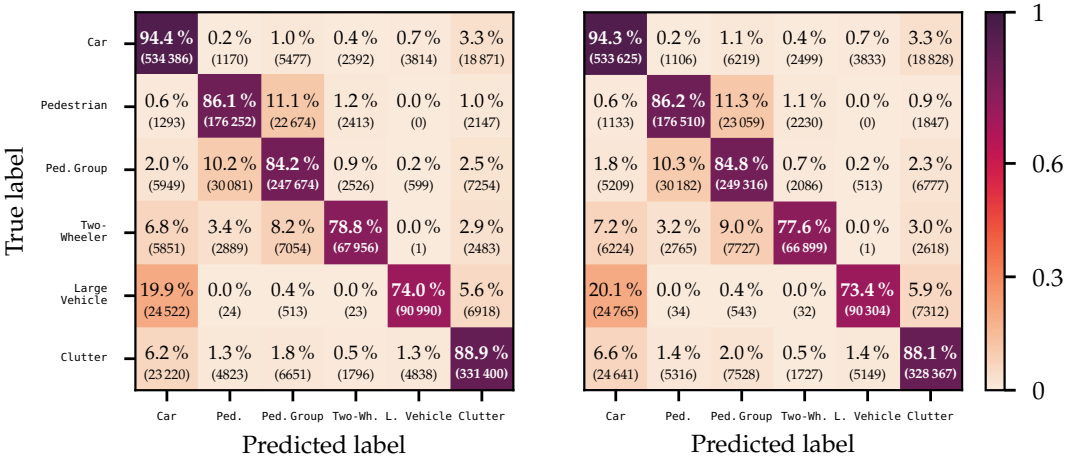


Figure 6.6: Confusion matrix based on feature vectors. Left: Basic feature set. Right: Extended feature set.

First, a random forest is trained and evaluated only on feature vectors obtained from ground truth clusters while the two different feature sets are used. In Fig. 6.6, the confusion matrices for both feature calculation methods are displayed. The macro averaged F_1 scores of the two methods are 0.858 for the basic feature set and 0.855 for the extended feature set, see also Table 6.1. The majority class `Passenger Car` is classified best by the two approaches, followed by the `Clutter` class. Minority

classes like Two-Wheeler and Large Vehicle show a significantly lower performance than the classes with more objects. The mixture between the two classes Pedestrian and Pedestrian Group is not surprising: instances of these two classes have similar spatial extent, about the same Doppler velocities and the average RCS values does not differ too much. Additionally, uncertainties from the labeling process itself come into play here. The class Pedestrian Group was designed as a fallback for situations in which it is apparent from the camera image that multiple Pedestrian are present but the radar targets cannot be assigned properly to individual Pedestrian clusters. Some labelers feel more confident and create multiple Pedestrian instances in situations where other labelers would have created one large Pedestrian Group cluster. This causes some extra overlap between the classes which also shows up here. Similar arguments can be used to explain the confusion between Passenger Car and Large Vehicle feature vectors. As explained in Section 4.3, the transition between these two classes is almost continuous and labelers can easily come to different conclusions for the same object. Small trucks are then difficult to discern from larger cars and the situation becomes even more complicated if the objects are far away so that only few targets are available for the classification task. The same holds true if a Large Vehicle is only seen from behind since then the most discriminating feature – the spatial extent of the cluster – becomes less useful. Occlusions caused by guard rails or other road users are another source which can make a Large Vehicle appear shorter than it actually is.

The extended feature set caused a small increase in the true positives of the Pedestrian and Pedestrian Group classes but at the same time, the true positives of the other classes slightly decreased. Hence, no real performance boost can be obtained from the additional features even though they show up as more important than some features from the basic feature set when the permutation importance measure is used. This finding is also interesting in so far that it allows the statement that more features are not always helpful for the classification task, even though a classifier built solely from these additional features performs better than random guess.

Up to now, only the performance on feature vectors extracted from ground truth clusters was discussed. Since a clustering step is necessary for the extraction of feature vectors, the complete system performance is only obtained when the performance is evaluated on the feature vectors extracted from the predicted clusters. Therefore, a classifier was trained on feature vectors extracted from ground truth clusters and evaluated on feature vectors extracted from DBSCAN$^-$ and DBSCAN$^+$ clusters. The resulting confusion matrices are displayed in Fig. 6.7.

Performance on the DBSCAN$^+$ clusters is slightly better than on the DBSCAN$^-$ clusters (F_1 scores of 0.486 and 0.438, respectively), especially on the Pedestrian and Two-Wheeler classes, but a clear performance degradation is visible for all

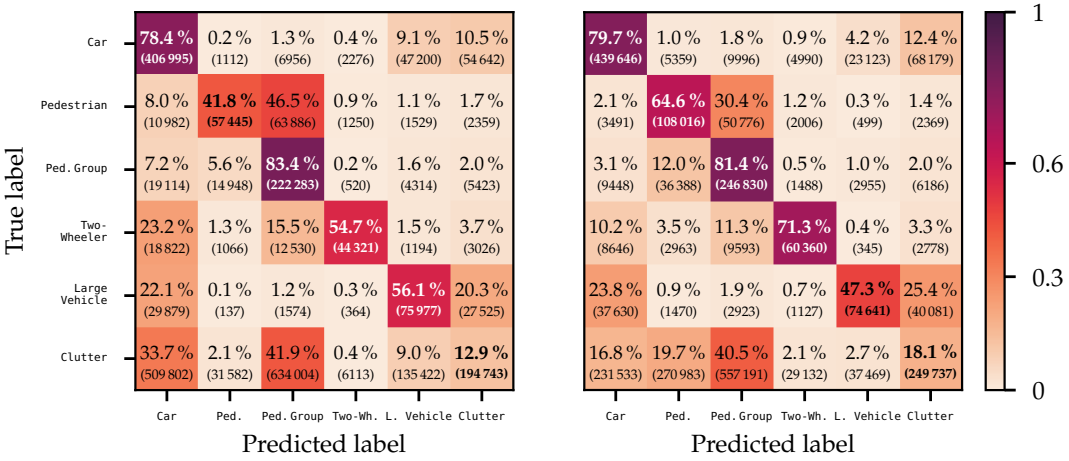


Figure 6.7: Confusion matrix based on feature vectors. A classifier was trained on feature vectors from ground truth clusters. Left: Evaluation on feature vectors from DBSCAN$^-$ clusters. Right: Evaluation on feature vectors from DBSCAN$^+$ clusters.

classes. The `Clutter` class experienced its largest drop from about 89 % to 18 %. The reason for this is quite understandable: The labelers got a precise definition of the cases when they should create a `Clutter` cluster. These rules, however, do not necessarily correspond to the false-positive objects that a clustering algorithm creates and which then should be classified as clutter. One approach could be to change the labeling task so that already clustered data is provided. The labelers task would then be to correct only those clusters that correspond to real objects and only assign the `Clutter` label to the false-positive clusters. This approach has the downsides that a) the labeling task depends on the parameters of the clustering algorithm and any change in these would require a re-labeling of the already annotated data and b) it turned out that correcting pre-clustered data is more time-consuming than starting from scratch. A more feasible approach is to assign the label `Clutter` automatically to all clusters that do not have any overlap with an existing ground truth cluster. With this method, the statistics of the `Clutter` class still depend on the clustering algorithm and its parameters, but since the labeling of this class can be done automatically this comes at no cost.

The previous evaluation clearly shows that the clustering step has a great influence on the classifier's performance and that a classifier trained on feature vectors of one clustering method (here: ground truth labeling) does not necessarily perform well on feature vectors of another source. One approach to heal this is by re-clustering the ground truth clusters so that a clustering algorithm takes ground truth clusters as input and outputs (possibly multiple) new clusters which all contain a subset of the ground truth cluster's points. An ideal clustering algorithm would create only one cluster, which is identical to the ground truth cluster. This

method has the advantage that only those targets are added to a cluster which truly belong to the object. Any targets from the surrounding cannot enter any of the clusters since only the ground truth cluster is used as input. The clusters are hence "clean" in the sense that all targets in the cluster have the same ground truth label. This approach was used in the previously published papers [232], [234]. The second possibility is that the whole recorded sequences are re-clustered with the respective clustering algorithm. This has the advantage that all the peculiarities of the respective algorithm are covered and the extracted features take this into account. The downside is that the targets in the created clusters do not necessarily all have the same label since nearby objects might be merged into one cluster or targets from the static environment might be added. Nevertheless, this second method is chosen here since it turned out to yield far better results than the first method. The ground truth label for a newly created cluster is chosen to be the ground truth label of the individual targets which appear most often in the cluster. That is, a simple majority vote is used to assign labels to the created clusters.

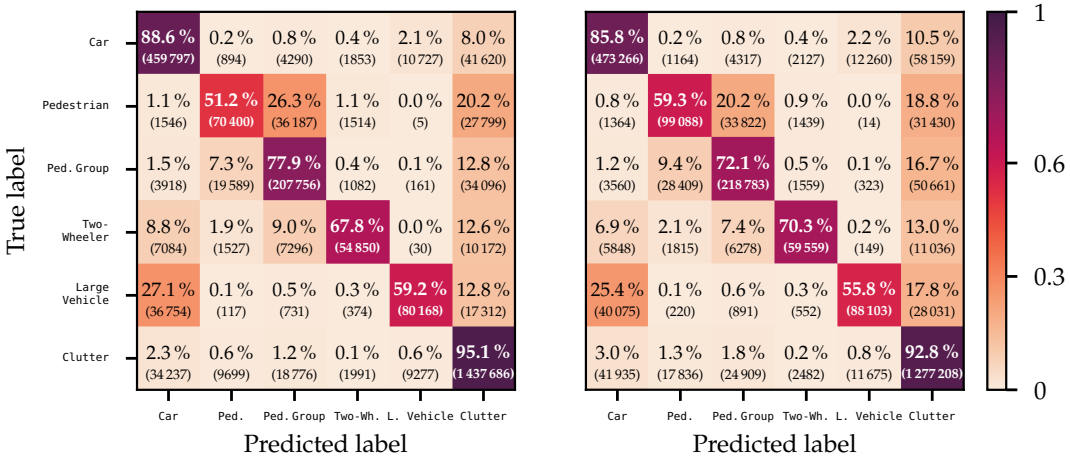


Figure 6.8: Confusion matrix based on feature vectors. Left: Training and evaluation on feature vectors from DBSCAN$^-$ clusters. Right: Training and evaluation on feature vectors from DBSCAN$^+$ clusters.

The training and evaluation pipeline then consists of the following steps:

1. Re-clustering with DBSCAN$^-$ or DBSCAN$^+$.
2. Propagation of ground truth labels to the newly created clusters.
3. Extraction of features from both the ground truth clusters and the newly created clusters.
4. Training of a random forest with the two sets of feature vectors.
5. Evaluation of the trained classifier on the feature vectors of the re-clustered data.

Feature Set	Clustering	Eval. Basis	F_1	Prec.	Rec.	Conf. Matrix
Basic	ground truth (gt)	feat. vec.	0.858	0.876	0.844	Fig. 6.6 left
Extended	ground truth (gt)	feat. vec.	0.855	0.875	0.840	Fig. 6.6 right
Basic	train: gt eval: DBSCAN$^-$	feat. vec.	0.438	0.493	0.546	Fig. 6.7 left
Basic	train: gt eval: DBSCAN$^+$	feat. vec.	0.486	0.493	0.604	Fig. 6.7 right
Basic	train: gt & DBSCAN$^-$ eval: DBSCAN$^-$	feat. vec.	0.767	0.816	0.733	Fig. 6.8 left
Basic	train: gt & DBSCAN$^+$ eval: DBSCAN$^+$	feat. vec.	0.758	0.800	0.727	Fig. 6.8 right
Basic	train: gt & DBSCAN$^-$ eval: DBSCAN$^-$	targets	0.719	0.753	0.708	Fig. 6.9 left
Basic	train: gt & DBSCAN$^+$ eval: DBSCAN$^+$	targets	0.761	0.785	0.746	Fig. 6.9 right

Table 6.1: Scores obtained with a random forest as classifier for different configurations.

It should be noted that the evaluation is solely done on feature vectors extracted from the re-clustered data since this is the data the classifier would get in a true application. Cross validation of the training with DBSCAN$^+$ re-clustering results in an F_1 score of 0.758 and the training with DBSCAN$^-$ yields a score of 0.767. The confusion matrices are shown in Fig. 6.8 and can be directly compared to the ones displayed in Fig. 6.7, where only ground truth data was used for training. A drastic increase in the overall performance can be observed even though the performance on ground truth clusters is unmatched, cf Fig. 6.6. It is noteworthy that the confusion between instances of the `Large Vehicle` class with `Passenger Car` samples is still quite high. This is most likely due to imperfect clustering which causes that larger objects are not clustered together but rather many small clusters are created which then resemble `Passenger Car` objects. For driver assistance systems, the confusion between `Pedestrian` and `Pedestrian Group` is possibly not too relevant. Nevertheless, for tracking applications it might be valuable information that an object is a group of pedestrian and not a single pedestrian so that possibly multiple pedestrian objects could emerge from the `Pedestrian Group` at a later point in time. Another interesting fact is that the F_1 score for the DBSCAN$^+$ training is lower than for the DBSCAN$^-$ training. This could indicate that the improved clustering does not help in the classification task. However, in the per-target evaluation in the next section, different results are found.

In Table 6.1, the results of all approaches are summarized again.

6.2.2.5 *Per-Target Evaluation*

To compare the random forest performance to methods presented later in this thesis, it is necessary to introduce a common evaluation scheme. Since a feature vector based evaluation is not feasible for the other methods, a per-target evaluation is used. To this end, the predicted label of each feature vector is propagated to the targets that were used for the creation of the feature vector. Since overlapping time windows are used during feature extraction, each target appears in multiple feature vectors and therefore a target might get different labels. This is resolved by using only the label which is predicted the first time a target contributes to a feature vector since this case is probably the one relevant in a driver assistance system where new information should be directly processed.

In Fig. 6.9, the two confusion matrices displayed in Fig. 6.8 are repeated on a per-point scale and F_1 scores are re-computed from them. The per-point scores with 0.719 and 0.761 (see also Table 6.1) differ far more from each other than the scores computed on the feature vector basis. It should be noted that the situation is reversed from the evaluation on the last section where the DBSCAN$^-$ approach performed better. This indicates that with DBSCAN$^+$ on average less feature vectors are classified correctly but at the same time more individual targets are ascribed to the correct class. This is not a contradiction but rather shows that feature vectors built from many targets are classified better than the ones created from only a few targets and that apparently more of these "large" feature vectors are present in the DBSCAN$^+$ case. Feature vectors created from clusters with a small amount of targets have a lower impact on the here presented per-target score.

Just as in the previous evaluation, the weak performance on the `Pedestrian` class becomes apparent with about 26 % and 18 % "overlooked" targets, i.e., targets which were classified either as clutter or not even added to a cluster and hence could not be classified.

6.2.3 *Long Short-Term Memory Network*

Using an LSTM as a classifier instead of a random forest allows to consider temporal correlations by using *sequences* of feature vectors as input instead of just one single feature vector. Details on how LSTMs work can be found in Section 2.1.2.2 and in the references mentioned in there.

The performance of different LSTM networks is analyzed in this section. Common to all experiments is that the LSTM layers are followed by one fully connected layer, which maps the output of the last LSTM to the six (unscaled) class probabilities.

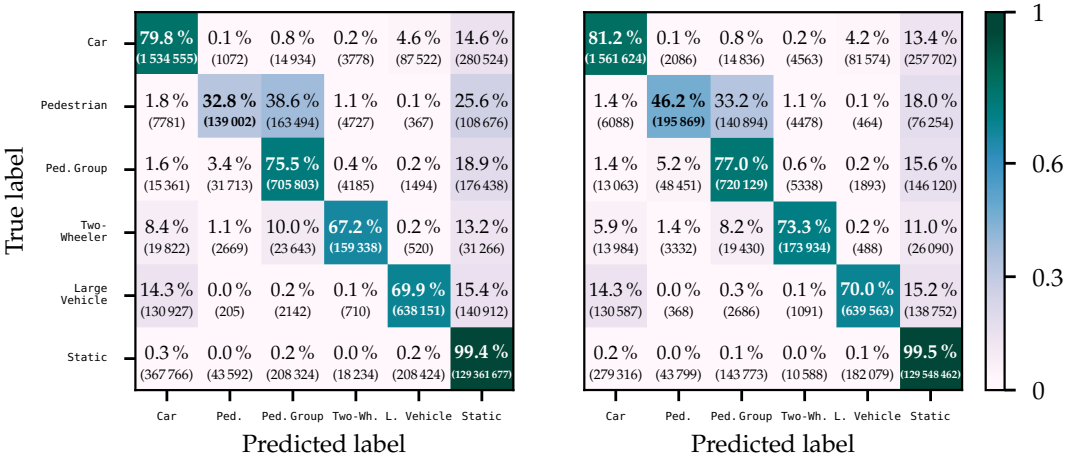


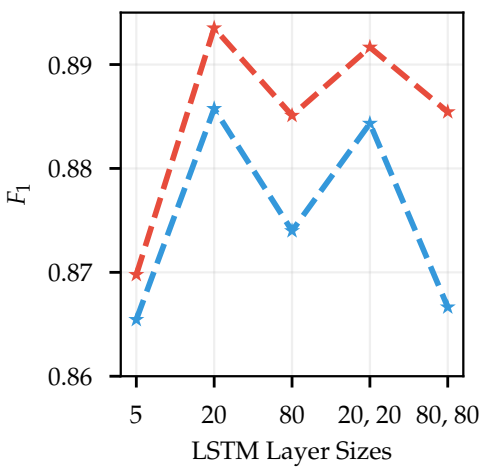
Figure 6.9: Random forest confusion matrix based on individual targets after label propagation from the respective feature vectors. Left: Training and evaluation on feature vectors from DBSCAN$^-$ clusters. Right: Training and evaluation on feature vectors from DBSCAN$^+$ clusters.

As input data, sequences of eight feature vectors are used, which are generated according to the two schemes described in 6.2.1. It is evaluated which of the two schemes is more beneficial for the classifier and how normalization of the feature vectors influences the performance.

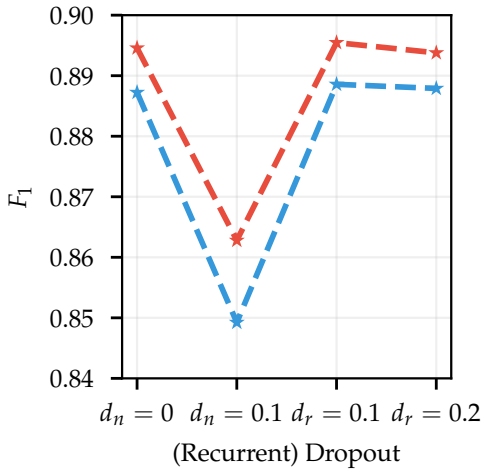
6.2.3.1 *Parameter Sweep*

Prior to training of the network, choices have to be made about the hyper-parameters of the classifier. This includes the number of LSTM layers, the number of neurons in each of the layers, the ratio of both dropout and recurrent dropout and whether or not normalization of input data should be used. Dropout is a regularization method that helps to prevent overfitting by randomly dropping connections between neurons during training [246]. With normal dropout d_n, the ratio of dropped neurons in the input layer of the LSTM is described and the recurrent dropout d_r describes the ratio of dropped units of the recurrent kernel.

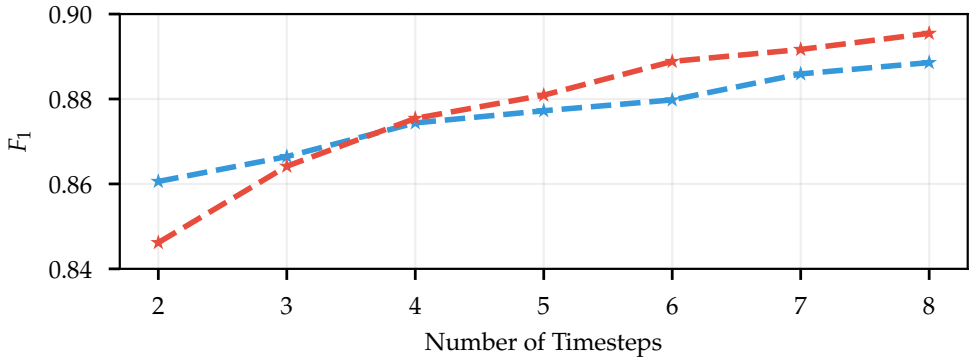
In Fig. 6.10, variations of different parameters and their effect on the F_1 score are displayed. Similar to the discussion of the random forest parameters, evaluation is done on feature vectors extracted from ground truth clusters and the scores are obtained after five-fold cross validation. In all three sub-figures, the blue lines stand for the evaluation with constant size feature vectors and red lines show the performance if feature vectors with increasing size are used.



a) Variation of the number and size of the LSTM layers .



b) Effect of applying normal dropout d_n and recurrent dropout d_r during training.



c) Variation of the number of time steps used in the feature vector sequences. The LSTM was both trained and evaluated on sequences of the respective length.

Figure 6.10: Classifier performance as a function of different parameter settings. The red curves symbolize the score if the increasing time windows are used and the blue curve marks the case where features from constant time windows are calculated, see also Section 6.2.1.

Variation of the LSTM layer sizes is shown in Fig. 6.10a. A sequence consisting of eight feature vectors, $d_n = 0$ and $d_r = 0.1$ was used in this experiment. The numbers on the horizontal axis stand for the size H of the internal matrices of the LSTM, see also Eq. (2.22)ff. Comma separated numbers indicate that multiple stacked LSTM layers with the given sizes are used. Interestingly, there is no simple relationship between the number of trainable variables in the network and the performance since at first the scores increase with increasing number of neurons and decrease

later if the matrix sizes are further enlarged. Unsurprisingly, the network shows the lowest scores among all tested configurations if only five neurons are used. But for the case of fixed size feature vectors in the input sequence (blue curve), a similar performance is reported for a two-layer LSTM in which both layers have 80 neurons. The "sweet spot" seems to lie around 20 neurons in a single-layer LSTM since the highest scores are reported here for both feature extraction methods. Overfitting could be a possible explanation for this finding even though neither dropout nor recurrent dropout changed the general behavior. This suggests that a single layer LSTM with only 20 neurons should be used.

In Fig. 6.10b, the impact on the F_1 score of using dropout and recurrent dropout during training is displayed. Using dropout $d_n = 0.1$ on the input connections decreases the performance considerably. In contrast, slight recurrent dropout of $d_r = 0.1$ results in a small performance increase compared to the case in which neither dropout nor recurrent dropout were used (first $\star$ in the figure). However, increasing d_r further is not beneficial so that $d_r = 0.1$ is selected for all following experiments.

Finally, the impact of the sequence length is displayed in Fig. 6.10c. A single layer LSTM with 20 neurons and $d_r = 0.1$ was trained and evaluated on sequences of varying length and the F_1 scores of the test sets were collected. If only two time steps are used, the method using constant size feature vectors shows a higher performance. This relation switches if more than four time steps make up the input sequence. As explained in Section 6.2.1, the constant size feature vectors all have a length of $T = 150\,\text{ms}$ and are separated by the varying time range Δt. If sequences of length two are extracted with the second method, then only data from two different radar scans is used which in this data set is typically less information than in the $300\,\text{ms} + \Delta t$ time window used in the first method. It is hence not surprising that for short sequences (in the sense of number of feature vectors in the sequence) the first feature extraction method is superior. For longer sequences, however, accumulating data shows higher performance even though the total time (in seconds) of the data presented to the network is smaller than if fixed size feature vectors were used. This indicates that it is easier for the LSTM to make use of the time information if all available data is presented instead of showing only the new feature vectors. Nevertheless, the score saturates at about eight time steps so that this number will be used in the following experiments.

It is often quoted that normalizing (and scaling) input data for a neural network is an important pre-processing step [13], which a) stops gradients from becoming too large for some of the feature dimensions and thereby "shadowing" the other dimensions and b) avoids the tails of the sigmoid activation function at which the gradients are all close to zero. In the results shown before, all input features were normalized by subtracting the mean and dividing by the standard deviation

of the respective feature dimension. The means and standard deviations of the training step were saved and test data was normalized with the same values. The best performing LSTM architecture achieves an F_1 score of 0.896, whereas the same architecture without normalization scores a bit lower with 0.890. Normalization therefore seems to have some positive impact even though the changes are rather small in this case. This is possibly due to the fact that all features vary on the same scale and that there is no feature that is many orders of magnitudes different from the others.

6.2.3.2 *Performance Evaluation*

Now that the optimal LSTM configuration was found (namely a single layer with 20 neurons, $d_r = 0.1$, eight time steps using feature extraction method two with normalization), confusion matrices and further scores of the best performing architecture can be discussed.

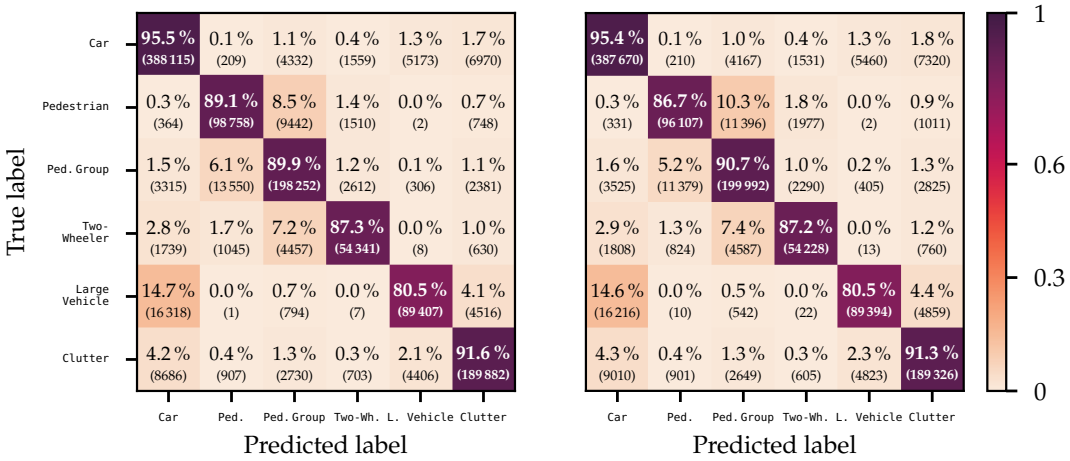


Figure 6.11: Confusion matrices of the LSTM classifier. Training and evaluation is done on features from ground truth clusters. Left: Basic feature set. Right: Extended feature set.

To give an upper limit for the classifier's performance, Fig. 6.11 shows the confusion matrices for LSTMs trained on feature vectors of ground truth clusters with the basic (left) and the extended (right) feature set. The F_1 scores of the two approaches are with 0.896 and 0.893 very similar, with slightly higher performance if the basic feature set is used. This result is in accordance with the findings made for the random forest. Similar to the random forest results, also the LSTM struggles most with the distinction between `Passenger Car` and `Large Vehicle` objects – probably for the same reasons as discussed before in Section 6.2.2.4. A detailed comparison between the approaches will be done in Section 6.6.

To assess the performance in a real system, i.e. in the case where no ground truth clusters are present but rather DBSCAN is used for the cluster creation, classifiers are trained and evaluated on feature vectors of DBSCAN$^-$ and DBSCAN$^+$. For training, also feature vectors from ground truth clusters are added for augmentation. The resulting confusion matrices are displayed in Fig. 6.12, where for features from DBSCAN$^-$ a score of 0.782 is obtained and for DBSCAN$^+$ the F_1 score is given by 0.793. In contrast to the random forest, a clear improvement can be seen on a feature vector level if DBSCAN$^+$ clusters are used. This allows the interesting conclusion that different classifiers work differently well with clustering methods and therefore optimizing clustering in one direction (here: towards on average larger clusters with a little more false positive targets, see Section 5.2.3) does not necessarily help all classifiers which work on these clusters.

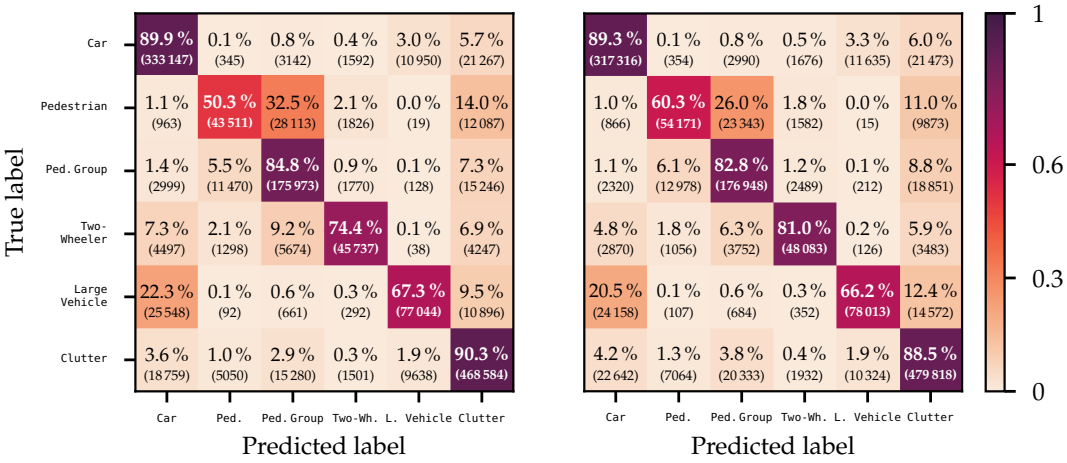


Figure 6.12: Confusion matrices of the LSTM classifier. The basic feature set is used and evaluation is done *per feature vector*. Left: Training with features from ground truth and DBSCAN$^-$ clusters. Right: Training with features from ground truth and DBSCAN$^+$ clusters.

The same relation between the scores of the two clustering approaches can be seen in a per-target evaluation: in this case the scores are given by 0.721 and 0.766 for DBSCAN$^-$ and DBSCAN$^+$, respectively. The confusion matrices are shown in Fig. 6.13 and can be directly compared to the ones of the random forest shown in Fig. 6.9 since the two classifiers are evaluated on the same basis. The trends already visible in the evaluation on feature vector basis are also present in these matrices: the LSTM shows higher performance on all classes except for the `Large Vehicle` category if DBSCAN$^+$ is used as a clustering algorithm instead of DBSCAN$^-$.

Another positive aspect is the reduction of false positives, i.e. the reduction of targets that are truly `Static` but classified as something different. The confusion matrices shown in Fig. 6.12 could lead to the opposite conclusion since the DBSCAN$^+$

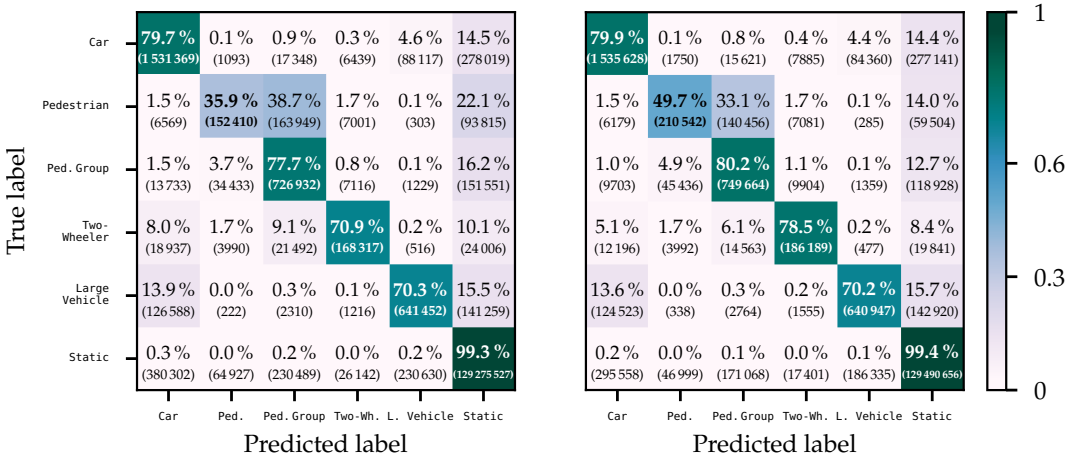


Figure 6.13: Confusion matrices of the LSTM classifier. The basic feature set is used and evaluation is done *per target*. Left: Training with features from ground truth and DBSCAN$^-$ clusters. Right: Training with features from ground truth and DBSCAN$^+$ clusters.

approach has with 88.5 % true positives for the `Clutter` class on a relative scale less correct predictions than the LSTM trained on DBSCAN$^-$ clusters (90.3 %). Since the occurrence of `Clutter` highly depends on the clustering algorithm itself and hence a different number of feature vectors with different statistics are extracted, a comparison on feature level basis is not directly possible and the per-target evaluation has to be used for this kind of comparison. For example, a very sensitive clustering algorithm creates many clusters in areas where no true dynamic objects exist (e.g. due to noise) so that a classifier has to reject these clusters as `Clutter` in order to obtain a high score. A less sensitive clustering algorithm creates fewer of these `Clutter` clusters so that for many targets it is implicitly clear that they belong to the `Static` class as they are not a member of any of the proposed clusters.

Finally, in Table 6.2 the scores mentioned in this section are summarized again.

6.3 AUTOMATIC FEATURE EXTRACTION

The methods presented in the last sections rely on a manual feature extraction step. These features are handcrafted with the hope in mind that they allow to discern the individual classes from each other. However, the most recent history of machine learning indicated that methods which do not only classify the input data but also calculate the needed features on their own often outperform methods which are based on manually created features. As there is no strong argument why this

Feature Set	Clustering	Eval. Basis	F_1	Prec.	Rec.	Conf. Matrix
Basic	ground truth (gt)	feat. vec.	0.895	0.902	0.890	Fig. 6.11 left
Extended	ground truth (gt)	feat. vec.	0.893	0.901	0.886	Fig. 6.11 right
Basic	train: gt & DBSCAN$^-$ eval: DBSCAN$^-$	feat. vec.	0.782	0.812	0.761	Fig. 6.12 left
Basic	train: gt & DBSCAN$^+$ eval: DBSCAN$^+$	feat. vec.	0.793	0.810	0.780	Fig. 6.12 right
Basic	train: gt & DBSCAN$^-$ eval: DBSCAN$^-$	targets	0.721	0.733	0.723	Fig. 6.13 left
Basic	train: gt & DBSCAN$^+$ eval: DBSCAN$^+$	targets	0.766	0.774	0.763	Fig. 6.13 right

Table 6.2: Scores obtained with LSTM classifiers for different configurations.

should be different for classification tasks using radar data, it is worthwhile to investigate how these methods perform in comparison to the approaches discussed before.

6.3.1 *Classification with PointNet++*

With the neural networks called *PointNet* [199] and *PointNet++* [198], Qi et al. introduced novel architectures, which directly operate on bare point clouds, see also Section 2.1.2.1. From the different architectures and variants introduced in their work, in this section the *classification* network from PointNet++ is used, which incorporates a configurable number of *Multi-Scale Grouping (MSG)* blocks followed by pooling layers and finally fully connected layers for the mapping of the created high dimensional feature vectors onto the class probabilities. The details of why the network is designed that way and how the MSG modules work can be found in [198].

6.3.1.1 *Input Data*

In contrast to the random forest and LSTM approaches discussed before, now the bare *clusters* are used as input to the network. This means that the individual targets which form a cluster are used as input, where each target is defined by its position in car coordinates $x^{(cc)}$ and $y^{(cc)}$, the ego-motion compensated Doppler velocity $\hat{v}_r$, the RCS value σ and the measurement time t at which the target was recorded.

To keep the approach as similar as possible to the previous ones, time slices of length $T = 150\,\text{ms}$ are extracted from the clusters. All targets within this time window are transformed to the car coordinate system of the earliest present time stamp so that all targets lie in the same reference system. The time dimension is normalized so that $t_0 = 0$ indicates the earliest measurement time and all other times are given in seconds after t_0.

With 200 allowed input targets, the network's input layer is designed large enough so that all targets measured typically during 150 ms on one object can be processed. In the rare case in which there are more targets within a cluster, a random subset of targets is chosen and used as input. Usually, less than 200 targets are measured so that in order to guarantee a fixed size input tensor, one target is repeated the required number of times. As the network is invariant with respect to repetitions of targets, this does not alter the result.

To increase the number of training samples and to help the network to generalize better to unseen data, the following three augmentation steps are done during training. With a probability of 80 %, the targets are shifted randomly in space, time, velocity and RCS. The distances of the shifts are chosen from a normal distribution centered at zero and are clipped to 0.2 m for the spatial shift, 0.5 m/s for the velocities and 0.5 dBsm for the RCS. Independent of this first augmentation step, the cluster is mirrored with a probability of 30 % on the origin of the car coordinate system so that an object which was initially recorded at the front left of the ego-vehicle is then mirrored to the rear right side. Thirdly, random dropout was applied to the targets so that a target was removed from the cluster with a 30 % chance. The effectiveness of these augmentation steps is evaluated later on.

6.3.1.2 *Network Structure and Training*

The structure of the classification network is shown in Fig. 6.14. Two MSG modules are followed by a set abstraction module whose output is then fed into three consecutive fully connected layers. The definition of the MSG modules in [198] assumes that the input data consists only of 3D points with x, y, and z coordinates. In this case, however, the targets have two spatial dimensions and a variable number c of further feature dimensions. Most often, $c = 3$ for the three extra dimensions Doppler velocity, RCS value and measurement time. The MSG modules were modified so that this new input data can be processed, which means that the neighborhood searches are done in two dimensions but all feature dimensions are taken into account during the convolution operations.

The first MSG module uses 64 of the 200 input targets as core points (abbreviated in Fig. 6.14 as "MSG @ 64"). As described in [198], the core points are selected via

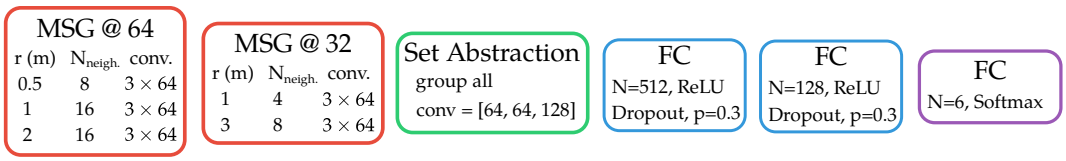


Figure 6.14: Network structure of the classification network. See text for details about the used abbreviations.

farthest-point-sampling. Neighborhoods with radii 0.5 m, 1 m and 2 m are created and targets within this area are grouped together so that each of the 64 core points has for each of the three neighborhoods either 8 or 16 neighbors. The core point – neighborhood tensors are then passed three times through a convolution layer with 64 kernels for feature extraction (see also Section 2.1.2.1, especially Fig. 2.4 for details). The generated feature vectors for each of the three neighborhoods are concatenated and used as input for the next MSG module along with the chosen 64 core points. Max pooling is used to select for each core point only one feature vector (instead of N_{neigh} feature vectors) with the highest activation caused by one of the N_{neigh} neighboring points. That is, one high dimensional feature vectors are ascribed to each of the core points which contain information about the neighborhood.

The second MSG module then selects 32 core points out of the 64 input points and the same process as in the first MSG module is repeated, except for different hyper-parameters, see Fig. 6.14.

The set abstraction module groups all of the 32 core points of the last step together (instead of selecting a subset, as done in the MSG modules) and creates one large feature vector which encodes information from all core points via convolution.

The network described until now could be seen as a *feature extractor* and the following part consisting of fully connected layers (abbreviated as FC in the figure) is then the part which classifies the generated feature vectors into the six classes. The first two fully connected layers with $N_1 = 512$ and $N_2 = 128$ neurons, respectively, are followed by dropout layers during training to avoid overfitting. In these dropout layers, connections are left out with probability $p = 0.3$ during both the forward and the backward path of the training. ReLu activation functions are used at all layers except for the last fully connected layer, where a softmax function creates a normalized probability vector.

Adam [123] is used as an optimizer with learning rate of $\alpha = 0.001$. In total, about 222 000 free parameters were tuned during training.

One training *epoch* is defined here as passing 20 000 batches with 100 samples through the network. The usual definition of an epoch is that *all* samples of the training set are passed once through the network but since data augmentation is

used and samples could be shown with and without augmentation, it becomes difficult to define when the network has seen each sample once. Choosing samples randomly from all available clusters proofed to yield stable results. No weighting between the classes was done so that each cluster contributes equally to the total loss.

After four of these epochs, the loss saturated at a minimum so that the training was stopped at that point. Each of the epochs took about one hour to train on an Nvidia GTX 1080 GPU.

6.3.1.3 *Results*

Just as in the previous experiments, evaluation was done using five-fold cross validation. The general trends observed there, namely that a) using only ground truth clusters for both training and inference yields higher results than working with the clusters created by DBSCAN$^{\pm}$ and b) using DBSCAN^{+} shows slightly better results than using DBSCAN^{-}, also hold true in this case. These experiments are therefore not shown again but only the case is considered where training is done on both ground truth clusters and DBSCAN^{+} clusters and evaluation is performed solely on DBSCAN^{+} clusters of the respective test set.

The network architecture shown in Fig. 6.14 performed best in the experiments. Using less neurons in the fully connected layers (e.g. 256 instead of 512) led to a considerable decrease in performance. Increasing the number of neurons beyond 512 did, however, not result in higher scores. Different number of convolution layers and sizes were tested for the MSG modules. Again, increasing the size of the convolution layers beyond 64 was not beneficial.

With the best performing architecture, an F_1 score of 0.771 on a per-target evaluation was obtained. The corresponding confusion matrix is displayed in the left part of Fig. 6.15. Once again, highest true positive counts are obtained for the `Passenger Car` class and the already discussed confusions between `Pedestrian` and `Pedestrian Group` instances is visible. A more detailed comparison with the other approaches will be done in Section 6.6.

As an additional experiment, the augmentation steps described previously were left out so that no further modifications to the training data were done. Interestingly, the difference in the final score is negligible so that no clear performance increase due to augmentation can be stated. Possible explanations for this are manifold. One reason could be that the training set is already large enough so that the chosen augmentation steps do not add additional information. This would be the case if the augmented samples were identical to an already existing sample. A different possibility is that the augmentation (especially dropping some of the

input points) alters the instances too much so that they no longer resemble the unchanged instances and hence the additionally learned information is of no use. It is also possible that augmentation only helps at the final stages of the training period in which the general differences between the classes is already learned and only the edge cases are refined. This stage might not have been reached in the performed experiments and would require more computation time with maybe decaying learning rates, freezing of initial layers or other configuration changes. It is therefore likely that additional fine-tuning of the training is necessary to increases the classifier's performance further.

In a different setting, the manually computed feature vectors that were used as input for the random forest and the LSTM, were also provided and concatenated with the generated feature vector. This was done right before the first fully connected layer so that the "classification layers" had access to the manually computed features as well as to the features generated by the previous layers of the network. The scores obtained from this experiment do not show any significant difference to the case were the manually computed features were not provided. This suggests that the automatically generated features already contain the information of the manually computed features and therefore information is only repeated.

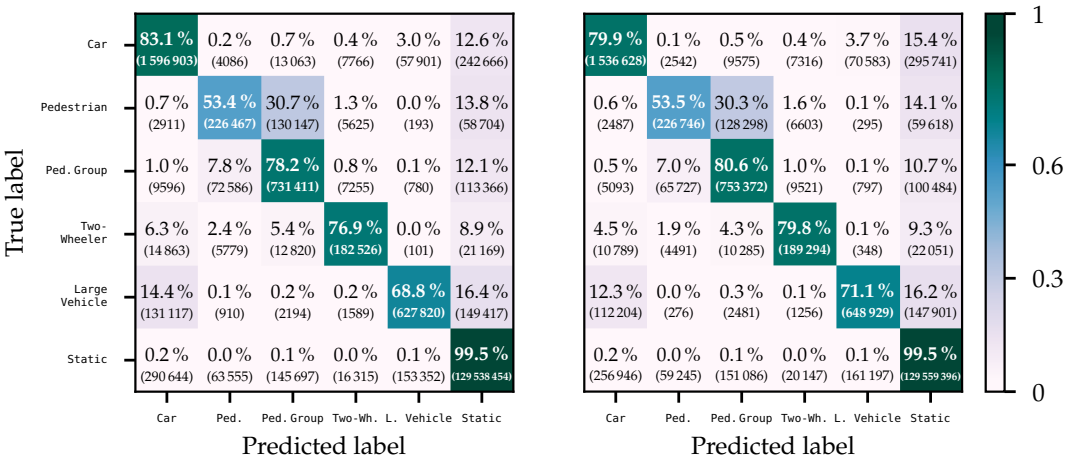


Figure 6.15: Per-target confusion matrices for the automatic feature extraction approaches. Left: Single time steps with PointNet++. Right: Combination of PointNet++ as feature extractor and an LSTM as classifier with eight time steps.

6.3.2 *PointNet++ as Feature Extractor, LSTM as Classifier*

A natural extension to the previously shown experiments is to interpret the PointNet++ architecture as a feature extractor and to train a different classifier on these

features. That is, the fully connected layers at the end of the network shown in Fig. 6.14 are substituted by a different classifier. More specifically, in this section it is explored how time information can be incorporated. A straightforward way to achieve this is by using an LSTM as a classifier instead of the fully connected layers. This new architecture is shown in Fig. 6.16. Inspired by the results of the LSTM approach discussed in the previous section, eight time steps are used with increasing time windows, i.e. targets of the next time step are added to the existing cluster so that the number of targets in a cluster increases over time.

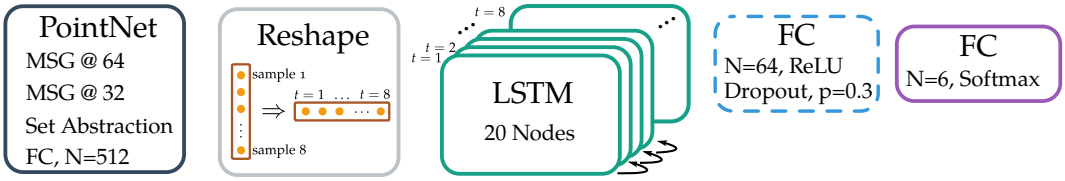


Figure 6.16: Network structure of the combination of PointNet++ with an LSTM. The fully connected block drawn with dashed lines is only used once to evaluate its necessity.

The dashed line around the fully connected layer in Fig. 6.16 indicates that this layer was only used once to assess whether or not fully connected layers after the LSTM increase the performance. This experiment revealed that having a single LSTM layer with 20 neurons followed by only one fully connected layer which maps the LSTM output onto the six classes, shows the highest performance. Including an extra fully connected layer or increasing the LSTMs layer size is not beneficial.

In the chosen architecture, the clusters of each time step are first passed as individual samples through the PointNet++ feature extractor. This results in one high dimensional feature vector for each of the "samples" (displayed as orange circles in the "Reshape" block in Fig. 6.16). The first part of the network is therefore agnostic to the fact that the presented samples have a time order. Reshaping the samples so that they form an array with eight time steps allows passing them through the LSTM as a sequence of feature vectors. Only the final output of the LSTM is used in the subsequent layers, i.e. the fully connected layers receive as input the output of the LSTM after all time steps have been processed.

The same optimizer as in the previous experiment is utilized in which only one time step was used. In contrast to the pure PointNet++ network, training is now split into two parts. In the first part, only the PointNet++ network is trained for one epoch and the LSTM output is discarded. In the second part of the training, the complete network including the LSTM is trained for three more epochs. The idea behind this is to ease the learning so that in the first stage the simpler PointNet++ architecture can be trained without interference of the LSTM, which would otherwise have to deal with initially less meaningful feature vectors. With this training setup,

however, the LSTM is only involved after it was assured that useful feature vectors are extracted in the previous stage by the PointNet.

After five fold cross-validation, an F_1 score of 0.776 was obtained with this method. The confusion matrix is displayed on the right side of Fig. 6.15 and is very similar to the one obtained if only a single time step is used. Small increases can be observed in the `Pedestrian Group`, `Two-Wheeler` and `Large Vehicle` class. However, performance on the `Passenger Car` class slightly decreases and the additional errors result almost entirely from classifying the `Passenger Car` instances as `Clutter`. This can be read off the two confusion matrices in Fig. 6.15 by comparing the two entries in the top right corner. The slight performance increases in the other classes overcompensate this so that in total the score is a little higher than in the case with one time step.

6.4 HUMAN PERFORMANCE

Until now, only the performance of different algorithms on the task of classification of clustered radar data was discussed. In this section, however, the classification performance of three different human labelers is evaluated. To this end, three skilled labelers who all collected several hours of annotation experience beforehand were challenged to ascribe class labels to some selected clusters. The goal of this small experiment is to obtain an estimate, how well humans perform on the task discussed in the last sections.

6.4.1 *Experimental Setup*

From all available and already annotated data, small sequences were extracted and the ground truth labels of the clusters were removed. For each of the three different lengths $T \in \{0.2\,\mathrm{s},\ 0.6\,\mathrm{s},\ 1\,\mathrm{s}\}$, 15 sequences were extracted which all contained a variety of different dynamic objects. All three labelers received the radar point clouds of the same sequences in which the ground truth clusters were still present. They were then asked to ascribe labels to each of the available clusters. The labelers had no access to the camera images they usually used during the labeling process. Otherwise, they were allowed to move forward and backward in the sequences in time steps of 100 ms and to display radar data from multiple scenes just as they did during normal labeling, see also Section 4.2. In contrast to the algorithms discussed before, whole scenes including the static environment were shown so that the labelers were not only confronted with the targets of one cluster (in a given time range) but rather with all dynamic (ground truth) clusters in the scene

Time bin	Basis	Passenger Car	Pedestrian	Pedestrian Group	Two Wheeler	Large Vehicle	Static
0.2 s	targets	612	75	141	107	306	28 562
	objects	46	12	10	9	11	–
0.6 s	targets	1757	256	285	327	609	81 281
	objects	127	44	30	28	27	–
1 s	targets	2949	145	825	247	1691	161 735
	objects	215	34	78	39	47	–

Table 6.3: Number of targets and objects used in the experiment for each of the three time bins.

as well as all other measured targets. This extra information could for example be used by the labelers to assess where roads and sidewalks are located in the scene and hence to verify if their selected class label is reasonable. In Table 6.3, the number of objects and the number of targets presented to the test candidates are listed per class and per sequence length. In this table the imbalance in the data set becomes apparent again: since in most scenes with pedestrians also cars were present, the `Passenger Car` class makes up the largest fraction of all objects. No `Clutter` clusters were shown to the labelers but they were allowed to assign this label to any cluster. Additionally, the three labelers were asked to note how long they needed for the classification of each sequence.

6.4.2 *Results*

Evaluating the classification scores and the time the labelers needed for completing the task shows that the classification skills are very differently distributed among the three labelers L_1, L_2 and L_3. In Fig. 6.17, two plots are displayed which show how long the three labelers needed for the "classification" of one sequence (left figure) and how well they performed in terms of the F_1 score (right figure). Labeler L_1 needed on average the least time to complete the task and at the same time achieved the highest scores (red curves in Fig. 6.17). In contrast, L_2 needed a considerable larger amount of time to classify the sequences but achieved only mediocre scores (pink curves). Labeler L_3 achieved scores similar to L_2 but was considerably faster (blue curves in Fig. 6.17). Interestingly, both L_1 and L_2 needed more time to complete the $T = 0.6\,\mathrm{s}$ sequences than for the longer $T = 1\,\mathrm{s}$ sequences.

Two factors influence how long the test candidates need to finish the classification task: the total number of objects for which a decision has to be made and the

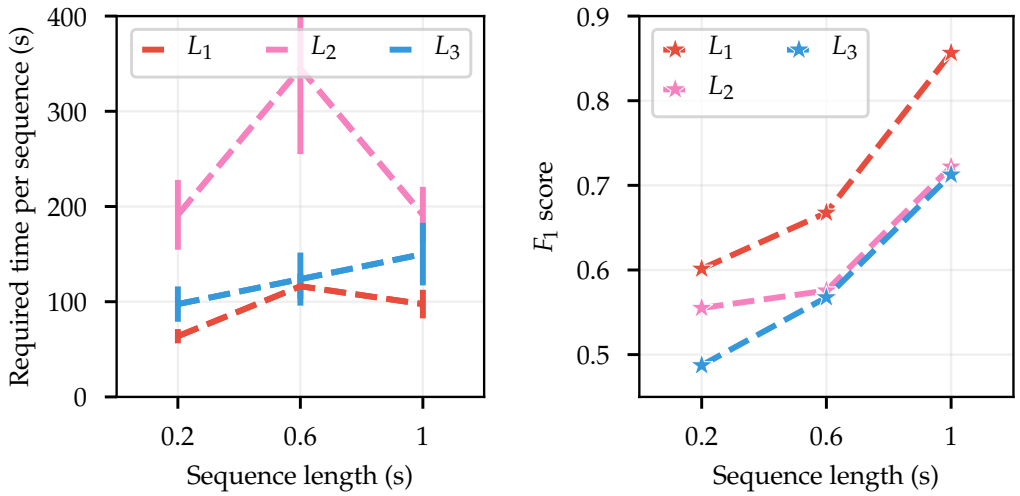


Figure 6.17: Left: Average duration each of the labeleres needed for "classification" of one sequence. Right: F_1 scores as a function of the length of the sequences.

difficulty of assigning a label to one object. As the scores in the right panel of Fig. 6.17 show, the classification task becomes easier if objects are visible for a longer period so that the test candidates can infer information from the movement of the objects within the scene. This indicates that for longer sequences the decision for one object becomes easier and can hence be made faster. But since the number of objects increases with sequence length (see Table 6.3), two opposing effects influence the labeling duration. For short sequences the difficulty is high but since there are only a few objects per sequence, the needed time is rather small. If the difficulty would remain constant, the labelers would need about thrice as much time for the $T = 0.6\,\mathrm{s}$ sequences since they contain about three times as many objects. However, for all three labelers the increase is smaller. Finally, for $T = 1\,\mathrm{s}$ the reduction of complexity outweighs the increase in objects (for L_1 and L_2). In total, the labelers needed between 1.2 and 3 hours to complete the experiment.

The two confusion matrices of L_1 and L_3 at the time bin $T = 0.6\,\mathrm{s}$ are displayed in Fig. 6.18. This time window was chosen because T is comparable to the lengths of the sequences presented to the LSTM classifier in Section 6.2.3. Labeler L_1 performed best at this sequence length and L_3 worst. Since no truly `Static` objects were presented to the labelers as candidates for dynamic objects, they naturally achieved 100 % true positives on this class. However, L_3 rejected some of the truly moving objects as `Clutter` while L_1 never decided to do so. From the confusion matrices it is visible that both test candidates were biased towards the `Passenger Car` class since they predicted this class most frequently (left column of the matrices). Unsurprisingly, they achieved high true positive rates for the `Passenger Car` class

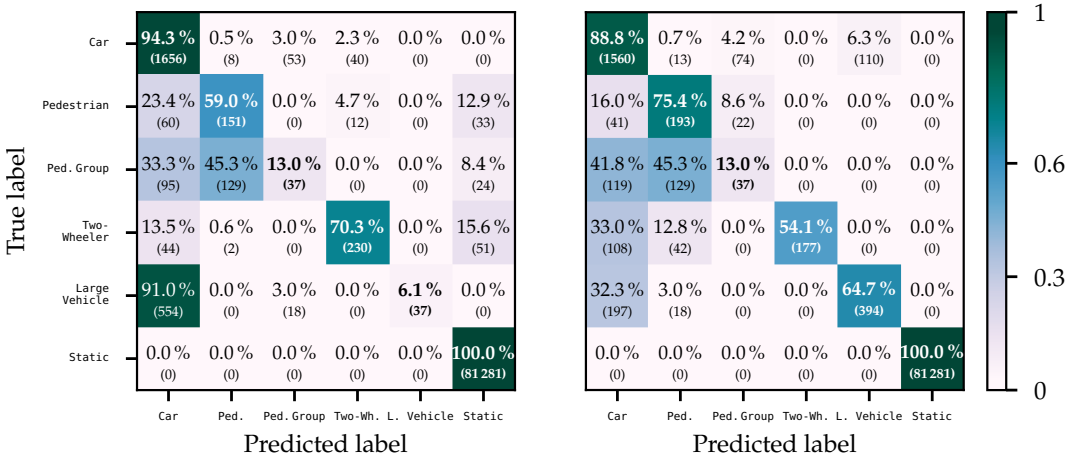


Figure 6.18: Per-target confusion matrices for the best performing labeler L_1 (right) and the worst performing labeler L_3 (left) for sequences of length $T = 0.6\,\mathrm{s}$.

itself. Both labelers struggled with identifying `Pedestrian Groups` properly and assigned either the `Pedestrian` or `Passenger Car` label. Candidate L_1 did a far better job in identifying `Large Vehicle` objects whereas L_3 usually assigned them to the `Passenger Car` class.

The results of this experiment show that temporal evolution of the clusters is a very important feature for "human classifiers" and that the performance fluctuations are quite large. Labeler L_1 had by far the most experience with annotating radar data, had also trained one of the other labelers and had assisted in writing labeling instructions. This leads to the unsurprising result that more experience in the task leverages both the speed and accuracy of the labeler. At short sequence lengths with $T < 1\,\mathrm{s}$, the algorithms shown in the previous section outperform the human test candidates by a great margin. However, for longer sequences humans can make great use of the temporal evolution of the clusters and use this information to reach similar or even higher performance. A larger number of skilled labelers would be needed to allow for more reliable statements.

Finally, Fig. 6.19 shows an excerpt of a scene the test persons had to annotate. Colors indicate to which semantic class the points belong to and convex hulls mark the individual clusters. The text labels above the objects mark to which class the three labelers ascribed the object. Correct assignments are drawn with a green background whereas errors are highlighted in red. Interestingly, L_2 classified the car next to the ego-vehicle as a `Two-Wheeler` even though the width of the cluster clearly indicates that this cannot be a motorbike, bicycle or any other member of the `Two-Wheeler` class. The same test candidate did not ascribe a label to one of the

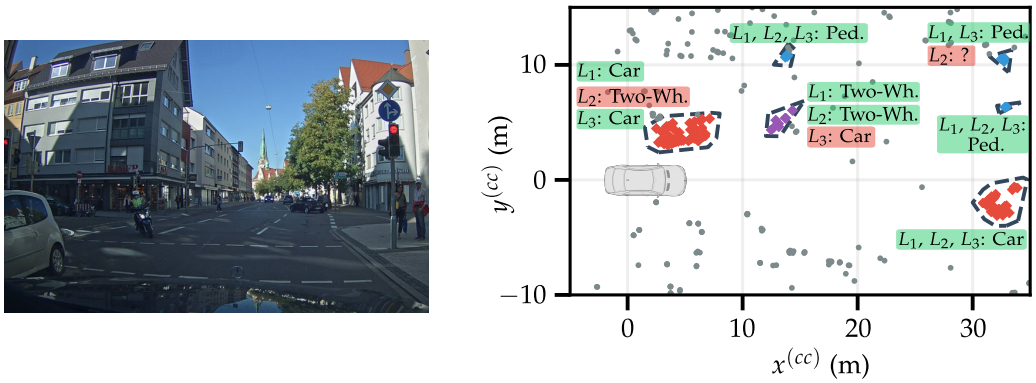


Figure 6.19: Example scene that was presented to the test persons. Left: Camera image of the scene (not shown during the experiment). Right: Point cloud with per-class colors. The text labels describe to which class the three test persons assigned the respective object.

pedestrians and later mentioned that he was too unsure to which class this object belongs to and hence left the cluster unassigned.

6.5 MOTION CLASSIFICATION AND BODY HEIGHT ESTIMATION OF PEDESTRIANS

The algorithms presented before are all designed to work on radar data of rather small time windows of a few hundred milliseconds. The intention is that these algorithms can provide semantic information as early as possible after the sensor delivered the data so that subsequent applications like tracking can benefit from this. Nevertheless, it is also worthwhile to investigate which additional semantic information can be extracted if data from longer time windows are available, e.g. if a pedestrian was tracked for over more than 3 s. Together with a master's student, the following two questions were investigated:

1. Is it possible to distinguish different motion types like walking, running, jumping or walking with crutches from each other?
2. Is it possible to estimate a pedestrian's body height solely from radar data if no elevation angle can be measured?

The results of the research activities are explained in detail in the resulting master's thesis [104], which was supervised by the author of this thesis, and in one further publication [105]. The contents of this section are therefore similar to these earlier publications.

This section summarizes briefly the individual approaches and their results. It is not intended to repeat all the findings described in the master's thesis but rather a short overview is presented. For a more elaborate discussion of the two topics, [104] should be consulted.

6.5.1 *Motion Type Classification*

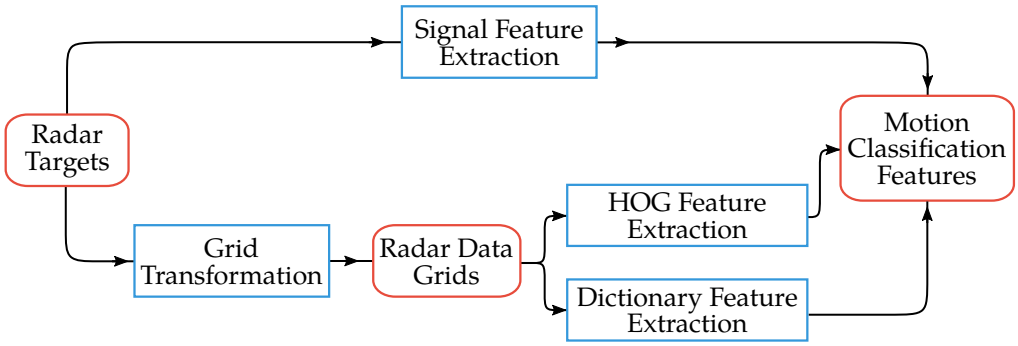


Figure 6.20: Feature extraction pipeline for the motion type classification of pedestrians.

As discussed in the previous sections, it is already a great challenge to distinguish pedestrians from other road users if only radar data from short time ranges are used. If, however, a combination of a clustering algorithm and a classifier is able to group all points belonging to the same pedestrian together, then the natural question arises whether the motion type of the pedestrian can be classified in more detail.

To this end, a data set was recorded in which eleven different test subjects performed the motion types `walk`, `run`, `jump`, `crutches`, `skateboard` and driving in a `wheelchair` repeatedly. Similar to the labeling process described before, ground truth clusters were created of all measured targets that belong to the same repetition of one test subject.

The radar targets of each cluster are the starting point of a feature extraction pipeline. Since only a small data set could be recorded in the limited time available for the master's student, an automatic feature extraction approach with for example a PointNet++ architecture was not considered but rather meaningful manual features were created.

A flowchart of the feature extraction process is displayed in Fig. 6.20. Red rounded boxes symbolize data input or output and blue rectangles mark a computation step. The feature set for the motion classification consists of three parts:

1. Signal features
2. Histograms of oriented gradients (HOG)
3. One-hot encoded classification results from a dictionary learning approach

The signal features are directly computed from the ego-motion compensated Doppler velocities $\hat{v}_{r,i}$ of the N radar targets in a cluster via

$$\mu_1 = \frac{1}{N-1}\sum_{i=1}^{N}|\hat{v}_{r,i} - \mu| \tag{6.1}$$

$$\mu_k = \frac{1}{N-1}\sum_{i=1}^{N}(\hat{v}_{r,i} - \mu)^k \qquad \text{for } k = 2,3,4, \tag{6.2}$$

where μ is the average value of all $\hat{v}_{r,i}$.

For the computation of the other two feature sets, a special grid map is computed first. As a precondition for the grid map computation, the pedestrian's trajectory has to be estimated. A linear trajectory fit on the measured targets is done via the *Random Sample Consensus (RANSAC)* scheme to obtain an estimation for the pedestrian's position for each point in time. Additionally, the pedestrian's Cartesian velocity v_{ped} can be directly extracted. After the trajectory estimation, a Frenet transformation is used to transform the Cartesian coordinates of the measurements into pairs of the form (d_i, n_i), where d_i is the coordinate parallel to the trajectory and n_i is the component orthogonal to the estimated route. For one example trajectory, this is illustrated in the left part of Fig. 6.21. These newly generated coordinate pairs can then be used to compute a 2D grid map: the deviation of each measured Doppler velocity from the estimated velocity of the pedestrian v_{ped} is entered in the grid cell corresponding to (d_i, n_i), and Gaussian distributed weights are applied to the central cell and its neighbors. Two examples of these grid maps are shown in the right part of Fig. 6.21. Further details about the trajectory estimation and the computation of the grid map can be found in [104].

These grid maps are then used for the computation of histograms of oriented gradients, which form the first set of features extracted from the grid maps. The second set of features stems from a dictionary learning approach [265]. Dictionary learning in combination with sparse coding [152] can be used as a stand-alone classifier. However, the one-hot encoded vector of class predictions – in which the entry associated with the predicted class is set to one and the remaining entries are set to zero – can also be used as an additional feature for a different classifier. More information about the dictionary learning approach can be found in both [104] and [105].

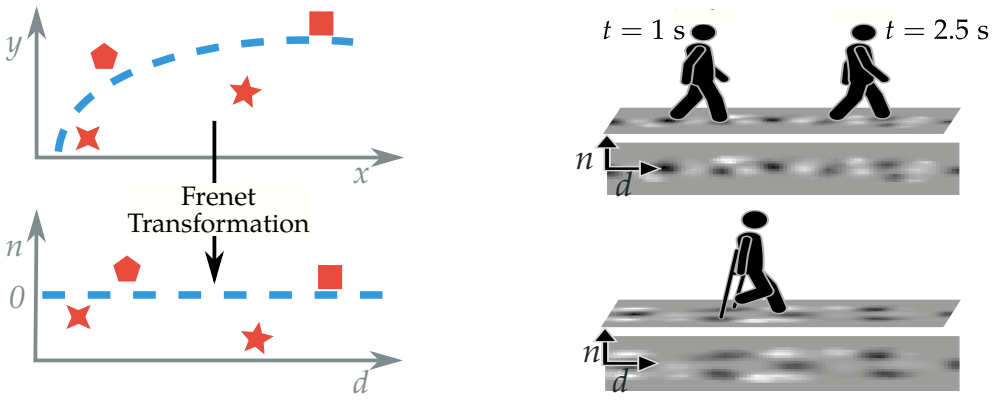


Figure 6.21: Left: Example for a Frenet transformation. Right: Two radar grids for the motion types `walking` and `crutches`. Images are based on figure 4.3 and 4.6 in [104].

Approach	F_1 Score
Signal Feat. only	0.64
HOG only	0.61
Dictionary only	0.75
Ensemble	0.79

Table 6.4: Motion classification scores.

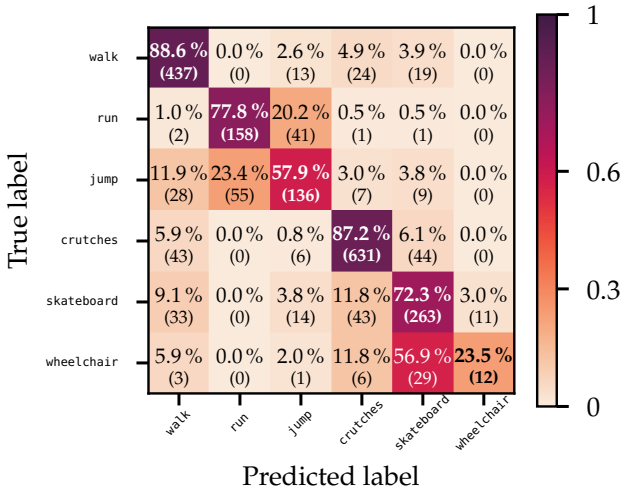


Figure 6.22: Confusion matrix of the motion classification approach. The absolute numbers in the matrix represent clusters with length 3 s.

The generated features are then used to train a random forest classifier. For evaluation, cross-validation is used to obtain reliable scores. In Fig. 6.22, the confusion matrix of the best performing random forest configuration is shown. The classes `walk` and `crutches` are classified best – possibly due to their characteristic grid map representations. The `wheelchair` class is identified worst, which might be caused by the fact that only data from a single individual were available. Increasing the data set would certainly leverage the classifier's performance on this class or at

least allow for a more detailed analysis so that more expressive features can be extracted.

In Table 6.4, F_1 scores of different classification approaches are summarized. If only the signal features μ_k are used, a random forest classifier achieves a score of 0.64 and if only the features obtained from the histogram of oriented gradients are used, the score drops to 0.61. On the other hand, if dictionary learning in combination with sparse coding is used as a classifier instead of a random forest and only the grid maps are used for feature extraction, the score increases to 0.75. Finally, if a random forest is used as a classifier and the three feature sets are used in combination, the highest score of 0.79 is reached.

6.5.2 *Height Estimation*

The second question raised in the introduction of this chapter deals with estimating the height of a person solely from radar data with two spatial measurement dimensions. This apparent paradox can be resolved when the other measurement dimensions of a radar sensor are included in the considerations. Since also the Doppler velocities are available, the fluctuations in the Doppler profile can be used to estimate either the stride length or the stride duration of a walking person. The stride length in turn is correlated with the body height of a person so that with the help of a reliable model, the body height can be calculated. This model can be either constructed from analyzing the physiology of humans and their gait or alternatively it can be learned from data.

The idea is based on the observation that during walking, a person has always one stationary leg on the ground while the other leg swings forward. Radar targets obtained from the legs of a moving person will then show an oscillation in the measured Doppler velocity whose frequency is directly proportional to the stride length l_s of the person. In the left part of Fig. 6.23, the Doppler velocities obtained from a walking person are plotted against the distance d along the person's trajectory where the oscillations become visible.

To extract the step frequency or alternatively the stride length l_s of a pedestrian, the computation pipeline displayed in Fig. 6.24 is used. In a first step, the person's trajectory is estimated and the Frenet transformation is applied, resulting in coordinate pairs (d_i, n_i) for each target. This step is identical to the one performed during feature extraction of the motion classification algorithm described in the previous section. With the radar targets transformed to this new coordinate system, the data points in the $d - \hat{v}_r$ space are smoothed with a moving average filter for non-uniform data so that evenly spaced data points can be sampled. One resulting curve

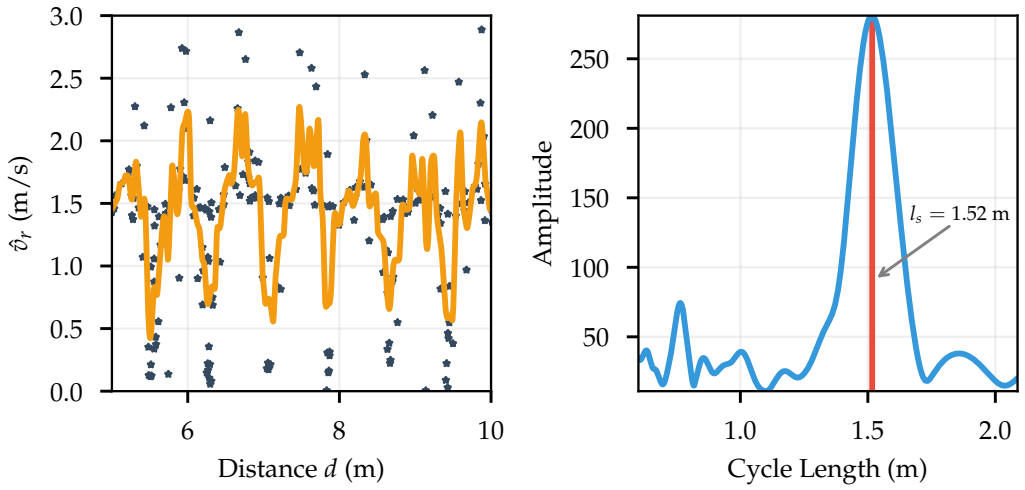


Figure 6.23: Left: Doppler over ground $\hat{v}_r$ vs. distance d. Blue stars show measurements, the orange line displays the low-pass filtered signal. Right: Fourier transform of the filtered signal (blue line) with highlighted maximum, corresponding to the stride length l_s (red line). Based on figure 4.11 in [104].

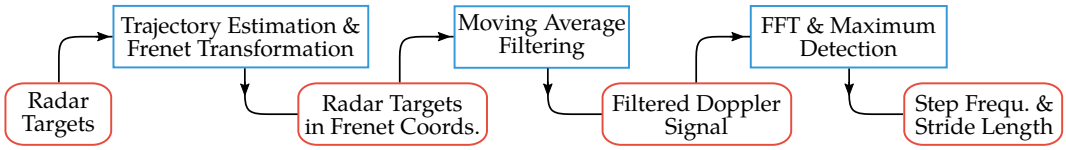


Figure 6.24: Pipeline of the stride length extraction for height estimation of pedestrians.

is displayed in the left part of Fig. 6.23 as an orange line on top of the measured targets. An FFT on this smoothed signal is used to find the dominating frequency of the oscillation. After appropriate scaling of the abscissa, the stride length l_s can be directly read off. Notice that the stride length l_s is commonly defined as twice the length of a single step so that $l_s = 2/f_{\max}$ where $f_{\max}$ corresponds to the Fourier frequency at which the maximum occurs. This is illustrated in the right part of Fig. 6.23.

A model created by Boulic and Thalmann [24] describes the relation between a person's stride length and body height as

$$h = \frac{l_s^2}{v_{\text{ped}} \cdot 1.346^2 \cdot 0.53}, \tag{6.3}$$

where the factor 0.53 stems from the observation that the height of thigh of a human is typically 53 % of their body height and the factor 1.346 was obtained from different experiments presented in [112]. The walking speed v_{ped} can be obtained from the trajectory estimation, just as it was done for the motion classification.

A data set with over 50 different persons walking towards and away from the radar sensors was recorded and manually labeled. The person's body heights were determined and stored together with the measured radar targets. From time windows of length 3 s, the stride length l_s and the pedestrians velocity v_{ped} was extracted via the procedure discussed before. Inspired by the Boulic-Thalmann model, the terms l_s^2/v_{ped}, l_s/v_{ped}^2, v_{ped}, l_s/v_{ped}, l_s, $v_{\text{ped}} \cdot l_s$, $v_{\text{ped}}^2 \cdot l_s$ and $v_{\text{ped}} \cdot l_s^2$ were calculated and provided as features to a random forest regressor. Training and evaluation of this approach was again done using cross-validation. In the left part of Fig. 6.25, the feature importances of the different terms are displayed. In accordance with the Boulic-Thalmann model, the term l_s^2/v_{ped} was ranked as the most important feature for the height estimation. However, the regression forest outperformed the Boulic-Thalmann model by a considerable margin as it becomes apparent from the right part of Fig. 6.25. In this plot, the absolute values of the differences between the estimated and true body height are displayed for the different body heights. The blue bars indicate the mean error for a given body height range and the black lines on top visualize the standard deviation of the error. Additionally, the average error of the Boulic-Thalmann model is inserted as a red line. The mean absolute error of the random forest approach lies with about 6.7 cm far below the average error of the Boulic-Thalmann model. The best performance is achieved for heights around 175 cm. Given that the majority of persons recorded in this data set have a body height around this value and only few very tall or very short people were available, the data set is highly imbalanced and the random forest tends to focus on the majority values. Nevertheless, this proof of concept shows that body height estimation is – up to a certain extent – indeed possible even though only two spatial dimensions are measured by the radar sensor.

6.6 COMPARISON AND SUMMARY

In this chapter, different methods to classify clusters of radar targets into the six classes `Passenger Car`, `Pedestrian`, `Pedestrian Group`, `Two-Wheeler`, `Large Vehicle` and `Clutter` were introduced, tuned and evaluated on a recorded data set. First, methods based on manual feature extraction were applied and afterwards two methods were employed which generate features from the raw clusters so that the manual feature calculation step becomes obsolete.

The per-target scores of all four approaches are summarized in Table 6.5. Only the best performing setting is displayed in the table. Except for the first entry in the table, training was done in all cases with ground truth clusters as well as clusters from DBSCAN$^+$, and evaluation only on DBSCAN$^+$ clusters. For a detailed comparison with basic DBSCAN, the individual subsections of this chapter can be

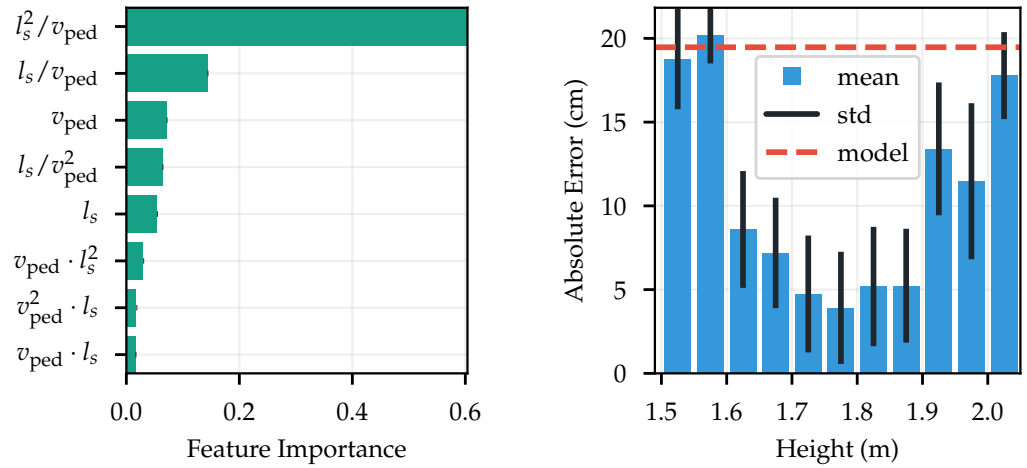


Figure 6.25: Left: Feature importances during the height estimation. Right: Absolute values of the predicted heights. The mean errors for each height bin are drawn along with the standard deviation. The red line indicates the mean error of the Boulic-Thalmann model. Based on figures 4 and 5 in [105].

consulted. The first entry in the table belongs to the most basic setting in which DBSCAN$^-$ is used for clustering. This random forest approach can be seen as the baseline set by the previous works [289], so that the comparison highlights the improvements obtained through methods introduced in this thesis. From this table it becomes obvious that if DBSCAN$^+$ is used, the difference in F_1 score between the best and worst performing classifier is with about 0.015 rather small. However, a considerable jump to the baseline of up to 0.06 can be observed. The rather simple approach of using a manually created feature set with rather obvious choices for the individual features and feeding those into a random forest classifier performs surprisingly well. The more elaborate methods using PointNet++ or a

Method	F_1	Precision	Recall	Confusion Matrix
Random Forest, DBSCAN$^-$	0.719	0.753	0.708	Fig. 6.9 left
Random Forest	0.761	0.785	0.746	Fig. 6.9 right
LSTM	0.766	0.774	0.763	Fig. 6.13 right
PointNet++	0.771	0.777	0.767	Fig. 6.15 left
PointNet++ & LSTM	0.776	0.781	0.774	Fig. 6.15 right

Table 6.5: Comparison of the scores obtained with the four different approaches.

combination of PointNet++ and an LSTM perform only slightly better. Crucial for a well performing algorithm is that the exact same clustering method is used for both training and evaluation. If training is done only on ground truth clusters and evaluation on the automatically generated clusters, then the performance of the total system decreases considerably. A simple re-clustering of existing ground truth clusters as it was done in [232] turned out to be not sufficient.

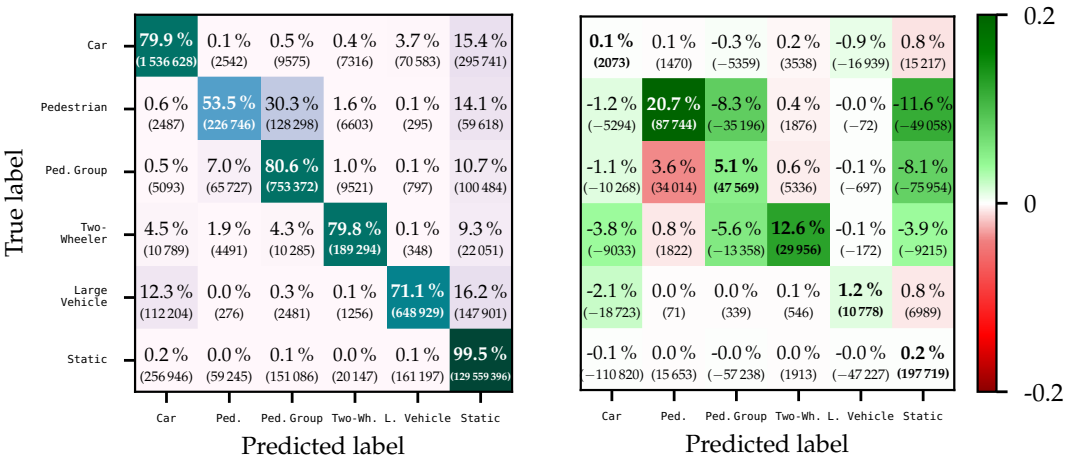


Figure 6.26: Comparison of the most basic approach using standard DBSCAN for clustering and a random forest as classifier with the one using DBSCAN$^+$ for clustering and PointNet++ & LSTM as classifier. Left: Confusion matrix of the best performing PointNet++ & LSTM architecture. Right: Differences to the random forest confusion matrix.

In Fig. 6.26, the confusion matrix of the best performing method which combines PointNet++ with an LSTM and uses DBSCAN$^+$ for clustering is repeated (original in the right panel of Fig. 6.15) and the differences to the baseline random forest approach with DBSCAN$^-$ are displayed (confusion matrix in the left panel of Fig. 6.9). In the right panel of Fig. 6.26, green color is used to highlight improvements and red color shows where the random forest approach yielded better scores. This definition implies that positive values on the diagonal and negative values on the off-diagonal of the confusion matrix are colored green whereas negative values on the diagonal and positive values on off-diagonal elements are colored red. The color bar on the right side of the figure is hence valid for the diagonal entries. For off-diagonal entries, the color bar ranges from -0.2 (green) to +0.2 (red). The listed percentage values are relative to the total number of targets of the respective class. The greatest performance increase can be seen at the `Pedestrian` class where the true positive counts increases by over 20 percentage points. Main driver of this performance boost is that less targets are erroneously classified as `Static`. This in turn is caused by two effects: Firstly, less `Pedestrian` clusters are overlooked with DBSCAN$^+$ which are otherwise automatically classified as `Static` and secondly,

the PointNet++ & LSTM combination shows higher performance in the distinction between `Pedestrian` and `Clutter` clusters. Since `Clutter` predictions are mapped onto the `Static` class for the per-target evaluation, higher performance on the `Clutter` class directly influences the score of the `Static` class.

The second highest performance increase stems from the `Two-Wheeler` class, where the confusion with `Passenger Car` objects and `Pedestrian Group` instances could be decreased in favor of more true positives. The biggest decrease in performance is visible at the `Pedestrian Group` – `Pedestrian` confusion. However, closer inspection of the `Pedestrian Group` row shows that this is a non-critical performance decline. Firstly, the true positive count of `Pedestrian Group` targets increases. Secondly, the increase in wrong `Pedestrian` predictions can be entirely explained by the decrease of `Clutter` (or `Static`) predictions. Therefore, the error shifted from predicting a `Pedestrian Group` as `Clutter` (or not clustering it all – both cases would result in a `Static` label) towards predicting a `Pedestrian Group` as a single `Pedestrian` which is much more favorable.

In Fig. 6.3, different decision surfaces of the random forest classifier are shown. Since visualization of the complete feature space with 17 dimensions is impossible, the surfaces are shown for classifiers for which only two features are used. A different method to display the interplay of the features is to embed the high-dimensional feature space into two dimension with the help of t-SNE [257], see also Section 2.2. This is done in Fig. 6.27 where for 300 clusters of each class manually created features and automatically created features of the PointNet++ approach are displayed after the embedding in two dimensions. Since the scaling of the axes has no real meaning after the embedding, labels on the axes were left out for brevity. Each point in the figure symbolizes the embedding of one feature vector and color indicates for which class the feature vector was created. For the PointNet++ approach, feature vectors are defined as the output of the set abstraction module. A clear structure can be seen in the embedding of the automatically generated features that allows even in two dimensions the creation of simple decision boundaries. For the manually created features, however, many regions can be identified where multiple classes overlap each other and feature vectors of each class are spread out over the whole space instead of being clustered together. It is therefore understandable that a classifier which uses the manually created feature vectors as input has a harder job to do than a classifier which processes the automatically extracted features. Since both the random forest and the LSTM performed reasonably well on the manually created feature vectors, the full feature space does allow the creation of reasonable decision boundaries and the embedding into two dimensions does not reflect the information content of the features properly.

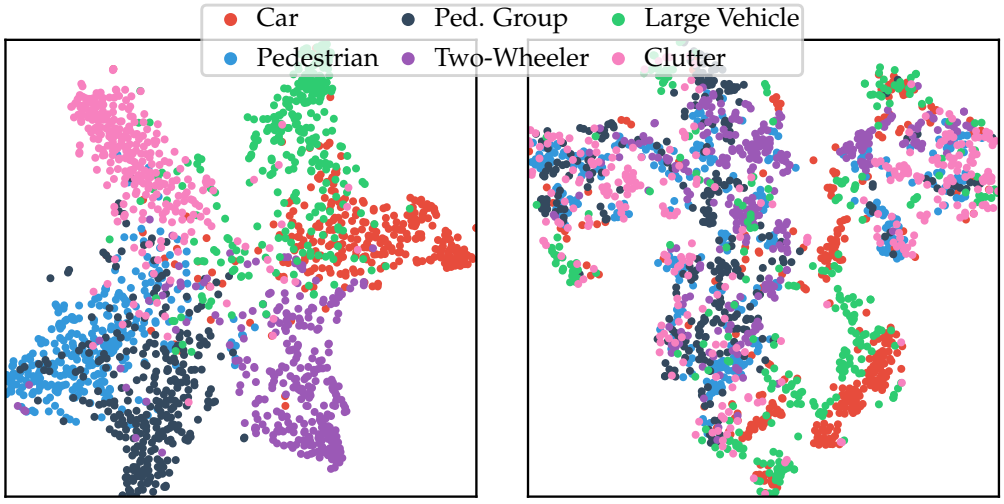


Figure 6.27: Embedding of the automatically generated features (left) and the manually computed features (right). For embedding into two dimensions, t-SNE was used.

Incorporating time information in the classification step proves to be non-trivial. Even though the best performing architecture includes an LSTM and makes use of up to eight time steps, the difference to methods that do not make use of the temporal evolution of the clusters is small. This stands in great contrast to the performance humans show on the same task. A clear correlation between the length of the sequences shown to the annotators and the achieved score is visible, cf. Fig. 6.17. For sequences with a length of 1 s, the best performing test person outperformed the proposed algorithms. This hints that the way temporal evolution was incorporated into the algorithms was not optimal and that there is more potential to utilize this information. However, during the experiment the subjects could see the entire scene and not only the extracted clusters so that they had access to more information than the proposed classifiers.

7

SEMANTIC (INSTANCE) SEGMENTATION OF RADAR POINT CLOUDS

In the last chapter it was shown in detail how variation of the clustering method influences the classifier's performance. With perfect clustering, i.e. with a clustering algorithm that reproduces ground truth clusters, a classifier has a far easier job than a classifier that has to work on the output of an imperfect clustering algorithm. Especially larger objects like trucks tend to break into multiple smaller clusters since there are often larger gaps between targets originating from the front and back parts of the vehicle. This in turn influences the feature extraction process and finally the classifier's output. Further complexities of the clustering process are listed in section 5.2. It is therefore a natural thing to look for alternatives to the clustering step so that feature extraction and classification are no longer affected by this prepossessing step. Therefore, in this chapter semantic segmentation and recurrent instance segmentation algorithms are constructed and evaluated. The results shown in this chapter were previously published in [233] and [235]. It should be noted that in [235] a complete processing pipeline for the classification of both static and the dynamic objects is shown. However, only the part regarding moving objects is discussed here since classification of static objects was contributed by the second author of the article.

As input to these algorithms, the spatio-temporal point cloud constructed from the measured radar targets is used. An alternative to using point clouds as input would be a grid mapping approach. This is for example used in the classification and semantic segmentation of static radar targets [144]–[147]. One problem that arises when grid maps are used is the varying density of the individual radar targets. In typical scenes, only a small fraction of the 2D measurement space is filled with targets while the rest remains empty. In areas where targets are detected, i.e. especially where dynamic objects are located, the target distribution is rather high and the spatial distribution of the targets is an important feature for the classification task. This fact necessitates that a grid with cell sizes $< 1\,\mathrm{m}$ is needed even though most parts of the grid remain empty. Another issue is that grid mapping algorithms are designed for situation where data are accumulated over

longer periods, so that measurement noise is averaged out and true structures become sharper. It is a non-trivial task to align this with the requirement that the classification information of dynamic objects should be available on short time scales, as it is required for the algorithms discussed in this work.

After related work is discussed in the following section, details about the chosen algorithms for semantic segmentation and recurrent instance segmentation are given and the training procedure is explained. Thereafter, the results obtained from the two approaches are presented and discussed. Similarly to the previous chapter where classification scores were shown, the performance of human annotators on the instance segmentation task is analyzed.

The following own contributions are made in this chapter:

- Introduction of a novel neural network architecture for recurrent instance segmentation on point cloud data
- Performance comparison of a semantic segmentation approach with an instance segmentation network
- Assessment of human performance on the instance segmentation task
- Evaluation of ensemble methods for instance segmentation

7.1 STATE OF THE ART

Related work from three categories is displayed in this section: Firstly, semantic (instance) segmentation network architectures are presented which highly influenced the whole machine learning community. These pioneering works all use images as input but inspired the subsequently published articles that work with point cloud data. These kind of networks are presented in the second part of this section. Finally, literature about semantic (instance) segmentation of radar data is discussed.

7.1.1 *Neural Network Architectures for Semantic (Instance) Segmentation*

The foundation for neural networks which perform a semantic segmentation of an image was laid in 2015 by the authors of [148]. They proposed that a CNN could not only be used as a feature extractor and classification network but could also be used to predict a class label for each pixel in an image. Several network architectures like GoogLeNet [252] – originally designed for classification tasks –

were extended to semantic segmentation networks and their superior performance over other approaches was demonstrated. Notably, they outperformed also the famous R-CNN network [77] by a great margin. The R-CNN architecture was until then one of the best performing structures. Updated versions of R-CNN like "Fast R-CNN" [76], "Faster R-CNN" [204] and "Mask R-CNN" [92] were developed thereafter. In addition to bounding boxes and class labels for each box, "Mask R-CNN" also creates a pixel-wise classification of the predicted objects.

Despite many improvements in this direction, one shortcoming of these networks is still their relatively large inference time of about 200 ms per frame [92]. This observation caused the development of network architectures that deliver results faster at the cost of being slightly less accurate. Prominent examples are the YOLO network [203] developed by Facebook and the SSD [142] developed by researches from Zoox and Google. Both networks are object detectors, i.e. they predict bounding boxes and class labels for each object in an image but do not perform a semantic segmentation.

A different approach is used in [138], where a polygon fit is used to predict object instances based on proposed instance masks, which in turn are generated by a segmentation network.

A different way of predicting instances is used in [256]. For each pixel in the image, they predict the direction to its corresponding instance center. The predicted angle is discretized into a fixed number of classes so that the task is tailored for a fully convolutional network. Yet another way to create object instances from a semantic segmentation output is to learn the energy levels of a watershed transformation [6], [11]. A direction network is used to predict the direction of descent of the watershed energy level for each pixel. This is used as input to another network which predicts the final energy level so that a single cut through this energy landscape produces the desired instances.

A more thorough overview about semantic segmentation networks is given in [143].

7.1.2 *Works on Point Cloud Data*

Working with raw point clouds instead of images requires novel network architectures since the input data is usually not ordered and neighborhood relations are a priori unknown. As almost all CNNs for images make great use of the neighborhoods of each pixel and use them for feature generation, new ways have to be developed to obtain this information. Three categories emerged into which most approaches can be sorted: voxel-based models, point-based models and graph-based

architectures. Each of these categories is discussed in the following subsections individually.

7.1.2.1 *Voxel-Based Approaches*

One natural idea to create neighborhood relations is by simply dividing the measurement space into equally sized cells. For 2D data, this corresponds to a grid-mapping process so that an image-like structure is created. For data with three spatial dimensions, so-called voxels are created which are simply elements in a regular 3D grid. The advantage of these approaches is that in the 2D case all network architecture used for images can be directly applied. In the case with three spatial dimensions, the usual convolution operation can be extended to 3D so that neighborhoods in all directions are considered. Two large disadvantages exist with this approach. Firstly, slicing the input space into equally sized parts and storing values for each of the cells is very memory demanding. For sparse input data, many cells remain unoccupied so that large parts of the required memory remain unused. Secondly, a cell size has to be fixed so that a resolution limit is directly enforced. If the input data shows large variations in density and is at the same time very sparse, then it is a non-trivial task to find a resolution which covers the essential properties of the data but at the same time is not too memory demanding. Most commonly, the chosen resolution causes that multiple input points fall into the same cell so that the network does not see the true input data but rather a sampled version.

Prominent examples for voxel-based networks are VoxelNet [291] (created by a research team at Apple), MSNet [266] or 3D U-Net [41]. VoxelNet operates directly on the input point cloud so that the embedding is learned during training. Based on these pioneering works, many network variants emerged which for example use different encoding schemes to transform the point clouds into a regular grid [131].

7.1.2.2 *Point-Based Approaches*

Point-based approaches directly use point clouds as input and do not demand a voxeling step during processing. PointNet [199] gained widespread attention due to its novel approach to deal with unstructured data. The architecture takes three properties of point clouds as design principles: a) the input points are unordered and order of the points should not matter, b) local structures in the point cloud contain relevant information, c) a set of transformations (e.g. rotation or translation) applied to the whole point cloud should not alter the semantics. Key to working with unordered input is to apply a symmetric function at relevant positions in the network so that the output of this function does not depend on the order of the input values. In PointNet, maxpooling is used as the symmetric function so that

only the point with the highest activation for a given convolution kernel contributes to the subsequent layers. In PointNet++ [198], the feature extraction step is extended so that local properties of the point cloud are better captured. As demonstrated in the previous chapter, the PointNet structure can be used for classification and in this chapter it will be used as a basis for semantic segmentation. More details on the individual PointNet modules are given for example in Section 7.2.1.1. The so-called "Frustrum PointNet" [196] is an extension of PointNet designed to fuse information from camera and point cloud data in a single network.

The great advantage of the PointNet approach is that the network operates directly on the true input data so that no resolution artifacts are introduced and empty areas do not have to be modeled. However, to obtain information from the local neighborhoods of a point, explicit nearest neighbor searches have to be performed at various stages in the network.

Combinations of the PointNet architecture with voxel- or grid-based approaches allow to use PointNet only for embedding and 2D or 3D convolutions at later stages [23], [205].

Improvements of PointNet++ include for example MHNet [139] which extracts similarly to the original PointNet++ features at different scales but explicitly fuses the features from different scales to one combined representation. Thereby, local features of different abstraction levels are combined again, which the original PointNet++ does not do. Conditional random fields are used as a post-processing step to smooth the predictions. SpiderCNN introduces parametrized convolution kernels which can operate directly on point clouds [283]. In PCPNet, surface normals are estimated with the help of local patches that are encoded using a variant of PointNet modules [86].

An entirely differently route is taken by Kd-networks [125]. The authors use the kd-tree indexing structure as a basis representation of the point cloud data instead of uniform grids. Parameter sharing of the trainable weights is implemented similarly to regular convolution operations and standard backpropagation can be used for training.

7.1.2.3 *Graph-Based Approaches*

Finally, graph-based approaches make up the third category. The idea of these approaches is that convolution operations can be defined on graphs so that contextual information can be propagated. In [130], the input point cloud is first partitioned into geometrically homogeneous subsets, the so-called superpoints. From these superpoints, a new graph is created which is much smaller than a graph constructed

on the whole point cloud so that the subsequent convolution operations can be performed more efficiently. Spectral graph convolution on local point neighborhoods is used in [264] in combination with recursive cluster pooling. In their network, some of PointNet's modules are used for farthest point sampling and grouping but the max-pooling step employed in PointNet and PointNet++ is replaced with a recursive clustering and pooling strategy. The proposed architecture can be used for both classification and semantic segmentation.

A mixture of a graph-based approach and a grid-based approach is shown in [64]. Instance segmentation is performed with the help of a bird's eye view on a 3D point cloud, which in turn is obtained by a grid-mapping process. A fully-convolutional network is applied to this 2D map to predict object instances and feature vectors for each point visible from the top-down view. The points left out in this process are treated in a second step, where a graph neural network is used to propagate the existing feature vectors. Finally, instance grouping based on the predicted feature vectors is performed by using the mean shift algorithm. The graph constructed in their work is borrowed from [267], where the so-called "EdgeConv" network module is introduced. This network module can be directly integrated into the PointNet architecture while reducing memory footprint and inference time drastically.

An extension to simple graph convolutional neural networks [30] is presented in [281] where features from multiple scales are extracted for a more fine-grained segmentation of the input point cloud.

7.1.3 *Works on Radar Data*

Semantic segmentation using (automotive) radar data is dealt with only in few articles.

Depending on the data representation, different approaches are used. For radar grid maps, neural networks similar to the ones used for images can be utilized. For example, semantic segmentation of the static environment is performed in [147], [194], [235]. A grid mapping algorithm precedes the actual segmentation so that in a first step an image-like structure is created from the radar targets. Grid maps describing radar specialties, e.g. the distribution of RCS values of objects, can be created to facilitate all measurement dimensions of the sensor. Since radar data is relatively sparse, large portions of the emerging grid map remain empty so that careful weighting of the loss function is necessary during training [235].

In [163], camera and radar data is fused together as a combined input to a neural network that predicts bounding boxes for cars. From the 3D radar data, grid maps

with multiple height layers are created so that the third dimension of the point cloud is encoded in different layers of the grid map. A small toy data set [162] was released to illustrate the abilities of the radar used in the classification task.

Another approach dealing with grid maps is presented in [243]. The goal of their work is not to provide semantic information but rather to replace the traditional occupancy grid mapping with a learning procedure so that a complex inverse sensor model is obtained. Input to the network is a simple 2D grid of clustered radar data. An encoder-decoder architecture with skip connections is chosen which creates an occupancy grid map with the three classes "occupied", "free" and "unobserved".

Working directly on radar point clouds for semantic segmentation is done in [52]. They use PointNet++ in combination with "Frustum PointNet" [196] to obtain a semantic instance segmentation of the point cloud. Even though their approach can be used for an arbitrary amount of different classes, the algorithm is only evaluated on the two classes `Passenger Car` and `Clutter` due to a lack of annotated data. For data generation, a single test vehicle is recorded for which its ground truth position is known. Thereby, all radar targets recorded in the vicinity of the car's position can be automatically labeled. The downside of this approach is that the network gets a high bias towards the object size of the single test vehicle. This is especially problematic since a bounding box estimation is used for the instance segmentation. In principle, the network could have learned the car's size by heart to achieve high scores but without the ability to generalize to different objects or classes.

7.2 METHODS

In this section, the network structures and training pipelines for the semantic segmentation approaches as well as for the recurrent instance segmentation are presented. In the first part, only the semantic segmentation network is discussed, before in the second part of this section a novel architecture is introduced. In each of these two subsections, first the general architecture of the networks is described, before details about training parameters and other hyper-parameter choices are explained.

7.2.1 *Semantic Segmentation*

The presentation of the semantic segmentation approach is split into two parts. First, the general network structure is introduced and the processing pipeline of the

input data is explained. Then, training parameters like learning rate and weights for the loss function are listed.

7.2.1.1 *Network Structure*

The basis for the neural network is the PointNet++ semantic segmentation architecture [198] which is similar to the classification architecture already used in section 6.3.1. The network was originally designed for 3D point clouds, i.e. for points with three spatial dimensions. The radar data used in this work has, however, only two spatial dimensions but with the Doppler velocity $\hat{v}_r$, the measurement time t and the RCS value σ three additional feature dimensions exist which have to be considered. Secondly, radar data shows much greater differences in density and sampling rate than the 3D data for which the network was constructed. Therefore, the individual network modules were adapted to reflect these different properties.

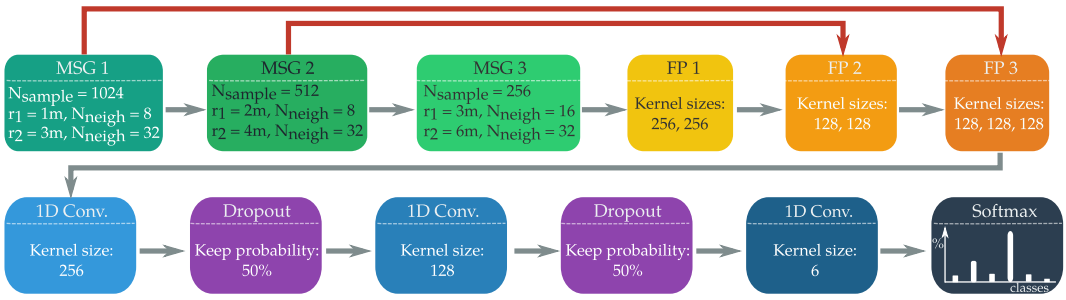


Figure 7.1: Structure of the semantic segmentation network. The convolution kernel sizes of the three MSG modules are [[32, 32, 64], [64, 64, 128]], for the first two modules and [[64, 64, 128], [64, 64, 128]] for MSG 3. The figure was published beforehand in [233].

The structure of the network is shown in Fig. 7.1. Input to the network is a point cloud $\mathcal{P}_0$ with N_{in} points p_i. Each of these points consists of its two spatial coordinates $(x^{(cc)}, y^{(cc)})$, the measured Doppler velocity (over ground) $\hat{v}_r$, the RCS value σ and the measurement time t. The measurement time is normalized so that the earliest timestamp occurring in the scene is set to zero and all other measurement times are relative to it.

Three MSG modules are followed by three feature propagation modules. The MSG modules create high dimensional feature vectors for a subset of the input points. These feature vectors contain information about the surrounding measurements so that semantic information can be extracted. Since, however, a class probability vector is desired for *each* of the input points, the generated features have to be propagated back to the complete point cloud. This step is performed in the *feature propagation* modules. Their task is to transfer the high dimensional feature vectors

from sparser point clouds to the next higher layer. For example, the output point cloud $\mathcal{P}_3$ of MSG module three contains only 256 points. The next denser point cloud $\mathcal{P}_2$ is the one created by MSG module 2 and contains 512 points. To propagate the features from $\mathcal{P}_3$ to $\mathcal{P}_2$, for each point in $\mathcal{P}_2$, the three nearest neighbors in $\mathcal{P}_3$ are selected. The weighted average of the feature vectors of the three points is then used for the point in $\mathcal{P}_2$, after passing them through a set of convolution layers. Weighting is done via the inverse spatial distance, i.e. points closer to the seed point are weighted higher.

After the third feature propagation layer, each of the N_{in} input points is attributed with a high-dimensional feature vector f_i. Using 1D convolutions, the feature vectors are mapped sequentially down to $N_{\text{class}} = 6$ dimensions, see also Eq. (2.16). During training, dropout layers are used for regularization and to avoid overfitting. Finally, a softmax layer is applied to convert the six values to a probability vector whose entries all lie in $[0, 1]$ and sum to one.

7.2.1.2 *Training Parameters*

As with all neural network architectures, hyper-parameters have to be chosen prior to the final training and evaluation rounds. This was done in accordance with [233] on randomly selected subsets of the training data. The results of this hyper-parameter tuning are not presented here but rather the resulting best performing architecture is shown. The chosen parameters of each module are listed in Fig. 7.1 and the number of input points is set to $N_{\text{in}} = 3072$.

For both training and evaluation, point clouds of a fixed time length T are used as input. The architecture is optimized for $T = 0.5\,\text{s}$ but different time lengths are evaluated in section 7.3.3. Similarly to the classification network in section 6.3.1, the input point cloud is over-sampled if less than N_{in} targets were measured during time T. If more than N_{in} targets are measured in the respective time period, the N_{in} points with highest Doppler velocity $\hat{v}_r$ are chosen. During inference, the same value for T is chosen and if targets are left out due to the input size restriction, the class label is propagated from the nearest neighbor for which a prediction was made. In this way, predictions for all input points can be made.

Cross-entropy is used as a loss function. Let y_{ik} and $\hat{y}_{ik}$ describe the ground truth probabilities and predicted class probabilities of point i. Then $y_{ic} = 1$ if point p_i belongs to class c and $y_{ik} = 0$ for all $k \neq c$. The six values $\hat{y}_{ik}$ with $k = 1, 2, \ldots, 6$ are

the output of the network after the softmax layer. The loss function is then defined as

$$L_{\text{sem}} = \frac{1}{N_{\text{in}}} \sum_{i=1}^{N_{\text{in}}} \omega_i \sum_{k=1}^{6} y_{ik} \log \frac{1}{\hat{y}_{ik}}, \tag{7.1}$$

where ω_i is the weight factor of point i. In preceding experiments, the weights ω_i were fixed to $\omega^{\text{static}} = 1$ and $\omega^{\text{dynamic}} = 7$ for points of the static and any dynamic class, respectively.

Adam [123] was used as an optimizer with an initial learning rate of $\alpha = 0.001$. The learning rate was decreased every 300 000 samples by a factor of 2. Training was done with a batch size of 24 for 25 epochs. Including dynamically loading of training data from disk, training takes about 45 min per epoch on an Nvidia GTX 1080 GPU.

Just as for the PointNet++ classification network, the following two augmentation steps are done during training to enrich the input data. With a probability of 80 %, the targets are shifted randomly in space, time, velocity and RCS. The distances of the shifts are chosen from a normal distribution centered at zero and are clipped to 0.2 m for the spatial shift, 0.5 m/s for the velocities and 0.5 dBsm for the RCS. Secondly, random dropout was applied to the targets so that a target was removed from the scene with a probability of 30 %.

7.2.2 *Recurrent Instance Segmentation*

Similar to the semantic segmentation network presented in the last section, now the novel recurrent neural network structure is presented in two steps. Again, the general structure is explained first, before training parameters and the loss functions are described in a second step.

7.2.2.1 *Network Structure*

The network architecture used for the recurrent instance segmentation is best described by splitting the network into a set of modules, where each module fulfills one individual sub-task. These modules and their interaction is described in the following subsections. The recurrent nature of the network is achieved through the two modules *memory abstraction* and *memory update*. A simple LSTM cannot be incorporated into the network since if it was applied to the individual targets, an ordering of the points would be needed. If it was applied to the created instances, an association of the instances of the different time step would be needed which is

in general a complex topic on its own. An overview of the network structure and the interaction of the different modules is given in Fig. 7.2.

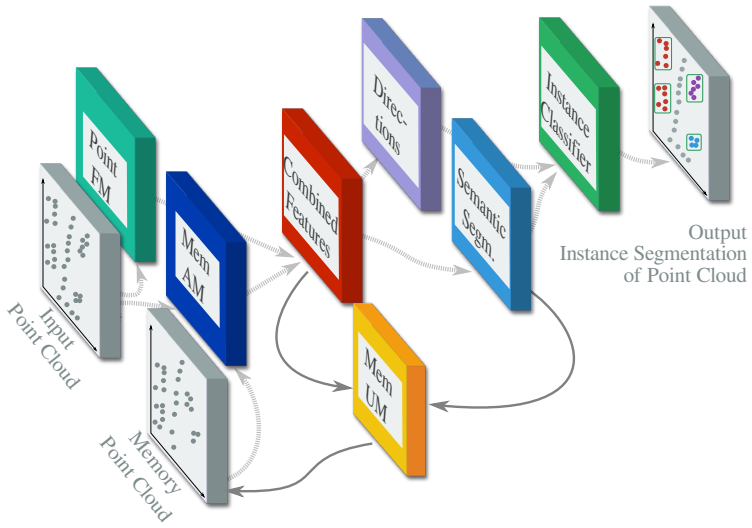


Figure 7.2: Structure of the recurrent instance segmentation network. The input point cloud is passed to the point feature generation module (teal box) and the memory abstraction module (blue box), where feature vectors for each point are generated and afterwards combined (red box). The direction module and the semantic segmentation module (violet and light blue box, respectively) support the final instance classifier (green box). The light grey dashed arrows symbolize the forward path whereas the dark grey arrows show the update step of the memory point cloud during which the memory update module (yellow box) is utilized.

INPUT POINT CLOUD The network expects as input a point cloud with N_{in} points $p_1, \dots, p_{N_{\text{in}}}$. As before, each point (radar target) p_i is defined by two spatial coordinates, the measured RCS value σ_i, the ego-motion compensated Doppler velocity $\hat{v}_{r,i}$ and the measurement time Δt_i relative to the earliest measurement in the complete input data. The spatial coordinates are given in both car coordinates $(x_i^{(cc)}, y_i^{(cc)})$ as well as in global coordinates $(x_i^{(gc)}, y_i^{(gc)})$. The global coordinates are later used for a lookup of neighboring points, whereas the car coordinates are used as input features. The input point cloud is passed into the two blocks *point feature generation module* and *memory abstraction module*.

MEMORY POINT CLOUD As the name suggests, the memory point cloud works as a memory block for the network. It consists of a list of N_m points $q_1, \dots, q_{N_m}$ of previous inputs. Each of these points is defined by the same six parameters as the input points p_i as well as one additional k_m-dimensional feature vector g_j. The

construction of this feature vector as well as the filling of the memory point cloud is described in the paragraph about the *memory update module*.

POINT FEATURE GENERATION MODULE For each input point p_i, a k_f-dimensional feature vector f_i is created by passing the input point cloud through a cascade of PointNet++ MSG modules and matching feature propagation modules, see also section 7.2.1.1. The car coordinates $(x_i^{(cc)}, y_i^{(cc)})$ as well as $\hat{v}_{r,i}$, t_i and σ_i are used as input features to the first MSG module. Just as described for the semantic segmentation network, the time dimension is normalized so that the earliest measurement occurs at $t = 0$. The dimension k_f of the created feature vectors f_i is a tunable parameter and depends on the size of the convolution layers in the feature propagation modules. This module is similar to the first part of the semantic segmentation network described in section 7.2.1.1 since for each point a high-dimensional feature is generated which takes neighboring points into account. In Fig. 7.2, the module is displayed as a teal box with label *Point FM*.

MEMORY ABSTRACTION MODULE To fuse the input point cloud with information from previous time steps, the *memory abstraction module* (*Mem AM*, blue box in Fig. 7.2) performs neighborhood searches for each point p_i from the input point cloud within the memory point cloud. That is, each input point p_i with global coordinates $(x_i^{(gc)}, y_i^{(gc)})$ is considered in a first step as a center of a circular search area with radius r. From all points q_j of the memory point cloud within this area, the first n_{sample} points are selected along with their k_m-dimensional feature vectors g_j and are then grouped together. If for one input point no neighboring points in the memory point cloud could be found, all n_{sample} feature vectors are set to zero. If less than n_{sample} different neighbors are found, the already detected points are repeated to guarantee a fixed size tensor in the subsequent computations. The neighborhood searches are performed in the global coordinate system since due to the ego-motion of the vehicle the car coordinates are in general time dependent and a comparison of target positions from different time steps is not possible in this reference frame. The collection of neighboring points is passed through a set of convolution layers followed by max-pooling so that a new k_{ma}-dimensional feature vector is created. Applying max-pooling after the convolution operation causes that only the neighbor with the highest activation with respect to the convolution kernel contributes to the output which in turn makes the operation invariant to permutations of the points. The whole process is similar to the MSG modules in PointNet++ with the exception that the neighborhood search is performed in the memory point cloud and the central points stem from the input data. The memory abstraction module is responsible for propagating information from past inputs to the current one so that not only information from the current point cloud is

used for the segmentation task but also the evolution of the input data is taken into account.

FEATURE VECTOR COMBINATION The memory abstraction module outputs a k_{ma} dimensional feature vector for each input point p_i and the point feature generation module creates k_f-dimensional feature vectors. To create one single feature vector per input point, the outputs are simply concatenated, resulting in one $k_f + k_{ma}$ dimensional feature vector (displayed as a red box in Fig. 7.2). This feature vector then encodes information about the current input point, its surrounding, and earlier measurements in the neighborhood.

SEMANTIC SEGMENTATION MODULE To obtain a vector of class probabilities for each point of the input point cloud, the tensor holding all the combined $k_f + k_{ma}$ dimensional feature vectors f_{in} is passed through a set of 1D convolution layers. These convolution layers can be interpreted as applying one shared fully connected layer to each of the individual points: $f_{\text{out}} = W \cdot f_{\text{in}}$. This approach is discussed in more detail in section 2.1.2.1 and illustrated in Fig. 2.4. The final convolution layer of this module creates a six-dimensional output vector for each input point, which contains the unscaled probabilities of the six classes.

DIRECTION MODULE The architecture of the direction module is very similar to the one of the semantic segmentation module. Its task is to generate displacement vectors $dr = (dx_i^{(cc)}, dy_i^{(cc)})$ for each input point p_i so that the shifted points $\tilde{p}_i = \left(x_i^{(cc)} + dx_i^{(cc)}, y_i^{(cc)} + dy_i^{(cc)}, \sigma_i, \hat{v}_{r,i}, t_i\right)$ lie at the center of the instance the respective point belongs to. If a point does not belong to a moving object, the displacements $(dx_i^{(cc)}, dy_i^{(cc)})$ should be zero. If for example p_i is one of the $|\mathcal{I}|$ points that belong to instance $\mathcal{I}$ and the average spatial position of all points within $\mathcal{I}$ is given by $(\bar{x}, \bar{y})$, then the ideal outputs of the direction module for this point are $dx_i^{(cc)} = \bar{x} - x_i^{(cc)}$ and $dy_i^{(cc)} = \bar{y} - y_i^{(cc)}$. The two numbers $dx_i^{(cc)}$ and $dy_i^{(cc)}$ are the final output of the direction module after a number of one-dimensional convolution operations is applied to the input features f_{in}, similar to the semantic segmentation module.

MEMORY UPDATE MODULE Integration of targets of the current point cloud into the internal memory point cloud is done in the *memory update module* (*Mem UM*, orange box in Fig. 7.2). The class probabilities predicted in the semantic segmentation module are concatenated with the feature vectors generated for the current input point cloud to form the input to a set of convolution layers. The output of these layers is one k_m-dimensional feature vector for each input point. These feature vectors are then stored in the memory point cloud along with the

global coordinates $(x_i^{(gc)}, y_i^{(gc)})$ of the point to which this feature vector belongs to. Since the memory point cloud cannot grow arbitrarily but is restricted to N_m points, the oldest N_{in} points are removed to make space for the new entries. It should be noted that even though the size of the memory point cloud is restricted and old points are removed, the update scheme allows for passing information over arbitrarily long times. This is possible since the combined feature vectors of the current input point cloud and the previous memory point cloud contain information from the last state of the memory and hence information from the neighborhoods of the previous memory points. Even if points are removed from the memory point cloud, their information content can still be passed on if they were processed once by the memory abstraction module.

INSTANCE CLASSIFIER The final output of the network is a list of instances along with a predicted class label for each instance. This output is generated from the combined feature vectors, the semantic segmentation output and the displacement vectors created in the direction module. In a first step, instance candidates are created from the displacement vectors. This is done by shifting all points by the predicted values of $dx_i^{(cc)}$ and $dy_i^{(cc)}$, i.e. $p_i \rightarrow \tilde{p}_i = \left(x_i^{(cc)} + dx_i^{(cc)}, y_i^{(cc)} + dy_i^{(cc)}, \sigma_i, \hat{v}_{r,i}, t_i\right)$. The better the direction module performs, the closer all points of the same instance lie together in the shifted point cloud. Based on the predicted labels of the semantic segmentation module, m_{dyn} points of the five dynamic object classes are chosen from the input point cloud. For each of these points, m_r points within a radius r_{dyn} around the seed point are selected. The created neighborhoods are then merged together to a maximum of m_{inst} instances with $n_{p,\text{inst}}$ points per instance. Two neighborhoods are merged if they have a non-empty intersection, i.e. if at least one point belongs to both neighborhoods. To guarantee fixed size tensors, resampling of the number of points per instance $n_{p,\text{inst}}$ as well as the number of instances m_{inst} is done. That is, if less than m_{inst} instances are created (or less than $n_{p,\text{inst}}$ points make up one instance), the first detected instance (or point within an instance) is repeated the appropriate number of times. The number of unique instances is stored so that later on only the correct number of instances is used for both training and evaluation. The parameters $n_{p,\text{inst}}$ and m_{inst} are chosen large enough so that the typical number of points per instance and number of instances in a scene can be predicted. The instance creation process is depicted for one example scene in Fig. 7.3. The semantic segmentation module initially predicted in this case the two labels `Large Vehicle` and `Passenger Car` for one single `Large Vehicle` object. After shifting the targets by the predicted direction vectors, performing neighborhood searches and merging the resulting groups, a single instance is created. The instance classifier then predicts one single label for all targets of this instance – in this case the correct label `Large Vehicle` is chosen.

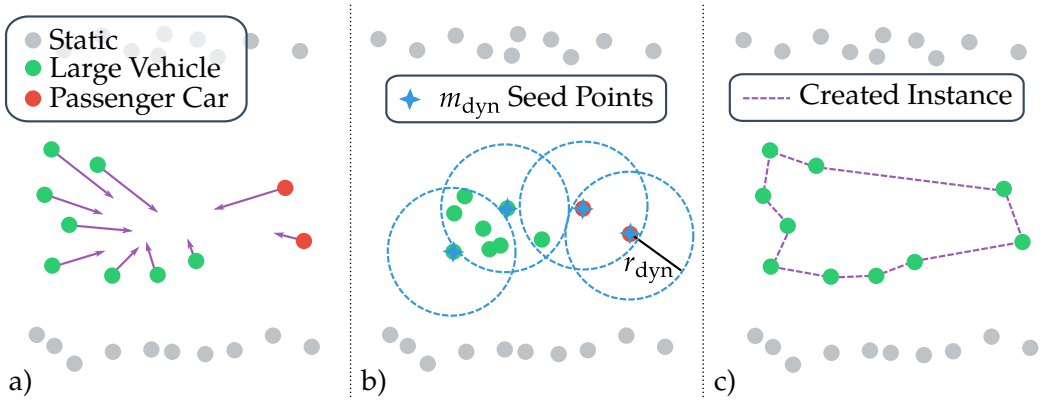


Figure 7.3: Instance creation for *one* example object. a) Predictions from semantic segmentation module and predicted direction vectors. b) Shifted targets, selection of seed points, neighborhood search. c) Merged neighborhoods form one instance, which is classified by the instance classifier.

Applying the instance generation algorithm on the shifted point cloud allows to use a greater search radius r_{dyn} since points of the same instance are closer together and points not belonging to the instance are on average further away so that it becomes less likely that points are added to an instance which are in fact member of a different instance or belong the static environment. After creation of the instances, all points belonging to this instance are passed together through a PointNet++ classification network, similar to the one described in section 6.3.1. In the *shape feature* generation part of this network, the five basic features of each point $p_i = \left(x_i^{(cc)}, y_i^{(cc)}, \sigma_i, \hat{v}_{r,i}, t_i\right)$ are used as input to a series of MSG modules. The combined feature vectors generated before are used as input for a set of convolution layers. This allows incorporating the encoded time information into the classification step. The feature vectors generated in these two paths are concatenated and finally passed through fully connected layers to obtain the final class probability vector.

7.2.2.2 *Training Parameters*

The size of the input point cloud is set to $N_{\text{in}} = 1000$ and points are collected from a fixed time range of $T_{\text{rec.}} = 150\,\text{ms}$. The memory point cloud can hold up to $N_m = 2 \times N_{\text{in}} = 2000$ targets. The other free hyper-parameters of the network are listed in the appendix in Section A.5.

Training of the network is done in four consecutive steps. Firstly, the semantic segmentation branch of the network is trained so that both the *direction module* and the *instance classifier* are neglected. Just as in the previous setting where only a

semantic segmentation network was considered, the cross-entropy loss function on a per-target basis is used:

$$L_{\text{sem}} = \frac{1}{N_{\text{in}}} \sum_{i=1}^{N_{\text{in}}} \omega_i \sum_{k=1}^{6} y_{ik} \log \frac{1}{\hat{y}_{ik}}. \tag{7.2}$$

Just as before, ω_i is the weight factor of point i and the two symbols y_{ik} and $\hat{y}_{ik}$ describe the ground truth probabilities and predicted class probabilities, respectively. In preceding experiments, the weights ω_i were fixed to $\omega^{\text{static}} = 0.5$ and $\omega^{\text{dynamic}} = 8$ for points of the static and dynamic class, respectively. Training of the semantic segmentation branch is done for 10 epochs.

In the second training step, the direction module is trained for another six epochs while all other trainable weights are kept constant. That is, only the parameters of the direction module are altered and all weights of the preceding modules remain unchanged. Therefore, the features extracted in the *point feature generation module* and in the *memory abstraction module* remain the same and only the 1D convolution layers in the direction module itself can be tuned. The following loss function is used during this training stage:

$$L_{\text{dir}} = \frac{1}{\hat{N}_{\mathcal{I}}} \sum_{j=1}^{\hat{N}_{\mathcal{I}}} \sum_{l=1}^{|\mathcal{I}_j|} \frac{||d_{jl} - \hat{d}_{jl}||^2}{|\hat{\mathcal{I}}_j|}. \tag{7.3}$$

The value $\hat{N}_{\mathcal{I}}$ stands for the number of ground truth instances $\mathcal{I}_j$, which in turn are constructed from a set of $|\mathcal{I}_j|$ targets. The 2D vector d_{jl} describes the ground truth distance from target l of instance j to the instance's center and similarly, $\hat{d}_{jl}$ symbolizes the prediction. The inner sum runs over all targets within one ground truth instance $\mathcal{I}_j$ so that each target enters the loss function only once for the ground truth instance it belongs to. Distances to other instances are not considered. For example, if point p_i is the lth point of instance j, then $d_{jl} = (dx_i^{(cc)}, dy_i^{(cc)})$ and the point p_i does not enter the loss function in any other sum. Similar approaches for predicting vectors to instance centers are used in [6], [256].

In the third phase, the sum of the two previously used functions

$$L_{\text{comb.}} = L_{\text{sem}} + L_{\text{dir}} \tag{7.4}$$

is used as a combined loss function for another 10 epochs. In contrast to the previous stage, now all parameters can be optimized, i.e. also the weights of the preceding modules. Avoiding the second training step and training directly with the combined loss function yielded poor results since the initially random weights of the direction module did not converge to useful values.

Finally, the instance classifier is trained while the weights of all other layers remain fixed. Similar to the semantic segmentation training phase, cross-entropy is used as a loss function. However, now the loss is calculated per instance and not per target:

$$L_{\text{inst}} = \frac{1}{N_{\mathcal{I}}} \sum_{i=1}^{N_{\mathcal{I}}} w_i \sum_{k=1}^{6} y_{ik} \log \frac{1}{\hat{y}_{ik}}. \tag{7.5}$$

The factor $N_{\mathcal{I}}$ stands for the predicted number of unique instances, while w_i are weights for instances created for truly dynamic objects and for instances which contain only static points. The predicted and ground truth class probabilities are denoted by the two symbols $\hat{y}_{ik}$ and y_{ik}, respectively. Since a predicted instance can contain targets from different ground truth classes, a scheme is needed to ascribe a single ground truth label to the instance. If all targets of the predicted instance are in fact members of the `Static` class, then the ground truth label is defined as `Static`. If at least one of the targets is a member of one of the other five classes, then the maximum occurring label of these five classes is chosen, i.e. the *mode* of the ground truth class label distribution of the instance's targets is used as a ground truth label for the whole instance. For instances with a `Static` label, the weight factor $w^{\text{static}} = 0.5$ is used and for instances with a ground truth label from a dynamic object class, the weight $w^{\text{dynamic}} = 1.5$ is selected. Training of the instance classifier is done for 15 epochs with a batch size of 20.

All training steps are performed with the same sequence length of six consecutive time steps. Adam [123] is used as an optimizer with learning rate $\alpha_1 = 0.001$ for all but the third training phase where a reduced learning rate of $\alpha_2 = 0.0002$ is chosen so that the already learned features are only refined.

7.3 RESULTS

Outcomes of the two different neural network architectures are presented in this chapter. Since similar experiments are performed for both approaches, results are shown side-by-side in the subsequent sections. Except for the section about the classification performance on *objects*, all scores given here are calculated per *target*, allowing for a direct comparison with the classifiers presented in the last chapter.

7.3.1 *Per-Target Evaluation*

During training of the networks, augmentation of the input data was used to help the network generalize to unseen data. The impact of using augmentation is ana-

lyzed for the semantic segmentation network by training once with augmentation and once without it.

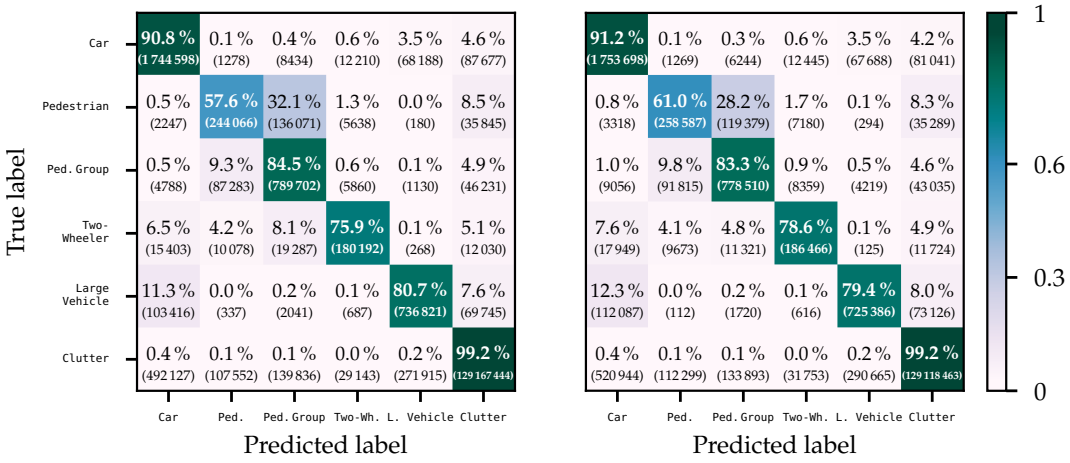


Figure 7.4: Confusion matrices from the semantic segmentation network. Left: Training without augmentation. Right: Training with augmentation.

The two resulting confusion matrices are displayed in Fig. 7.4, showing that using augmentation increases the performance slightly ($F_1 = 0.776$ with augmentation versus $F_1 = 0.775$ without augmentation). Due to this finding, augmentation is also used during training of the recurrent instance segmentation network. The confusion matrix on the right hand side of Fig. 7.4 shows the results of the best performing semantic segmentation network. The achieved F_1 score is identical to the one of the PointNet++ & LSTM approach presented in the previous chapter. However, the underlying precision and recall values are quite different. While for the PointNet++ & LSTM classifier the precision value is slightly higher than the recall value (see Table 6.5), this relationship is inverted for the semantic segmentation network: the recall value is with 0.821 considerably higher than the precision, which ranges at 0.739. The precision value is mostly influenced by the number of targets that were assigned to one of the dynamic classes even though they truly belong to the `Static` class (last row of the confusion matrix). The influence of the `Static` class is large simply because it contains so many locations.

This allows for the conclusion that the semantic segmentation network misses way less truly moving targets in comparison to the classification approach at the cost of creating many more false positive targets that are truly members of the `Static` class. Depending on the application, either of the two behaviors is favorable.

Training of the recurrent instance segmentation network was done with six consecutive time steps. Due to the recurrent nature of the network, it is however possible to perform the evaluation with arbitrarily many time steps. As it will be shown

in the next sections, the more time steps are used the better the score gets. The confusion matrices shown in Fig. 7.5 are created with $n_t = 14$ consecutive time steps.

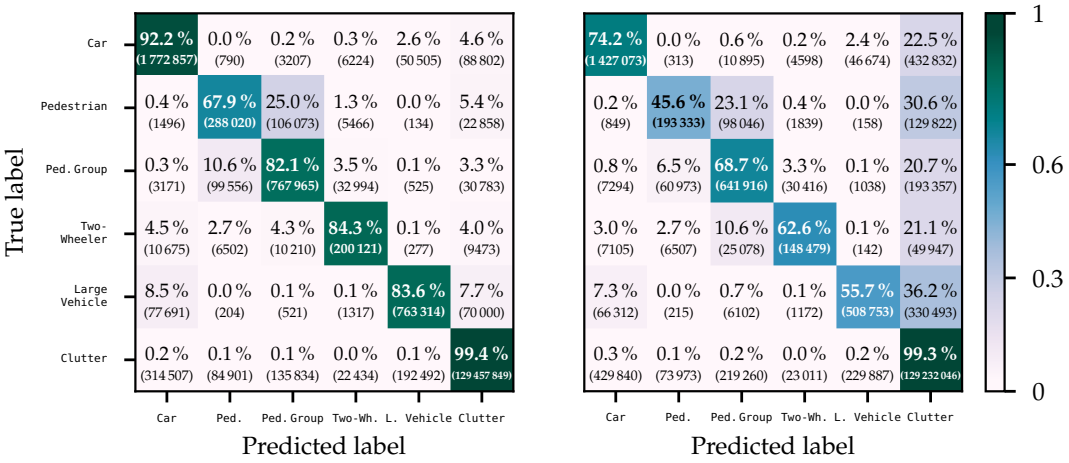


Figure 7.5: Confusion matrices of the recurrent instance segmentation network. Left: All five target-features are used and evaluation is done with 14 time steps. Right: Resulting confusion matrix if only the two position features are used as input.

In comparison to the output of the semantic segmentation network, the recurrent instance classifier achieves not only a higher recall value, but also produces less false positives so that the precision increases as well. The true positive rates are greater than the ones from the semantic segmentation network for almost all classes with the only exception being the `Pedestrian Group` class where a minor decrease can be found. Noteworthy confusions appear once again between the `Pedestrian` and `Pedestrian Group` classes as well as between the `Passenger Car` and `Large Vehicle` categories. The origins of these errors are most probably the same as already discussed in Section 6.2.2.4. As described in Section 7.2.2.1, the recurrent network outputs not only predictions for each instance but also has a semantic segmentation branch. These predictions can now also be compared to the pure semantic segmentation network with no recurrent structure. This comparison shows a slight superiority of the recurrent network over the simpler feed-forward network but at the same time, the scores of the recurrent network's semantic segmentation branch are considerably lower than the ones from the instance segmentation.

To provide an insight on how the directions module performs, Fig. 7.6 displays three ground truth instances in the left panel and the predicted displacement vectors in the right panel. All targets belonging to the `Passenger Car` are noticeably shifted inwards so that the network successfully learned that these targets belong together. The instance creation module then has no difficulty in grouping all these

points together so that the instance classifier can finally predict a single class label for all of the targets.

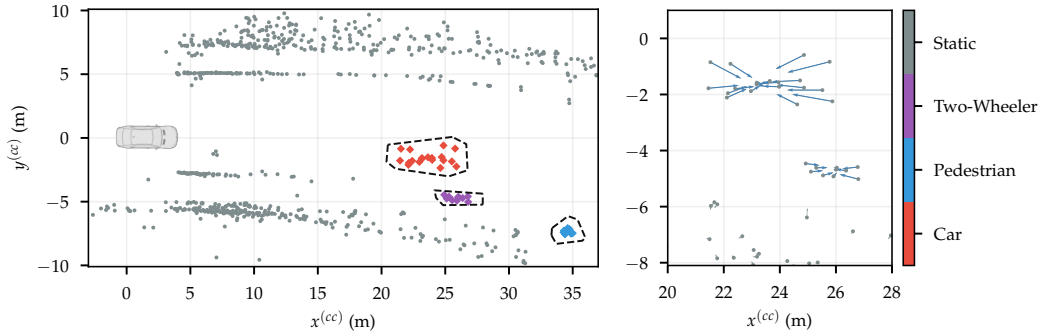


Figure 7.6: Left: Ground truth instances. Right: Predicted displacement vectors for each target.

Similarly, the predicted displacements for the bicycle's targets also point towards the center of the object. A problem that was noticed for multiple objects is that the network manages to predict the correct direction of the vectors but their extent remains too small so that even though a small shift is done in the right direction, multiple instances are created for one object. This is analyzed in more detail in Section 7.3.4, where the number of created and missed objects is evaluated.

7.3.2 *Feature Importance*

To evaluate how the different features influence the classification performance, they are removed one after the other from the input data. Thereby, an F_1 score is obtained for different input combinations so that it can be assessed how much each feature contributes to the total score. This approach is similar to the *permutation* importance measure where the values of a single feature are interchanged among all inputs and the decrease in performance is monitored, see also the last part in Section 2.1.1.2. Removing the feature entirely from the input data and re-training the classifier on this smaller number of features has very similar expressiveness.

In Table 7.1, the F_1 scores as well as precision and recall values are listed for the semantic segmentation network and the instance classification approach for different input feature combinations. The two spatial positions $x^{(cc)}$ and $y^{(cc)}$ of the locations are abbreviated as x and y in the table, σ is the RCS value, $\hat{v}_r$ describes the ego-motion compensated Doppler velocity and Δt is the time difference between a target's measurement time and the earliest timestamp in the point cloud.

	Semantic Segmentation			Instance Segmentation		
Features	F_1	Precision	Recall	F_1	Precision	Recall
$x, y, \sigma, \hat{v}_r, \Delta t$	0.776	0.739	0.821	0.811	0.777	0.849
$x, y, \hat{v}_r, \Delta t$	0.769	0.732	0.814	0.799	0.770	0.831
$x, y, \sigma, \hat{v}_r$	0.760	0.721	0.808	0.799	0.768	0.833
$x, y, \sigma, \Delta t$	0.736	0.731	0.740	0.727	0.731	0.723
x, y	0.591	0.602	0.583	0.694	0.717	0.677

Table 7.1: Scores obtained for different input combinations, displaying the importance of each feature.

Since the network needs the spatial positions of each location to generate feature vectors for a target and its neighborhood, these two input feature cannot be removed without altering the whole network architecture. The first observation that can be made from this table is that in all cases the recurrent instance classification network outperforms the semantic segmentation approach.

Secondly, removing either the RCS value σ or the measurement time Δt from the input data (second and third row) results in similar F_1 scores so that these two features are of almost the same importance. This statement holds true for both neural networks. If, however, the Doppler information is removed, a more considerable drop in performance can be observed (fourth row). This indicates that among the three features σ, $\hat{v}_r$ and Δt, the Doppler velocity is the most important feature for the classification.

Using only the two spatial positions as input shows an even greater decrease in performance. The resulting confusion matrix is shown on the right side of Fig. 7.5. The extent of the decrease differs highly between the two architectures: for the semantic segmentation network, the difference in F_1 score between the case where both σ and Δt are still available to the case where only x and y are used as input is much larger than for the instance classification network. It is therefore interesting to note that removing either σ or Δt has a very small impact on the score if $\hat{v}_r$ is still available (rows two and three) but if also the Doppler information is removed, the semantic segmentation network makes great use of these two features (cf. rows four and five of the table). For the instance classification network, the importance of σ and Δt remains more constant, i.e. the performance drop between rows four and five is of similar magnitude as between row one and two as well as between row one and three where σ and Δt are removed individually. This also shows that

the instance segmentation approach makes far more use of the spatial positions of the targets than the simpler semantic segmentation network. This effect can be possibly explained by the fact that with the *direction module* the network is forced to make use of the spatial relationship between the targets in order to create instances out of them.

7.3.3 *Dependence on Sequence Length*

The performances of the two approaches depend on the length of input point cloud T (for the semantic segmentation network) and the number of input time steps n_t (in the case of the recurrent network).

For the simpler semantic segmentation network, it can be expected that the performance increases the greater T gets since then the properties of the dynamic objects become more pronounced. However, this only holds true up to the point where the number of measured targets exceeds the number of input points N_{in} of the network. If this happens, then additional targets cannot be processed anymore by the network or alternatively, these points have to replace some of the earlier measured points, e.g. by selecting the N_{in} points with the highest $\hat{v}_r$ values. This replacement would then cause that the network has less information to infer from the static environment where dynamic objects could be and how they relate to the overall scene. Therefore, the structure of the semantic segmentation network sets an upper limit to the number of input points that can be processed and hence the sequence length T cannot be extended arbitrarily. The network was designed for a sequence length of $T = 0.5\,\text{s}$ with $N_{\text{in}} = 3072$. In this section it is analyzed how the performance changes, if data from shorter time windows is used as input.

The recurrent structure of the instance classification network allows for arbitrarily long input sequences with n_t steps. Each of the input point clouds has the same time length $T_{\text{rec.}}$ so that in contrast to the semantic segmentation approach, the network structure itself has not to be modified if sequences with larger n_t are used as input.

In Fig. 7.7, the scores of the semantic segmentation network as well as the scores of the recurrent instance segmentation approach are shown. For the recurrent algorithm, the two branches of the semantic segmentation and the instance segmentation are displayed. It should be noted that the network was trained with a fixed number of time steps $n_t = 6$ but evaluated with varying n_t. The semantic segmentation network was newly trained and evaluated for each value of T.

From the plot of the scores achieved by the semantic segmentation network it becomes obvious that the network was designed for $T = 0.5\,\text{s}$. With shorter scenes

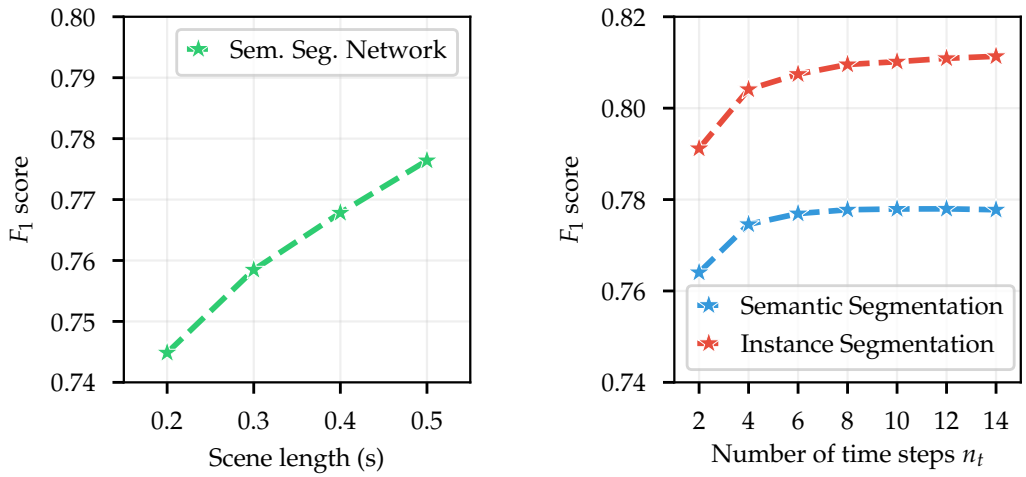


Figure 7.7: Dependence of the F_1 score on the temporal length of the input data. Left: Scores from the semantic segmentation network. Right: F_1 scores of the instance classification network, showing both the sem. seg. branch of the network and the instance classifier.

and subsequently less input targets, the performance drops considerably. Evaluating with larger values of T is not directly possible since then the network architecture would have to be changed. Further experiments proofed that performance saturates around $F_1 = 0.78$ even if the network structure is altered to process more input targets.

For the recurrent network, saturation is visible for both output branches. Since the memory point cloud of the network can hold only twice the number of input points, one could expect that after $n_t = 3$ the score cannot increase any further. Nevertheless, due to the design of the memory update module, features of earlier inputs can be propagated to much later time steps. Even though a clear saturation is visible, the scores of the instance segmentation output continuously increase. Because of this finding, the confusion matrices shown in Fig. 7.5 and the scores listed in Table 7.1 are extracted from the evaluation with $n_t = 14$ time steps. The recurrent network is hence able to make use of the temporal evolution of the input scenes and to propagate the extracted information over multiple time steps. For the recurrent network, the abscissa in Fig. 7.7 shows the number of time steps n_t, whereas for the semantic segmentation network the abscissa lists the sequence length T. This choice is made because for a recurrent network it is more natural to consider the number of sequence elements n_t rather than the total time. The total duration of data shown to the recurrent network can be recovered by simply multiplying n_t with the duration $T_{\text{rec.}} = 150\,\text{ms}$ of each single step.

Since training the network with larger n_t requires more memory on the GPU, this step was restricted to $n_t = 6$. Experiments with $n_t = 4$ during training showed an

overall decrease in performance so that possibly with $n_t > 6$ even higher scores could be obtained.

7.3.4 *Evaluation per Object Instance*

Up to now, evaluation was done on a per-target basis. Since the recurrent network predicts not only a class label per target but creates instances as well, it is possible to evaluate the classifier's performance on a per-object basis.

To this end, the *Intersection Over Union (IoU)* between the predicted instances and the ground truth instances is calculated for each output of the network. The predicted object with the largest IoU for one ground truth instance is then associated with it. If no prediction overlaps with a ground truth object, this object is counted as an *overlooked object*. If multiple predictions have an IoU greater than zero with a ground truth object, only the one with the largest IoU is associated and the other ones are considered as *additional predictions*. If a prediction has no overlap with any ground truth object, it is counted as a *false positive*.

For the overlooked objects it is measured how long they remain unobserved. Since the instance segmentation network processes point clouds of length $T = 0.15\,\mathrm{s}$, objects are considered as overlooked for time ranges which are multiples of T. The processing time of the network is not considered in this calculation. To allow for a direct comparison, the same evaluation is also performed for the approaches described in Chapter 6 where a clustering algorithm is used for the instance creation.

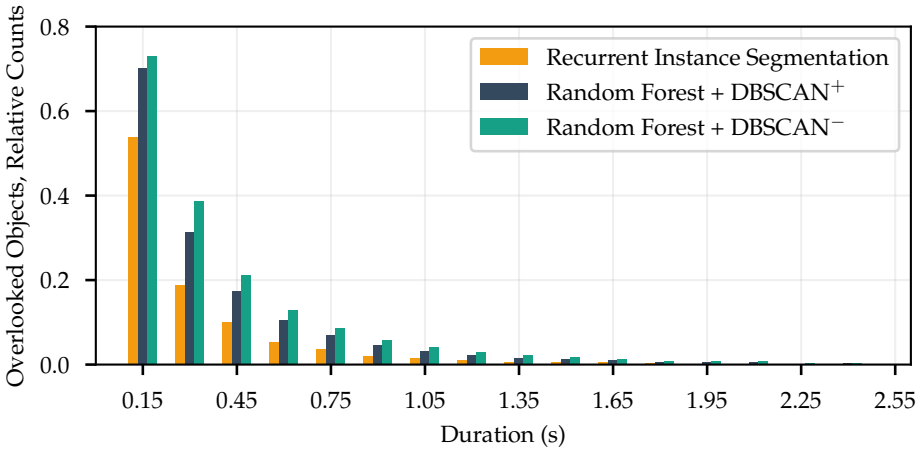


Figure 7.8: Relative number of overlooked objects for three different approaches: recurrent instance segmentation network, DBSCAN$^+$ and DBSCAN$^-$.

In Fig. 7.8, the relative number of these missed objects is shown. The chart is normalized so that the sum of the bar heights corresponding to the instance segmentation network equals one. The same normalization weights were also used for the two random forest approaches so that the bar heights can be directly compared. This choice implies that the bar heights of these two algorithms do not sum to one. The general trend of the three distributions is identical: the most common case is that if an object was missed, it is only overlooked for one time step. For the instance segmentation network, this event accounts for over 50 % of all cases. Of all missed objects, less than 20 % are overlooked for two consecutive time steps and this number drops to about 10 % for three time steps.

The values for both DBSCAN$^+$ and DBSCAN$^-$ are higher than for the network. For example, about twice as many objects are overlooked by DBSCAN$^-$ than by the network in the 0.3 s bin. This finding is well aligned with the results discussed in Section 7.3.1, where it was shown that on a per-target basis the neural network predicts fewer points as static that truly belong to a dynamic object. As one could expect from the evaluation performed in the last chapters, with DBSCAN$^+$ less objects are overlooked than with DBSCAN$^-$.

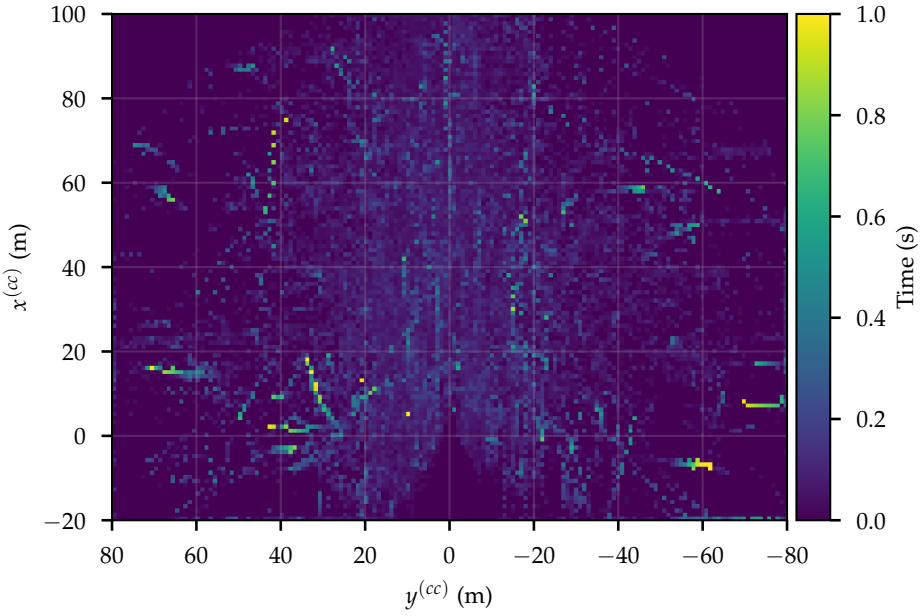


Figure 7.9: Positions of objects overlooked by the instance segmentation network. The average time an object was not identified in a certain cell is color-coded.

To assess how the objects overlooked by the network are spatially distributed, Fig. 7.9 displays the object's centers in a Cartesian grid map. The average value of the times an object was overlooked in one of the 10 cm × 10 cm cells is color

Method	Passenger Car	Pedestrian	Pedestrian Group	Two-Wheeler	Large Vehicle
Ground Truth	13.4	7.3	11.5	18.7	31.4
Instance Segm. Network	10.5	7.4	8.8	19.3	14.3
DBSCAN^{+}	5.1	3.1	5.1	7.2	10.6
DBSCAN^{-}	5.4	3.4	5.7	7.5	11.9

Table 7.2: Average number of targets per ground truth and per predicted object in a time bin of size 0.15 s.

coded in this plot. The visualization displays that in general the network's mistakes are distributed homogeneously. This means, for example, that an object far away is equally likely missed as an object nearby the ego-vehicle. This result is rather surprising given that considerably less targets are measured on far distant objects, see also Fig. 4.12.

To quantify the differences between the network's object predictions and the ones from DBSCAN$^{\pm}$ with a random forest as classifier, it is instructive to look at the total numbers of predicted instances. It should be kept in mind that the numbers shown here strongly depend on the chosen time window size, since of course the number of predicted objects increases if the network is queried more often with shorter time windows. Accumulating all ground truth instances in the 0.15 s time windows shows that in total about 402 000 individual objects exist. The network predicted in total about 525 000 instances, DBSCAN^{+} over 985 000 objects and DBSCAN^{-} created around 911 000 instances. That is, the clustering methods created more than twice as many objects as needed and the network predicted only about 30 % more objects than the number of ground truth objects actually is. At the same time, the number of overlooked objects is with about 39 000 objects much smaller than for DBSCAN^{+} (73 000 overlooked objects) and DBSCAN^{-} (87 000 missed objects).

It is also instructive to compare the sizes of the created objects in terms of the number of targets they contain. These numbers are listed in Table 7.2 for each class along with the average sizes of the ground truth objects. Both DBSCAN approaches create considerably smaller clusters than the network's instance creation module. This finding highlights again that a large part of the classification problem actually lies at the clustering stage since no classifier can perform well if the input data is too noisy and contains too little class specific information. However, the network still fails to assign targets of a `Large Vehicle` instance to one single object since the average number of targets per predicted object is less than half of the number in ground truth objects. This in turn explains the large confusion between `Passenger`

`Car` and `Large Vehicle` targets seen in the matrices in Fig. 7.5: If not all targets of a `Large Vehicle` instance are grouped together properly, the resulting object looks rather like a `Passenger Car` instance than a `Large Vehicle`.

7.3.5 *Ensemble Learning*

The recurrent instance segmentation network can be interpreted also in one further way: Everything up to the instance classification can be considered as a clustering algorithm and the final part of the network does the actual classification of the predicted instances. With this interpretation, one obvious experiment is to exchange the instance classification head of the network with a different classifier. For example, a random forest can be trained on the instances predicted by the network and later on used for evaluation as well. This degrades the whole network to a substitute for DBSCAN$^{\pm}$ but allows for a comparison with these clustering algorithms. Since a random forest cannot work with the bare points of an instance as input data, features are calculated in the same way as described in Section 6.2.1.

Another possibility is to add these manually calculated features to the instance classification network in order to check if these additional features are helpful for the classification task or if the network performs well enough with the automatically generated features.

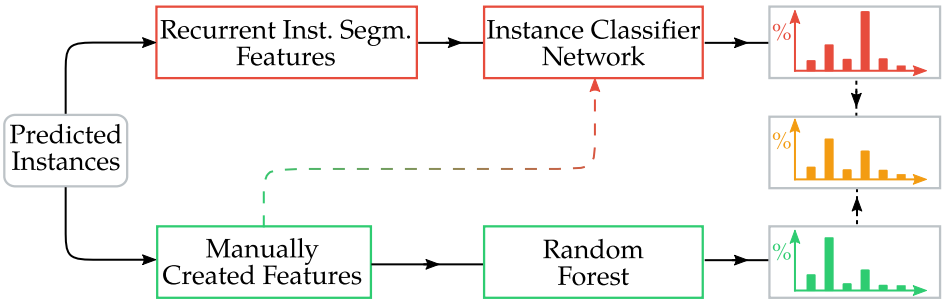


Figure 7.10: Flowchart showing the different evaluation schemes for the ensemble classifier.

Finally, the predictions made by the random forest and the instance classification network can be averaged to obtain an ensemble result. The possible evaluation ways are depicted in Fig. 7.10. The red boxes indicate the normal path of the recurrent network that was evaluated in the previous sections. The green boxes show the evaluation strategy if the network is used only for instance prediction and a random forest is used for classification of these instances. The dashed line displays the branch in which the manually created features are added to the instance classification network. Averaging the probabilities predicted by the random forest and the neural network is indicated in the last column of the figure.

If only a random forest is used for classification with features identical to the ones used in the last chapters, then the combination of recurrent network for "clustering" and random forest for classification achieves an F_1 score of 0.777. The best performing random forest classifier using DBSCAN$^+$ for clustering achieves a score of 0.761, cf. Table 6.5. The confusion matrix of this newly trained random forest classifier is shown on the left side of Fig. 7.11. This experiment demonstrates that the clusters created by the network are more beneficial to the subsequent classifier than the ones from the DBSCAN algorithm. At the same time, the achieved score is considerably lower than the one obtained by the full network, which is 0.811. The difference between these two cases is not only the different classifier, but also the different input features. The instance classifier network uses the feature vectors generated by the recurrent neural network which can possibly contain information about the neighborhood of the instance under consideration and the temporal evolution of the region at which the instance is located. This information is not available in the manually created feature vectors the random forest utilizes.

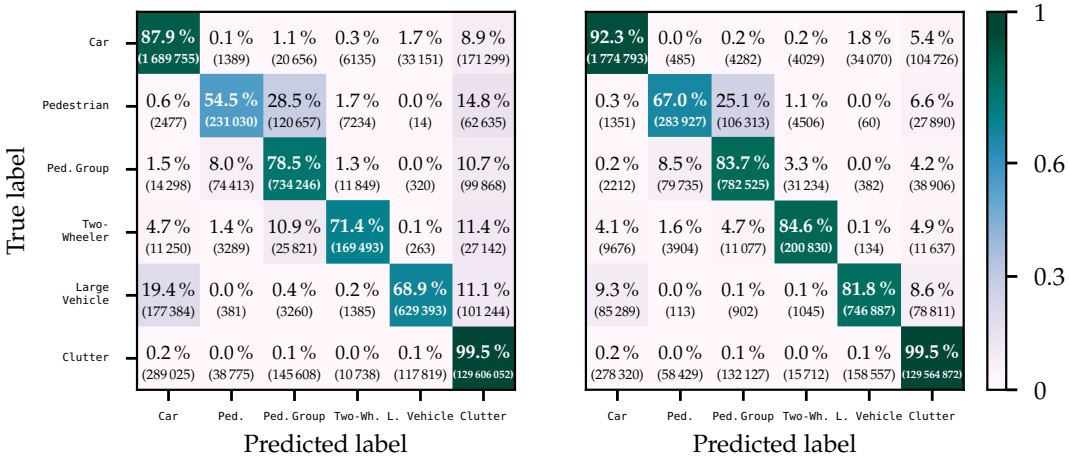


Figure 7.11: Confusion matrices of the ensemble learning. Left: Random forest classifier trained on the instances proposed by the network. Right: Combination of random forest and instance classification network.

Adding the manually created feature vectors to the instance classification network does not yield any performance increase. In fact, the F_1 score remains with 0.811 the same. This finding is identical to the one made for the PointNet++ classification network in Section 6.3.1.3.

In the final experiment, the scores of the random forest classifier and the instance classification network are combined by computing the average value of the two

probability vectors $p^{(\mathrm{RF})}$ and $p^{(\mathrm{NN})}$. The instance is then assigned to the class c for which the corresponding averaged probability is the highest:

$$c = \arg\max_i \frac{1}{2}\left(p_i^{(\mathrm{RF})} + p_i^{(\mathrm{NN})}\right). \tag{7.6}$$

With this method, an F_1 score of 0.825 is achieved. The confusion matrix of this ensemble classifier is displayed on the right side of Fig. 7.11. The main reason for the higher score is an increase in the precision value, which in turn is caused by a reduction of false positive dynamic objects (last row of the confusion matrix). The random forest tends to classify a truly dynamic object rather as `Clutter` then creating a false positive object. This contrasts the neural network, which tends to predict more false positives but therefore misses less objects. Combining these two methods now leads to an ensemble classifier that overlooks more objects than the single network but at the same time reduces the false positives. As the true positive rates remain on a similar level, it depends on the final system which of the two classifiers is more desirable, i.e. if it is better for the system to deal with false positives than to overlook targets of a dynamic object or vice versa.

7.4 HUMAN PERFORMANCE

Similar to section 6.4, where the performance of human annotators on the classification task was evaluated, the performance of the same three labelers is now evaluated on the instance segmentation task.

7.4.1 *Experimental Setup*

The semantic instance segmentation task was presented to the three labelers in an analogous way to the classification task. Short snippets of four different time lengths $T \in \{0.2\,\mathrm{s}, 0.6\,\mathrm{s}, 1\,\mathrm{s}, 5\,\mathrm{s}\}$ were cut out of all annotated sequences so that 15 snippets per length T were available for testing. All three labelers received the same snippets that contained only the measured radar data and in contrast to the classification experiment, no ground truth clusters were present. The labelers were then asked to create clusters for each object as well as assigning a class label to each of the created clusters. This exercise was identical to the actual labeling process described in section 4.2 with the only difference that no camera data was accessible. That is, the three test candidates had only the measured radar targets as information source to identify where dynamic objects are located and what class these objects belong to. Just as in the classification experiment, the labelers

Time bin	Basis	Passenger Car	Pedestrian	Pedestrian Group	Two-Wheeler	Large Vehicle	Static
0.2 s	targets	575	281	116	165	286	32 040
	objects	40	41	11	10	12	–
0.6 s	targets	3562	460	455	350	894	88 768
	objects	173	67	33	30	32	–
1 s	targets	4426	217	830	381	1570	160 391
	objects	213	50	52	44	40	–
5 s	targets	21 673	1180	4115	2932	5493	775 917
	objects	1249	155	324	229	138	–

Table 7.3: Number of targets and objects used in the experiment for each of the four time bins.

could jump forwards and backwards in time through the scenes and make use of the Doppler velocity information. The time the labelers needed to annotate each snippet was denoted for later evaluation. The number of targets and objects per class present in the snippets is summarized for each length T in Table 7.3.

7.4.2 *Results*

The same evaluation as done for the *classification* experiment is repeated for this instance segmentation task. In Fig. 7.12, both the annotation duration and the achieved F_1 score are plotted against the sequence length. For the three sequence lengths $T \in \{0.2\,\text{s}, 0.6\,\text{s}, 1\,\text{s}\}$, the trend of the curves describing the time the labelers needed to "classifiy" a sequence is directly comparable to the ones shown in Fig. 6.17 for the classification task. The labeling time needed for sequences of length 1 s is on average shorter than for 0.6 s long sequences (at least for labelers L_1 and L_3) which is possibly for the same reasons as discussed in Section 6.4: The task becomes easier when objects are visible for a longer period of time but at the same time the number of present objects increases with increasing sequence length so that a local minimum can be predicted. However, the longer the extracted snippets get the more time is needed in general so that after this local minimum an increase in labeling time can expected. It should be noted that the time needed to annotate the 5 s long sequences is shorter than five times the average time needed for annotation of the 1 s snippets so that having more information about the temporal evolution of the objects is still beneficial for the annotation speed. Just as in the classification experiment, test candidate L_1 is the fastest for almost all sequence lengths – except for the longest snippets where L_2 is the fastest. This is in so far remarkable as L_2 is

by far the slowest for all other sequence lengths. This speed up cannot be explained with a lack of motivation or less focused work since the performance of L_2 on the longest snippets is better than the one of L_3 who needed the most time to finish the job.

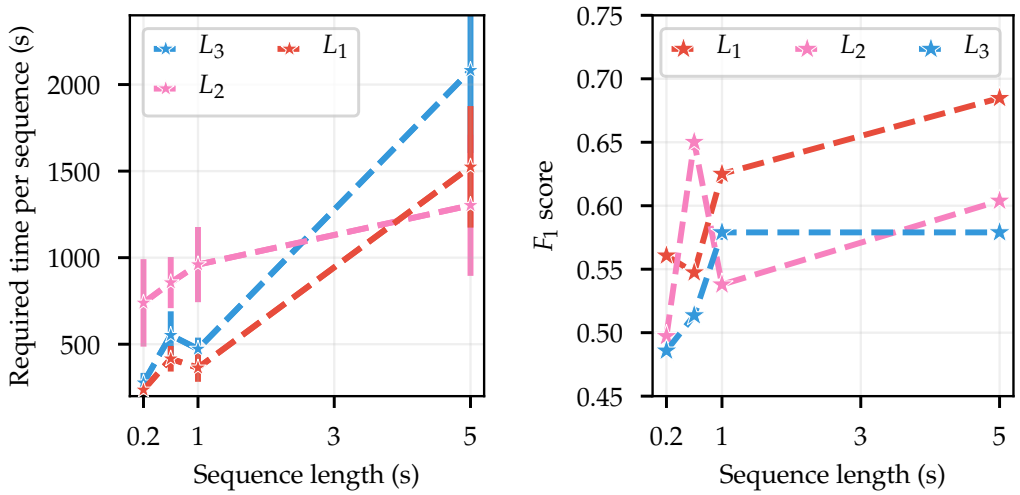


Figure 7.12: Left: Average duration each of the labeleres needed for instance segmentation of one sequence. Right: F_1 scores as a function of the length of the sequences.

In general, the scores depicted in the right part of Fig. 7.12 show that the longer the sequence lengths are the better the score gets on average. However, there are also considerable deviations from this trend since for example L_2 performs by far the best on the 0.6 s long sequences where the other two test candidates achieved much lower scores. One possible explanation is that the snippets chosen for this time length are easier to comprehend for this test candidate for example because he annotated similar sequences during his normal working task before. Another explanation could be that the number of targets and objects in the short sequences is rather small so that rather minute differences in the labeling result cause large deviation in the score. The highest overall score – achieved by L_1 for the longest sequences – is with about 0.69 considerably below the scores that the algorithms presented in this chapter accomplish (between 0.75 and 0.8). So in contrast to the classification task where the human test candidates outperformed the algorithm, the machine learning algorithm wins in the instance segmentation task. This leads to the conclusion that humans struggle more with the detection of dynamic objects in a radar point cloud than the presented algorithms but once a clustering is presented, the human test candidates outperform the algorithms since they make better use of the information contained in the temporal evolution of the objects.

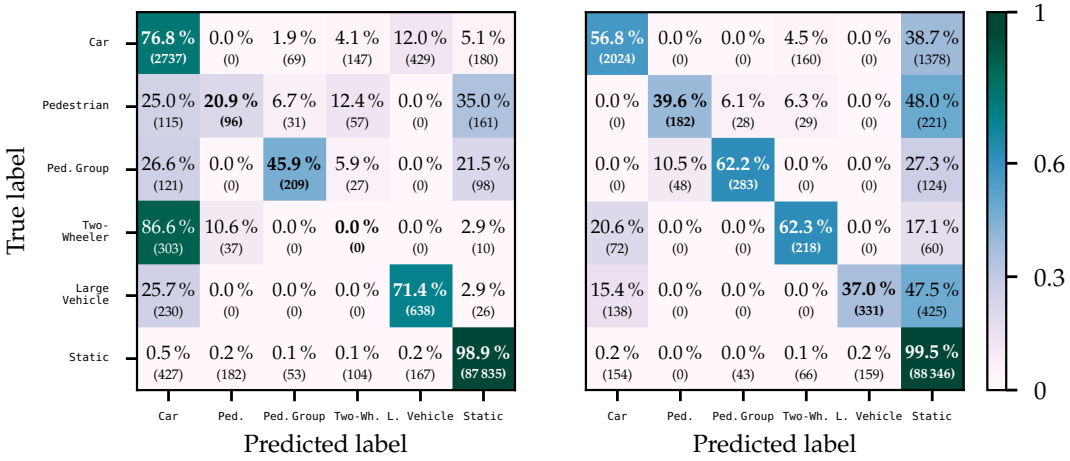


Figure 7.13: Per-target confusion matrices for the best performing labeler L_2 (right) and the worst performing labeler L_3 (left) for sequences of length $T = 0.6\,\mathrm{s}$.

In Fig. 7.13, the confusion matrices of the two test candidates L_3 and L_2 are shown for the $T = 0.6\,\mathrm{s}$ snippets. In contrast to the classification task, where the last column of the confusion matrices indicated that for some reason no label was attached to a ground truth cluster, this column can now be interpreted as the number of targets left out of a created cluster. The trends in the two results are quite different. Whereas L_3 is strongly biased towards predicting the `Passenger Car` class (first column of the confusion matrix), L_2 overlooked more targets and left them unlabeled (last column). If however L_2 creates a cluster for a dynamic object, the assigned label is most often correct since the greatest confusions happen usually with the `Static` class. Additionally, L_2 created much less false positive objects or put differently, L_2 assigned less truly `Static` targets to a dynamic object (last row in matrices). To summarize, the instance segmentation task naturally proved to be much harder than the classification task for the human annotators and the machine learning algorithms outperformed the manual work – at least the one from the three labelers.

7.5 COMPARISON AND SUMMARY

In this chapter, one feed-forward neural network for semantic segmentation and one recurrent network for instance segmentation were introduced and evaluated with different performance measures. Both networks take bare point clouds as input in which each point is defined by five features: two spatial coordinates, the ego-motion compensated Doppler velocity, the RCS value and the measurement time relative to the earliest measurement in the point cloud.

Method	F_1	Precision	Recall	Confusion Matrix
Semantic Segmentation	0.776	0.739	0.821	Fig. 7.4 right
Instance Segmentation, 14 time steps	0.811	0.777	0.849	Fig. 7.5 left
RF, Clusters from Network	0.777	0.793	0.768	Fig. 7.11 left
Ensemble	0.825	0.803	0.848	Fig. 7.11 right

Table 7.4: Comparison of the scores obtained with the segmentation approaches.

The instance segmentation approach outperformed the pure semantic segmentation algorithm by a great margin. The longer the input sequence to the network, the higher the performance gets, even though a clear saturation is visible. This finding underlines that the proposed network architecture makes use of temporal correlations in the data with the upside that only data from a limited period has to be used as input. In the case of the semantic segmentation network, the architecture strongly depends on the number of input points and hence on the temporal length of the input data.

Evaluating the importance of the input features showed that for both networks the Doppler velocity $\hat{v}_r$ has the greatest impact on the final score, see Table 7.1. Even with only spatial information present, the network is still able to distinguish most classes from each other.

The scores of the approaches shown in this chapter are listed again in Table 7.4. The ensemble classifier consisting of the recurrent instance segmentation network and a random forest performs best with respect to the F_1 score. As discussed in section 7.3.5 about the ensemble, this classifier has the downside of predicting more targets of truly dynamic objects as `Static` than the pure instance segmentation network. This has a small negative effect on the recall value. On the other hand, the ensemble classifier predicts less false positive targets of dynamic objects, which increases the precision value considerably. Which behavior is favorable, depends on the entire system in which the algorithm is used and what further processing is done with the data.

Predictions of the instance segmentation network are illustrated in the appendix in Section A.6. Positive as well as negative examples are shown and a camera image of the scene is displayed for reference. The arrows in the figures symbolize the ego-motion compensated Doppler velocity.

Finally, Fig. 7.14 again displays the confusion matrix of the instance segmentation network and the differences to the scores predicted by the baseline approach, in

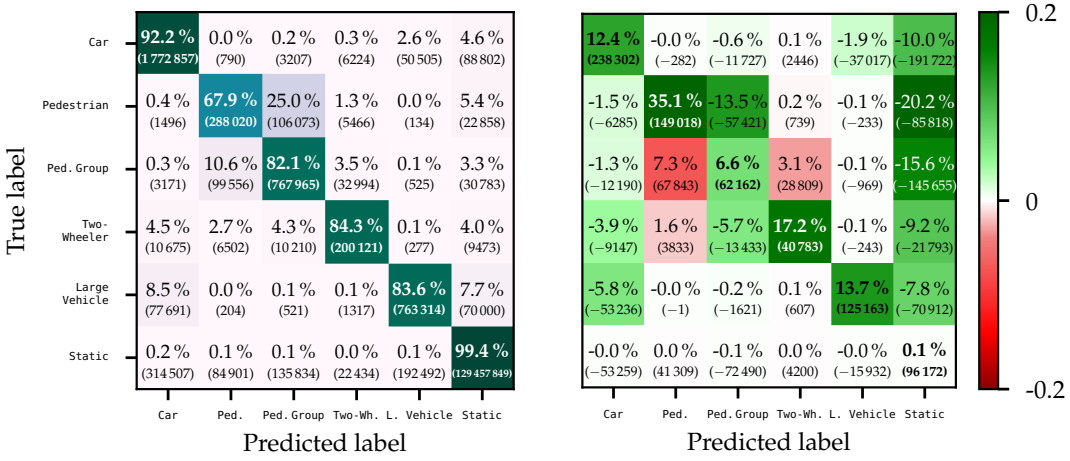


Figure 7.14: Comparison between the results from the instance segmentation network and the basic approach using standard DBSCAN for clustering and a random forest as classifier. Left: Confusion matrix of the instance segmentation network. Right: Differences to the random forest confusion matrix.

which DBSCAN$^-$ is used for clustering and a random forest predicts the class label based on handcrafted feature vectors. In the confusion matrix on the right hand side, green color indicates areas in which the instance classification outperforms the random forest and red areas mark the cells in which the random forest is superior, see also the description in Section 6.6. The true positive rate increased for all classes, with the greatest increase at the `Pedestrian` class with over 35 percentage points. The number of overlooked targets (last column) decreased significantly for all object types, with again the greatest difference in the `Pedestrian` class. While the confusion of `Pedestrian Group` targets with the `Pedestrian` and `Two-Wheeler` class increased, also the true positive rate of the `Pedestrian Group` category got higher. The incorrect predictions of the `Pedestrian` and `Two-Wheeler` class therefore stem on average from targets that were predicted as `Static` by the random forest. This finding is similar to the one already described in Section 6.6.

Experiments with three human annotators showed that their performance increases significantly with the duration of the presented sequences. However, even on sequences with 5 s length, the scores of the labelers remains far below the ones achieved by the network. The greatest difficulty in this task seems to be the correct identification of targets that belong to a moving object since the greatest confusion exists for all test candidates between the true class of a target and the `Static` class. With the limited number of test candidates in mind and the therefore limited expressiveness of the experiment, these findings hint that the machine learning approach is superior to a human at this task.

8

CONCLUSION

In this thesis, various methods for classification and semantic (instance) segmentation of moving road users were introduced. Solely radar measurements from a large real-world data set were used to construct, train and evaluate the machine learning based algorithms. Sequences were recorded over a period of about 1.5 years and afterwards annotated according to a newly developed label guideline for radar data.

From the various approaches discussed in this work, the newly developed network architecture that performs semantic instance segmentation on the raw radar point cloud achieves the highest scores. A general trend can be noticed between the different algorithms: the greater the extent to which the algorithm can work with raw data, the higher the resulting scores. For example, the classification network which performs the feature extraction step of previously generated clusters on its own, performs better than an algorithm that relies on manually created features, cf. Section 6.6. Similarly, if also the clustering step is included into a combined learning step, the score increases even further.

Handcrafted features are therefore less expressive than features generated by a neural network and a fully supervised instance creation is favorable over an unsupervised or semi-supervised clustering algorithm. The correct grouping of all radar targets that belong to the same object proved to be one of the critical steps in the classification task. If ground truth clusters are used as a basis, even a simple random forest classifier working with manually extracted features is able to discern most classes from each other. However, performance rapidly decays, if clusters from a DBSCAN algorithm are used, i.e. if a system is considered in which clusters are automatically created.

In Table 8.1, scores of some approaches discussed in this thesis are repeated again. The baseline is given by a random forest classifier with basic DBSCAN for clustering. Extending the clustering algorithm by a learning phase to incorporate domain knowledge into the hyper-parameters of the approach (cf. Chapter 5), results in a noticeable performance increase (second row in the table). Incorporating time information into the classification algorithms increases the scores slightly (rows three and five), even though much more data can be used for the decision. A

Method	F_1	Precision	Recall	Confusion Matrix
Random Forest, DBSCAN$^-$	0.719	0.753	0.708	Fig. 6.9 left
Random Forest, DBSCAN$^+$	0.761	0.785	0.746	Fig. 6.9 right
LSTM	0.766	0.774	0.763	Fig. 6.13 right
PointNet++	0.771	0.777	0.767	Fig. 6.15 left
PointNet++ & LSTM	0.776	0.781	0.774	Fig. 6.15 right
Semantic Segmentation	0.776	0.739	0.821	Fig. 7.4 right
Instance Segmentation	0.811	0.777	0.849	Fig. 7.5 left

Table 8.1: Scores of the different approaches presented in this work.

similar statement can be made for the semantic (instance) segmentation approaches, where using data from multiple time steps raises the F_1 score, see also Section 7.3.3, especially Fig. 7.7.

Experiments with human annotators showed that in contrast to the proposed algorithms, a considerable performance increase can be measured if sequences of longer duration have to be classified or segmented. This suggests that the way temporal information was incorporated into the algorithms was not yet optimal and that investigations into this direction could pay off. However, in the instance segmentation task the neural network outperformed the three test candidates by a great margin, especially due to its superiority in discerning truly dynamic objects from the static environment.

The lack of a publicly available radar data set to compare the algorithms of different researchers makes it difficult to put the proposed approaches into perspective. However, since more and more companies and research institutes release data sets, chances increase that an open benchmark will become possible.

The results of this thesis indicate that with the advent of the next generation radar sensors, which provide height information, better angular resolution and less clutter, the classification performance can be further increased since especially spatial features always proofed to be important to the algorithms. Integration of one additional spatial dimension is trivially possible in the proposed recurrent neural network architecture.

A

APPENDIX

A.1 FIVE INGREDIENTS FOR SUPERVISED LEARNING

In Section 2.1, supervised learning methods were introduced. Five ingredients were listed which are needed for each algorithm from this category. In this section, more details about these five ingredients are given and some examples are presented.

Data Set

The *data set* $\mathcal{X}$ is the basis from which the algorithm should learn the defined task. For example, each datum x in the data set can be a single image [216], an audio sequence [88], a track resulting from a particle collision in the LHC [116], [213] or any other abstract feature vector.

Depending on the task and chosen algorithm, pre-processing of the inputs might be necessary. The pre-processing steps can be as simple as re-sizing an image or as complex as changing the representation of x by calculating hand-crafted feature vectors or apply dimensionality reduction algorithms to reduce the number features which define x.

To assess later on the generalization power of the algorithm, not all elements in $\mathcal{X}$ are used in the training stage. Some parts are left out during training and used only in the test phase for evaluation. There are two common ways to split the data. The first way is to split $\mathcal{X}$ into three parts: $\mathcal{X}_{\text{train}}$, $\mathcal{X}_{\text{val}}$ and $\mathcal{X}_{\text{test}}$. The largest part of the data goes into $\mathcal{X}_{\text{train}}$ which is used to train the algorithm and adapt the weights θ_j. Every machine learning algorithm contains some parameters h_i which define the structure of the algorithm itself and cannot be optimized during the training stage. These parameters are called hyper-parameters. In order to find suitable values for these non-trainable parameters, the performance of a trained algorithm is estimated on the *validation data set* $\mathcal{X}_{\text{val}}$ and the h_i are modified manually until the performance on $\mathcal{X}_{\text{val}}$ saturates at a maximum. After that, a final evaluation on $\mathcal{X}_{\text{test}}$ is done, to obtain unbiased metrics that describe the generalization ability of the trained model. If tuning of the h_i was performed on the test data, information

would leak from the supposed to be unknown data into the training phase and thereby an over-confident performance metric would be reported, see e.g. chapter 5 in [79]. The second way to split the data is the so-called *cross-validation* scheme. During n-fold cross-validation, the data is split into n parts and each of the n parts is used in turn as a test set while the remaining $n-1$ parts are used for training. Hyper-parameters should be fixed beforehand on a separate validation set. This method is often used to obtain more reliable metrics for the generalization abilities of the algorithm since one single test set might be biased in some way. For example, in a classification task the test set might have an imbalanced class distribution or contain disproportionately many difficult (or easy) samples. Splitting data into the different sets has to be done carefully to avoid that information leaks from the test set into the training data. For example, if for a classification task of camera images a video was captured and the splitting into training and test data would be done by creating blocks of ten seconds and using the first eight seconds of each block for training and the last two seconds for testing, then there would be a large temporal correlation between the training and the test set. Frames at the beginning and the end of the two-second test data block would be very similar to frames of the previous and following training data block so that overconfident metrics would be reported. The splitting of the radar data used in this work is described at the respective later chapters.

Ground Truth Labels

Collecting data is often inexpensive with regards to time and money. In contrast, obtaining *ground truth labels* $\mathcal{Y}$ for the acquired data requires usually human experts who go through the collected data and attach labels y to each datum x. Modern machine learning algorithms like deep convolutional neural networks require often large amounts of labeled data to achieve high performance. For example, for the popular object localization task in the "Large Scale Visual Recognition Challenge", training data with 1.2 million manually labeled images are provided and further 150 000 annotated images are available for validation [216]. Companies developed different strategies to obtain labeled data without spending too much money on the labeling process. The most ingenious method is Google's reCAPTCHA system [80]. The system is advertised as a free service that protects web projects from spam and automated requests by presenting a *Completely Automated Public Turing test to tell Computers and Humans Apart (CAPTCHA)* to peculiar users. Depending on the version of the system, these CAPTCHAs include tasks like selecting all parts of an image that contain traffic lights, typing in the letters of a distorted and scanned text or typing in the house number that can be read off from a photograph. Owners of web services make use of reCAPTCHA because it is reliable, free as well as easy

to implement. The advantage for Google is that they obtain labeled data from the users of the third party web services free of charge and can use the data to increase the performance of their own machine learning algorithms. Other strategies to decrease the demand of manually labeled data is to use auto-labeling concepts. Auto-labeling means that the desired label information is obtained automatically from a different algorithm. For example, in [188] a strategy for obtaining per-point labels for lidar data from the classification results of simultaneously recorded camera images is presented. The idea is that semantic information can be obtained easier from images because plenty of pre-trained neural networks [40], [81], [195] already exist which can provide pixelwise class information. If the lidar point cloud and the classified camera image are aligned, the semantic information from the image-classifier can be propagated to the lidar points and hence no manual labeling is required.

Depending on the task, the complexity of the labeling job varies. For a simple classification task of an image, only a single label has to be attached to the image. These labels are usually represented by integer indices so that $y_i \in \{1, 2, \ldots, N_{\text{cls}}\}$, where N_{cls} is the number of different classes. For a pixelwise semantic segmentation of an image, every pixel has to be annotated so that for an image X_i with width W and height H the label result y_i is also an $W \times H$ matrix where $(y_i)_{mn} \in \{1, 2, \ldots, N_{\text{cls}}\}$. If not only semantic segmentation but semantic *instance* segmentation is desired, the label result y_i is an $W \times H \times 2$ tensor of two stacked $W \times H$ matrices where the first matrix is identical to the one from the semantic segmentation labeling task and the second matrix encodes the instance id. The instance id is used to differentiate between objects of the same kind in an image, e.g. the pixels of two chairs in an image would all get the same semantic label id but the pixels of the first chair would get a different instance id than the pixels of the second chair. These examples illustrate that the labeling costs can vary dramatically depending on the actual task that should be performed in the end. Unfortunately, labeling of radar data is extremely difficult for a number of reasons. The labeling process is discussed extensively in section 4.2. Irrespective of the concrete task, labels fulfill the need of machine learning algorithms to compare the predicted output with ground truth data.

Algorithm

Machine learning algorithms have usually two operation modes, namely *training* and *testing*. During training, the internal parameter θ_j, $j = 1, \ldots, N_{\text{param}}$ are adapted so that the predicted outcome $\hat{y}$ resembles the ground truth label y. After the training phase, the internal parameters are held constant and only the algorithm's output

(i.e. the algorithm's prediction) is used for whatever application the algorithm was designed for. The exact way in which the θ_j are updated depends on both the algorithm itself and the optimizer that is used for training. Of course, not every machine learning algorithm is suitable for a given task but rather an algorithm has to be chosen whose *capacity* fits the complexity of the task. For example, *logistic regression* has way smaller capacity than a large convolutional neural network like GoogLeNet [252] and hence logistic regression is not expected to perform well on image classification tasks. In other words, algorithms with a lower capacity tend to *underfit* a given problem, i.e. the learned decision function is too simple to represent the complexity of the given task. The other extreme called *overfitting* happens when the algorithm has too many degrees of freedom so that the decision function is fitted tightly to the training data. A very noise und highly varying decision function is the result of overfitting. To allow models to have a large number of tunable parameters so that also complex tasks can be learned, *regularization* schemes have been developed which aim to reduce overfitting, e.g. by penalizing very large weights (c.f. ℓ_1 and ℓ_2 regularization in [79]). In the following sections 2.1.1 and 2.1.2, the random forest classification algorithm and artificial neural networks will be introduced.

Loss Function

The fourth ingredient for supervised learning is the *loss function*. The loss function takes as input the predicted value and the ground truth label and returns an error measure that indicates how well the algorithm performed on the current input sample. Often, either the loss function can be interpreted as a distance between $\hat{y}$ and y or the loss is directly implemented as a distance function. For example, the ℓ_2-loss function is simply the sum of the squared distances of the output and ground truth vectors:

$$L_{\ell_2}(\hat{y}, y) = ||\hat{y} - y||^2. \tag{A.1}$$

For classification tasks, often the so-called *cross-entropy* is used. Let $\vec{q} \in \mathbb{R}^{N_{\text{cls}}}$ be the predicted probability vector over the N_{cls} possible classes of a single sample x so that $\sum_i q_i = 1$ and $\hat{y} = \arg\max_i q_i$. Accordingly, let $\vec{p} \in \mathbb{R}^{N_{\text{cls}}}$ be the ground truth probability vector for this sample which has zeros everywhere except for position y, where $p_y = 1$. The cross-entropy is then defined as

$$H(p, q) = -\sum_{i}^{N_{\text{cls}}} p_i \log q_i. \tag{A.2}$$

Often, the cross entropy is not calculated for only a single sample x but for a *batch* of N samples which introduces just another sum in the calculation of $H(p, q)$ where

p_{ij} and q_{ij} are now the ground truth and predicted probabilities for class i of sample j:

$$H(p,q)_N = -\frac{1}{N}\sum_{j=1}^{N}\sum_{i=1}^{N_{\text{cls}}} p_{ij}\log q_{ij}. \tag{A.3}$$

The cross-entropy is tightly linked to the Kullback-Leibler divergence $D(p||q)$ which is a measure for the similarity of two probability distributions p and q over the random variable x:

$$D(p||q) = \sum_x p(x)\log\frac{p(x)}{q(x)} = H(p,q) + \sum_x p(x)\log p(x). \tag{A.4}$$

The point is now that a maximum likelihood estimation of the model parameters θ can be interpreted as minimizing the dissimilarity between the empirical distribution of the training data $\mathcal{X}$ and the learned model distribution where the dissimilarity is measured by the Kullback-Leibler divergence. Therefore, finding a maximum likelihood solution for the θ_j of the model corresponds directly to *minimizing* $D(p||q)$ and therefore it corresponds also to minimizing the cross entropy $H(p,q)$. For details see for example Chapter 5 in [79] or Chapter 2 in [46].

Optimizer

The last ingredient is finally the *optimizer*. The optimizer is responsible for updating the internal weights θ_j after each forward pass of training data. The way the weights are updated from training step t to the next step $t+1$ highly depends on the actual machine learning algorithm itself. For example, the decision trees in a random forest classifier are constructed in a greedy fashion by selecting as a split criterion the feature which results in the maximum information gain. Details about random forests are discussed in the following section. In this case, the optimizer is tightly linked to the algorithm itself. In contrast, there exist several different methods to optimize the weights in a neural network. At the root of these optimizers lies the idea that the gradient of the loss function with respect to the weights gives information on how to change the weights so that the loss decreases. These *gradient descent* methods proofed to be very successful even for large networks with millions of parameters. The basic form of gradient descent, namely

$$\theta_j^{t+1} = \theta_j^t - \alpha\frac{\partial}{\partial\theta_j}L(\hat{y}(\theta_1,\ldots,\theta_{N_{\text{param}}}),y) \tag{A.5}$$

is only rarely used in real world application but its simplicity allows for an easy understanding of the update scheme and is therefore often cited in introductory

works and tutorials [29], [164], [251]. Since the gradient of a function L points in the direction of steepest *ascent*, subtracting the gradient from the current value of the parameter causes a *decrease* of the function. The idea is hence to follow the (negative) gradient until the loss function converges. The parameter α is called the *learning rate* of the optimizer and defines how far the current gradient is followed until the next θ_j is found. Manual tuning of the learning rate proofed to be difficult so that nowadays often the Adam optimizer [123] is used which internally controls the learning rate for each of the N_{param} parameters by incorporating information from the first and second stochastic moments of the gradients into the update rule:

$$\theta_j^{t+1} = \theta_j^t - \alpha \frac{m_t}{\sqrt{v_t} + \epsilon}. \tag{A.6}$$

The parameter ϵ is a small number that is added to avoid division by zero, m_t is a bias-corrected moving average of the past gradients that were computed for θ_t and v_t is the corresponding bias-correct second moment of θ_t's past gradients. The value for α has to be understood as a *maximum* learning rate since the actually used learning rate in an update step will mostly be different from α due to the scaling with m_t and v_t. Many other variants of (stochastic) gradient descent algorithms exist, e.g. AdaGrad [58], AdaMax [123] or RMSprop [99].

The great advantage of gradient descent based optimizer is that the gradients can be computed efficiently using the *backpropagation* scheme [215]. In its essence, this scheme describes only the repetitive application of the chain rule from calculus. Derivative free optimization schemes like simulated annealing are rarely found in the context of neural networks since their performances degrades quickly with increasing number of parameters [208]. Already for $N_{\text{param}} > \mathcal{O}(100)$ most of the methods mentioned in [208] perform sub-optimally so that application of these algorithms for networks with several million parameters is unlikely to succeed. Frameworks like Google's tensorflow use *reverse-mode automatic differentiation* for the computation of the gradients [9], [74]. A computation graph is created which has nodes for each operation in the network. For example, the addition of two numbers u and v is represented by three nodes: one node for the input u, one node for the input v and one node for the sum $u + v$. The computation graph is used for both the forward pass through the network and the reverse pass during which the gradients are computed. For the latter, the output of one node is considered as a function of the inputs to this node. For each node, not only the definition of the forwards pass is needed but also the definition of its gradient with respect to all inputs. Since after one forward pass the values of all nodes are known, the gradients for each output variable can be obtained in a reverse pass through the computation graph during which all partial derivatives of the nodes are combined using the chain rule. That is, for a parameter update of all weights in the neural network, one forward pass plus one reverse pass for each output variable is needed.

In contrast, numerical differentiation would require at least one forward pass for each of the N_{param} parameters of the network making this method unfeasible. It should be noted that *backpropagation* is *not* an optimizer. It is only a way to calculate gradients efficiently.

A.2 SPATIAL DISTRIBUTION OF RADAR TARGETS

The following images display how the radar targets of the five classes `Passenger Car`, `Pedestrian`, `Pedestrian Group`, `Two-Wheeler` and `Large Vehicle` are distributed around the field of view of the four sensors.

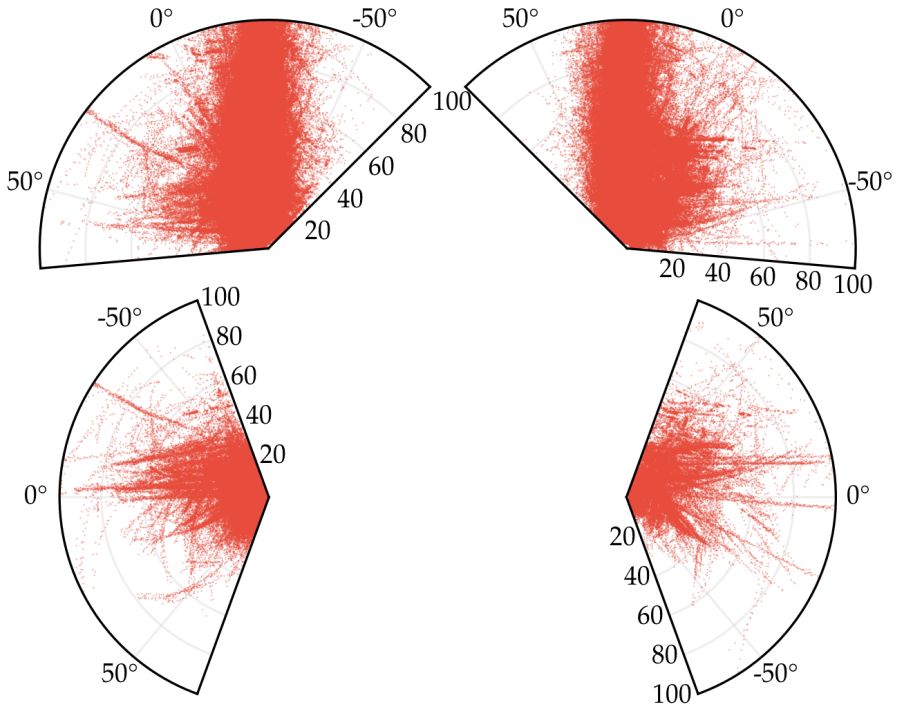


Figure A.1: Target distribution in a polar plot. The measured targets of each sensor are displayed in a separate diagram. The diagrams are rotated by the mounting position of the respective sensor. Only targets of the class `Passenger Car` are displayed.

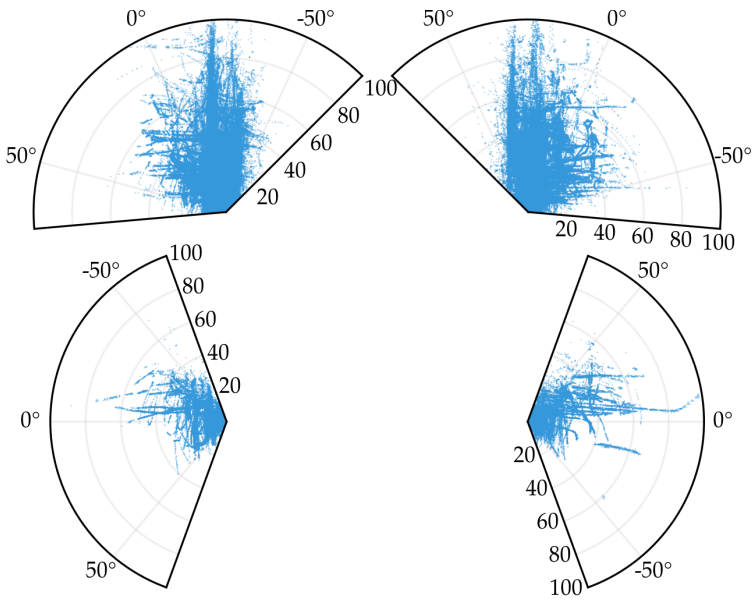


Figure A.2: Target distribution in a polar plot. The measured targets of each sensor are displayed in a separate diagram. The diagrams are rotated by the mounting position of the respective sensor. Only targets of the class `Pedestrian` are displayed.

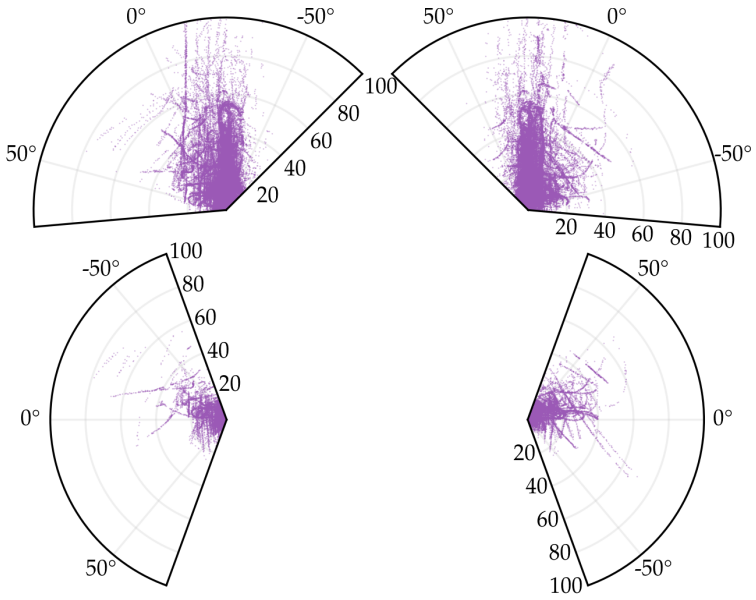


Figure A.4: Target distribution in a polar plot. The measured targets of each sensor are displayed in a separate diagram. The diagrams are rotated by the mounting position of the respective sensor. Only targets of the class `Two-Wheeler` are displayed.

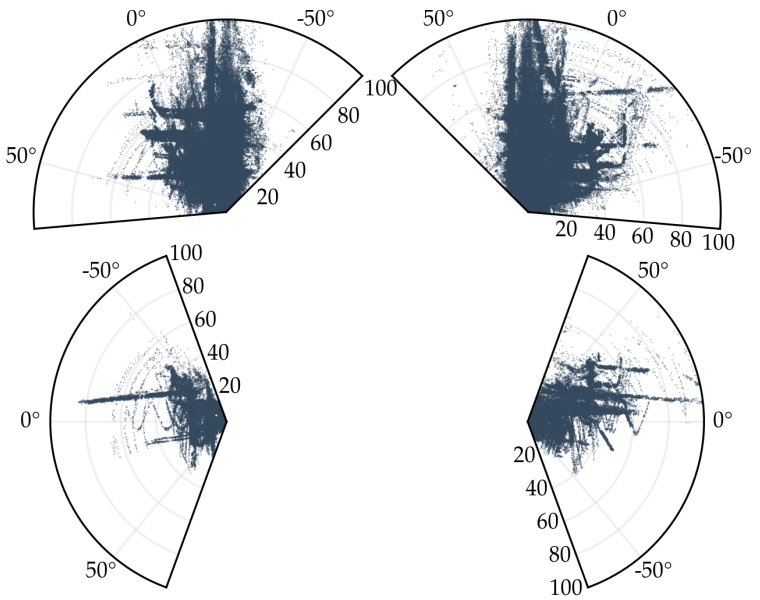


Figure A.3: Target distribution in a polar plot. The measured targets of each sensor are displayed in a separate diagram. The diagrams are rotated by the mounting position of the respective sensor. Only targets of the class `Pedestrian Group` are displayed.

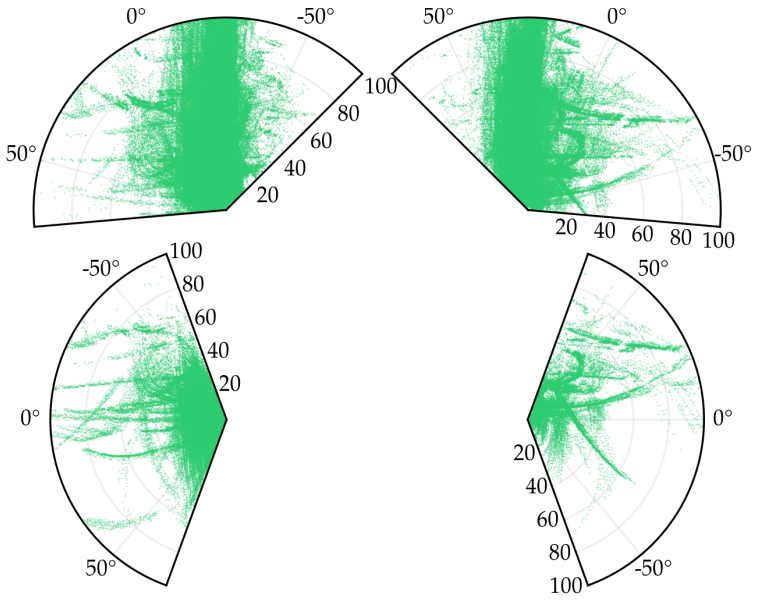


Figure A.5: Target distribution in a polar plot. The measured targets of each sensor are displayed in a separate diagram. The diagrams are rotated by the mounting position of the respective sensor. Only targets of the class `Large Vehicle` are displayed.

A.3 RADAR CROSS SECTIONS

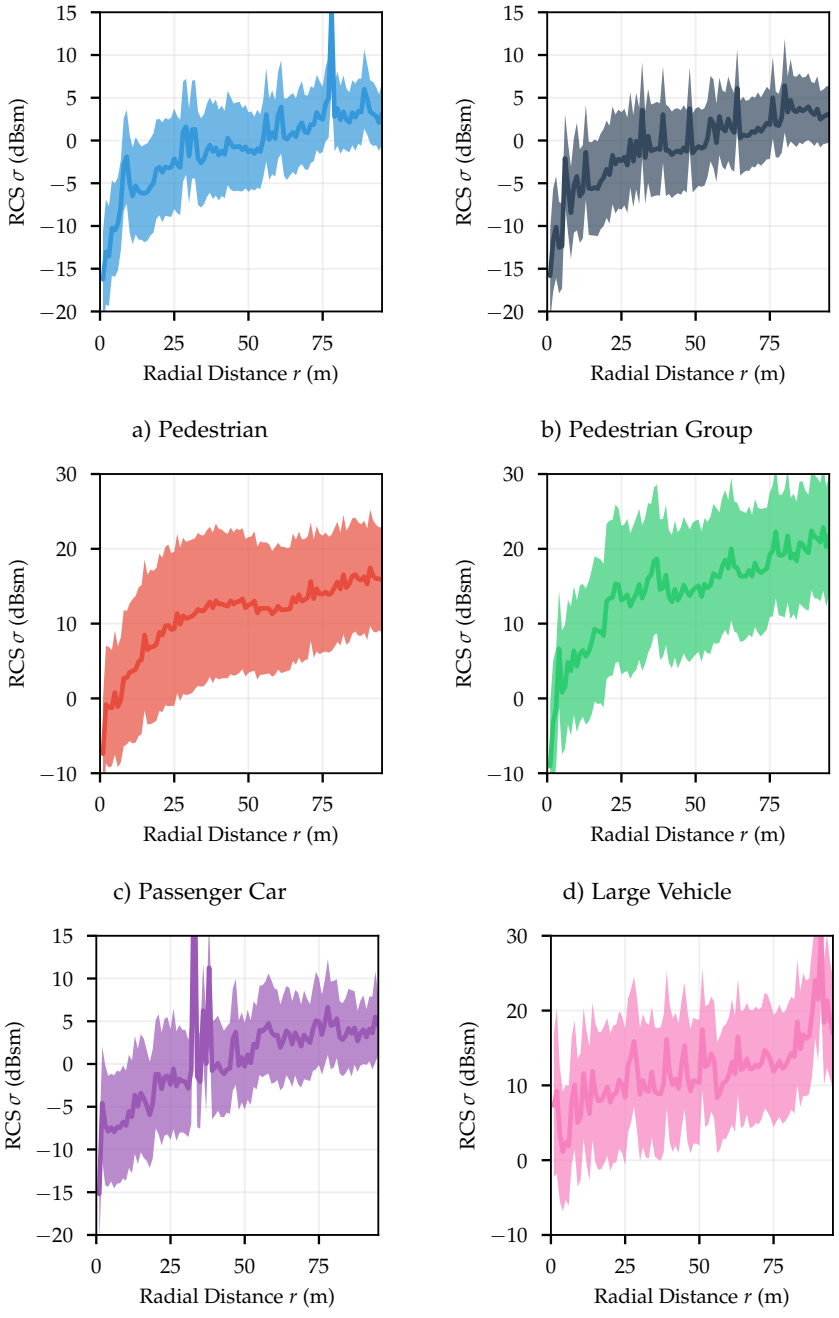


Figure A.6: Measured RCS values of the different semantic classes as a function of the radial distance at which the respective targets were measured.

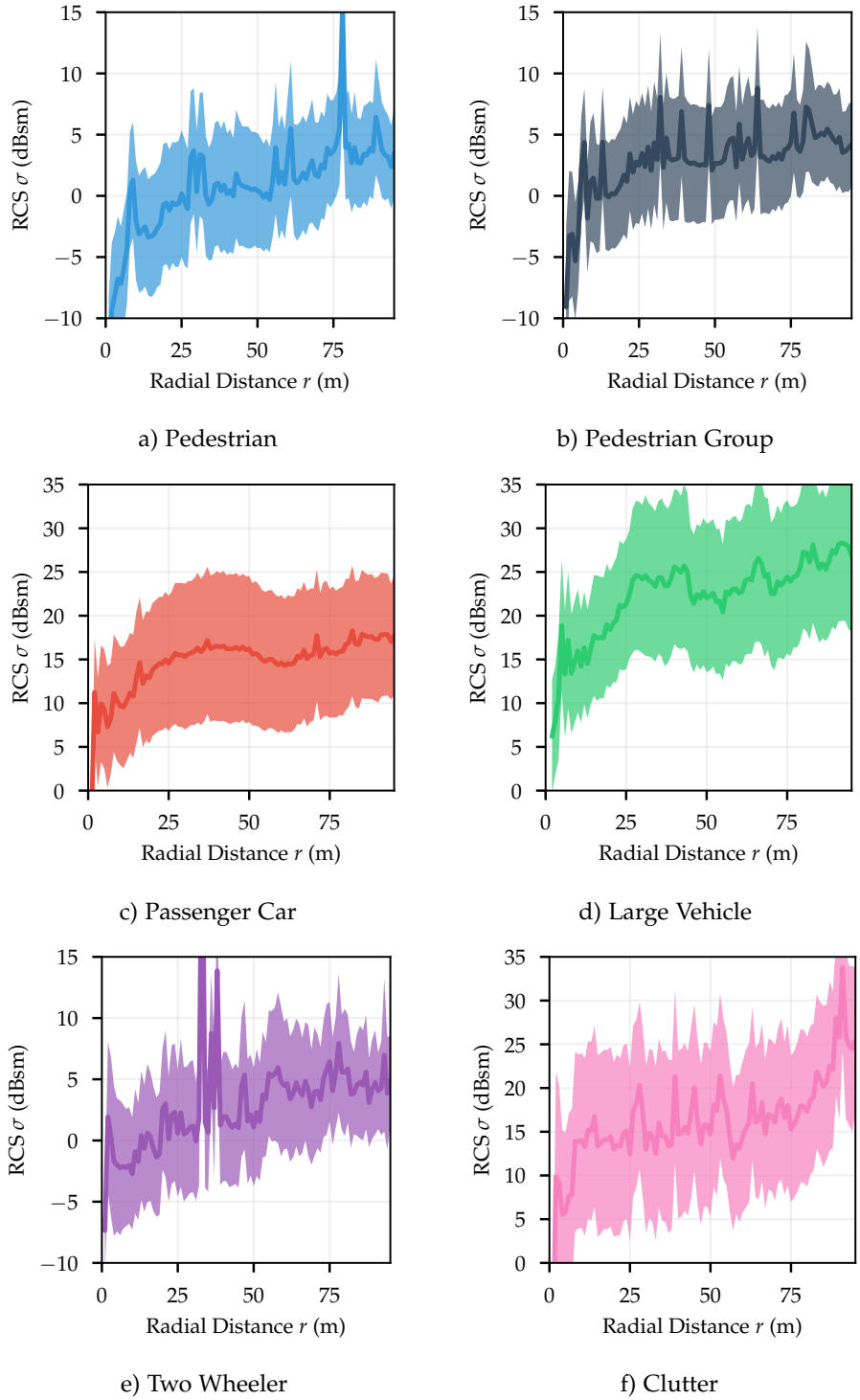


Figure A.7: Measured RCS values of the different semantic classes as a function of the radial distance at which the respective objects were measured. The RCS values are summed for each object first and then the average value for each range bin is calculated.

A.4 CLUSTER LEARNING RESULTS

Resulting parameters of the supervised clustering approach discussed in Section 5.2 are displayed in the following.

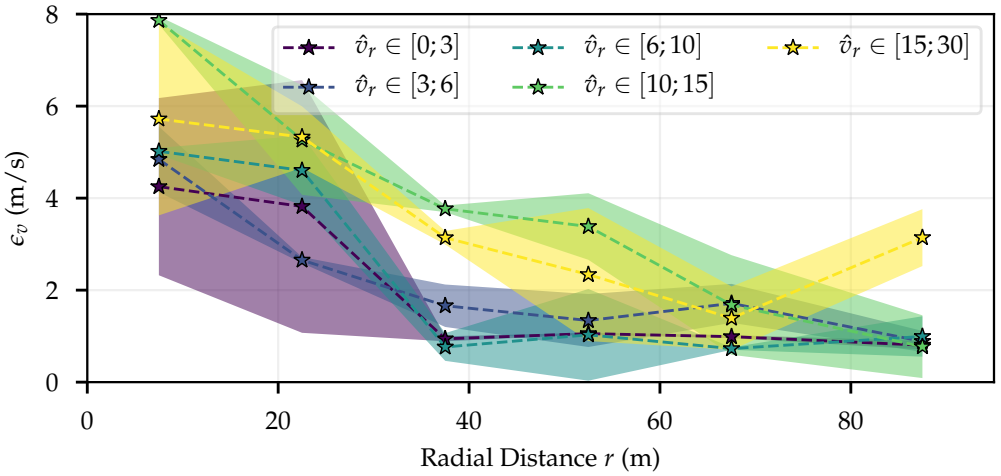


Figure A.8: Averaged values for ϵ_v plotted against radial distance for each of the five different velocity intervals. Shaded areas indicate regions with plus/minus one standard deviation around the mean value.

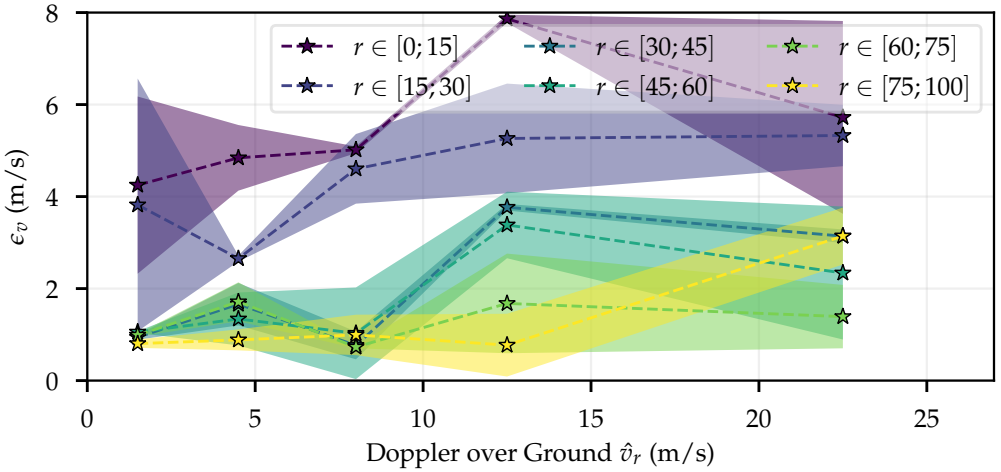


Figure A.9: Averaged values for ϵ_v plotted against Doppler velocity for each of the six range regions. Shaded areas indicate regions with plus/minus one standard deviation around the mean value.

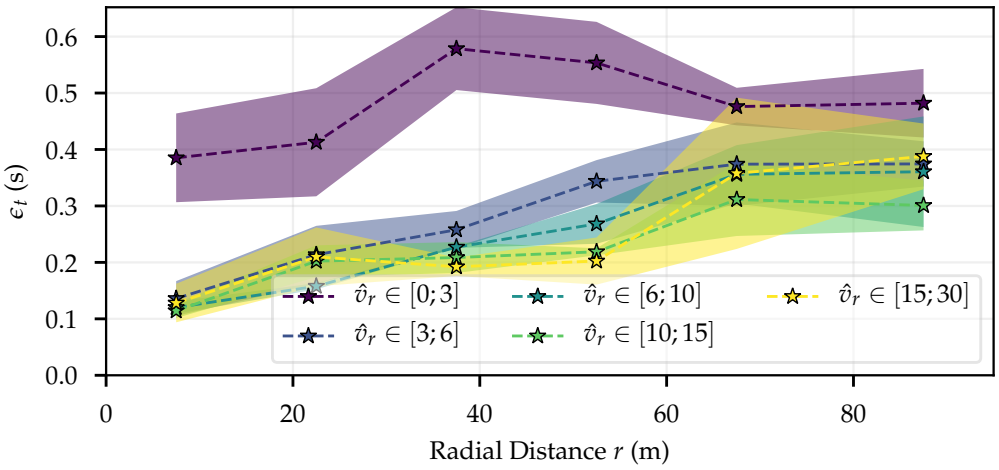


Figure A.10: Averaged values for ϵ_t plotted against radial distance for each of the five different velocity intervals. Shaded areas indicate regions with plus/minus one standard deviation around the mean value.

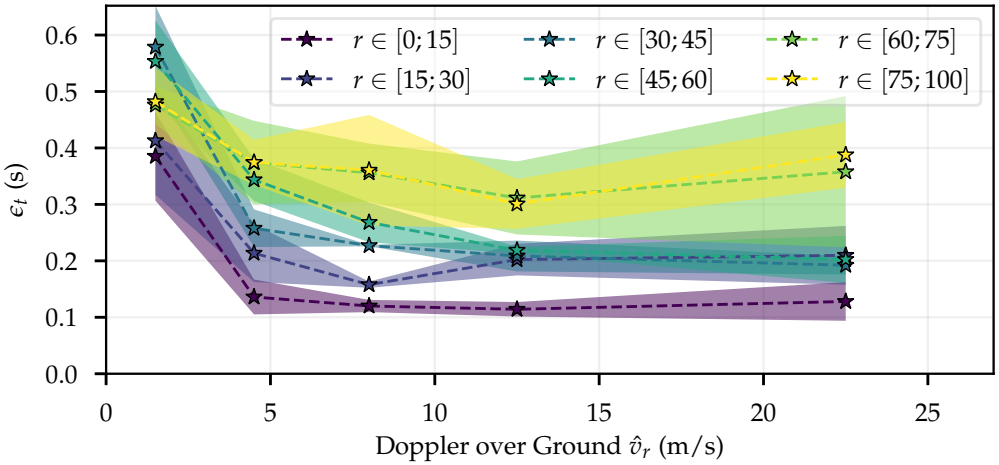


Figure A.11: Averaged values for ϵ_t plotted against Doppler velocity for each of the six range regions. Shaded areas indicate regions with plus/minus one standard deviation around the mean value.

A.5 PARAMETERS OF THE INSTANCE SEGMENTATION NETWORK

This section lists the hyper-parameters that were used for training and evaluation of the recurrent instance segmentation network. For a description of the used symbols see Section 7.2.2.1.

Point Feature Generation Module

- MSG @768, $r = [1, 2, 5]$, $n = [4, 8, 20]$, $c = [[64, 64, 128], [64, 64, 128], [64, 64, 128]]$
- MSG @512, $r = [3, 6]$, $n = [8, 20]$, $c = [[96, 96, 128], [96, 96, 128]]$
- MSG @256, $r = [4, 7]$, $n = [8, 20]$, $c = [[128, 128, 128], [128, 128, 128]]$
- Set Abstr. $[128, 128, 256]$
- Feat. Prop. $[128, 128]$
- Feat. Prop. $[128, 128]$
- Feat. Prop. $[128, 128]$
- Feat. Prop. $[128, 256]$

Memory Abstraction Module

Lookup in Memory Pointcloud

- Search Radii: $r = [1, 2, 5]$
- Number of Neighbors: $n = [4, 8, 20]$
- Convolutions: $c = [[64, 64, 128], [64, 64, 128], [64, 64, 128]]$

Semantic Segm. Module

- 1D Convolution: $c = 128$
- Dropout: $p = 0.5$
- 1D Convolution: $c = 128$
- Dropout: $p = 0.5$
- 1D Convolution: $c = 6$

Direction Module

- 1D Convolution: $c = 128$
- Dropout: $p = 0.5$
- 1D Convolution: $c = 128$
- Dropout: $p = 0.5$
- 1D Convolution: $c = 2$

Memory Update Module

- 1D Convolution: $c = 128$
- 1D Convolution: $c = 128$
- 1D Convolution: $c = 128$

Instance Creation

- $m_{\text{dyn}} = 250$
- $m_{\text{r}} = 10$
- $r_{\text{dyn}} = 0.8$ m
- $m_{\text{inst}} = 70$
- $n_{p,\text{inst}} = 150$

Instance Classification

- MSG @5, $r = [0.5, 1, 3]$, $n = [2, 4, 8]$, $c = [[16, 16, 32], [16, 16, 32], [16, 16, 32]]$
- MSG @3, $r = [2, 4]$, $n = [5, 5]$, $c = [[16, 16, 32], [16, 16, 32]]$

} shape features

- 2D Convolutions: $c = [64, 64, 128, 256, 512]$
- Fully Connected: $c = 256$
- Fully Connected: $c = 6$

A.6 EXAMPLE PREDICTIONS OF THE INSTANCE SEGMENTATION NETWORK

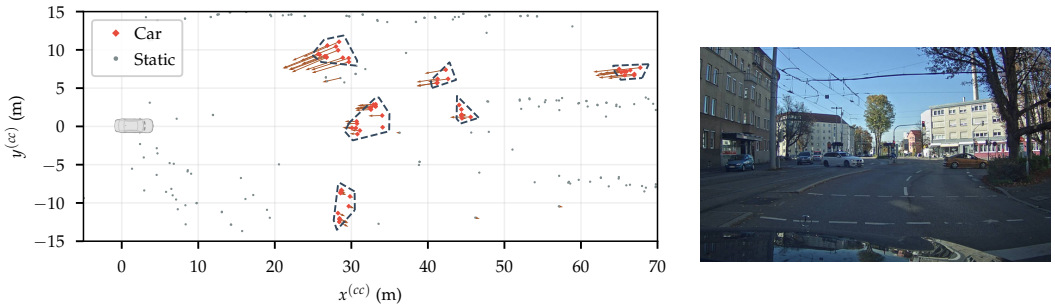


Figure A.12: Almost perfect instance segmentation of multiple cars driving straight or performing a left turn.

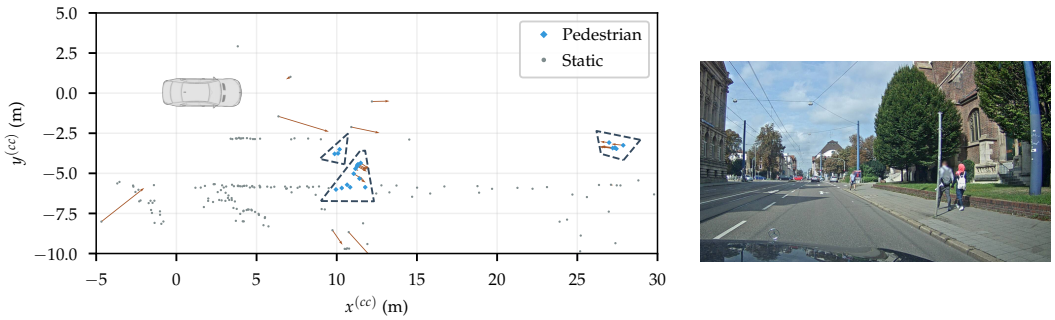


Figure A.13: The network correctly created three `Pedestrian` instances. One of the instance contains measurements from the nearby wall.

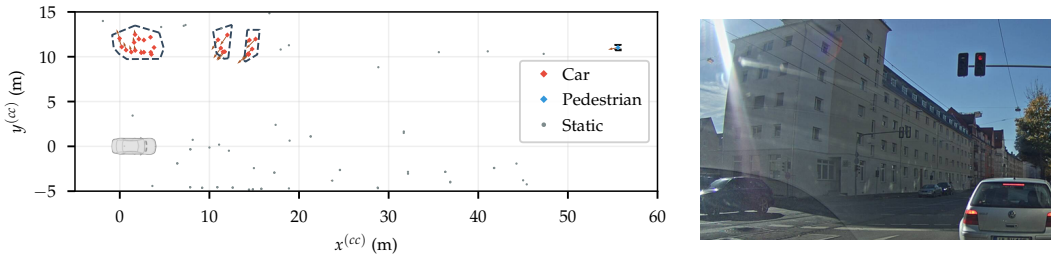


Figure A.14: Correct segmentation of one car left of the ego vehicle. For the second car, two instances are predicted. The predicted `Pedestrian` does not exist.

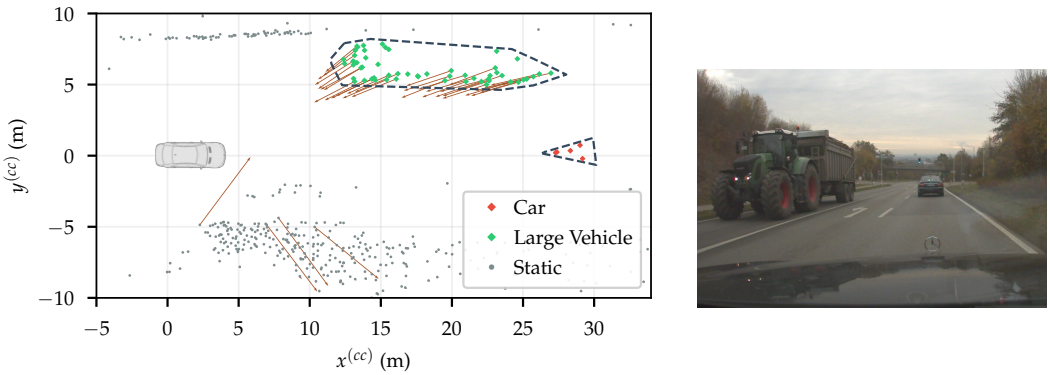


Figure A.15: Perfect classification and segmentation of a tractor and a car.

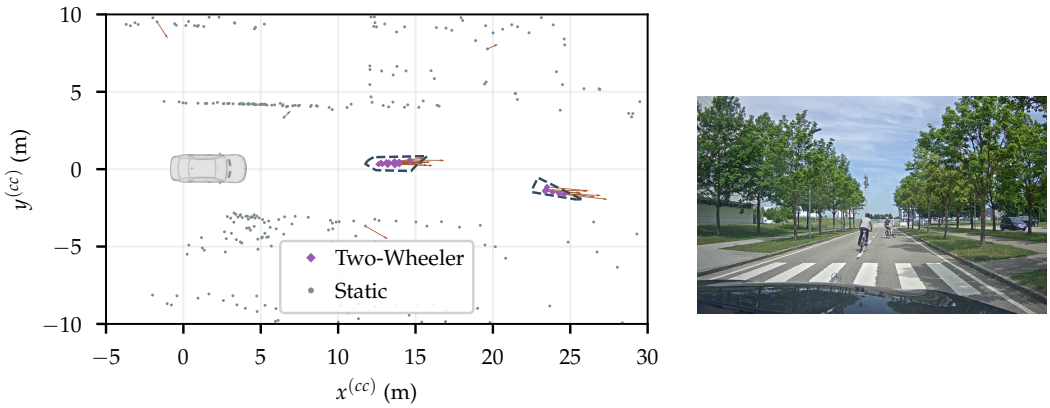


Figure A.16: Two bicycles in front of the ego-vehicle are correctly identified.

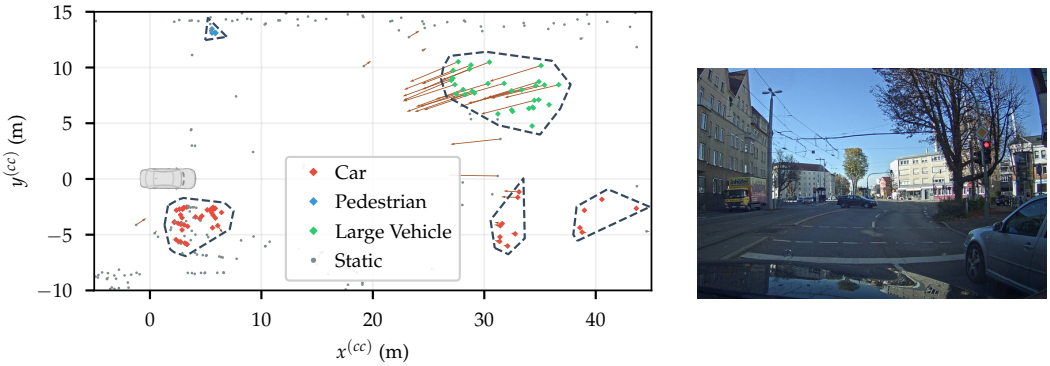


Figure A.17: The predicted width of car right of the ego-vehicle is too large. A ghost object is created behind the car which performs the left turn.

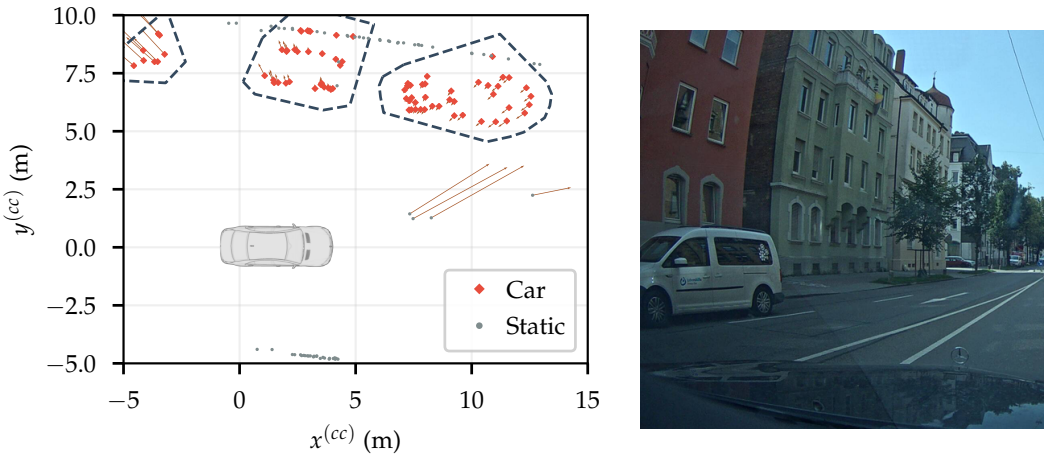


Figure A.18: The approaching cars extend a bit too much into the sidewalk but are otherwise well captured.

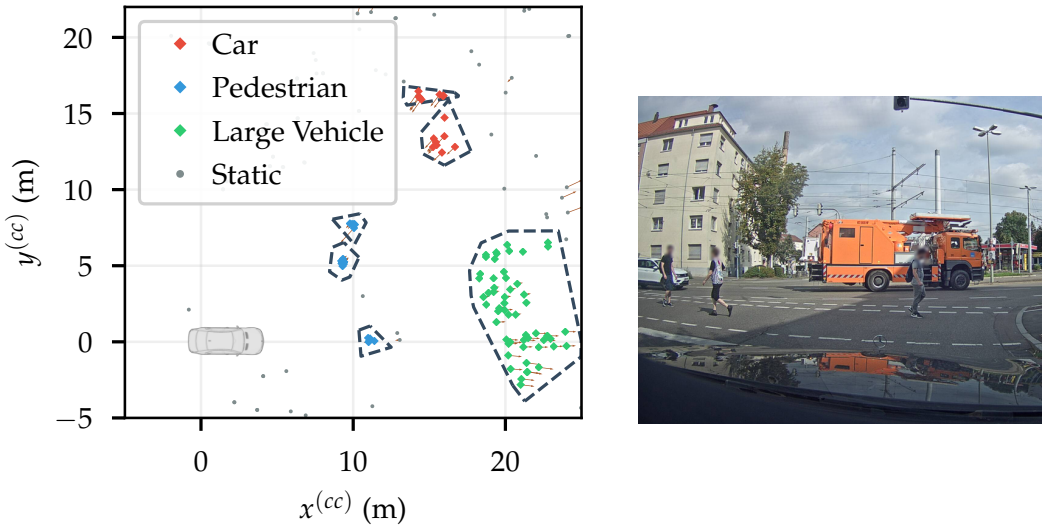


Figure A.19: All three pedestrians, the truck and the car are correctly classified. However, two car instances are created instead of only one.

BIBLIOGRAPHY

[1] A. Angelov, A. Robertson, R. Murray-Smith, *et al.*, "Practical classification of different moving targets using automotive radar and deep neural networks", *IET Radar, Sonar & Navigation*, vol. 12, no. 10, pp. 1082–1089, Oct. 2018.

[2] M. Ankerst, M. M. Breunig, H.-P. Kriegel, *et al.*, "OPTICS: Ordering Points To Identify the Clustering Structure", *ACM SIGMOD international conference on Management of data*, vol. 28, no. 2, pp. 49–60, Jun. 1999.

[3] Apple: Frameworks Natural Language Processing Team, *Can Global Semantic Context Improve Neural Language Models?*, 2018. [Online]. Available: `https://machinelearning.apple.com/2018/09/27/can-global-semantic-context-improve-neural-language-models.html` (visited on 08/15/2019).

[4] Audi, *Adaptive Cruise Control mit Stop & Go-Funktion*, 2011. [Online]. Available: `https://www.audi-technology-portal.de/de/elektrik-elektronik/fahrerassistenzsysteme/adaptive-cruise-control-mit-stop-go-funktion` (visited on 06/06/2019).

[5] Audi, *Fahrerassistenzsysteme*, 2017. [Online]. Available: `https://www.audi-mediacenter.com/de/technik-lexikon-7180/fahrerassistenzsysteme-7184` (visited on 06/06/2019).

[6] M. Bai and R. Urtasun, "Deep Watershed Transform for Instance Segmentation", in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, IEEE, Jul. 2017, pp. 2858–2866.

[7] M. Barjenbruch, *Drive U - Übersichtsvortrag Radar*, Ulm, 2015.

[8] G. E. A. P. A. Batista, A. L. C. Bazzan, and M. C. Monard, "Balancing Training Data for Automated Annotation of Keywords: a Case Study", *In Proceedings of the Second Brazilian Workshop on Bioinformatics*, 2003.

[9] A. G. Baydin, B. A. Pearlmutter, and A. A. Radul, "Automatic Differentiation in Machine Learning: a survery", *arXiv preprint*, 2015.

[10] S. Bernard, L. Heutte, and S. Adam, "Influence of hyperparameters on random forest accuracy", in *Lecture Notes in Computer Science*, Berlin, Heidelberg: Springer, 2009.

[11] S. Beucher and C. Lantuejoul, "Use of Watersheds in Contour Detection", in *International Workshop on Image Processing: Real-time Edge and Motion Detection/Estimation*, Rennes, 1979.

[12] S. Birchfield, *Image Processing and Analysis*, 1st ed. Cengage Learning, 2017.

[13] C. M. Bishop, *Neural Networks for Pattern Recognition*, 1st ed. Oxford: Clarendon Press, 1995.

[14] J. M. Blackledge, *Digital Signal Processing: Mathematical and Computational Methods, Software Development and Applications*, 2nd ed. 2006.

[15] S. Blake, "OS-CFAR Theory for Multiple Targets and Nonuniform Clutter", *IEEE Transactions on Aerospace and Electronic Systems*, 1988.

[16] C. Blakemore and F. W. Campbell, "On the existence of neurones in the human visual system selectively sensitive to the orientation and size of retinal images", *The Journal of Physiology*, vol. 203, no. 1, pp. 237–260, Jul. 1969.

[17] C. Blakemore, *Video - The Visual Cortex of the Cat*, 1972. [Online]. Available: https://www.youtube.com/watch?v=RSNofraG8ZE (visited on 08/15/2019).

[18] B. Borah and D. K. Bhattacharyya, "An Improved Sampling-Based DBSCAN for Large Spatial Databases", in *Proceedings of International Conference on Intelligent Sensing and Information Processing, ICISIP 2004*, 2004.

[19] A. Borghesi, A. Bartolini, M. Lombardi, *et al.*, "Anomaly Detection Using Autoencoders in High Performance Computing Systems", *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, no. 01, pp. 9428–9433, 2019.

[20] Bosch, *Fourth generation long-range radar sensor*, Abstatt, 2014. [Online]. Available: https://cds.bosch.us/themes/bosch%7B%5C_%7Dcross/amc%7B%5C_%7Dpdfs/LRR4%7B%5C_%7D292000P0ZH%7B%5C_%7DEN%7B%5C_%7Dlow.pdf (visited on 02/13/2019).

[21] Bosch, *Radar-based driver assistance systems: Mid-range radar sensor*, Stuttgart, 2017. [Online]. Available: https://bit.ly/2WOFJD4 (visited on 06/06/2019).

[22] A. Bott, *Synoptische Meteorologie : Methoden der Wetteranalyse und -prognose*, 1st ed. Berlin Heiderlberg: Springer, 2012.

[23] A. Boulch, J. Guerry, B. Le Saux, *et al.*, "SnapNet: 3D point cloud semantic labeling with 2D deep segmentation networks", *Computers & Graphics*, vol. 71, pp. 189–198, Apr. 2018.

[24] R. Boulic, N. M. Thalmann, and D. Thalmann, "A global human walking model with real-time kinematic personification", *The Visual Computer*, 1990.

[25] L. Breiman, "Bagging Predictors", *Machine Learning*, vol. 24, pp. 123–140, 1996.

[26] L. Breiman, "Random Forests", *Machine Learning*, vol. 45, no. 1, pp. 5–32, 2001.

[27] L. Breiman, *Manual On Setting Up, Using, And Understanding Random Forests V3.1*, Berkeley, 2002. [Online]. Available: https://www.stat.berkeley.edu/%7B~%7Dbreiman/Using%7B%5C_%7Drandom%7B%5C_%7Dforests%7B%5C_%7DV3.1.pdf.

[28] D. Brodeski, I. Bilik, and R. Giryes, "Deep Radar Detector", in *2019 IEEE Radar Conference (RadarConf)*, IEEE, Apr. 2019, pp. 1–6.

[29] J. Brownlee, *Gradient Descent For Machine Learning*, 2016. [Online]. Available: https://machinelearningmastery.com/gradient-descent-for-machine-learning/ (visited on 08/12/2019).

[30] J. Bruna, W. Zaremba, A. Szlam, *et al.*, "Spectral Networks and Locally Connected Networks on Graphs", *2nd International Conference on Learning Representations, ICLR 2014 - Conference Track Proceedings*, Dec. 2013.

[31] H. Caesar, V. Bankiti, A. H. Lang, *et al.*, "nuScenes: A multimodal dataset for autonomous driving", *arXiv preprint arXiv:1903.11027*, 2019.

[32] R. J. G. B. Campello, D. Moulavi, and J. Sander, "Density-Based Clustering Based on Hierarchical Density Estimates", in *Advances in Knowledge Discovery and Data Mining*, Springer, Berlin, Heidelberg, 2013, pp. 160–172.

[33] R. Caruana and A. Niculescu-Mizil, "An Empirical Comparison of Supervised Learning Algorithms Using Different Performance Metrics", in *Proceedings of the 23rd international conference on Machine Learning*, 2006.

[34] M.-F. Chang, J. W. Lambert, P. Sangkloy, *et al.*, "Argoverse: 3D Tracking and Forecasting with Rich Maps", in *Conference on Computer Vision and Pattern Recognition (CVPR)*, IEEE, 2019.

[35] N. V. Chawla, K. W. Bowyer, L. O. Hall, *et al.*, "SMOTE: Synthetic minority over-sampling technique", *Journal of Artificial Intelligence Research*, vol. 16, pp. 321–357, 2002.

[36] C. Y. Chen and P. P. Vaidyanathan, "Beamforming issues in modern MIMO radars with Doppler", in *Conference Record - Asilomar Conference on Signals, Systems and Computers*, 2006.

[37] V. Chen, Fayin Li, Shen-Shyang Ho, *et al.*, "Micro-doppler effect in radar: phenomenon, model, and simulation study", *IEEE Transactions on Aerospace and Electronic Systems*, vol. 42, no. 1, pp. 2–21, Jan. 2006.

[38] V. C. Chen, D. Tahmoush, and W. J. Miceli, Eds., *Radar Micro-Doppler Signatures: Processing and Applications*. Institution of Engineering and Technology, May 2014.

[39] K. Cho, B. van Merrienboer, C. Gulcehre, *et al.*, "Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation", in *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, Stroudsburg, PA, USA: Association for Computational Linguistics, 2014, pp. 1724–1734.

[40] F. Chollet, *Trained image classification models for Keras*, 2016. [Online]. Available: `https://github.com/fchollet/deep-learning-models/` (visited on 08/13/2019).

[41] Ö. Çiçek, A. Abdulkadir, S. S. Lienkamp, *et al.*, "3D U-net: Learning dense volumetric segmentation from sparse annotation", in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2016.

[42] Continental, *ARS 300 Long Range Radar Data Sheet*, 2009. (visited on 06/12/2019).

[43] Continental, *ARS 408-21 Premium Radar Datasheet*, 2017. [Online]. Available: `https://www.continental-automotive.com/getattachment/8e4678e1-9358-48e1-8d5b-a0c2de942edb/ARS408-21%7B%5C_%7DDatenblatt%7B%5C_%7Dde%7B%5C_%7D170707%7B%5C_%7DV07.pdf.pdf` (visited on 03/10/2019).

[44] M. Cordts, M. Omran, S. Ramos, *et al.*, "The Cityscapes Dataset for Semantic Urban Scene Understanding", in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, IEEE, Jun. 2016, pp. 3213–3223.

[45] P. Cortez and A. Morais, "A Data Mining Approach to Predict Forest Fires using Meteorological Data", in *Proceedings of 13th Portugese Conference on Artificial Intelligence*, 2007.

[46] T. M. Cover and J. A. Thomas, *Elements of Information Theory*, 2nd ed. Wiley, 2005.

[47] A. Criminisi, "Decision Forests: A Unified Framework for Classification, Regression, Density Estimation, Manifold Learning and Semi-Supervised Learning", *Foundations and Trends in Computer Graphics and Vision*, vol. 7, no. 2-3, pp. 81–227, 2011.

[48] Cycling Sports Group Europe B.V., *GT Bicycles*, 2019. [Online]. Available: `www.gtbicycles.com` (visited on 09/25/2019).

[49] A. D'Ambrosio and V. A. Tutore, "Conditional classification trees by weighting the gini impurity measure", in *Studies in Classification, Data Analysis, and Knowledge Organization*, 2011.

[50] Daimler AG, *So funktioniert DISTRONIC PLUS*, 2015. [Online]. Available: `https://blog.daimler.com/2015/03/18/einfach-technik-so-funktioniert-distronic-plus/` (visited on 05/15/2019).

[51] Daimler AG, *New assistance systems: Active Brake Assist 4 emergency braking assistant featuring pedestrian recognition and sideguard assist*, 2017. [Online]. Available: `https://media.daimler.com/marsMediaSite/en/instance/ko.xhtml?oid=12367326` (visited on 06/06/2019).

[52] A. Danzer, T. Griebel, M. Bach, *et al.*, "2D Car Detection in Radar Data with PointNets", in *2019 IEEE Intelligent Transportation Systems Conference (ITSC)*, IEEE, Oct. 2019, pp. 61–66.

[53] S. K. Dehkordi, N. Appenrodt, J. Dickmann, *et al.*, "Region of interest based adaptive high resolution parameter estimation with applications in automotive radar", in *Proceedings International Radar Symposium*, 2018.

[54] J. Deng, W. Dong, R. Socher, *et al.*, "ImageNet: A large-scale hierarchical image database", in *2009 IEEE Conference on Computer Vision and Pattern Recognition*, IEEE, Jun. 2009, pp. 248–255.

[55] J. Dickmann, J. Klappstein, H.-L. Bloecher, *et al.*, "Automotive Radar — " quo vadis ?"", in *2012 9th European Radar Conference*, IEEE, 2012, pp. 18–21.

[56] J. Dickmann, J. Klappstein, M. Hahn, *et al.*, "Present research activities and future requirements on automotive radar from a car manufacturer's point of view", in *2015 IEEE MTT-S International Conference on Microwaves for Intelligent Mobility, ICMIM 2015*, Heidelberg, Apr. 2015, pp. 15–18.

[57] A. Dominguez, "A History of the Convolution Operation", *IEEE Pulse*, vol. 6, no. 1, pp. 38–49, Jan. 2015.

[58] J. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization", *Journal of Machine Learning Research*, 2011.

[59] C. Duofang, C. Baixiao, and Q. Guodong, "Angle estimation using ESPRIT in MIMO radar", *Electronics Letters*, vol. 44, no. 12, p. 770, 2008.

[60] B. Efron, "Better bootstrap confidence intervals", *Journal of the American Statistical Association*, vol. 82, no. 397, pp. 171–185, Mar. 1987.

[61] C. Eick, N. Zeidat, and Z. Zhao, "Supervised clustering - algorithms and benefits", in *16th IEEE International Conference on Tools with Artificial Intelligence*, IEEE Comput. Soc, 2004, pp. 774–776.

[62] M. B. El Mashade, "Heterogeneous performance evaluation of sophisticated versions of CFAR detection schemes", *Radioelectronics and Communications Systems*, vol. 59, no. 12, pp. 536–551, Dec. 2016.

[63] E. Elhamifar and R. Vidal, "Sparse Subspace Clustering: Algorithm, Theory, and Applications", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 35, no. 11, pp. 2765–2781, Nov. 2013.

[64] C. Elich, F. Engelmann, T. Kontogianni, *et al.*, "3D Bird's-Eye-View Instance Segmentation", in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 11824 LNCS, 2019, pp. 48–61.

[65] M. Ester, H. P. Kriegel, J. Sander, *et al.*, "A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise", in *Proceedings of the 2nd International Conference on Knowledge Discovery and Data Mining*, AAAI Press, Aug. 1996, pp. 226–231.

[66] T. Finley and T. Joachims, "Supervised clustering with support vector machines", in *Proceedings of the 22nd international conference on Machine learning - ICML '05*, New York, New York, USA: ACM Press, 2005, pp. 217–224.

[67] C. Fischer, F. Ruf, H.-L. Bloecher, *et al.*, "Evaluation of different superresolution techniques for automotive applications", in *IET International Conference on Radar Systems (Radar 2012)*, Institution of Engineering and Technology, 2012, pp. 33–33.

[68] Y. Freund and R. E. Schapire, "Experiments with a New Boosting Algorithm", *Proceedings of the 13th International Conference on Machine Learning*, 1996.

[69] J. Friedman, T. Hastie, and R. Tibshirani, "Additive logistic regression: a statistical view of boosting", *The Annals of Statistics*, vol. 28, no. 2, pp. 337–407, Apr. 2000.

[70] Y. Gal and Z. Ghahramani, "A theoretically grounded application of dropout in recurrent neural networks", in *Advances in Neural Information Processing Systems*, 2016.

[71] J. Gehring, M. Auli, D. Grangier, *et al.*, "Convolutional sequence to sequence learning", in *34th International Conference on Machine Learning, ICML 2017*, 2017.

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

[73] S. Geman, E. Bienenstock, and R. Doursat, "Neural Networks and the Bias/Variance Dilemma", *Neural Computation*, vol. 4, no. 1, pp. 1–58, Jan. 1992.

[74] A. Géron, *Hands-on machine learning with Scikit-Learn and TensorFlow : concepts, tools, and techniques to build intelligent systems*, 1st ed. Sebastopol: O'Reilly, 2017.

[75] T. Giese, J. Klappstein, J. Dickmann, *et al.*, "Road course estimation using deep learning on radar data", in *Proceedings International Radar Symposium*, 2017.

[76] R. Girshick, "Fast R-CNN", in *IEEE International Conference on Computer Vision (ICCV)*, 2015.

[77] R. Girshick, J. Donahue, T. Darrell, *et al.*, "Rich feature hierarchies for accurate object detection and semantic segmentation", in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2014.

[78] A. B. Goldberg and X. Zhu, "Introduction to semi-supervised learning", *Synthesis Lectures on Artificial Intelligence and Machine Learning*, 2009.

[79] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*, 1st ed. MIT Press, 2016.

[80] Google LLC, *reCAPTCHA*, 2009. [Online]. Available: `https://www.webcitation.org/6HultOqbu?url=http://recaptcha.net/learnmore.html` (visited on 08/12/2019).

[81] Google LLC; Google Brain, *Tensorflow detection model zoo*, 2019. [Online]. Available: `https://github.com/tensorflow/models/blob/master/research/object%7B%5C_%7Ddetection/g3doc/detection%7B%5C_%7Dmodel%7B%5C_%7Dzoo.md` (visited on 08/13/2019).

[82] A. Graves, "Supervised Sequence Labeling with Recurrent Neural Networks", PhD thesis, Technical University Munich, 2008.

[83] A. Graves, S. Fernández, and J. Schmidhuber, "Bidirectional LSTM networks for improved phoneme classification and recognition", in *Lecture Notes in Computer Science*, Springer, Berlin, Heidelberg, 2005, pp. 799–804.

[84] A. Graves, N. Jaitly, and A.-r. Mohamed, "Hybrid speech recognition with Deep Bidirectional LSTM", in *2013 IEEE Workshop on Automatic Speech Recognition and Understanding*, IEEE, Dec. 2013, pp. 273–278.

[85] K. Greff, R. K. Srivastava, J. Koutnik, *et al.*, "LSTM: A Search Space Odyssey", *IEEE Transactions on Neural Networks and Learning Systems*, vol. 28, no. 10, pp. 2222–2232, Oct. 2017.

[86] P. Guerrero, Y. Kleiman, M. Ovsjanikov, *et al.*, "PCPNet Learning Local Shape Properties from Raw Point Clouds", *Computer Graphics Forum*, vol. 37, no. 2, pp. 75–85, May 2018.

[87] S. Haag, B. Duraisamy, W. Koch, *et al.*, "Classification Assisted Tracking for Autonomous Driving Domain", in *2018 Sensor Data Fusion: Trends, Solutions, Applications (SDF)*, IEEE, Oct. 2018, pp. 1–8.

[88] P. Hamel and D. Eck, "Learning features from music audio with deep belief networks", in *Proceedings of the 11th International Society for Music Information Retrieval Conference, ISMIR 2010*, 2010.

[89] S. H. Al-Harbi and V. J. Rayward-Smith, "Adapting k-means for supervised clustering", *Applied Intelligence*, vol. 24, no. 3, pp. 219–226, Jun. 2006.

[90] D. Harrison and D. L. Rubinfeld, "Hedonic housing prices and the demand for clean air", *Journal of Environmental Economics and Management*, vol. 5, 1978.

[91] J. Hasch, E. Topak, R. Schnabel, *et al.*, "Millimeter-wave technology for automotive radar sensors in the 77 GHz frequency band", *IEEE Transactions on Microwave Theory and Techniques*, vol. 60, no. 3 PART 2, pp. 845–860, 2012.

[92] K. He, G. Gkioxari, P. Dollar, *et al.*, "Mask R-CNN", in *Proceedings of the International Conference on Computer Vision (ICCV)*, 2017.

[93] Y. He, Y. Yang, Y. Lang, *et al.*, "Deep Learning based Human Activity Classification in Radar Micro-Doppler Image", in *2018 15th European Radar Conference (EuRAD)*, IEEE, Sep. 2018, pp. 230–233.

[94] Hermann Hartje KG, *Excelsior Fahrrad*, 2019. [Online]. Available: `https://www.excelsior-fahrrad.de/` (visited on 09/25/2019).

[95] S. Heuel and H. Rohling, "Pedestrian recognition based on 24 GHz radar sensors", in *11th International Radar Symposium (IRS)*, Vilnius: IEEE, 2010, pp. 1–6.

[96] S. Heuel and H. Rohling, "Two-stage pedestrian classification in automotive radar systems", *12th International Radar Symposium (IRS)*, pp. 477–484, 2011.

[97] S. Heuel and H. Rohling, "Pedestrian classification in automotive radar systems", in *13th International Radar Symposium (IRS)*, Warsaw: IEEE, May 2012, pp. 39–44.

[98] S. Heuel and H. Rohling, "Pedestrian recognition in automotive radar sensors", *14th International Radar Symposium (IRS)*, 2013.

[99] G. E. Hinton, N. Srivastava, and K. Swersky, "Neural Networks for Machine Learning Lecture 6a Overview of mini-batch gradient descent.", *COURSERA: Neural Networks for Machine Learning*, 2012.

[100] S. Hochreiter, "Untersuchungen zu dynamischen neuronalen Netzen", *Master's thesis, Institut für Informatik, Technische Universität, Munchen*, 1991.

[101] S. Hochreiter and J. Schmidhuber, "Long Short Term Memory", Technical Report FKI-207-95, München, Tech. Rep., 1995, p. 8.

[102] S. Hochreiter and J. Schmidhuber, "Long Short-Term Memory", *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[103] F. Hofele, "An innovative CFAR algorithm", in *2001 CIE International Conference on Radar Proceedings (Cat No.01TH8559)*, IEEE, 2002, pp. 329–333.

[104] M. Horn, "Motion Classification and Height Estimation of Pedestrians Using Sparse Radar Data", PhD thesis, Ulm University, 2018.

[105] M. Horn, O. Schumann, M. Hahn, *et al.*, "Motion Classification and Height Estimation of Pedestrians Using Sparse Radar Data", in *2018 Sensor Data Fusion: Trends, Solutions, Applications (SDF)*, IEEE, Oct. 2018, pp. 1–6.

[106] D. H. Hubel and T. N. Wiesel, "Receptive fields of single neurones in the cat's striate cortex", *The Journal of Physiology*, vol. 148, no. 3, 1959.

[107] D. H. Hubel and T. N. Wiesel, "Receptive fields, binocular interaction and functional architecture in the cat's visual cortex", *The Journal of Physiology*, vol. 160, no. 1, p. 106, 1962.

[108] D. Hubel and T. Wiesel, *Brain and Visual Perception: The Story of a 25-year Collaboration*. Oxford University Press, Nov. 2004.

[109] E. Hyun and Y.-S. Jin, "Human-vehicle classification scheme using doppler spectrum distribution based on 2D range-doppler FMCW radar", *Journal of Intelligent & Fuzzy Systems*, vol. 35, no. 6, S. O. Hwang, Ed., pp. 6035–6045, Dec. 2018.

[110] L. Ingber, *Adaptive Simulated Annealing (ASA)*, Pasadena, 1993. [Online]. Available: `https://www.ingber.com/%7B%5C#%7DASA-CODE`.

[111] L. Ingber, "Adaptive simulated annealing (ASA): Lessons learned", *Control and Cybernetics*, vol. 25, no. 1, pp. 32–54, 1996.

[112] V. T. Inman, H. J. Ralsto, and F. Todd, *Human Walking*, 1st ed. Michigan: Williams & Wilkin, 1981.

[113] H. Iqbal, M. Schartel, F. Roos, *et al.*, "Implementation of a SAR Demonstrator for Automotive Imaging", in *2018 18th Mediterranean Microwave Symposium (MMS)*, IEEE, Oct. 2018, pp. 240–243.

[114] ISO, "ISO 8855 - Road Vehicles, Vehicle dynamic and road-holding ability", International Organization for Standardization, Geneva, Tech. Rep., 2011, p. 11. [Online]. Available: `https://www.sis.se/api/document/preview/914200/`.

[115] W. B. Johnson and J. Lindenstrauss, "Extensions of Lipschitz mappings into a Hilbert space", in *Conference on Modern Analysis and Probability*, R. Beals, A. Beck, A. Bellow, *et al.*, Eds., 1984, pp. 189–206.

[116] Kaggle Competition, *TrackML Particle Tracking Challenge - Kaggle*, 2018. [Online]. Available: `https://www.kaggle.com/c/trackml-particle-identification` (visited on 08/12/2019).

[117] Y. Kang, H. Yin, and C. Berger, "Test Your Self-Driving Algorithm: An Overview of Publicly Available Driving Datasets and Virtual Testing Environments", *IEEE Transactions on Intelligent Vehicles*, vol. 4, no. 2, pp. 171–185, Jun. 2019.

[118] C. Karnfelt, A. Peden, A. Bazzi, *et al.*, "77 GHz ACC radar simulation platform", in *2009 9th International Conference on Intelligent Transport Systems Telecommunications, (ITST)*, IEEE, Oct. 2009, pp. 209–214.

[119] D. Katz, "State of Stealth", *Aviation Week & Space Technology*, no. Special Issue, 2017.

[120] D. Kellner, "Verfahren zur Bestimmung von Objekt- und Eigenbewegung auf Basis der Dopplerinformation hochauflösender Radarsensoren", PhD thesis, Ulm University, 2017.

[121] D. Kellner, J. Klappstein, and K. Dietmayer, "Grid-based DBSCAN for clustering extended objects in radar data", in *IEEE Intelligent Vehicles Symposium, Proceedings*, Alcala de Henares, 2012, pp. 365–370.

[122] R. Kesten, M. Usman, J. Houston, *et al.*, *Lyft Level 5 AV Dataset 2019*, 2019. [Online]. Available: `https://level5.lyft.com/dataset/` (visited on 08/10/2019).

[123] D. P. Kingma and J. L. Ba, "Adam: a Method for Stochastic Optimization", in *International Conference on Learning Representations 2015*, May 2015, pp. 1–15.

[124] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, "Optimization by simulated annealing", *Science*, 1983.

[125] R. Klokov and V. Lempitsky, "Escape from Cells: Deep Kd-Networks for the Recognition of 3D Point Cloud Models", in *2017 IEEE International Conference on Computer Vision (ICCV)*, vol. 2017-Octob, IEEE, Oct. 2017, pp. 863–872.

[126] A. Komjathy, "Ionospheric Effects on the Propagation of Electromagnetic Waves", in *Encyclopedia of Remote Sensing*, N. E.G, Ed., New York, 2014, pp. 286–291.

[127] H.-P. Kriegel, P. Kröger, E. Schubert, *et al.*, "Outlier Detection in Axis-Parallel Subspaces of High Dimensional Data", in *Advances in Knowledge Discovery and Data Mining. PAKDD 2009. Lecture Notes in Computer Science*, T. T., K. B., C. N., *et al.*, Eds., Berlin Heidelberg: Springer, 2009, pp. 831–838.

[128] A. Krizhevsky, "Learning Multiple Layers of Features from Tiny Images", Tech. Rep., 2009. [Online]. Available: `https://www.cs.toronto.edu/%7B~%7Dkriz/cifar.html`.

[129] J. Kwon, S. Lee, and N. Kwak, "Human Detection by Deep Neural Networks Recognizing Micro-Doppler Signals of Radar", in *2018 15th European Radar Conference (EuRAD)*, IEEE, Sep. 2018, pp. 198–201.

[130] L. Landrieu and M. Simonovsky, "Large-Scale Point Cloud Semantic Segmentation with Superpoint Graphs", in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2018.

[131] A. H. Lang, S. Vora, H. Caesar, *et al.*, "Pointpillars: Fast encoders for object detection from point clouds", in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2019.

[132] A. Laribi, M. Hahn, J. Dickmann, *et al.*, "Performance Investigation of Automotive SAR Imaging", in *2018 IEEE MTT-S International Conference on Microwaves for Intelligent Mobility (ICMIM)*, IEEE, Apr. 2018, pp. 1–4.

[133] Y. LeCun, B. Boser, J. S. Denker, *et al.*, "Backpropagation Applied to Handwritten Zip Code Recognition", *Neural Computation*, vol. 1, no. 4, pp. 541–551, Dec. 1989.

[134] K. Lee, H. Cheng, W. Jou, *et al.*, "The influence of carbon fiber orientation on the mechanical and tribological behavior of carbon fiber/LCP composites", *Materials Chemistry and Physics*, vol. 102, no. 2-3, pp. 187–194, Apr. 2007.

[135] S. Lee, Y.-J. Yoon, J.-E. Lee, *et al.*, "Human-vehicle classification using feature-based SVM in 77-GHz automotive FMCW radar", *IET Radar, Sonar & Navigation*, vol. 11, no. 10, pp. 1589–1596, Oct. 2017.

[136] Li Mu, Tong Xiangqian, Shen Ming, *et al.*, "Research on key tchnologies for collision avoidance automotive radar", in *2009 IEEE Intelligent Vehicles Symposium*, IEEE, Jun. 2009, pp. 233–236.

[137] Z. Li and X. Wang, "High Resolution Radar Data Fusion Based on Clustering Algorithm", in *2010 2nd International Workshop on Database Technology and Applications*, 2010, pp. 1–4.

[138] J. Liang, N. Homayounfar, W.-C. Ma, *et al.*, "PolyTransform: Deep Polygon Transformer for Instance Segmentation", *arXiv preprint*, 2019.

[139] X. Liang and Z. Fu, "MHNet: Multiscale Hierarchical Network for 3D Point Cloud Semantic Segmentation", *IEEE Access*, vol. 7, pp. 173 999–174 012, 2019.

[140] S. Lim, S. Lee, and S. C. Kim, "Clustering of detected targets using DBSCAN in automotive radar systems", in *Proceedings International Radar Symposium*, vol. 2018-June, 2018.

[141] T.-Y. Lin, M. Maire, S. Belongie, *et al.*, "Microsoft COCO: Common Objects in Context", in *Computer Vision – ECCV 2014. Lecture Notes in Computer Science*, Springer, Cham, 2014, pp. 740–755.

[142] W. Liu, D. Anguelov, D. Erhan, *et al.*, "SSD: Single Shot MultiBox Detector", in *Lecture Notes in Computer Science*, vol. 9905 LNCS, 2016, pp. 21–37.

[143] X. Liu, Z. Deng, and Y. Yang, "Recent progress in semantic image segmentation", *Artificial Intelligence Review*, 2019.

[144] J. Lombacher, M. Hahn, J. J. Dickmann, *et al.*, "Object classification in radar using ensemble methods", in *International Conference on Microwaves for Intelligent Mobility (ICMIM)*, H. Okazaki, Ed., Nagoya, Japan: IEEE, Mar. 2017, pp. 87–90.

[145] J. Lombacher, M. Hahn, J. Dickmann, *et al.*, "Detection of arbitrarily rotated parked cars based on radar sensors", in *16th International Radar Symposium (IRS)*, IEEE, Jun. 2015, pp. 180–185.

[146] J. Lombacher, M. Hahn, J. Dickmann, *et al.*, "Potential of radar for static object classification using deep learning methods", in *2016 IEEE MTT-S International Conference on Microwaves for Intelligent Mobility (ICMIM)*, IEEE, May 2016, pp. 1–4.

[147] J. Lombacher, K. Laudt, M. Hahn, *et al.*, "Semantic radar grids", in *2017 IEEE Intelligent Vehicles Symposium (IV)*, IEEE, Jun. 2017, pp. 1170–1175.

[148] J. Long, E. Shelhamer, and T. Darrell, "Fully Convolutional Networks for Semantic Segmentation", in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun. 2015.

[149] G. Louppe, "Understanding Random Forests: From Theory to Practice", PhD thesis, University of Liege, 2014.

[150] M. Emre Celebi (ed), *Partitional Clustering Algorithms*, 1st ed. Heidelberg: Springer International Publishing, 2015.

[151] M. Emre Celebi (ed), *Unsupervised Learning Algorithms*, 1st ed. Springer International Publishing, 2015.

[152] J. Mairal, F. Bach, J. Ponce, *et al.*, "Online dictionary learning for sparse coding", in *ACM International Conference Proceeding Series*, 2009.

[153] M. Mann, "Benutzerorientierte Entwicklung und fahrergerechte Auslegung eines Querführungsassistenten", PhD thesis, 2008.

[154] J. Martinez, D. Kopyto, M. Schutz, *et al.*, "Convolutional Neural Network Assisted Detection and Localization of UAVs with a Narrowband Multi-site Radar", in *2018 IEEE MTT-S International Conference on Microwaves for Intelligent Mobility (ICMIM)*, IEEE, Apr. 2018, pp. 1–4.

[155] W. Mayer, "Abbildender Radarsensor mit sendeseitig geschalteter Gruppenantenne", PhD thesis, Ulm University, Ulm, 2008, p. 196.

[156] H. H. Meinel, "Millimeterwaves for automotive applications", in *26th European Microwave Conference, EuMC 1996*, vol. 2, 1996, pp. 830–835.

[157] H. H. Meinel, "Automotive Millimeterwave Radar: History and Present Status", in *28th European Microwave Conference Amsterdam*, Amsterdam, 1998, pp. 619–629.

[158] H. H. Meinel, "Evolving automotive radar - From the very beginnings into the future", *8th European Conference on Antennas and Propagation, EuCAP 2014*, no. EuCAP, pp. 3107–3114, 2014.

[159] S. Melacci, M. Maggini, and L. Sarti, "Semi-supervised clustering using similarity neural networks", in *2009 International Joint Conference on Neural Networks*, IEEE, Jun. 2009, pp. 2065–2072.

[160] A. Metallinou, *Amazon Scientists Use Transfer Learning to Accelerate Development of New Alexa Capabilities*, 2018. [Online]. Available: `https://developer.amazon.com/de/blogs/alexa/post/b6187c9a-2faa-4c17-a90c-142ab0e48b7e/amazon-scientists-use-transfer-learning-to-accelerate-development-of-new-alexa-capabilities` (visited on 08/15/2019).

[161] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, *et al.*, "Equation of state calculations by fast computing machines", *The Journal of Chemical Physics*, 1953.

[162] M. Meyer and G. Kuschk, "Automotive radar dataset for deep learning based 3D object detection", in *EuRAD 2019 - 2019 16th European Radar Conference*, IEEE, 2019.

[163] M. Meyer and G. Kuschk, "Deep learning based 3D object detection for automotive radar and camera", in *EuRAD 2019 - 2019 16th European Radar Conference*, IEEE, 2019.

[164] L. Miller, *Machine Learning week 1: Cost Function, Gradient Descent and Univariate Linear Regression*, 2018. [Online]. Available: `https://link.medium.com/LgHTbpfj64` (visited on 08/12/2019).

[165] T. Mitchell, *Machine Learning*, 1st ed., C. Liu, Ed. Boston: McGraw-Hill, 1997.

[166] P. Molchanov, J. Astola, K. Egiazarian, *et al.*, "On micro-doppler period estimation", in *Proceedings - 19th International Conference on Control Systems and Computer Science, CSCS 2013*, 2013.

[167] P. Molchanov and A. Vinel, "Radar frequency band invariant pedestrian classification", *Radar Symposium (IRS ...*, pp. 1–6, 2013.

[168] H. Moravec, *Mind Children*, 1st ed. Harvard University Press, 1988.

[169] M. C. Mozer, "A Focused Backpropagation Algorithm for Temporal Pattern Recognition", *Complex Systems*, vol. 3, no. 4, pp. 349–381, 1995.

[170] R. Munroe, *Tasks - xkcd*, 2014. [Online]. Available: `https://xkcd.com/1425/` (visited on 08/12/2019).

[171] V. Nair and G. E. Hinton, "Rectified linear units improve Restricted Boltzmann machines", in *ICML 2010 - Proceedings, 27th International Conference on Machine Learning*, 2010.

[172] S. Nawar and A. M. Mouazen, "Comparison between random forests, artificial neural networks and gradient boosted machines methods of on-line Vis-NIR spectroscopy measurements of soil total nitrogen and total carbon", *Sensors (Switzerland)*, 2017.

[173] V. Nordenmark, "Radar-detection based classification of moving objects using machine learning methods", PhD thesis, KTH Industrial Engineering and Management, 2015, p. 112.

[174] J. O. Ogutu, H. P. Piepho, and T. Schulz-Streeck, "A comparison of random forests, boosting and support vector machines for genomic selection", in *BMC Proceedings*, 2011.

[175] C. Olah, *Understanding LSTM Networks*, 2015. [Online]. Available: `https://colah.github.io/posts/2015-08-Understanding-LSTMs/` (visited on 08/16/2019).

[176] M. Özcan, S. Z. Gürbüz, A. R. Persico, *et al.*, "Performance Analysis of Co-Located and Distributed MIMO Radar for Micro-Doppler Classification", in *EuRAD 2016*, 2016, pp. 85–88.

[177] A. Palffy, J. Dong, J. F. P. Kooij, *et al.*, "CNN Based Road User Detection Using the 3D Radar Cube", *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 1263–1270, Apr. 2020.

[178] Y. Pang, Y. Li, J. Wang, *et al.*, "Carbon fiber assisted glass fabric composite materials for broadband radar cross section reduction", *Composites Science and Technology*, vol. 158, pp. 19–25, Apr. 2018.

[179] T. Parr, K. Turgutlu, C. Csiszar, *et al.*, *Beware Default Random Forest Importances*, 2008. [Online]. Available: `https://explained.ai/rf-importance/index.html` (visited on 08/15/2019).

[180] R. Pascanu, T. Mikolov, and Y. Bengio, "On the difficulty of training recurrent neural networks", in *30th International Conference on Machine Learning, ICML 2013*, Atlanta: Journal of Machine Learning Research, 2013.

[181] K. Patel, "Deep Learning-based Object Classification on Automotive Radar Spectra", in *Radar Conf 2019*, 2019.

[182] S. M. Patole, M. Torlak, D. Wang, *et al.*, "Automotive radars: A review of signal processing techniques", *IEEE Signal Processing Magazine*, vol. 34, no. 2, pp. 22–35, Mar. 2017.

[183] K. Pearson, "LIII. On lines and planes of closest fit to systems of points in space", *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, vol. 2, no. 11, pp. 559–572, Nov. 1901.

[184] Pending System GmbH & Co. KG, *Cube Bikes*, 2019. [Online]. Available: https://www.cube.eu/cube-bikes/ (visited on 09/25/2019).

[185] R. Pérez, F. Schubert, R. Rasshofer, *et al.*, "A machine learning joint lidar and radar classification system in urban automotive scenarios", *Advances in Radio Science*, 2019.

[186] T. Peters, "Ableitung einer Beziehung zwischen der Radarreflektivität, der Niederschlagsrate und weiteren aus Radardaten abgeleiteten Parametern unter Verwendung von Methoden der multivariaten Statistik", PhD thesis, 2012.

[187] R. Photonics, *Instantaneous Frequency*. [Online]. Available: https://www.rp-photonics.com/instantaneous%7B%5C_%7Dfrequency.html (visited on 06/11/2019).

[188] F. Piewak, P. Pinggera, M. Schäfer, *et al.*, "Boosting LiDAR-Based Semantic Labeling by Cross-modal Training Data Generation", in, Springer, Cham, Sep. 2019, pp. 497–513.

[189] Porsche, *Intelligente Unterstützung für den Fahrer*, 2015. [Online]. Available: https://presse.porsche.de/presskits%7B%5C_%7Duntil%7B%5C_%7D2015/products/2014/macan/html/de%7B%5C_%7D23544%7B%5C_%7D0.html (visited on 06/06/2019).

[190] P. Probst and A.-L. Boulesteix, "To tune or not to tune the number of trees in random forest", *The Journal of Machine Learning Research*, vol. 18, no. 1, pp. 6673–6690, 2017.

[191] P. Probst, M. N. Wright, and A. L. Boulesteix, *Hyperparameters and tuning strategies for random forest*, 2019.

[192] R. Prophet, M. Hoffmann, A. Ossowska, *et al.*, "Pedestrian Classification for 79 GHz Automotive Radar Systems", in *2018 IEEE Intelligent Vehicles Symposium (IV)*, vol. 2018-June, IEEE, Jun. 2018, pp. 1265–1270.

[193] R. Prophet, M. Hoffmann, M. Vossiek, *et al.*, "Pedestrian Classification with a 79 GHz Automotive Radar Sensor", in *2018 19th International Radar Symposium (IRS)*, IEEE, Jun. 2018, pp. 1–6.

[194] R. Prophet, G. Li, C. Sturm, *et al.*, "Semantic Segmentation on Automotive Radar Maps", in *2019 IEEE Intelligent Vehicles Symposium (IV)*, vol. 2019-June, IEEE, Jun. 2019, pp. 756–763.

[195] PyTorch, *Torchvision.Models*, 2019. [Online]. Available: https://pytorch.org/docs/stable/torchvision/models.html%7B%5C#%7Dsemantic-segmentation (visited on 08/13/2019).

[196] C. R. Qi, W. Liu, C. Wu, *et al.*, "Frustum PointNets for 3D Object Detection from RGB-D Data", in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, IEEE, Jun. 2018, pp. 918–927.

[197] C. R. Qi, L. Yi, H. Su, *et al.*, *PointNet++, Tensorflow Implementation*, 2017. [Online]. Available: https://github.com/charlesq34/pointnet2 (visited on 02/10/2019).

[198] C. R. Qi, L. Yi, H. Su, *et al.*, "PointNet++: Deep Hierarchical Feature Learning on Point Sets in a Metric Space", *arXiv preprint*, pp. 1–14, Jun. 2017.

[199] C. R. Qi, H. Su, K. Mo, *et al.*, "PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation", in *Proc. Computer Vision and Pattern Recognition (CVPR), IEEE*, Dec. 2017, pp. 77–85.

[200] S. Rao, "MIMO Radar", Texas Instruments, Tech. Rep., 2017, p. 13. [Online]. Available: http://www.ti.com/lit/an/swra554a/swra554a.pdf.

[201] S. Raschka, *Python Machine Learning*, 1st ed. Packt Publishing, 2015.

[202] S. Raschka, *Python Machine Learning Book - FAQ*, 2015. [Online]. Available: https://github.com/rasbt/python-machine-learning-book/blob/master/faq/decision-tree-binary.md (visited on 08/14/2019).

[203] J. Redmon, S. Divvala, R. Girshick, *et al.*, "You Only Look Once: Unified, Real-Time Object Detection", in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, vol. 2016-Decem, IEEE, Jun. 2016, pp. 779–788.

[204] S. Ren, K. He, R. Girshick, *et al.*, "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks", *Neural Information Processing Systems (NIPS)*, 2015.

[205] D. Rethage, J. Wald, J. Sturm, *et al.*, "Fully-Convolutional Point Networks for Large-Scale Point Clouds", in *Lecture Notes in Computer Science*, vol. 11208 LNCS, Springer Verlag, 2018, pp. 625–640.

[206] M. A. Richards, *Fundamentals of Radar Signal Processing*. McGraw-Hill, 2005.

[207] E. J. Riley, E. H. Lenzing, and R. M. Narayanan, "Characterization of radar cross section of carbon fiber composite materials", in *SPIE 9461, Radar Sensor Technology XIX; and Active and Passive Signatures VI*, K. I. Ranney, A. Doerry, G. C. Gilbreath, *et al.*, Eds., vol. 9461, International Society for Optics and Photonics, May 2015.

[208] L. M. Rios and N. V. Sahinidis, "Derivative-free optimization: A review of algorithms and comparison of software implementations", in *Journal of Global Optimization*, 2013.

[209] A. J. Robinson and F. Fallside, "The utility driven dynamic error propagation network", Cambridge University, Engineering Department, Cambridge, Tech. Rep., 1987.

[210] H. Rohling, "Radar CFAR Thresholding in Clutter and Multiple Target Situations", *IEEE Transactions on Aerospace and Electronic Systems*, vol. AES-19, no. 4, pp. 608–621, Jul. 1983.

[211] H. Rohling, "Ordered Statistic CFAR Technique - an Overview", *Processing*, 2011.

[212] A. Rosenberg and J. Hirschberg, "V-Measure: A conditional entropy-based external cluster evaluation measure", in *EMNLP-CoNLL 2007 - Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, 2007.

[213] D. Rousseau, S. Amrouche, P. Calafiura, *et al.*, *The TrackML Particle Tracking Challenge*, 2018. [Online]. Available: `https://hal.inria.fr/hal-01680537` (visited on 08/12/2019).

[214] R. Roy and T. Kailath, "ESPRIT-Estimation of signal parameters via rotational invariance techniques", in *Adaptive Antennas for Wireless Communications*, 1989.

[215] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors", *Nature*, vol. 323, no. 6088, pp. 533–536, Oct. 1986.

[216] O. Russakovsky, J. Deng, H. Su, *et al.*, "ImageNet Large Scale Visual Recognition Challenge", *International Journal of Computer Vision (IJCV)*, vol. 115, no. 3, pp. 221–252, 2015.

[217] I.-h. Ryu, I. Won, and J. Kwon, "Detecting Ghost Targets Using Multilayer Perceptron in Multiple-Target Tracking", *Symmetry*, vol. 10, no. 1, p. 16, Jan. 2018.

[218] H. Sahbi, "A particular Gaussian mixture model for clustering and its application to image retrieval", *Soft Computing*, vol. 12, no. 7, pp. 667–676, May 2008.

[219] J. Sander, M. Ester, H. P. Kriegel, *et al.*, "Density-based clustering in spatial databases: The algorithm GDBSCAN and its applications", *Data Mining and Knowledge Discovery*, 1998.

[220] F. Sarholz, J. Mehnert, J. Klappstein, *et al.*, "Evaluation of different approaches for road course estimation using imaging radar", in *IEEE International Conference on Intelligent Robots and Systems*, 2011.

[221] P. Saville, "Review of Radar Absorbing Materials", *Defence Research and Development Canada*, 2005.

[222] N. Scheiner, N. Appenrodt, J. J. Dickmann, *et al.*, "Radar-based Feature Design and Multiclass Classification for Road User Recognition", in *2018 IEEE Intelligent Vehicles Symposium (IV)*, IEEE, Jun. 2018, pp. 779–786.

[223] N. Scheiner, N. Appenrodt, J. Dickmann, *et al.*, "A Multi-Stage Clustering Framework for Automotive Radar Data", in *IEEE 22nd Intelligent Transportation Systems Conference (ITSC)*, Acukland: IEEE, Oct. 2019.

[224] N. Scheiner, N. Appenrodt, J. Dickmann, *et al.*, "Radar-based Road User Classification and Novelty Detection with Recurrent Neural Network Ensembles", May 2019.

[225] N. Scheiner, S. Haag, N. Appenrodt, *et al.*, "Automated Ground Truth Estimation for Automotive Radar Tracking Applications With Portable GNSS And IMU Devices", in *2019 20th International Radar Symposium (IRS)*, IEEE, Jun. 2019, pp. 1–10.

[226] N. Scheiner and O. Schumann, "Machine Learning Concepts For Multiclass Classification Of Vulnerable Road Users", in *16th European Radar Conference (EuRAD 2019) Workshops*, Paris, 2019.

[227] J. Schmidhuber, "Deep learning in neural networks: An overview", *Neural Networks*, vol. 61, pp. 85–117, Jan. 2015.

[228] R. Schmidt, "Multiple emitter location and signal parameter estimation", *IEEE Transactions on Antennas and Propagation*, vol. 34, no. 3, pp. 276–280, Mar. 1986.

[229] R. Schneider, H.-L. Blocher, and K. M. Strohm, "KOKON - Automotive high frequency technology at 77/79 GHz", in *2007 European Radar Conference*, IEEE, Oct. 2007, pp. 247–250.

[230] E. Schubert, F. Meinl, M. Kunert, *et al.*, "Clustering of high resolution automotive radar detections and subsequent feature extraction for classification of road users", in *2015 16th International Radar Symposium (IRS)*, IEEE, Jun. 2015, pp. 174–179.

[231] K. Schuler, M. Younis, R. Lenz, *et al.*, "Array design for automotive digital beamforming radar system", in *IEEE International Radar Conference, 2005.*, IEEE, 2005, pp. 435–440.

[232] O. Schumann, M. Hahn, J. Dickmann, *et al.*, "Comparison of random forest and long short-term memory network performances in classification tasks using radar", in *Sensor Data Fusion: Trends, Solutions, Applications (SDF)*, IEEE, Oct. 2017, pp. 1–6.

[233] O. Schumann, M. Hahn, J. Dickmann, *et al.*, "Semantic Segmentation on Radar Point Clouds", in *2018 21st International Conference on Information Fusion (FUSION)*, IEEE, Jul. 2018, pp. 2179–2186.

[234] O. Schumann, M. Hahn, J. Dickmann, *et al.*, "Supervised Clustering for Radar Applications: On the Way to Radar Instance Segmentation", in *2018 IEEE MTT-S International Conference on Microwaves for Intelligent Mobility, ICMIM 2018*, IEEE, Apr. 2018, pp. 1–4.

[235] O. Schumann, J. Lombacher, M. Hahn, *et al.*, "Scene Understanding with Automotive Radar", *IEEE Transactions on Intelligent Vehicles*, vol. 5, no. 2, 2020.

[236] M. Schuster and K. Paliwal, "Bidirectional recurrent neural networks", in *IEEE Transactions on Signal Processing*, vol. 45, 1997, pp. 2673–2681.

[237] M. Seul, L. O'Gorman, and M. Sammon, *Practical Algorithms for Image Analysis*, 1st ed. Cambridge: Cambridge University Press, 2000.

[238] C. E. Shannon, "Communication in the Presence of Noise", *Proceedings of the IRE*, 1949.

[239] X. Shi, Z. Chen, H. Wang, *et al.*, "Convolutional LSTM Network: A Machine Learning Approach for Precipitation Nowcasting", in *Advances in Neural Information Processing Systems 28*, C. Cortes, N. D. Lawrence, D. D. Lee, *et al.*, Eds., Curran Associates, Inc., 2015, pp. 802–810.

[240] R. Sibson, "SLINK: An optimally efficient algorithm for the single-link cluster method", *The Computer Journal*, vol. 16, no. 1, pp. 30–34, Jan. 1973.

[241] S. Sileymanov, *Design and Implementation of an FMCW Radar Signal Processing Module for Automotive Applications*, Twente, 2016. [Online]. Available: `https://essay.utwente.nl/70986/1/Suleymanov%7B%5C_%7DMA%7B%5C_%7DEWI.pdf`.

[242] M. I. Skolnik, *Introduction to Radar Systems*, 3rd ed. McGraw-Hill, 2001.

[243] L. Sless, G. Cohen, B. E. Shlomo, *et al.*, "Self-Supervised Occupancy Grid Learning From Sparse Radar For Autonomous Driving", *arXiv preprint*, Mar. 2019.

[244] C. Solomon and T. Breckon, *Fundamentals of Digital Image Processing*. Chichester, UK: John Wiley & Sons, Ltd, Dec. 2010.

[245] M. Sonka, V. Hlavac, and R. Boyle, *Image Processing, Anaylsis, and Machine Vision*, 3rd ed. Thomson, 2007.

[246] N. Srivastava, G. Hinton, A. Krizhevsky, *et al.*, "Dropout: A simple way to prevent neural networks from overfitting", *Journal of Machine Learning Research*, 2014.

[247] M. Stolz, M. Li, Z. Feng, *et al.*, "High resolution automotive radar data clustering with novel cluster method", in *2018 IEEE Radar Conference, RadarConf 2018*, 2018.

[248] R. Storn and K. Price, "Differential Evolution - A Simple and Efficient Heuristic for Global Optimization over Continuous Spaces", *Journal of Global Optimization*, 1997.

[249] C. Strobl, A.-L. Boulesteix, T. Kneib, *et al.*, "Conditional variable importance for random forests", *BMC Bioinformatics*, vol. 9, no. 1, p. 307, Dec. 2008.

[250] C. Strobl, A.-L. Boulesteix, A. Zeileis, *et al.*, "Bias in random forest variable importance measures: Illustrations, sources and a solution", *BMC Bioinformatics*, vol. 8, no. 1, p. 25, Dec. 2007.

[251] S. Suryansh, *Gradient Descent: All You Need to Know*, 2018. [Online]. Available: `https://hackernoon.com/gradient-descent-aynk-7cbe95a778da` (visited on 08/12/2019).

[252] C. Szegedy, Wei Liu, Yangqing Jia, *et al.*, "Going Deeper with Convolutions", in *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, vol. abs/1409.4, IEEE, Jun. 2015, pp. 1–9.

[253] D. Tahmoush, "Review of micro-Doppler signatures", *IET Radar, Sonar & Navigation*, vol. 9, no. 9, pp. 1140–1146, Dec. 2015.

[254] F. Tony Liu, K. Ming Ting, Z.-H. Zhou, *et al.*, "Isolation Forest", in *2008 Eighth IEEE International Conference on Data Mining*, IEEE, Dec. 2008, pp. 413–422.

[255] TU Delft et al., *Private discussions with Professors from different Universities*, 2018.

[256] J. Uhrig, M. Cordts, U. Franke, *et al.*, "Pixel-Level Encoding and Depth Layering for Instance-Level Semantic Labeling", in *Pattern Recognition*, B. Rosenhahn and B. Andres, Eds., Springer International Publishing, Apr. 2016, pp. 14–25.

[257] L. J. P. Van Der Maaten and G. E. Hinton, "Visualizing high-dimensional data using t-sne", *Journal of Machine Learning Research*, vol. 9, pp. 2579–2605, Nov. 2008.

[258] A. Vaswani, N. Shazeer, N. Parmar, *et al.*, "Attention is all you need", in *Advances in Neural Information Processing Systems 30*, I. Garnett, Guyon, U. V. Luxburg, *et al.*, Eds., Curran Associates, Inc., 2017.

[259] Volkswagen, *Automatische Distanzregelung (Active Cruise Control - ACC)*, 2018. [Online]. Available: `https://www.volkswagen-newsroom.com/de/automatische-distanzregelung-active-cruise-control-acc-3664` (visited on 06/06/2019).

[260] T. Wagner, R. Feger, and A. Stelzer, "Modifications of the OPTICS Clustering Algorithm for Short-Range Radar Tracking Applications", in *2018 15th European Radar Conference, EuRAD 2018*, 2018.

[261] A. Waibel, T. Hanazawa, G. Hinton, *et al.*, "Phoneme recognition using time-delay neural networks", *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 37, no. 3, pp. 328–339, Mar. 1989.

[262] J. Wainer, "Comparison of 14 different families of classification algorithms on 115 binary datasets", *CoRR*, vol. abs/1606.0, 2016.

[263] C. Waldschmidt and H. Meinel, "Future trends and directions in radar concerning the application for autonomous driving", in *EuMW 2014; EuRAD 2014: 11th European Radar Conference*, European Microwave Association - EuMA, Oct. 2014, pp. 416–419.

[264] C. Wang, B. Samari, and K. Siddiqi, "Local Spectral Graph Convolution for Point Set Feature Learning", in *Lecture Notes in Computer Science*, vol. 11208 LNCS, 2018, pp. 56–71.

[265] H. Wang, C. Yuan, W. Hu, *et al.*, "Supervised class-specific dictionary learning for sparse modeling in action recognition", *Pattern Recognition*, 2012.

[266] L. Wang, Y. Huang, J. Shan, *et al.*, "MSNet: Multi-Scale Convolutional Network for Point Cloud Classification", *Remote Sensing*, vol. 10, no. 4, p. 612, Apr. 2018.

[267] Y. Wang, Y. Sun, Z. Liu, *et al.*, "Dynamic graph Cnn for learning on point clouds", *ACM Transactions on Graphics*, 2019.

[268] Waymo, *Waymo Open Dataset*, 2019. [Online]. Available: `https://waymo.com/open/` (visited on 08/10/2019).

[269] P. J. Werbos, "Generalization of backpropagation with application to a recurrent gas market model", *Neural Networks*, 1988.

[270] V. Winkler, "Range Doppler detection for automotive FMCW radars", in *2007 European Radar Conference, EURAD*, 2007.

[271] H. Winner, S. Hakuli, and G. Wolf, *Handbuch Fahrerassistenzsysteme*, 1st ed. Wiesbaden: Vieweg+Teubner, 2009.

[272] T. Winterling, J. Lombacher, M. Hahn, *et al.*, "Optimizing labelling on radar-based grid maps using active learning", in *Proceedings International Radar Symposium*, 2017.

[273] C. Wohler and J. Anlauf, "An adaptable time-delay neural-network algorithm for image sequence analysis", *IEEE Transactions on Neural Networks*, vol. 10, no. 6, pp. 1531–1536, 1999.

[274] C. Wöhler and J. Anlauf, "Real-time object recognition on image sequences with the adaptable time delay neural network algorithm - applications for autonomous vehicles", *Image and Vision Computing*, vol. 19, no. 9-10, pp. 593–618, Aug. 2001.

[275] C. Wöhler, J. Schürmann, and J. K. Anlauf, "Segmentation-Free Detection of Overtaking Vehicles with a Two-Stage Time Delay Neural Network Classifier", in *European Symposium on Artificial Neural Networks*, 1999, pp. 301–306.

[276] C. Wöhler and J. K. Anlauf, "A time delay neural network algorithm for estimating image-pattern shape and motion", *Image and Vision Computing*, vol. 17, pp. 281–294, 1999.

[277] C. Wöhler, J. K. Anlauf, T. Pörtner, *et al.*, "A Time Delay Neural Network Algorithm for Real-Time Pedestrian Recognition", in *IEEE Int. Conf. on Intelligent Vehicles*, Stuttgart, 1998.

[278] D. Wolf, *Code-moduliertes Radar für Automobilanwendungen*, Ulm, 2017.

[279] World Intellectual Property Organization, *WIPO Trademarks - Random Forests*, 2002. [Online]. Available: `https://www.wipo.int/branddb/en/showData.jsp?ID=EMTM.017891729` (visited on 08/14/2019).

[280] Y. Wu, M. Schuster, Z. Chen, *et al.*, "Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation", *CoRR*, Sep. 2016.

[281] M. Xu, W. Dai, Y. Shen, *et al.*, "MSGCNN: Multi-scale Graph Convolutional Neural Network for Point Cloud Segmentation", in *2019 IEEE Fifth International Conference on Multimedia Big Data (BigMM)*, IEEE, Sep. 2019, pp. 118–127.

[282] X. Xu, M. Ester, H. P. Kriegel, *et al.*, "A Distribution-based Clustering Algorithm for Mining in Large Spatial Databases", in *Proceedings - International Conference on Data Engineering*, 1998.

[283] Y. Xu, T. Fan, M. Xu, *et al.*, "SpiderCNN: Deep Learning on Point Sets with Parameterized Convolutional Filters", in *Lecture Notes in Computer Science*, vol. 11212 LNCS, Springer Verlag, 2018, pp. 90–105.

[284] Z. Xu and Q. Shi, "Interference mitigation for automotive radar using orthogonal noise waveforms", *IEEE Geoscience and Remote Sensing Letters*, 2018.

[285] M. Yamada and K. Hasuike, "Document image processing based on enhanced border following algorithm", in *Proceedings - International Conference on Pattern Recognition*, 1990.

[286] N. Yamada, Y. Tanaka, and K. Nishikawa, "Radar cross section for pedestrian in 76GHz band", in *2005 European Microwave Conference*, IEEE, 2005, 4 pp.–1018.

[287] R.-M. Yang, G.-L. Zhang, F. Liu, *et al.*, "Comparison of boosted regression tree and random forest models for mapping topsoil organic carbon concentration in an alpine ecosystem", *Ecological Indicators*, vol. 60, pp. 870–878, Jan. 2016.

[288] H. Zen and H. Sak, "Unidirectional long short-term memory recurrent neural network with recurrent output layer for low-latency speech synthesis", in *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, IEEE, Apr. 2015, pp. 4470–4474.

[289] X. Zhang, "Dynamic Objects Classification Based on High Resolution Doppler Radar and Laser Data", PhD thesis, Stuttgart University, 2017.

[290] W. Zhou, H. An, H. Yang, *et al.*, "Fast clustering of radar reflectivity data on GPUs", in *Lecture Notes in Electrical Engineering*, 2011.

[291] Y. Zhou and O. Tuzel, "VoxelNet: End-to-End Learning for Point Cloud Based 3D Object Detection", in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, IEEE, Jun. 2018, pp. 4490–4499.

LIST OF FIGURES

LIST OF TABLES

LIST OF ACRONYMS

ACC	Adaptive Cruise Control . . . 31
ADC	Analog-to-Digital Converter . . . 32
ANN	Artificial Neural Network . . . 15
CA-CFAR	Cell Averaging Constant False Alarm Rate . . . 44
CAPTCHA	Completely Automated Public Turing test to tell Computers and Humans Apart . . . 192
CDM	Code Division Multiplexing . . . 38
CFAR	Constant False Alarm Rate . . . 44
CNN	Convolutional Neural Network . . . 15
CUT	Cell Under Test . . . 44
DBCLASD	Distribution Based Clustering of Large Spatial Databases . . . 90
DBSCAN	Density-Based Spatial Clustering of Applications with Noise . . . 87
DGPS	Differential Global Positioning System . . . 46
ESPRIT	Estimation of Signal Parameter via Rotational Invariance Technique 38
FMCW	Frequency Modulated Continuous Wave . . . 30
FFT	Fast Fourier Transform . . . 35
GRU	Gated Recurrent Unit . . . 25
HDBSCAN	Hierarchical Density-Based Spatial Clustering of Applications with Noise . . . 91
IoU	Intersection Over Union . . . 178
LSTM	Long Short-Term Memory . . . 15
MIMO	Multiple Input Multiple Output . . . 31
MSG	Multi-Scale Grouping . . . 133
MUSIC	Multiple Signal Classification . . . 38
NLP	Natural Language Processing . . . 21
OPTICS	Ordering Points To Identify the Clustering Structure . . . 91
OS-CFAR	Ordered Statistics Constant False Alarm Rate . . . 45
PCA	Principal Component Analysis . . . 26
RANSAC	Random Sample Consensus . . . 145
RCS	Radar Cross Section . . . 30
ReLu	Rectified Linear Unit . . . 22
RNN	Recurrent Neural Network . . . 21
SAR	Synthetic Aperture Radar . . . 31
SMOTE	Synthetic Minority Over-sampling Technique . . . 120

SVM	Support Vector Machine 109
TDM	Time Division Multiplexing 38
TDNN	Time Delay Neural Network 21
t-SNE	t-Distributed Stochastic Neighbor Embedding 26
UUID	Universally Unique Identifier 62