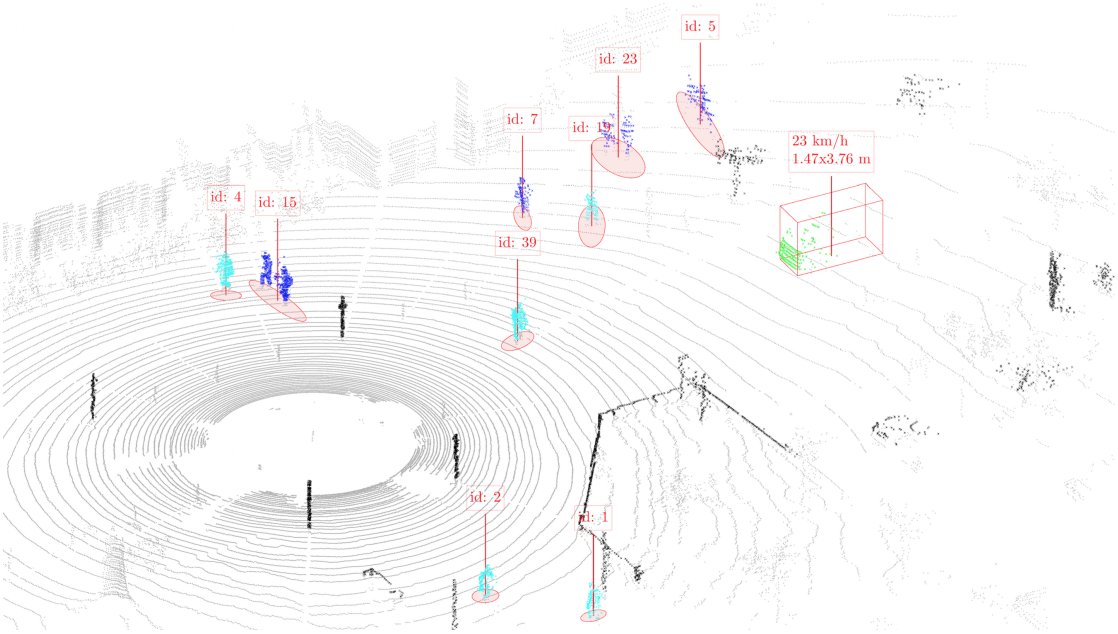
CHALMERS



Lidar-based Methods for Tracking and Identification

Master's Thesis in Systems, Control & Mechatronics

Marko Cotra
Michael Leue

Department of Signals & Systems
Chalmers University of Technology
Gothenburg, Sweden 2016
Master's Thesis 2016:1

cotra.marko@gmail.com
michael@mleue.com

Abstract

Active safety systems and autonomous drive functionality are being developed as an increasingly important competitive advantage for car manufacturers. Modern cars are equipped with a variety of sensors such as radar, ultrasound, GPS and camera. All of those information streams are then fused into an optimal estimation of the surrounding environment, in order for the car to follow a certain trajectory or avoid collisions with other objects. In order to test accuracy of the aforementioned sensors, costly high precision GPS sensors are placed on objects of interest around the car. Therefore a more cost effective and easy to use system for sensor validation is desirable.

A promising method for creating such a system is to utilize high-resolution data created by a lidar sensor. State-of-the-art lidar sensors are capable of capturing very detailed information about the surroundings in the form of a point cloud with up to 150,000 points per scan. Not only can this data be used to verify other sensors, it also opens up a new level of tracking dynamic (non-stationary) objects around the vehicle. In addition to the kinematic state (position, velocity, acceleration) the highly structured point data can be used to estimate the shape and size of objects of interest very accurately. However, the sheer amount of data requires a quite sophisticated pre-processing pipeline that removes irrelevant parts and partitions the point cloud into clusters. One way to achieve this is by utilizing prior knowledge about the surrounding environment.

This thesis describes the implementation of a tracking algorithm able to both estimate the kinematic state as well as shape and size of a time varying and unknown number of dynamic objects in lidar data. A prerequisite for the algorithm is available prior knowledge about the static surrounding environment. The algorithm is based on Bayesian inference. Methods and techniques normally used with other sensor types, such as radar and camera, are modified for usage with lidar data and used by the algorithm. Inferences on the shape of objects are modelled as either elliptical or rectangular, based on the type of target. In addition to that, a supervised neural network is trained to accurately classify detected objects (e.g. car, cyclist, pedestrian). This enables the algorithm to apply specific tracking approaches for different kinds of targets and more effectively dismiss clutter measurements that are not deemed to be important.

The tracking algorithm is evaluated using real data with four defined categories of targets: pedestrian, car, bicycle and pedestrian group. Rectangular shape estimation is used for cars and elliptical shape estimation for the remaining classes. Both tracking and identification of objects are achieved with high precision. However, due to a lack of reference system only visual evaluation of algorithm performance is made. Additionally, benefits of the detect-before-track approach utilized in this thesis are evaluated, mainly in regards to computational complexity.

The algorithm proposed in this thesis indicates that lidar sensor data can be used to successfully identify and track dynamic objects of interest around a car. Testing the algorithm using data where high precision GPS systems are attached on all objects of interest is needed in order to determine how suitable of a replacement a lidar based system would be. Additionally, an implementation of the algorithm on an embedded system is necessary in order to evaluate potential real-time use.

Acknowledgements

The authors would like to thank Karl Granström. A large part of this thesis is based on his research, and his input and support has been very helpful in key areas of this work. Additionally, the authors thank Lennart Svensson for his guidance during the initial stages of this thesis.

Marko would like to thank his friends and family for all their support. He would especially like to thank his father Nenad Cotra for inspiring him to become an engineer.

Michael would like to thank his parents Hartmut and Kathrin Leue for their continuous support throughout his life. He would not be where he is without them, not even close. *Danke für alles.*

He also wants to express his sincere gratitude towards the many free and open pieces of software that were used to realize this thesis, mainly Ubuntu GNU/Linux with the GNU base system and the Gnome software suite, Firefox, Vim, Octave and LaTeX. Thanks to all the helpful souls out there that make those tools available to everyone, free of charge.

Marko Cotra & Michael Leue, Gothenburg, June 2016

Contents

1 Introduction **1**

1.1 Thesis Objectives 2

1.1.1 Thesis Delimitations 2

1.1.2 Thesis Limitations 2

1.2 Major Contributions 3

1.3 Thesis Outline 3

2 Theory **4**

2.1 Single Target State Estimation 4

2.1.1 Kalman Filter 5

2.1.2 Non-linear Extensions of the Kalman Filter 7

2.1.3 Interacting Multiple Models 10

2.2 Multiple Target State Estimation 13

2.2.1 Multiple Target Tracking Using Random Finite Sets 13

2.2.2 The Probability Hypothesis Density Filter 14

2.3 Extended Target Tracking 18

2.3.1 Tracking Under Assumed Rectangle Shape 19

2.3.2 Tracking With Random Matrices 22
2.4 Neural Networks for Classification 28
2.4.1 Logistic Regression 28
2.4.2 Neural Network 33

3 Algorithm **37**
3.1 Sensor Properties 38
3.2 Static Point Removal 40
3.2.1 Ground Removal 40
3.2.2 Static Objects 41
3.3 Clustering 44
3.4 Classification 47
3.4.1 Features 47
3.4.2 Training and Testing 49
3.5 Elliptical PHD filtering 50
3.6 Car Cluster Pre-Processing 52
3.6.1 L-Shape Fitting With Height Based Filtering 52
3.6.2 Estimating length of viewed sides 53
3.6.3 Measurement Selection 54
3.7 Rectangular PHD filter 57
3.7.1 Choice of Motion Models 57
3.7.2 Generating Measurement Generating Points 58
3.7.3 Incorporating Rectangular Targets in PHD recursion 61

4 Results **63**
4.1 Scenario Description 63
4.2 Algorithm Performance 64

4.2.1 Pre-Processing 64
4.2.2 Neural Network 65
4.2.3 Tracking Filter 66

5 Discussion **72**

5.1 Benefits of Neural Network Classifier 72
5.2 Comparison of Extended Rectangular vs. Point Target Tracking 74

6 Conclusion **77**

Bibliography **81**

1

Introduction

Autonomous cars and active safety systems have transitioned from the realm of thought experiments and science fiction concepts into reality over the recent years. All major automobile manufacturers are developing some form of autonomous drive technology, examples of this being the 2017 Volvo Drive Me program [1] and Tesla Autopilot functionality [2]. Growing interest and demand for autonomous vehicles has pushed the development of vehicle sensor technology, increasing vehicle capability to better sense the surrounding environment. The key to successfully integrate and fuse together different sensor technologies is the ability to verify robustness and performance of both detection, identification and tracking of non-stationary objects of interest; such as pedestrians, cars and cyclists. This is currently an expensive process, involving the use of high-precision GPS systems attached to targets of interests around the autonomous vehicle. In order to verify sensor performance a cost effective and fast system of generating reference data (also called ground truth) is necessary.

In light of this problem, interest regarding lidar sensors has increased. A lidar sensor offers high accuracy measurements of range and bearing of the sensor's surrounding area, making it possible to construct a point cloud of the environment around a vehicle equipped with lidar. Lidar is frequently used in autonomous concept vehicles, such as the Google autonomous car [3]. Unfortunately, high cost and relatively large size have limited its usefulness for consumer offerings of autonomous vehicles. The lidar sensor might, however, offer the possibility to serve as a comparatively cheap and easy to use reference system for test and verification purposes of autonomous cars. Instead of equipping objects of interest around the autonomous car with expensive GPS systems, the car itself could be equipped with a lidar sensor, offering accurate enough ground truth for sensor verification.

This thesis shows how different methods can be used and integrated in order to provide an accurate reference system solely based on a lidar sensor. Target detection, identification and tracking are of interest, with a focus on finding robust and computationally effective methods. Necessary modifications to existing and previously used methods for other sensor types, such as radar and camera, are investigated in order to enable their usage with lidar sensor data.

1.1 Thesis Objectives

The objective behind the work presented in this thesis is to find, modify if necessary, and implement methods useful for detecting, tracking and identifying dynamic objects of interest around an autonomous car by using lidar data. The explicit goals are formulated as implementing a system capable of:

- Identifying and classifying objects of interest: cars, cyclists, pedestrians and pedestrian groups.
- Detecting and accurately tracking multiple objects of interest.
- Handling objects with multiple measurements associated to them.
- Removing uninformative measurements.
- Scaling well with additional objects and measurements.
- Capable of real-time implementation without significant changes to chosen algorithms.

1.1.1 Thesis Delimitations

Given the wide range of thesis objectives, several delimitations were necessary in order to fulfill the objectives within a time-frame of 20 weeks. Every delimitation was chosen in such a way that the work presented in this thesis could serve as a basis for further extensions, with less imposed delimitations. The following delimitations are present in this work:

- All objects of interest are seen as moving on perfectly flat horizontal plane. As such, no vertical motion is considered.
- Spatial extent (shape and size) is only estimated in regards to the length and width of objects, not height.
- The position of the lidar sensor is considered to be fixed, with no motion or rotation.
- Prior information of the surrounding environment is available, containing information of stationary objects such as houses, walls and so on.
- The capability of identifying objects is limited to four types: cars, cyclists, pedestrians and pedestrian groups.

1.1.2 Thesis Limitations

The work presented in this thesis is subject to several limitations, mainly connected to how the presented results are evaluated. Due to scheduling difficulties, data from a controlled test environment in the form of a crossing was not available. As such, no numeric reference measure (attained from a high precision GPS-system) is available for evaluating the proposed algorithm.

Consequently, there is no way to explicitly compare the accuracy of the algorithm presented in this work to that of a high precision GPS-system.

The scenario used for evaluating the proposed algorithm was obtained from the public KITTI dataset [4]. Efforts were made to replicate conditions from the originally planned test environment. This resulted in choosing data recorded at a college campus, where the lidar sensor stood still while several groups of pedestrians, two cars and two cyclists moved across the campus. From this scenario, an evaluation of the algorithm's capability to identify and track objects was performed by visual inspection.

1.2 Major Contributions

This thesis proposes a rigorous framework for tracking and identification of common road participants in lidar data. The overall algorithm makes heavy use of the structured and highly detailed data produced by a lidar sensor in order to solve both problems.

Extended targets are used as an improvement over center point targets, and enable tracking of both kinematic state as well as the extent of a target. This added complexity pays off in the form of a very robust tracker with much lower estimation uncertainty.

A neural network is used to classify objects before they enter the filtering part of the algorithm. This is done to then be able to run different filters on different kinds of objects to be able to employ different motion models and extent estimations for e.g. cars or pedestrians.

This integrated identification and tracking approach is very promising. It enables use of diverse tracking strategies for different classes of objects and lowers the complexity of multitarget tracking at the same time.

1.3 Thesis Outline

At first the basic underlying theory of the implemented algorithms is explained and references to related work are given. Then the different steps of the solution are explained thoroughly by following the data flow through the proposed algorithm. After that the results of this thesis are presented as a visual evaluation of the algorithm on a test scenario. This is followed by an interpretation and a discussion of the results. Lastly, a conclusion examines whether the thesis succesfully fulfilled the posed objectives and how the work could be extended in the future.

2

Theory

In this chapter a brief review of relevant theory, necessary in order to understand the implemented algorithm, is presented. The chapter begins with Bayesian single target state estimation, covering the general formulation as well as describing linear and non-linear recursive implementations. This is followed by a section on multi-target state estimation; focusing on the random finite set approach. Next, a section on extended target tracking is presented, illustrating how to incorporate shape and size when multiple measurements are associated to a target. Finally, the chapter closes with a section detailing how neural networks can be used for classification.

2.1 Single Target State Estimation

Single target state estimation concerns with the problem of estimating the state of a known target under uncertainty. There is uncertainty in how the state evolves over time, as well as uncertainty in how accurate sensor readings of the target are. In order to model and deal with uncertainty, a probabilistic framework is employed. By using a probabilistic framework, the single target tracking problem can be expressed in the following way.

$$\begin{aligned} \mathbf{x_k} &= \mathbf{f}_{k-1}(\mathbf{x}_{k-1},\mathbf{q}_{k-1}) \\ \mathbf{y_k} &= \mathbf{h}_k(\mathbf{x}_k,\mathbf{r}_k) \end{aligned} \tag{2.1}$$

The state, $\mathbf{x}_k$, of a known target is assumed to evolve over time according to a discrete time Markov model $\mathbf{f}(\cdot)$ and is subject to motion noise $\mathbf{q}_{k-1}$. Each incremental time step is denoted by k. The initial state is distributed according to $\mathbf{x}_0 \sim p(\mathbf{x}_0)$. At each time step a measurement, denoted $\mathbf{y}_k$, is generated as a function $\mathbf{h}(\cdot)$ of the state $\mathbf{x}_k$ and the measurement noise $\mathbf{r}_k$.

The optimal filtering problem is to compute the posterior density $p(\mathbf{x}_k|\mathbf{y}_{1:k})$ when $k \geq 1, k \in \mathbb{N}$. By using Bayesian statistics, the state conditioned on all measurements up to time k can be expressed as

$$p(\mathbf{x}_k|\mathbf{y}_{1:k}) = \int p(\mathbf{x}_{0:k}|\mathbf{y}_{1:k})d\mathbf{x}_{0:k-1} \tag{2.2}$$

$$p(\mathbf{x}_{0:k}|\mathbf{y}_{1:k}) = \frac{p(\mathbf{y}_{1:k}|\mathbf{x}_{0:k})p(\mathbf{x}_{0:k})}{p(\mathbf{y}_{1:k})} \propto p(\mathbf{x}_0)\prod_{i=1}^{k} p(\mathbf{y}_i|\mathbf{x}_i)p(\mathbf{x}_i|\mathbf{x}_{i-1}) \tag{2.3}$$

A drawback with the formulation in (2.2)-(2.3) is that complexity grows with k. Instead, it is possible to express the posterior density as a recursive solution which utilises the previous posterior density. The recursive solution is expressed in two steps, the prediction step and the update step. The prediction step represents the expected posterior density at time k conditioned on all measurements up to time $k-1$, which can be expressed using the Chapman-Kolomogrov equation as

$$p(\mathbf{x}_k|\mathbf{y}_{1:k-1}) = \int p(\mathbf{x}_k|\mathbf{x}_{k-1})p(\mathbf{x}_{k-1}|\mathbf{y}_{1:k-1})d\mathbf{x}_{k-1} \tag{2.4}$$

The update step incorporates knowledge from the measurement $\mathbf{y}_k$ into the prediction step, yielding the updated posterior as

$$p(\mathbf{x}_k|\mathbf{y}_{1:k}) = \frac{p(\mathbf{y}_k|\mathbf{x}_k)p(\mathbf{x}_k|\mathbf{y}_{1:k-1})}{p(\mathbf{y}_k|\mathbf{y}_{1:k-1})} \tag{2.5}$$

The two equations (2.4) and (2.5) present a general recursive solution to the optimal filtering problem. However, they do not offer a general closed form solution, which makes them difficult to implement in practice. Nonetheless, under certain conditions closed form solutions can be found, such as the Kalman Filter.

2.1.1 Kalman Filter

The Kalman Filter (KF) offers a closed form solution to the recursive optimal filtering formulation found in (2.4)-(2.5) by imposing the following conditions

- State and measurements evolve over time according to a linear model.
- The underlying prior distribution $p(\mathbf{x}_0)$ is assumed to be Gaussian, i.e. $\mathbf{x}_0 \sim \mathcal{N}(\hat{\mathbf{x}}_0, \mathbf{P}_0)$.

- Motion noise $\mathbf{q}_k$ and measurement noise $\mathbf{r}_k$ are i.i.d. Gaussian.

With these conditions the general motion and measurement model equations (2.1) can be rewritten as

$$\begin{aligned} \mathbf{x}_k &= \mathbf{F}_{k-1}\mathbf{x}_{k-1} + \mathbf{q}_{k-1}, & \mathbf{q}_k &\sim \mathcal{N}(\mathbf{0},\mathbf{Q}_{k-1}) \\ \mathbf{y}_k &= \mathbf{H}_k\mathbf{x}_k + \mathbf{r}_k, & \mathbf{r}_k &\sim \mathcal{N}(\mathbf{0},\mathbf{R}_k) \end{aligned} \tag{2.6}$$

(Both motion and sensor noise are imposed as zero-mean to keep subsequent equations more clear). Since all densities are of Gaussian nature, only two moments are needed to describe the posterior density: the mean $\hat{\mathbf{x}}_{k|k}$ and the covariance $\mathbf{P}_{k|k}$. As such, key densities in the recursive formulation (2.4) and (2.5) can be expressed as

$$p(\mathbf{x}_{k-1}|\mathbf{y}_{1:k-1}) \sim \mathcal{N}(\hat{\mathbf{x}}_{k-1|k-1}, \mathbf{P}_{k-1|k-1}) \tag{2.7}$$
$$p(\mathbf{x}_k|\mathbf{y}_{1:k-1}) \sim \mathcal{N}(\hat{\mathbf{x}}_{k|k-1}, \mathbf{P}_{k|k-1}) \tag{2.8}$$
$$p(\mathbf{x}_k|\mathbf{y}_{1:k}) \sim \mathcal{N}(\hat{\mathbf{x}}_{k|k}, \mathbf{P}_{k|k}) \tag{2.9}$$

The prediction step in the KF corresponds to finding expressions for the moments found in (2.8). Since the models are linear, this corresponds to finding the expected value and covariance of a linearly scaled and shifted Gaussian density.

Prediction

$$\begin{aligned} \mathbb{E}[\mathbf{x}_k|\mathbf{y}_{1:k-1}] &\equiv \hat{\mathbf{x}}_{k|k-1} = \mathbf{F}_{k-1}\hat{\mathbf{x}}_{k-1|k-1} \\ \mathrm{Cov}[\mathbf{x}_k|\mathbf{y}_{1:k-1}] &\equiv \mathbf{P}_{k|k-1} = \mathbf{F}_{k-1}\mathbf{P}_{k-1|k-1}\mathbf{F}_{k-1}^T + \mathbf{Q}_k \end{aligned} \tag{2.10}$$

The update step in the KF is divided into several components which are used to find expressions for the moments of the posterior density. Detailed information on how these components are derived can be found in [5].

$$\mathbf{v}_k = \mathbf{y}_k - \mathbf{H}_k\hat{\mathbf{x}}_{k|k-1} \tag{2.11}$$
$$\mathbf{S}_k = \mathbf{H}_k\mathbf{P}_{k|k-1}\mathbf{H}_k^T + \mathbf{R}_k \tag{2.12}$$
$$\mathbf{K}_k = \mathbf{P}_{k|k-1}\mathbf{H}_k^T(\mathbf{S}_k)^{-1} \tag{2.13}$$

The term $\mathbf{v}_k$ in (2.11) is the so called *innovation* and represents the difference between the actual measurement and the predicted measurement. The innovation is zero-mean and has a covariance equal to $\mathbf{S}_k$. The covariance $\mathbf{S}_k$ in (2.12) can be viewed as representing the expected measurement covariance. In other words, it expresses a region around the predicted state where measurements are likely to be received from. The final term $\mathbf{K}_k$, known as the *Kalman gain*, can be interpreted as a measure of how much the innovation should be included in the posterior.

By using (2.11)-(2.13) the posterior mean and covariance can be formulated as

Update

$$\begin{aligned}\hat{\mathbf{x}}_{k|k} &= \hat{\mathbf{x}}_{k|k-1} + \mathbf{K}_k \mathbf{v}_k \\ \mathbf{P}_{k|k} &= \mathbf{P}_{k|k-1} - \mathbf{K}_k \mathbf{S}_k \mathbf{K}_k^T\end{aligned} \tag{2.14}$$

With these final two moments the complete KF recursion is formulated. Filter performance is heavily influenced by the ratio between motion and measurement noise, which in effect corresponds to signal-to-noise ratio (SNR). A high SNR (i.e. $\|\mathbf{R}\| \ll \|\mathbf{Q}\|$) will result in a filter which is more responsive to measurements and follows these more closely. Conversely, a low SNR (i.e. $\|\mathbf{R}\| \gg \|\mathbf{Q}\|$) will result in a less responsive filter which lags behind new measurements.

2.1.2 Non-linear Extensions of the Kalman Filter

One of the assumptions for deriving the Kalman Filter are linear motion and measurement models. By utilizing linear models, all densities in the filter are Gaussian, since the initial state is considered to be a Gaussian. In many practical applications however, the assumption of linear models is no longer valid. As such, non-linear extensions of the KF have been formulated. In general, two different approaches exist when dealing with non-linear models: linearization and Gaussian moment matching. A commonly used algorithm utilizing linearization is the Extended Kalman Filter (EKF). In the family of Gaussian moment matching algorithms [5] the Unscented Kalman Filter (UKF) is frequently used, mainly due to how well the algorithm scales with the number of states. What follows is a general description of each algorithm as well as a comparison of the two. For more in-depth derivations and information on each algorithm, the reader is referred to [5] and [6] respectively.

The Extended Kalman Filter

The Extended Kalman Filter enables non-linear motion and measurements models to be incorporated within the KF framework by utilizing linearization. For a system of motion and measurement models,

$$\begin{aligned}\mathbf{x}_{k|k-1} &= \mathbf{f}_{k-1}(\mathbf{x}_{k-1}) + \mathbf{q}_{k-1} \\ \mathbf{y}_k &= \mathbf{h}_k(\mathbf{x}_{k|k-1}) + \mathbf{r}_k\end{aligned} \tag{2.15}$$

where $\mathbf{f}_{k-1}(\cdot)$ and $\mathbf{h}_k(\cdot)$ are nonlinear, linearization through first-order Taylor expansion can be performed for each function around the expected values $\hat{\mathbf{x}}_{k-1|k-1}$ and $\hat{\mathbf{x}}_{k|k-1}$ respectively,

$$\begin{aligned}\mathbf{f}_{k-1}(\mathbf{x}_{k-1}) &\approx \mathbf{f}_{k-1}(\hat{\mathbf{x}}_{k-1|k-1}) + \frac{\partial \mathbf{f}_{k-1}(\cdot)}{\partial \hat{\mathbf{x}}_{k-1|k-1}}(\mathbf{x}_{k-1} - \hat{\mathbf{x}}_{k-1|k-1}) \\ \mathbf{h}_k(\mathbf{x}_{k|k-1}) &\approx \mathbf{h}_k(\hat{\mathbf{x}}_{k|k-1}) + \frac{\partial \mathbf{h}_k(\cdot)}{\partial \hat{\mathbf{x}}_{k|k-1}}(\mathbf{x}_{k|k-1} - \hat{\mathbf{x}}_{k|k-1})\end{aligned} \tag{2.16}$$

With linearizations derived for each model, the prediction and update steps in the KF recursion can now be expressed.

Prediction

$$\begin{aligned}\hat{\mathbf{x}}_{k|k-1} &= \mathbf{f}_{k-1}(\hat{\mathbf{x}}_{k-1|k-1}) \\ \mathbf{P}_{k|k-1} &= \left(\frac{\partial \mathbf{f}_{k-1}(\cdot)}{\partial \hat{\mathbf{x}}_{k-1|k-1}}\right)\mathbf{P}_{k-1|k-1}\left(\frac{\partial \mathbf{f}_{k-1}(\cdot)}{\partial \hat{\mathbf{x}}_{k-1|k-1}}\right)^T + \mathbf{Q}_{k-1}\end{aligned} \tag{2.17}$$

Update

$$\begin{aligned}\mathbf{v}_k &= \mathbf{y}_k - \mathbf{h}_k(\hat{\mathbf{x}}_{k|k-1}) \\ \mathbf{S}_k &= \left(\frac{\partial \mathbf{h}_k(\cdot)}{\partial \hat{\mathbf{x}}_{k|k-1}}\right)\mathbf{P}_{k|k-1}\left(\frac{\partial \mathbf{h}_k(\cdot)}{\partial \hat{\mathbf{x}}_{k|k-1}}\right)^T + \mathbf{R}_k \\ \mathbf{K}_k &= \mathbf{P}_{k|k-1}\left(\frac{\partial \mathbf{h}_k(\cdot)}{\partial \hat{\mathbf{x}}_{k|k-1}}\right)^T \mathbf{S}_k^{-1} \\ \hat{\mathbf{x}}_{k|k} &= \hat{\mathbf{x}}_{k|k-1} + \mathbf{K}_k\mathbf{v}_k \\ \mathbf{P}_{k|k} &= \mathbf{P}_{k|k-1} - \mathbf{K}_k\mathbf{S}_k\mathbf{K}_k^T\end{aligned} \tag{2.18}$$

It is important to note that the EKF, unlike the KF, is not an optimal solution in any way.

The Unscented Kalman Filter

The Unscented Kalman Filter (UKF) belongs to the family of Gaussian moment matching filters. The key difference between these filters and the EKF is that instead of linearizing non-linear functions, the resulting non-Gaussian distribution is approximated as a Gaussian. This is performed by deterministically selecting so called *σ-points* from a Gaussian distribution, where each point has an associated (also deterministically chosen) weight. The σ-points are then propagated through the non-linear function, and from the transformed σ-points a numerical estimation of the mean and covariance is calculated. These methods can be seen as sophisticated Monte Carlo methods that, through their use of deterministically chosen sample-points, require less computation time, although with less accurate estimations.

The difference between Gaussian moment matching filters is in how they select σ-points, where some filters better estimate higher order effects while others scale better with the number of states. The UKF guarantees estimation of a function up to second order effects and utilizes a total of $1 + 2n$ σ-points, where n is the number of states in the state vector. The UKF is a common choice when dealing with non-linear motion or measurement models, especially since its computational complexity is equal to the EKF [6].

For the UKF, the prediction and update steps are defined as the following:

Prediction

Form a set of $2n+1$ σ-points

$$\begin{aligned}
\mathcal{X}_{k-1}^{(0)} &= \hat{\mathbf{x}}_{k-1|k-1}, \quad W_0 = 1 - n/3 \\
\mathcal{X}_{k-1}^{(i)} &= \hat{\mathbf{x}}_{k-1|k-1} + \sqrt{3}(\mathbf{P}_{k-1|k-1}^{1/2})_i, \quad i = 1,2,\ldots,n \\
\mathcal{X}_{k-1}^{(i+n)} &= \hat{\mathbf{x}}_{k-1|k-1} - \sqrt{3}(\mathbf{P}_{k-1|k-1}^{1/2})_i, \quad i = 1,2,\ldots,n \\
W_i &= \frac{1 - W_0}{2n}, \quad i = 1,2,\ldots,2n
\end{aligned} \tag{2.19}$$

where $(\mathbf{P}^{1/2})_i$ denotes the i-th column of $\mathbf{P}^{1/2}$, which is the square-root[1] of the covariance matrix. The predicted moments are then computed as

$$\begin{aligned}
\hat{\mathbf{x}}_{k|k-1} &\approx \sum_{i=0}^{2n} f(\mathcal{X}_{k-1}^{(i)}) W_i \\
\mathbf{P}_{k|k-1} &\approx \mathbf{Q}_{k-1} + \sum_{i=0}^{2n} (f(\mathcal{X}_{k-1}^{(i)}) - \hat{\mathbf{x}}_{k|k-1})(\cdot)^T W_i
\end{aligned} \tag{2.20}$$

Update
Form a set of $2n+1$ σ-points

$$\begin{aligned}
\mathcal{X}_{k}^{(0)} &= \hat{\mathbf{x}}_{k|k-1}, \quad W_0 = 1 - n/3 \\
\mathcal{X}_{k}^{(i)} &= \hat{\mathbf{x}}_{k|k-1} + \sqrt{3}(\mathbf{P}_{k|k-1}^{1/2})_i, \quad i = 1,2,\ldots,n \\
\mathcal{X}_{k}^{(i+n)} &= \hat{\mathbf{x}}_{k|k-1} - \sqrt{3}(\mathbf{P}_{k|k-1}^{1/2})_i, \quad i = 1,2,\ldots,n \\
W_i &= \frac{1 - W_0}{2n}, \quad i = 1,2,\ldots,2n
\end{aligned} \tag{2.21}$$

The desired moments can be computed as

$$\begin{aligned}
\hat{\mathbf{y}}_{k|k-1} &\approx \sum_{i=0}^{2n} h(\mathcal{X}_{k}^{(i)}) W_i \\
\mathbf{P}_{xy} &\approx \sum_{i=0}^{2n} (\mathcal{X}_{k}^{(i)} - \hat{\mathbf{x}}_{k|k-1})(h(\mathcal{X}_{k}^{(i)}) - \hat{\mathbf{y}}_{k|k-1})^T W_i \\
\mathbf{P}_{yy} &\approx \mathbf{R}_k + \sum_{i=0}^{2n} (h(\mathcal{X}_{k}^{(i)}) - \hat{\mathbf{y}}_{k|k-1})(\cdot)^T W_i \\
\mathbf{K}_k &= \mathbf{P}_{xy}\mathbf{P}_{yy}^{-1}
\end{aligned} \tag{2.22}$$

The updated state can then be computed as

$$\begin{aligned}
\hat{\mathbf{x}}_{k|k} &= \hat{\mathbf{x}}_{k|k-1} + \mathbf{K}_k(\mathbf{y}_k - \hat{\mathbf{y}}_{k|k-1}) \\
\mathbf{P}_{k|k} &= \mathbf{P}_{k|k-1} - \mathbf{K}_k \mathbf{S}_k \mathbf{K}_k^T
\end{aligned} \tag{2.23}$$

Comparison of EKF and UKF

Both the EKF and UKF enable non-linear motion and measurement models to be incorporated into the recursive KF framework, although with different approaches. The EKF is capable of

[1] Lower triangular matrix, can be obtained through Cholesky decomposition

capturing first order effects, while the UKF guarantees accuracy up to second order effects. Computing Jacobians in the EKF can be time consuming as well as pose problems when an analytical solution is not possible, necessitating the use of numerical differentiation techniques. The UKF does not require calculation of any Jacobians. However, calculating the square-root of the state covariance can produce problems, since special care needs to be taken in order to guarantee positive semi-definiteness of the state-covariance [7].

An illustration of the difference between EKF and UKF can be seen in Figure 2.1

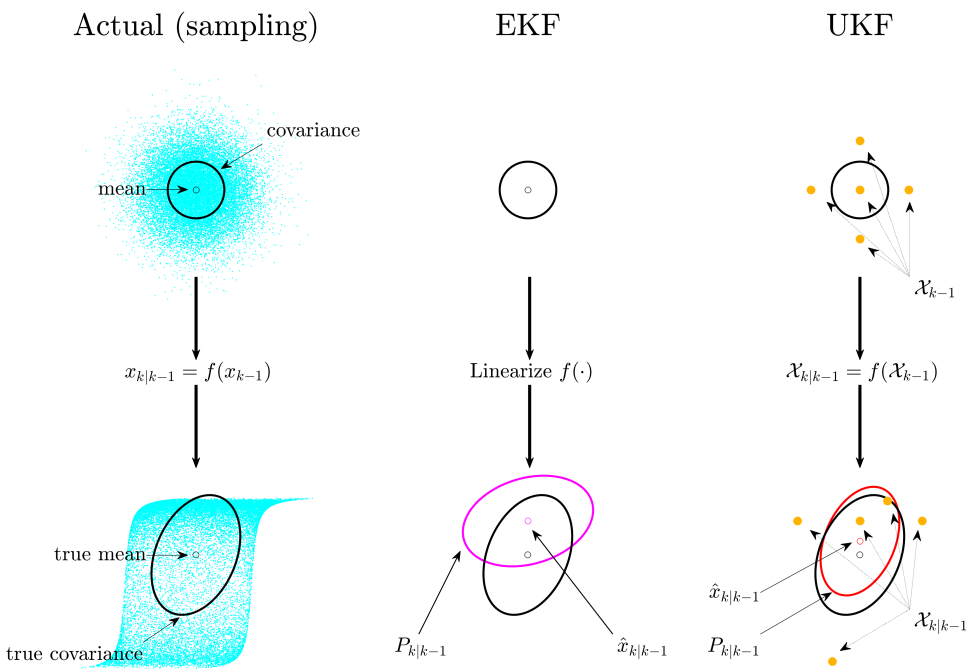


Figure 2.1: A conceptual illustration of the difference between EKF and UKF. The leftmost picture illustrates the true non-linearly transformed mean and covariance, achieved through sampling. The middle picture illustrates how the EKF through linearization produces an estimate of both moments. The rightmost picture illustrates how σ-points in the UKF are used to determine both transformed moments.

2.1.3 Interacting Multiple Models

In many practical applications a single motion model is not sufficient in order to accurately describe the behaviour of a target. For example, the dynamics for a vehicle are significantly different when driving in a straight line as compared to performing a turn. In literature, such actions are commonly referred to as maneuvers. A way to remedy this problem is to increase the motion noise $\mathbf{Q}_{k-1}$ in order to account for the model miss-match. However, this will result in an overall less accurate motion model, with reduced accuracy for state estimation as a result.

Another way of dealing with ambiguous state behaviour is through the use of multiple motion models. Multiple models utilize Jump Markov Linear Systems (JMLS), where each specified motion model corresponds to a discrete *mode* λ_k, which needs to be estimated from the received measurement $\mathbf{y}_k$ in addition to the state $\mathbf{x}_k$. Under JMLS, the general motion and measurement equations can be expressed as

$$\mathbf{x}_k = \mathbf{f}_{k-1}(\mathbf{x}_{k-1},\lambda_k) + \mathbf{q}_{k-1}(\lambda_k) \qquad \mathbf{q}_{k-1}(\lambda_k) \sim \mathcal{N}(\mathbf{0},\mathbf{Q}_{k-1}(\lambda_k)) \tag{2.24}$$

$$\mathbf{y}_k = \mathbf{h}_k(\mathbf{x}_k, \lambda_k) + \mathbf{r}_k(\lambda_k), \qquad \mathbf{r}_k(\lambda_k) \sim \mathcal{N}(\mathbf{0},\mathbf{R}_k(\lambda_k)) \tag{2.25}$$

where $\lambda_k = 1,2,\ldots,N_\lambda$ corresponds to model number, where $\mathbf{f}_{k-1}(\cdot), \mathbf{q}_{k-1}(\cdot), \mathbf{h}_k(\cdot), \mathbf{r}_k(\cdot)$ all depend on the mode λ_k. Two different approaches exist when dealing with mode transitions: hard and soft decisions. With hard decisions the most probable mode is chosen and used for determining the subsequent state $\mathbf{x}_k$. With soft decisions a weighted average of all modes is calculated for determining the state $\mathbf{x}_k$. An advantage with the soft decision approach is more robustness to model errors. For example, in a filter with two modes, one utilizing low motion noise and the other high motion noise, a soft decision approach makes it possible to in effect use motion noise in between the two. In practice, utilizing hard decision making will result in serial use of different KFs, while soft decision making results in parallel use of KFs at each timestep.

A frequently used filter for incorporating multiple models is the Interacting Multiple Models (IMM) filter, which utilizes soft decision making for determining mode. In the IMM filter the posterior state is modeled as

$$p(\mathbf{x}_k|\mathbf{y}_k) = \sum_{i=1}^{N_\lambda} w_k^i \mathcal{N}(\mathbf{x}_k; \hat{\mathbf{x}}_{k|k}^i, \mathbf{P}_{k|k}^i) \tag{2.26}$$

In order to determine the posterior state, a prior describing transitions between the modes is necessary. The prior is expressed as a Transition Probability Matrix[2] (**TPM**), where each element $\pi_{ij} = \Pr\{\lambda = j|\lambda = i\}$, i.e. describes the probability of transitioning from mode i to j.

$$\mathbf{TPM} = \begin{bmatrix} \pi_{11} & \pi_{12} & \cdots & \pi_{1N_\lambda} \\ \pi_{21} & \pi_{22} & \cdots & \pi_{1N_\lambda} \\ \vdots & \vdots & \ddots & \vdots \\ \pi_{N_\lambda 1} & \pi_{N_\lambda 2} & \cdots & \pi_{N_\lambda N_\lambda} \end{bmatrix} \tag{2.27}$$

An important step in the IMM algorithm is so called *Mixing*. For each mode j, the output of all filters in the IMM are merged, where each component is weighted with the probability that mode switches to j next time step. Without mixing, the number of components in the IMM

[2]It should be noted that the **TPM** can be modelled as state-dependent, i.e $\pi_{ij}(\hat{\mathbf{x}}_{k|k-1})$.

filter would grow combinatorially with each iteration. A complete iteration of the IMM filter is performed in the following way.

Mixing
Calculate mixing weights and merge the estimates and covariances.

$$\begin{aligned} w^{ji}_{k-1|k-1} &= \frac{w^{j}_{k-1}\pi_{ji}}{\sum_{\gamma=1}^{N_\lambda} w^{\gamma}_{k-1}\pi_{\gamma i}} \\ \hat{\mathbf{x}}^{0i}_{k-1|k-1} &= \sum_{j=1}^{N_\lambda} w^{ji}_{k-1|k-1}\hat{\mathbf{x}}^{j}_{k-1|k-1} \\ \mathbf{P}^{0i}_{k-1|k-1} &= \sum_{j=1}^{N_\lambda} w^{ji}_{k-1|k-1}\left[\mathbf{P}^{j}_{k-1|k-1} + (\hat{\mathbf{x}}^{j}_{k-1|k-1} - \hat{\mathbf{x}}^{0i}_{k-1|k-1})(\cdot)^T\right] \end{aligned} \tag{2.28}$$

Mode Matched Prediction
Perform a prediction for each mode.

$$\{\hat{\mathbf{x}}^{i}_{k|k-1}, \mathbf{P}^{i}_{k|k-1}\} = \text{PREDICT}\left(\mathbf{f}^{i}_{k-1}(\cdot), \mathbf{q}^{i}_{k-1}, \hat{\mathbf{x}}^{0i}_{k-1|k-1}, \mathbf{P}^{0i}_{k-1|k-1}\right) \tag{2.29}$$

Mode Matched Update
Perform an update for each mode and calculate mode probabilities.

$$\{\hat{\mathbf{x}}^{i}_{k|k}, \mathbf{P}^{i}_{k|k}, \hat{\mathbf{y}}^{i}_{k|k-1}, \mathbf{S}^{i}_{k}\} = \text{UPDATE}\left(\mathbf{h}^{i}_{k}(\cdot), \mathbf{r}^{i}_{k}, \hat{\mathbf{x}}^{i}_{k|k-1}, \mathbf{P}^{i}_{k|k-1}\right) \tag{2.30}$$

$$w^{i}_{k} = \frac{\mathcal{N}(\mathbf{y}_k; \hat{\mathbf{y}}^{i}_{k|k-1}, \mathbf{S}^{i}_{k}) \sum_{j=1}^{N_\lambda} w^{j}_{k-1}\pi_{ji}}{\sum_{\gamma=1}^{N_\lambda} \mathcal{N}(\mathbf{y}_k; \hat{\mathbf{y}}^{\gamma}_{k|k-1}, \mathbf{S}^{\gamma}_{k}) \sum_{j=1}^{N_\lambda} w^{j}_{k-1}\pi_{j\gamma}} \tag{2.31}$$

Output Estimate Calculation
Compute the estimated state by merging the estimates for each mode.

$$\begin{aligned} \hat{\mathbf{x}}_{k|k} &= \sum_{i=1}^{N_\lambda} w^{i}_{k}\hat{\mathbf{x}}^{i}_{k|k} \\ \mathbf{P}_{k|k} &= \sum_{i=1}^{N_\lambda} w^{i}_{k}\left[\mathbf{P}^{i}_{k|k} + (\hat{\mathbf{x}}^{i}_{k|k} - \hat{\mathbf{x}}_{k|k})(\cdot)^T\right] \end{aligned} \tag{2.32}$$

The terms PREDICT() and UPDATE() correspond to the prediction and update equations for the filter utilized in the IMM (e.g. KF or UKF, see Sections 2.1.1-2.1.2).

2.2 Multiple Target State Estimation

A single-target tracking filter can make several simplifying assumptions that are usually not given given in a real-world scenario. It works under the premise that there is only one target to track and that any measurement is therefore automatically associated with that one target. As soon as several targets and several measurements in each time-step are considered, the tracking problem becomes much more complicated.

One recursive solution to solving the multi-target tracking problem is the Multiple Hypothesis Tracking (MHT) filter. The MHT approach considers all possible hypotheses for associating measurements to existing targets, new targets and false alarms. The probability of each generated hypothesis is evaluated, and the most probable hypothesis is selected. The amount of tracks and corresponding states in the most probable hypothesis are presented as the filter output. Detailed information regarding the MHT filter can be found in [8].

A principal problem with the MHT filter arises from basic combinatorics. Data association occurs by initiating a separate hypothesis for every measurement that lies within the 3σ-gate of any of the predictions. It is therefore easy to see that the MHT filter does not scale well for neither an increased number of measurements nor targets. In addition to that, a robust filter implementation will have to keep lower-probability hypotheses for at least a few time steps and cannot discard them right away. Therefore, the effectiveness of hypothesis pruning techniques is always limited by the need for a certain robustness. Out of consideration for a computationally efficient algorithm other possible solutions for the multiple target tracking problem are required.

2.2.1 Multiple Target Tracking Using Random Finite Sets

A promising approach that does not suffer from the typical problem of associating a potentially high number of targets and measurements is to use random finite sets (RFS). A RFS is a set, i.e. an unstructured collection of elements, whose cardinality (the number of elements in the set) is a random variable. In this setting at any given time k there are two RFS:s of interest. One is the state RFS $\mathbf{X}_k$ containing individual targets and the other is the measurements RFS $\mathbf{Z}_k$ containing observations. Describing both the number of targets and the number of measurements as a random variable makes it possible to apply a Bayesian framework to the estimation of both the number of targets as well as their states [9].

RFS are used because they allow to model uncertainty in both the number of targets and the targets' states. In a single-target tracking scenario, uncertainty about the state and measurements is modelled by using random vectors. In the multi-target tracking case both the state and the measurements are modelled as RFS whose cardinality can be described by a discrete probability distribution (e.g. a Poisson distribution) and whose elements are characterized by a number of joint probability densities (e.g. several multivariate Gaussian components) [9].

Extending Bayesian reasoning to multi-target problems defined on RFS requires similar tools, that are available in general statistics, to derive predictions, likelihoods and posterior densities. For example set integrals and set derivatives have to be defined in order to be able to compute e.g. the multi-target prediction density $\mathbf{f}_{k|k-1}(\mathbf{X}_k|\mathbf{X}_{k-1})$ based on the state RFS $\mathbf{X}_k$ or the multi-target measurement density $\mathbf{f}_k(\mathbf{Z}_k|\mathbf{X}_k)$ based on the measurement RFS $\mathbf{Z}_k$ and the state

RFS $\mathbf{X}_k$. Those tools are provided by finite-set statistics (FISST) [10].

Each element in a given state RFS $\mathbf{x}_{k-1} \in \mathbf{X}_{k-1}$ might continue to exist at time k with probability $p_{S,k}$ or die with probability $1-p_{S,k}$. Those that continue to exist transition to state $\mathbf{x}_k$ according to a transition function $\mathbf{f}_{k|k-1}(\mathbf{x}_k|\mathbf{x}_{k-1})$. The state RFS at $\mathbf{X}_k$ can therefore be predicted as the transition to $\{\mathbf{x}_k\}$ of all surviving elements $\mathbf{x}_{k-1} \in \mathbf{X}_{k-1}$ or $\emptyset$ otherwise. This entire logic can be modelled as the transition $\mathbf{S}_{k|k-1}(\mathbf{x}_{k-1})$.[9]

In addition to previous targets surviving from the previous time-step there can also be new targets that are born at k, denoted as $\mathbf{\Gamma}_k$. In theory, the RFS approach can also handle new targets spawning from existing targets (e.g. a person leaving its car and walking away from it or a plane firing off a rocket) but as those were not relevant for this thesis, that part is omitted here. The multi-target RFS $\mathbf{X}_k$ can therefore be described as follows.

$$\mathbf{X}_k = \left[\bigcup_{\mathbf{x}_{k-1} \in \mathbf{X}_{k-1}} \mathbf{S}_{k|k-1}(\mathbf{x}_{k-1}) \right] \cup \mathbf{\Gamma}_k \tag{2.33}$$

The measurement RFS $\mathbf{Z}_k$ consists both of actual target observations as well as of clutter. The probability of detecting a target $\mathbf{x}_k \in \mathbf{X}_k$ is $p_{D,k}$ and missing it is $1 - p_{D,k}$. The likelihood function of any measurement $\mathbf{z}_k$ is described by $g_k(\mathbf{z}_k|\mathbf{x}_k)$ if it is detected or $\emptyset$ otherwise. This logic can be expressed by a transition function $\mathbf{\Theta}_k(x_k)$. In addition to measurements originating from actual targets there are also the aforementioned clutter observations denoted as $\mathbf{K}_k$. The entire measurement RFS $\mathbf{Z}_k$ is therefore the union of the following sets.

$$\mathbf{Z}_k = \left[\bigcup_{\mathbf{x}_k \in \mathbf{X}_k} \mathbf{\Theta}_k(\mathbf{x}_k) \right] \cup \mathbf{K}_k \tag{2.34}$$

As such, an optimal Bayesian multi-target filter has the following prediction and update steps, analogous to the general single-target Bayesian recursion described in equations (2.4) and (2.5).

$$\begin{aligned} p_{k|k-1}(\mathbf{X}_k|\mathbf{Z}_{1:k-1}) &= \int f_{k|k-1}(\mathbf{X}_k|\mathbf{X}_{k-1}) p_{k-1}(\mathbf{X}_{k-1}|\mathbf{Z}_{1:k-1}) \mu_s(d\mathbf{X}_{k-1}) \\ p_k(\mathbf{X}_k|\mathbf{Z}_{1:k}) &= \frac{g_k(\mathbf{Z}_k|\mathbf{X}_k) p_{k|k-1}(\mathbf{X}_k|\mathbf{Z}_{1:k-1})}{\int g_k(\mathbf{Z}_k|\mathbf{X}_k) p_{k|k-1}(\mathbf{X}_k|\mathbf{Z}_{1:k-1}) \mu_s(d\mathbf{X}_{k-1})} \end{aligned} \tag{2.35}$$

Here μ_s has to be a reference measure on the entire collection of subsets of $\mathbf{X}_k$ [9].

2.2.2 The Probability Hypothesis Density Filter

Deriving an optimal filtering solution to the multi-target tracking problem in a Bayesian framework follows similar ideas as the Kalman filter for the general single-target tracking case. Instead of propagating the entire posterior density, the state can usually be approximated well enough by

only propagating some of its statistical moments. For example, the constant gain Kalman filter only propagates the first-order statistical moment of the posterior density: the posterior expectation. Similarly, a first-order statistical moment of the multi-target posterior density is defined as the probability hypothesis density (PHD). The PHD is a function over the state-space that yields the expected number of targets over some sub-area S of the state-space when integrated over S [10]. An illustration of the PHD can be seen in Figure 2.2.

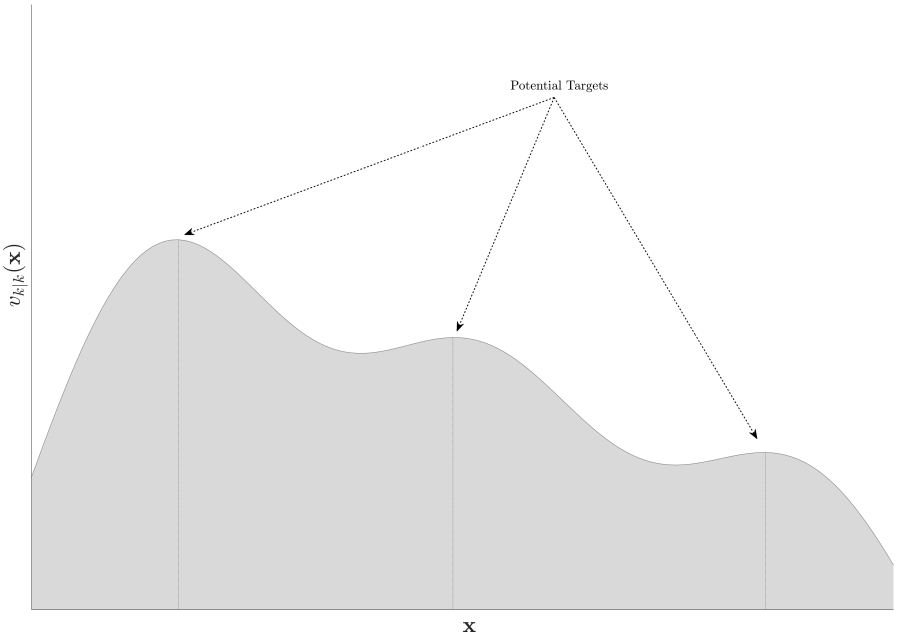


Figure 2.2: Example PHD visualizing three potential targets.

The PHD is defined over the entire state space $\mathbf{x}$, where the area blow the curve represents the number of targets present. Each peak in the figure represents a potential target. The corresponding $\mathbf{x}$ for each peak represents the state of the potential target. Higher peaks in the PHD represent more probable targets. A recursive filtering algorithm, again similar to the Kalman filter, that propagates the PHD over time is known as the probability hypothesis density filter (PHD filter).

There are several assumptions that have to be considered in order to be able to derive a recursive solution for propagating the PHD $v_k(\mathbf{x})$:

- Targets move independently of each other.
- Clutter is Poisson-distributed and independent of target-generated measurements.
- Predicted RFS, described by $p_{k|k-1}$, is Poisson-distributed.

Assuming that the predicted RFS is Poisson-distributed is important because a Poisson distribution is completely described by it's intensity. In a Poisson distribution, the first moment (the mean) is denoted as λ and all other moments can be derived from it [11, p.146]. The prediction and update step for recursively computing the PHD $v_k(\mathbf{x})$ are then as follows [9].

$$\begin{aligned} v_{k|k-1}(\mathbf{x}) &= \int p_{S,k}(\mathbf{x}_{k-1})\mathbf{f}_{k|k-1}(\mathbf{x}_k|\mathbf{x}_{k-1})v_{k-1}(\mathbf{x}_{k-1})d\mathbf{x}_{k-1} \\ &\quad + \int \beta_{k|k-1}(\mathbf{x}_k|\mathbf{x}_{k-1})v_{k-1}(\mathbf{x}_{k-1})d\mathbf{x}_{k-1} + \gamma_k(\mathbf{x}) \\ v_k(\mathbf{x}) &= [1 - p_{D,k}(\mathbf{x})]v_{k|k-1}(\mathbf{x}) \\ &\quad + \sum_{\mathbf{z}\in\mathbf{Z}_k} \frac{p_{D,k}(\mathbf{x})g_k(\mathbf{z}|\mathbf{x})v_{k|k-1}(\mathbf{x})}{\kappa_k(z) + \int p_{D,k}(\mathbf{x}_{k|k-1})g_k(\mathbf{z}|\mathbf{x}_{k|k-1})v_{k|k-1}(\mathbf{x}_{k|k-1})d\mathbf{x}_{k|k-1}} \end{aligned} \tag{2.36}$$

This recursion does not allow for a closed-form solution in the general case. However, a solution exists for the more specific case of linear motion and measurement models and considering only Gaussian noise on the models.

GM-PHD Filter For The Linear And Gaussian Case

Assuming linear models and only Gaussian noise, the following state transition $\mathbf{f}_{k|k-1}(\mathbf{x}_k|\mathbf{x}_{k-1})$ and likelihood $g_k(\mathbf{z}|\mathbf{x}_k)$ can be formulated. The variables are analogous to the ones used in the general motion and measurement model for the single-target tracking case described in equation (2.6).

$$\begin{aligned} \mathbf{f}_{k|k-1}(\mathbf{x}_k|\mathbf{x}_{k-1}) &= \mathcal{N}(\mathbf{x}_k; \mathbf{F}_{k-1}\mathbf{x}_{k-1},\ \mathbf{Q}_{k-1}) \\ g_k(\mathbf{z}_k|\mathbf{x}_k) &= \mathcal{N}(\mathbf{z}_k; \mathbf{H}_k\mathbf{x}_{k|k-1},\ \mathbf{R}_k) \end{aligned} \tag{2.37}$$

The resulting Gaussian distribution is the key component in the linear, Gaussian Mixture formulation of the PHD (GM-PHD) filter. Both the state RFS $\mathbf{X}_k$ and the measurement RFS $\mathbf{Z}_k$ consist of Gaussian components. This multimodal distribution over the state space for every time-step k is what can be seen in Figure 2.2.

Since the Bayesian recursion always needs prior information to work on, a birth RFS $\gamma_k(\mathbf{x})$ has to be defined. The components in the birth RFS describe regions in the state-space in which new targets are most likely to appear. This could e.g. be an airport in the case of a flight-surveillance system or the borders of the sensor-range around an autonomous car that wants to track new objects coming into view. The birth RFS is a multimodal distribution comprised as a Gaussian Mixture of components [9].

$$\gamma_k(\mathbf{x}) = \sum_{i=1}^{J_{\gamma,k}} w_{\gamma,k}^{(i)} \mathcal{N}(\mathbf{x}_k; \mathbf{m}_{\gamma,k}^{(i)}, \mathbf{P}_{\gamma,k}^{(i)}) \tag{2.38}$$

The mean and covariance of the Gaussian components are used to position the distribution in the state-space and the weight $w_{\gamma,k}^{(i)}$ is used to determine the expected number of targets within that distribution. $J_{\gamma,k}$ is the number of components in the set. Just like the Gaussian components, their associated weights will subsequently be updated in the Bayesian recursion and are then used to identify the most likely target states.

Now, assuming that the posterior PHD at time $k-1$ is a multimodal Gaussian distribution

$$\upsilon_{k-1}(\mathbf{x}) = \sum_{i=1}^{J_{k-1}} w_{k-1}^{(i)} \mathcal{N}(\mathbf{x}_k; \mathbf{m}_{k-1}^{(i)}, \mathbf{P}_{k-1}^{(i)}) \tag{2.39}$$

the predicted PHD at time k is a multimodal Gaussian distribution as well

$$\upsilon_{k|k-1}(\mathbf{x}) = \upsilon_{S,k|k-1}(\mathbf{x}) + \gamma_k(\mathbf{x}) \tag{2.40}$$

i.e. the predictions of the surviving targets from $k-1$ and the new births at k. Previous targets survive according to the defined survival probability $p_{S,k}$ and their next state is predicted according to the motion model defined in equation (2.37).

$$\upsilon_{S,k|k-1}(\mathbf{x}) = \sum_{j=1}^{J_{k-1}} w_{k-1}^{(j)} \mathcal{N}(\mathbf{x}_k; \mathbf{m}_{S,k|k-1}^{(j)}, \mathbf{P}_{S,k|k-1}^{(j)}) \tag{2.41}$$

From this multimodal Gaussian prediction, the update to the posterior can now be calculated as a sum of two terms. The first term consists of all predictions, assumed to not be detected by measurements at step k and are therefore down-weighted. The second term assumes that all predictions actually are detected and updates them towards each of the measurements received at step k analogous to a Kalman update [12].

$$
\begin{aligned}
v_k(\mathbf{x}) &= (1 - p_{D,k}) v_{k|k-1}(\mathbf{x}) + \sum_{\mathbf{z}_k \in \mathbf{Z}_k} v_{D,k}(\mathbf{x}_k; \mathbf{z}_k) \\
v_{D,k}(\mathbf{x}_k; \mathbf{z}_k) &= \sum_{j=1}^{J_{k|k-1}} w_k^{(j)}(\mathbf{z}_k) \mathcal{N}(\mathbf{x}_k; \mathbf{m}_{k|k}^{(j)}(\mathbf{z}_k), \mathbf{P}_{k|k}^{(j)}) \\
w_k^{(j)}(\mathbf{z}_k) &= \frac{p_{D,k} w_{k|k-1}^{(j)} q_k^{(j)}(\mathbf{z}_k)}{\kappa_k(\mathbf{z}_k) + p_{D,k} \sum_{l=1}^{J_{k|k-1}} w_{k|k-1}^{(l)} q_k^{(l)}(\mathbf{z}_k)} \\
q_k^{(j)}(\mathbf{z}_k) &= \mathcal{N}(\mathbf{z}_k; \mathbf{H}_k \mathbf{m}_{k|k-1}, \mathbf{H}_k \mathbf{P}_{k|k-1} \mathbf{H}_k^T + \mathbf{R}_k) \\
\mathbf{m}_{k|k}^{(j)}(\mathbf{z}_k) &= \mathbf{m}_{k|k-1}^{(j)} + \mathbf{K}_k^{(j)} \left(\mathbf{z}_k - \mathbf{H}_k \mathbf{m}_{k|k-1}^{(j)} \right) \\
\mathbf{P}_{k|k}^{(j)} &= \left[\mathbf{I} - \mathbf{K}_k^{(j)} \mathbf{H}_k \right] \mathbf{P}_{k|k-1}^{(j)} \\
\mathbf{K}_k^{(j)} &= \mathbf{P}_{k|k-1}^{(j)} \mathbf{H}_k^T \left(\mathbf{H}_k \mathbf{P}_{k|k-1}^{(j)} \mathbf{H}_k^T + \mathbf{R}_k \right)^{-1}
\end{aligned}
\tag{2.42}
$$

As mentioned before, apart from propagating the PHD of the state RFS, the PHD filter is also used to recursively update the weight associated with each Gaussian component. Those can then be used to estimate the number of targets present in the current time-step $\hat{N}_k$. Given a previous estimate $\hat{N}_{k|k-1}$ (or an initial estimate for $k = 0$) the current estimate can be computed in the following way.[9]

$$
\begin{aligned}
\hat{N}_{k|k-1} &= p_{S,k} \hat{N}_{k-1} + \sum_{j=1}^{J_{\gamma,k}} w_{\gamma,k}^{(j)} \\
\hat{N}_k &= (1 - p_{D,k}) \hat{N}_{k|k-1} + \sum_{\mathbf{z}_k \in \mathbf{Z}_k} \sum_{j=1}^{J_{k|k-1}} w_k^{(j)}(\mathbf{z}_k)
\end{aligned}
\tag{2.43}
$$

Thus, the PHD filter can be used to jointly estimate the number of targets and their states.

2.3 Extended Target Tracking

The different Bayesian state estimation methods presented in sections 2.1-2.2.1 are all introduced based on the assumption of dealing with point targets - targets that generate at most one measurement per sensor scan. In practice however, especially when using a range sensor such as a lidar, multiple measurements may arise from a target, especially when targets are relatively near the sensor. Such targets are commonly referred to as *Extended Targets*. The distribution of lidar distance measurements arising from a target is dependent on which section of the targets spatial extent is within sensor view. As such, at any given sensor scan only a subset of the extended target will be seen.

A simplistic approach when dealing with multiple measurements arising from a target is to calculate the mean value of all measurement. This enables the extended target to be treated as a point target. Such a method is suitable when dealing with targets whose spatial extent is relatively small, or in cases where the entire extended target is seen in the sensor field of view. However, large and partially viewed extended targets require more complex methods. Calculating the mean value of measurements will result in a loss of information about spatial extent, as well as induce large estimation errors for a target's state. An illustrative example can be seen in Figure 2.3.

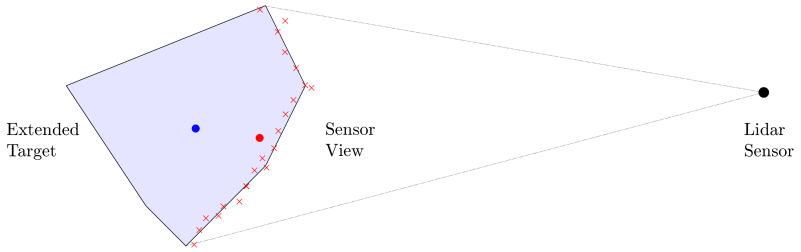


Figure 2.3: An example of how an extended target is viewed by a Lidar sensor. The red x's indicate measurements generated by the lidar Sensor. The small blue circle denotes the center point of the Extended target, and the red small circle denotes the mean value of all measurements.

Here the extended target has an elliptical shape, but only a circular sector is seen from the sensor's field of view. Calculating the mean of all received measurements (the red exes) and treating it as a single measurement (the red circle) of the target's center point will yield a far more inaccurate measurement than the inherent inaccuracy for each individual measurement. Changes in for example heading for the extended target will be seen as changes in position, even though the target might not have necessarily performed any translative movement. Additionally, calculating target extent based solely on the spread of received measurements will underestimate the true spatial extent of the target. Not utilizing prior and historical knowledge about shape severely limit practical usefulness of the method.

In order to accurately estimate the behaviour of extended targets more robust techniques need to be considered, where a target's size and shape are examined as well. The following sections describe two different ways of dealing with extended targets: assuming a rectangular shape or by utilizing random matrices.

2.3.1 Tracking Under Assumed Rectangle Shape

Rectangular shape modelling of targets is a suitable simplified model for describing the spatial extent of cars in cases where the shape and size of a car can be estimated to a high degree. By incorporating two additional parameters into the state vector, rectangle length l and width w [13], the extended state vector can be formulated as (time-step indexing with k is dropped in this section to make equations more readable)

$$\boldsymbol{\xi} = [\mathbf{x}^T,\ w,\ l]^T \tag{2.44}$$

Where $\mathbf{x}$ is the kinematic state-vector. In order to incorporate tracking of extended rectangular targets, an analytical expression of the likelihood must be formulated. For a set $\mathbf{Z} = \{\mathbf{z}^{(j)}\}_{j=1}^{|\mathbf{Z}|}$ of measurements, the likelihood can be expressed as

$$p(\mathbf{Z}|\boldsymbol{\xi}) = p(\mathbf{z}^{(1)}, \mathbf{z}^{(2)}, \ldots, \mathbf{z}^{(|Z|)}|\boldsymbol{\xi}) \tag{2.45}$$

By treating each measurement a as random independent variable drawn from a shape-dependent distribution, the joint likelihood can instead be expressed as the product of each individual measurement likelihood

$$p(\mathbf{Z}|\boldsymbol{\xi}) = \prod_{j=1}^{|\mathbf{Z}|} p(\mathbf{z}^{(j)}|\boldsymbol{\xi}) \tag{2.46}$$

The shape-dependent distribution can be approximated with a Gaussian mixture (GM) model, where the weights of the mixture sum to unity. The number of components N_c in the GM govern how well the mixture approximates the true underlying distribution, where $N_c \to \infty$ asymptotically approximates the distribution completely. For a rectangular shape, the GM approach results in placing N_c Gaussian kernels along the sides of a rectangle.

$$p(\mathbf{z}^{(j)}|\boldsymbol{\xi}) \approx \sum_{i=1}^{N_c} w^{(i)} \mathcal{N}(\mathbf{z}^{(j)}; \boldsymbol{\mu}^{(i)}(\boldsymbol{\xi}), \boldsymbol{\Sigma}^{(i)}(\boldsymbol{\xi})), \qquad \sum_{i=1}^{N_c} w^{(i)} = 1 \tag{2.47}$$

The two equations (2.46) and (2.47) give sufficient analytical expressions for handling extended targets in a Bayesian fashion. By collapsing the GM to just consisting of a single component, each measurement $\mathbf{z}^{(j)}$ can be seen as arising from a measurement generating point (MGP). As such, $\boldsymbol{\mu}^{(i)}(\boldsymbol{\xi})$ represents a non-linear function describing the location of the MGP as dependent on the state $\boldsymbol{\xi}$.

A benefit with this approach is that it allows (2.46) to be expressed in the form of (2.48), where $\mathbf{z}_Z, \boldsymbol{\mu}_Z(\boldsymbol{\xi})$ are vertical concatenations of all measurements and associated MGPs in the set $\mathbf{Z}$, and n_z is the size of each MGPs measurement vector $\mathbf{z}_Z$. This enables the use of standard KF equations for prediction and estimation, negating the use of sequential Monte Carlo (particle filter) methods to evaluate the general forms expressed in (2.46) and (2.47).

$$
\begin{aligned}
p(\mathbf{Z}|\boldsymbol{\xi}) &= \prod_{j=1}^{|\mathbf{Z}|} \mathcal{N}(\mathbf{z}^{(j)}; \boldsymbol{\mu}(\boldsymbol{\xi}), \boldsymbol{\Sigma}(\boldsymbol{\xi})) = \mathcal{N}(\mathbf{z}_Z; \boldsymbol{\mu}_Z(\boldsymbol{\xi}), \sigma_\Sigma^2 \mathbf{I}_{n_z|Z|}) \\
\mathbf{z}_Z &= [\mathbf{z}^{(1)}, \mathbf{z}^{(2)}, \ldots, \mathbf{z}^{(|\mathbf{Z}|)}]^T \\
\boldsymbol{\mu}_Z(\boldsymbol{\xi}) &= [\boldsymbol{\mu}(\boldsymbol{\xi})^{(1)}, \boldsymbol{\mu}(\boldsymbol{\xi})^{(2)}, \ldots, \boldsymbol{\mu}(\boldsymbol{\xi})^{(|\mathbf{Z}|)}]^T
\end{aligned}
\tag{2.48}
$$

However, a difficulty with (2.48) is that it introduces an association problem, since each measurement $\mathbf{z}^{(i)}$ must be associated to a corresponding MGP $\boldsymbol{\mu}(\boldsymbol{\xi})^{(i)}$. A solution to the association problem can be found in [13], however such a solution requires measurements to be sorted according to bearing and incorporates all measurements, making such a method computationally intensive for a high number of measurements. The solution to the association problem in this thesis is presented in Section 3.6.

The state of a rectangular target can be (minimally) described as consisting of positional coordinates x and y, for example specifying the center point of the target. This makes it possible to express positional coordinates in terms of rectangle length l and width w, which are also included in the state vector. Additionally, rotation of the rectangle can be described by its heading ϕ.

$$
\boldsymbol{\xi}_{\text{min}} = [x \;\; y \;\; \phi \;\; w \;\; l]^T \tag{2.49}
$$

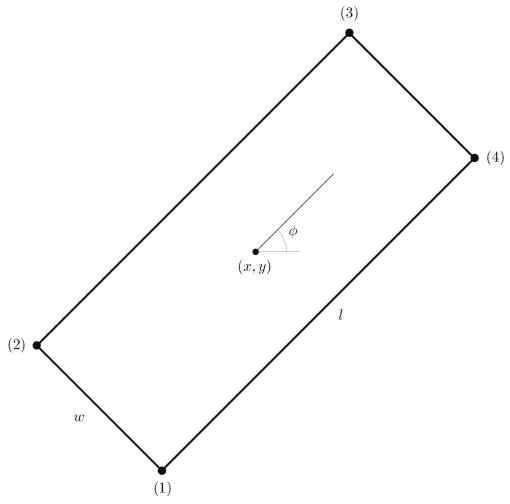


Figure 2.4: Car modelled as a rectangular shape.

Since each MGP is a function dependent on state, $\boldsymbol{\mu}(\boldsymbol{\xi})$, any arbitrary point along any of the sides of the rectangle must be expressed in terms of the components in the state vector. For any scan,

either one or two sides of a car will be seen by the range sensor. As such, each corner position $\{\mathbf{h}_c^i\}_{i=1}^4$ of the rectangle, visualized in Figure 2.4, can be used as base component for describing any arbitrary point along any of the rectangle's sides. For a rectangle where x and y are defined as placed in the center of the rectangular shape, each corner position can be expressed as

$$\begin{aligned}
\mathbf{h}_c^1(\boldsymbol{\xi}_{\min}) &= \begin{bmatrix} x - \frac{1}{2}\sqrt{(w^2+l^2)}\cos(\arctan(w/l)+\phi) \\ y - \frac{1}{2}\sqrt{(w^2+l^2)}\sin(\arctan(w/l)+\phi) \end{bmatrix} \\
\mathbf{h}_c^2(\boldsymbol{\xi}_{\min}) &= \mathbf{h}_c^1(\boldsymbol{\xi}_{\min}) + \begin{bmatrix} w\cos(\phi+\pi/4) \\ w\sin(\phi+\pi/4) \end{bmatrix} \\
\mathbf{h}_c^3(\boldsymbol{\xi}_{\min}) &= \mathbf{h}_c^2(\boldsymbol{\xi}_{\min}) + \begin{bmatrix} l\cos(\phi) \\ l\sin(\phi) \end{bmatrix} \\
\mathbf{h}_c^4(\boldsymbol{\xi}_{\min}) &= \mathbf{h}_c^1(\boldsymbol{\xi}_{\min}) + \begin{bmatrix} l\cos(\phi) \\ l\sin(\phi) \end{bmatrix}
\end{aligned} \tag{2.50}$$

Considering how range sensors work, a reasonable assumption is that measurements are uniformly distributed along each visible side of the rectangle. As such the position for N_{MGP} uniformly distributed MGPs along the width or length, relative to corner i, can be described by

$$\{\mathbf{h}_l^j(\boldsymbol{\xi}_{\min})\}_{j=1}^{N_{\text{MGP}}} = \mathbf{h}_c^i(\boldsymbol{\xi}_{\min}) + \frac{j}{1+N_{\text{MGP}}}\rho_l \begin{bmatrix} l\cos(\phi+\eta(i)) \\ l\sin(\phi+\eta(i)) \end{bmatrix}, \qquad \eta(i) \to \{0,0,\pi,\pi\} \tag{2.51}$$

$$\{\mathbf{h}_w^j(\boldsymbol{\xi}_{\min})\}_{j=1}^{N_{\text{MGP}}} = \mathbf{h}_c^i(\boldsymbol{\xi}_{\min}) + \frac{j}{1+N_{\text{MGP}}}\rho_w \begin{bmatrix} w\cos(\phi+\eta(i)) \\ w\sin(\phi+\eta(i)) \end{bmatrix}, \quad \eta(i) \to \left\{\frac{\pi}{2},\frac{3\pi}{2},\frac{3\pi}{2},\frac{\pi}{2}\right\} \tag{2.52}$$

Where $0 \leq \rho_w,\rho_l \leq 1$ represents how much of each side is in view of the sensor. This is an important parameter, since in practice range sensor may not see the entire side due to factors such as reflectivity or range to target. As such, distributing MGPs only along the viewed length and width is necessary in order to get good estimates of shape and (by extension) kinematic state.

2.3.2 Tracking With Random Matrices

The rectangular shape model for cars is specifically tailored for that particular use case. This, however, makes it impractical to use with other objects that don't show such a shape or general behaviour that can be accurately described by the state vector introduced in equation (2.49). Since this thesis is not only concerned with tracking cars but also other common traffic participants (namely pedestrians and cyclists), an efficient model for estimating their shape and size is required.

It is worth pointing out a principal difference of how cars, pedestrians and cyclists usually show their extent in lidar data. In general, a car is a solid object that is much longer and wider than a pedestrian or cyclist. It therefore has the potential to block out a large portion of its shape from sensor view. An example of this can be observed in Figure 2.5.

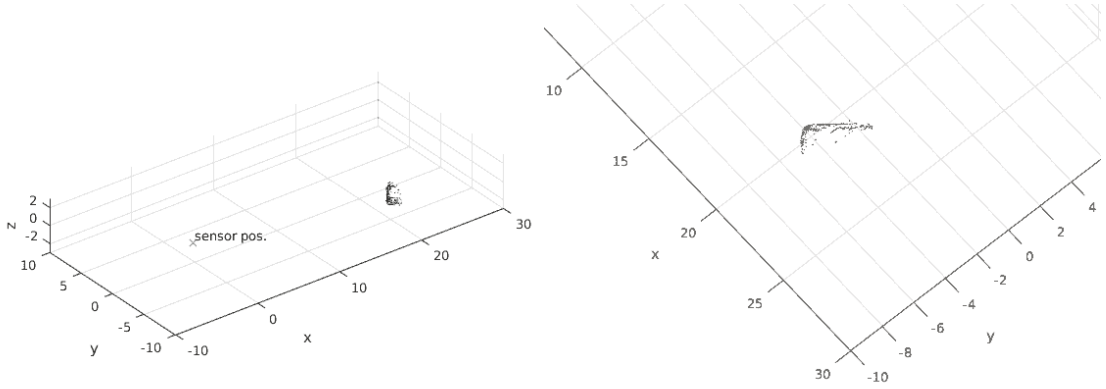


Figure 2.5: A car blocking parts of its own extent from the sensor view, isometric and top-down perspective.

Since a car has relatively large size, this leads to a loss of information about the target. Due to this, the rectangular tracking method is important, since it enables better estimates of a car's true center position. However, this does not hold true for cyclists and pedestrians. They are significantly smaller in size and thus their extent is directly seen in lidar data. This can be seen for the case of a cyclist in Figure 2.6. That is also true for pedestrians, since they are smaller in extent as compared to cyclists.

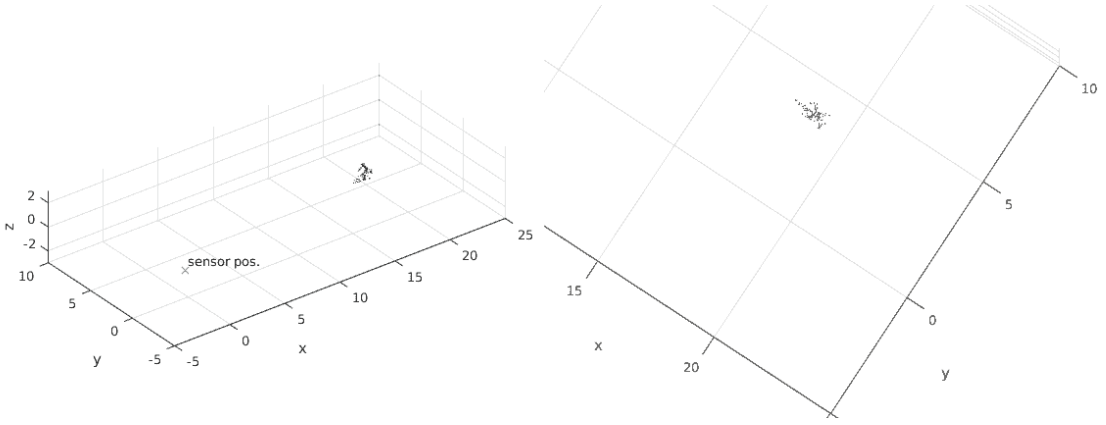


Figure 2.6: A cyclists true extent directly viewed in lidar data, isometric and top-down perspective.

With that in mind, a reasonable approach is to track the observed extent of pedestrians and cyclists in lidar data in order to infer sufficiently good estimates of their true shape and size. This can be achieved by approximating shape and size with ellipses. Ellipses are versatile since

they can approximate different shapes and sizes by adjusting length, width and eccentricity, in addition to being easy to model and incorporate into a tracking framework. Koch, who first used elliptical target tracking within a Bayesian framework, describes that ellipses can effectively be used to track the size, shape and orientation of an extended target [14].

Koch introduces an extended state

$$\boldsymbol{\xi} = \mathbf{x}_k, \mathbf{X}_k \tag{2.53}$$

which models both kinematic state $\mathbf{x}_k$ of a target (e.g. position, velocity, acceleration) as well as target extent $\mathbf{X}_k$, which contains the parameters needed to define the ellipse. $\mathbf{X}_k$ in a Bayesian framework is a random variable and can be modelled as a random matrix drawn from an inverse Wishart distribution [14].

Considering the elliptical shape of level curves for a covariance matrix in a 2-dimensional Gaussian distribution, it becomes clear how an elliptical shape can be derived from a random positive-definite matrix. This is also why it can be modelled as an inverse Wishart distribution, which is a probability distribution over positive-definite matrices. It can be used as a conjugate prior to estimate the random matrix that defines the extent ellipse [15]. Using it as a conjugate prior ensures that the random matrix will, throughout the entire filter recursion, remain inverse Wishart distributed. Drawing a sample from it will therefore always result in obtaining a positive-definite matrix.

An inverse Wishart distribution is described by the following probability density function [14, p.1056]

$$p(\boldsymbol{X}|\mathbf{V},\upsilon) = \frac{|\mathbf{V}|^{\frac{\upsilon}{2}}}{2^{\frac{\upsilon d}{2}}\boldsymbol{\Gamma}_d(\frac{\upsilon}{2})}|\boldsymbol{X}|^{-\frac{\upsilon+d-1}{2}}e^{-\frac{1}{2}tr(\mathbf{V}\boldsymbol{X}^{-1})} \tag{2.54}$$

where $\boldsymbol{\Gamma}_d$ is the multivariate gamma function, tr is the trace function, d is the size $d \times d$ of $\mathbf{X}$, υ is the degrees of freedom of the distribution and $\mathbf{V}$ is the positive definite scale matrix. The expected value of a draw from this distribution is defined as

$$\mathbb{E}[\mathbf{X}] = \frac{\mathbf{V}}{\upsilon - d - 1} \tag{2.55}$$

As can be seen, the probability density function is parametrized by υ and $\mathbf{V}$. The type of ellipses possible to draw from an inverse Wishart distribution can be visualized by drawing several samples for varying degrees of freedom υ. Equation (2.55) states that the expected value is the scale matrix $\mathbf{V}$ scaled down by the degrees of freedom υ subtracted with the dimensions d and 1. The expected value for a sample drawn from an inverse Wishart distribution therefore diminishes for an increasing υ. This can be seen in Figure 2.7. In addition, the variance in the samples decreases, such that the samples converge to a scaled down version of V with little variation in orientation or eccentricity.

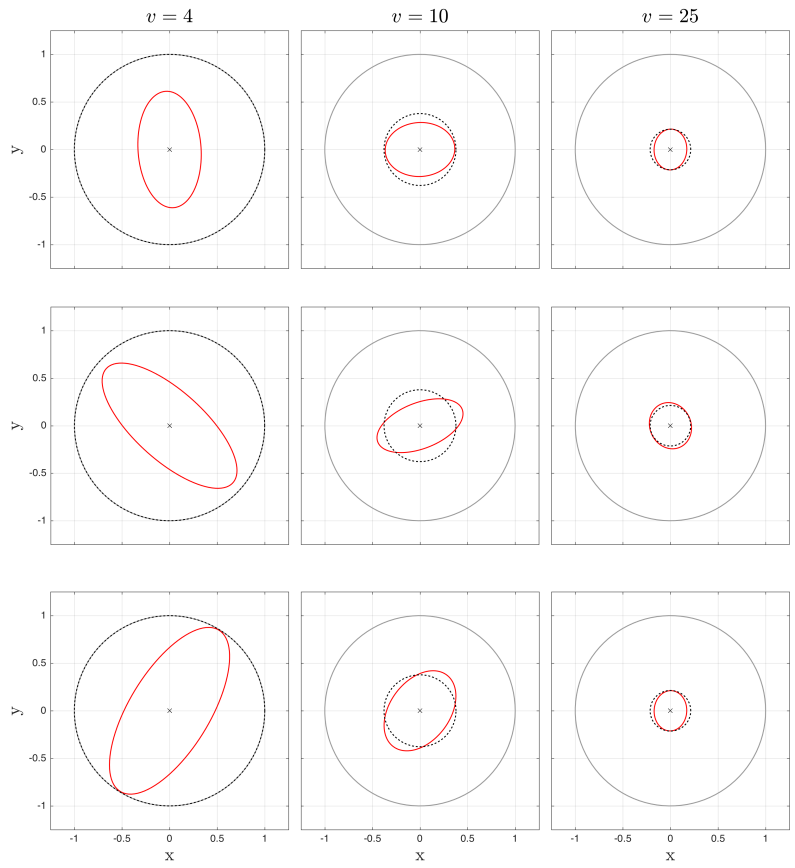


Figure 2.7: 9 inverse Wishart distribution samples for varying degrees of freedom $v = \{4,10,25\}$ and constant $\mathbf{V} = diag([1,1])$. Red circle is 1σ-plot of the drawn sample $\mathbf{X}$, gray circle is 1σ-plot of $\mathbf{V}$ and dashed black circle is $\mathbb{E}[\mathbf{X}]$. Note $\mathbb{E}[\mathbf{X}] = \mathbf{V}$ for $v = 4$.

A closed-form Bayesian recursion to propagate both the kinematic state $\mathbf{x}_k$ and the extent of a target $\mathbf{X}_k$ was developed by Koch in [14]. It follows the standard Bayesian filtering steps of prediction through a motion model and update through a likelihood function based on a measurement model. The kinematic state is modelled as a Gaussian distribution parametrized by the mean $\mathbf{x}_k$ and covariance $\mathbf{P}_k$, while the extent is modelled as an inverse Wishart distribution as described before.

Prediction
The kinematic part:

$$\begin{aligned} \mathbf{x}_{k|k-1} &= (\mathbf{F}_{k|k-1} \otimes \mathbf{I}_d)\mathbf{x}_{k-1|k-1} \\ \mathbf{P}_{k|k-1} &= \mathbf{F}_{k|k-1}\mathbf{P}_{k|k-1}\mathbf{F}_{k|k-1}^T + \mathbf{Q}_{k|k-1} \end{aligned} \tag{2.56}$$

The extent part:

$$\begin{aligned} \upsilon_{k|k-1} &= e^{-T/\tau}\upsilon_{k-1|k-1} \\ \boldsymbol{V}_{k|k-1} &= \frac{e^{-T/\tau}\upsilon_{k-1|k-1} - d - 1}{\upsilon_{k-1|k-1} - d - 1}\boldsymbol{V}_{k-1|k-1} \end{aligned} \tag{2.57}$$

Here T is the sample time of the process and τ a decay constant which determines how much υ and $\mathbf{V}$ decrease in the prediction step. Lower values for υ and $\mathbf{V}$ can be interpreted as higher uncertainty in the estimates drawn from the distribution, as illustrated in Figure 2.7. $\mathbf{I}_d$ is the identity matrix of dimension d. It should be noted that $\mathbf{F}_{k-1|k-1}$ models motion for a single dimension, which is why $\mathbf{I}_d$ is used to specify the number of total dimensions.

Update
The center $\mathbf{z}_k$ and scatter matrix $\mathbf{Z}_k$ of a measurement cluster with n_k measurements:

$$\begin{aligned} \mathbf{z}_k &= \frac{1}{n_k}\sum_{j=1}^{n_k}\mathbf{z}_k^j \\ \mathbf{Z}_k &= \sum_{j=1}^{n_k}(\mathbf{z}_k^j - \mathbf{z}_k)(\cdot)^T \end{aligned} \tag{2.58}$$

The kinematic part:

$$\begin{aligned} S_{k|k-1} &= \mathbf{H}_k\mathbf{P}_{k|k-1}\mathbf{H}_k^T + \frac{1}{n_k} \\ \mathbf{K}_{k|k-1} &= \mathbf{P}_{k|k-1}\mathbf{H}_k^T S_{k|k-1}^{-1} \\ \mathbf{x}_{k|k} &= \mathbf{x}_{k|k-1} + (\mathbf{K}_{k|k-1}\otimes\mathbf{I}_d)(\mathbf{z}_k - (\mathbf{H}_k\otimes\mathbf{I}_d)\mathbf{x}_{k|k-1}) \\ \mathbf{P}_{k|k} &= \mathbf{P}_{k|k-1} - \mathbf{K}_{k|k-1}S_{k|k-1}\mathbf{K}_{k|k-1}^T \end{aligned} \tag{2.59}$$

Where $\mathbf{S}_{k|k-1}$ is the innovation factor (in other literature often called the measurement covariance) and $\mathbf{K}_{k|k-1}$ is the gain.

The extent part:

$$\begin{aligned} \mathbf{N}_{k|k-1} &= S_{k|k-1}^{-1}(\mathbf{z}_k - (\mathbf{H}_k\otimes\mathbf{I}_d)\mathbf{x}_{k|k-1})(\mathbf{z}_k - (\mathbf{H}_k\otimes\mathbf{I}_d)\mathbf{x}_{k|k-1})^T \\ \upsilon_{k|k} &= \upsilon_{k|k-1} + n_k \\ \mathbf{V}_{k|k} &= \mathbf{V}_{k|k-1} + \mathbf{N}_{k|k-1} + \mathbf{Z}_k \end{aligned} \tag{2.60}$$

Where $\mathbf{N}_{k|k-1}$ is the innovation matrix.

For derivation of these equations, details on assumptions made and theoretical background the reader is referred to [14].

Random Matrix PHD Filter

Propagation of random matrices to estimate the extent of a target can be included into a basic PHD Filter framework as described in section 2.2.1 and 2.2.2. Instead of propagating a Gaussian mixture RFS there is now a Gaussian component (for the kinematic state) and an inverse Wishart component (for the extent) associated with every target. Granström and Orguner showed an implementation in [16].

The main problem to solve is to derive a suitable likelihood function that can be used to update the components' weights. Based on predictions about the kinematic state and extent of a target together with received measurements, the target for which measurements fit best needs to be found.

Prediction
At first, each of the $J_{k-1|k-1}$ existing prior component's weight is adjusted based on the survival probability p_S.

$$w_{k|k-1}^{(i)} = P_s w_{k-1|k-1}^{(i)}, \quad \forall i \in J_{k-1|k-1} \tag{2.61}$$

Update
The update step is partitioned into two parts. In the case of no available measurement (no detection) for a particular target the weight is lowered

$$w_{k|k}^{(j)} = (1 - (1 - e^{-\gamma(j)})p_D^{(j)})w_{k|k-1}^{(j)}, \quad \forall j \in J_{k|k-1} \tag{2.62}$$

with $\gamma(j)$ being the expected number of measurements for that target.

Otherwise the weight is updated according to a likelihood function that says how well the shape and position of the predicted ellipse matches the position and shape of the updated ellipse. Every measurement $\mathbf{z}_i \in \mathbf{Z}_k$ is used to update and calculate a likelihood for every existing predicted target $j \in J_{k|k-1}$.

$$\begin{aligned} w_{k|k}^{(j+J_{k|k-1}(i-1))} = & \frac{e^{-\gamma^{(j)}}(\gamma^{(j)})^{|\mathbf{z}_i|}p_D^{(j)}}{\beta_k^{|\mathbf{z}_i|}(\pi^{|\mathbf{z}_i|}|\mathbf{z}_i|S)^{\frac{d}{2}}} \frac{|\mathbf{V}_{k|k-1}^{(j)}|^{\upsilon_{k|k-1}^{(j)}/2}}{|\mathbf{V}_{k|k}^{(j+J_{k|k-1}(i-1))}|^{V_{k|k}^{(j+J_{k|k-1}(i-1))}/2}} \\ & \times \frac{\mathbf{\Gamma}_d(\upsilon_{k|k}^{(j+J_{k|k-1}(i-1))}/2)}{\mathbf{\Gamma}_d(\upsilon_{k|k}^{j}/2)} w_{k|k-1}^{j}, \forall j \in J_{k|k-1}, \forall \mathbf{z}_i \in \mathbf{Z}_k \end{aligned} \tag{2.63}$$

with $\mathbf{\Gamma}_d$ being the multivariate gamma function, $|\mathbf{z}_i|$ being the number of points in measurement $\mathbf{z}_i$, β_k being the expected number of clutter measurements in $\mathbf{Z}_k$ and $|\mathbf{V}|$ denoting the determinant of matrix $\mathbf{V}$. The number of measurements arising from a target is modelled by a Poisson distribution, with parameter γ. Both $\upsilon_{k|k}^{(j+J_{k|k-1}(i-1))}$ and $\mathbf{V}_{k|k}^{(j+J_{k|k-1}(i-1))}$ are the up-

dated parameters of the inverse Wishart distribution, the computation of which can be seen in [16].

All updated predictions for a certain measurement $\mathbf{z}_i$ are normalized amongst themselves in the following way.

$$\begin{aligned} d_{\mathbf{z}_i} &= \delta_{|\mathbf{z}_i|,1} + \sum_{j=1}^{J_{k|k-1}} w_{k|k}^{(j+J_{k|k-1}(i-1))} \\ w_{k|k}^{(j+J_{k|k-1}(i-1))} &= \frac{w_{k|k}^{(j+J_{k|k-1}(i-1))}}{d_{\mathbf{z}_i}}, \quad \forall j \in J_{k|k-1} \end{aligned} \tag{2.64}$$

$\delta_{|z_i|,1}$ is the Kronecker delta which evaluates to 1 if the two subscripts are equal or to 0 otherwise.

For more details on the Gaussian inverse Wishart PHD (GIW-PHD) filter the reader is referred to article [16] and the accompanying technical report [17] that provides pseudo-code for an implementation and also talks about implementation issues and the computational complexity of the filter.

2.4 Neural Networks for Classification

This section aims to explain the basic algorithms used by two common Machine Learning tools to solve classification tasks: Logistic Regression and Neural Networks. A Neural Network for classification can be seen as a combination of several logistic regression blocks. As such, this explains the order in which the concepts are introduced.

A neural network is trained on labeled data, i.e. there are output labels for every data sample. The employed algorithm therefore falls into the category of *supervised learning*.

The theory explained in this section is based on [18], [19] and [20]. The reader is encouraged to turn to those books for a much more in-depth explanation of the following topics.

2.4.1 Logistic Regression

At the root of logistic regression lies the desire to find an appropriate function that defines a boundary between different classes present in data. In this thesis this is e.g. the decision whether a certain object is a car or not. An example of such a boundary can be seen on the left side in Figure 2.8 for a case where a linear function works well to separate the two areas. The red circles are examples for one class in the dataset, the blue crosses signify the other class. More difficult cases like the one shown on the right side in Figure 2.8 cannot be separated by a linear function, a nonlinear function is needed to differentiate between them. For a growing complexity of separation between different classes or a growing number of input dimensions (both examples are in 2D) more and more complex separation functions are needed.

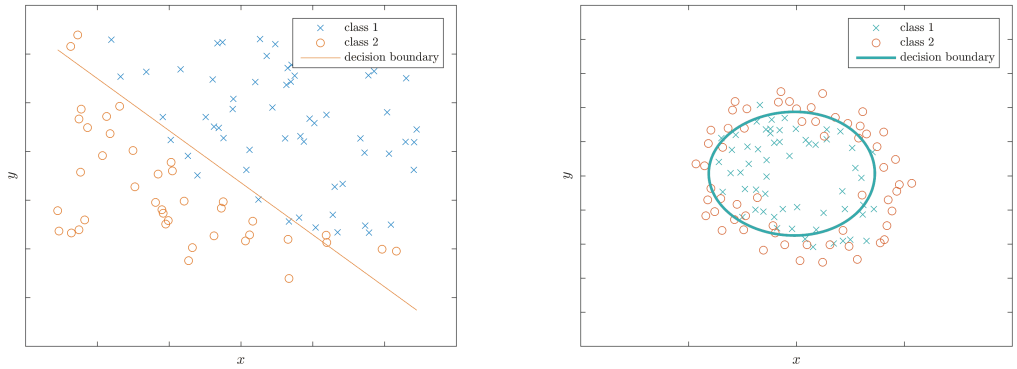


Figure 2.8: Logistic regression with a linear and a polynomial boundary.

The x- and y-domains in Figure 2.8 are the so-called features of that particular classification problem. Features are carefully chosen properties of the objects to be classified. In this thesis this could e.g. be the density and the number of points in a point cluster. Those features need to have somewhat unique values for different classes in the dataset such that the respective examples (i.e. the datapoints in Figure 2.8) actually end up in different parts of the plot and are therefore separable.

Plotting such datasets and their decision boundary is only useful for the 2-dimensional (data with 2 features, separated by a line or curve) and 3-dimensional case (data with 3 features, separated by a plane or a curved plane). Humans generally lack the the ability to visualize e.g. 4-dimensional features separated by a 3-dimensional plane. However, the following concepts and equations hold true for the general case of an n-dimensional space separated in the dimension of it's hyperspace.

Logistic Regression Cost Function

Considering a column vector of features $\mathbf{X}$ and a scalar binary class label (between 0 and 1) Y, the relation between them can be expressed as the linear combination of $\mathbf{X}$ with a column vector of weights $\mathbf{\Theta}$. For p features, the following hypothesis equation h_Θ holds.

$$Y = h_{\mathbf{\Theta}}(\mathbf{X}) = g(\mathbf{\Theta}^T\mathbf{X}) = g(\Theta_1 X_1 + \Theta_2 X_2 + \ldots + \Theta_p X_p) \tag{2.65}$$

With $g(x)$ being the sigmoid function which is applied in order to scale the output to an actual classification value between 0 and 1.

$$g(x) = \frac{1}{1 + e^{-x}} \tag{2.66}$$

A plot of the sigmoid function is shown in Figure 2.9.

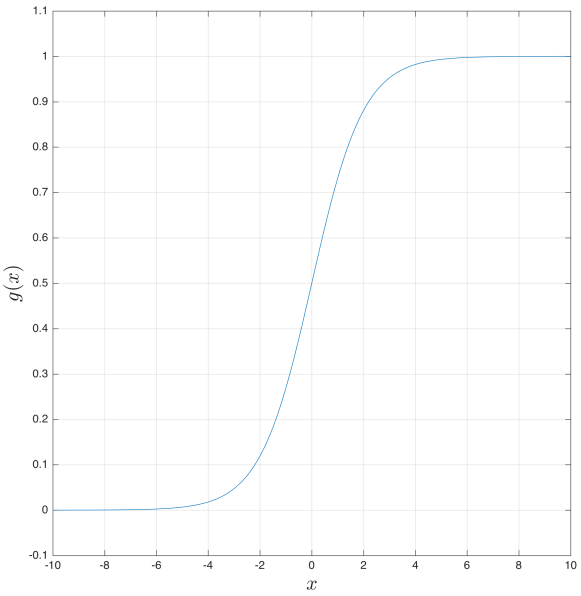


Figure 2.9: Sigmoid function.

In a supervised learning scenario, both the features $\mathbf{X}$ and the true class labels Y_{true} are known. The aim is therefore to find the weight vector $\boldsymbol{\Theta}$ that optimizes the decision boundary, such that the predicted class labels Y_{pred} resemble the true class labels Y_{true} as closely as possible.

This optimal decision boundary is found by optimizing a cost function, a function of $\boldsymbol{\Theta}$, that sums up the difference between the predicted outputs Y_{pred} with that particular weight configuration Θ and the true outputs Y_{true} from the training data. A plot of an example cost function over two $\boldsymbol{\Theta}$ values can be seen in Figure 2.10.

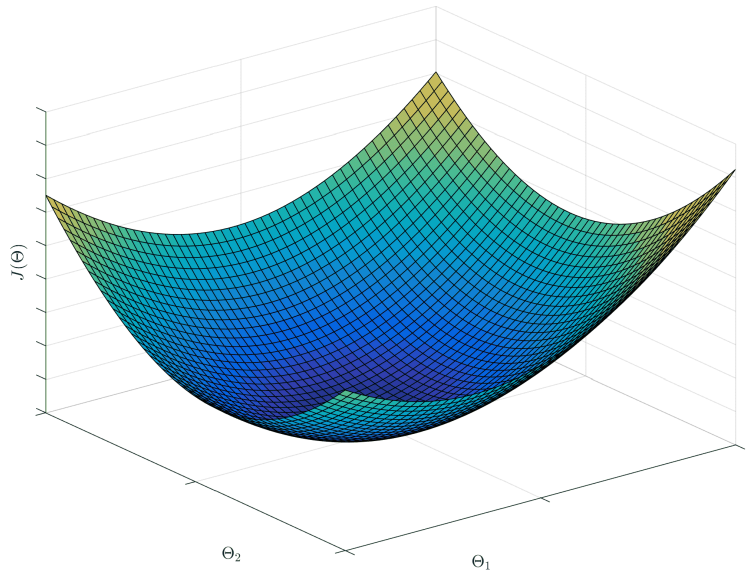


Figure 2.10: An example cost function $J(\boldsymbol{\Theta})$ over sample values in two dimensions of $\boldsymbol{\Theta}$.

The cost function in the case of logistic regression for m data samples and a binary classification class label output Y is as follows.

$$J(\boldsymbol{\Theta}) = \frac{1}{m}\left[\sum_{i=1}^{m} -Y_{\text{true}}^{(i)} \log(h_{\boldsymbol{\Theta}}(\mathbf{X}^{(i)})) - (1 - Y_{\text{true}}^{(i)}) \log(1 - h_{\boldsymbol{\Theta}}(\mathbf{X}^{(i)}))\right] \tag{2.67}$$

Here (i) denotes the ith sample. Extending binary classification to more than two classes is shown when explaining neural networks in 2.4.2.

Equation (2.67) includes two distinct cases for being part of a certain class ($Y = 1$) and the other for not being part of that class ($Y = 0$).

$$J(\boldsymbol{\Theta}) = \frac{1}{m}\sum_{i=1}^{m} \begin{cases} Y_{\text{true}}^{(i)} = 1, & -\log(h_{\boldsymbol{\Theta}}(\mathbf{X}^{(i)})) \\ Y_{\text{true}}^{(i)} = 0, & -\log(1 - h_{\boldsymbol{\Theta}}(\mathbf{X}^{(i)})) \end{cases} \tag{2.68}$$

If the true output Y_{true} is 1 then the cost of the prediction $h_{\boldsymbol{\Theta}}(\mathbf{X})$ is calculated by case 1 in (2.68). If Y_{true} is 0 then the cost is calculated by case 2 in (2.68). Considering the plots for both $-\log(\mathbf{X})$ and $-\log(1 - \mathbf{X})$ in Figure 2.11 it can be seen that the cost for predicting exactly the true output is zero but increases exponentially for deviations further and further away from the true output.

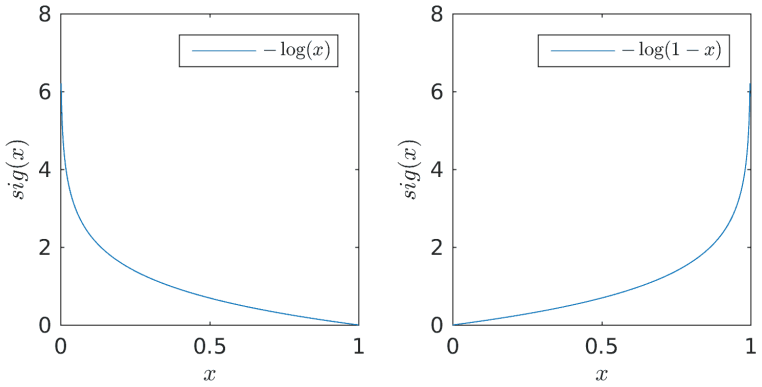


Figure 2.11: Both parts of the logistic regression cost function: $-\log(x)$ and $-\log(1-x)$.

The cost function $J(\mathbf{\Theta})$ sums up the cost for all examples a logistic regression classifier is being trained on. This is then the cost for a certain configuration of $\mathbf{\Theta}$. The aim is to try and find the values for $\mathbf{\Theta}$ that render the lowest possible cost for $J(\mathbf{\Theta})$. Considering the plot of the cost function in Figure 2.10 it can be seen that the goal is to find the bottom of the "bowl-shaped", convex function.

Gradient Descent

In order to find the minimum of the cost function $J(\mathbf{\Theta})$ a technique called gradient descent can be used. An intuitive understanding of the algorithm can be gained by considering Figure 2.10 once more. The starting point would be a random configuration for $\mathbf{\Theta}$. From there, you iteratively subtract the current gradient in all $\mathbf{\Theta}$ directions. You will descend into the "bowl" until you reach the bottom. This can be expressed as follows.

$$\Theta_j = \Theta_j - \alpha \frac{\partial}{\partial \Theta_j} J(\mathbf{\Theta}) \tag{2.69}$$

Here Θ_j is the jth element of the $\mathbf{\Theta}$ vector and α is a learning parameter that controls the speed with which $J(\mathbf{\Theta})$ approaches its minimum. Choosing α too low will result in a slow convergence towards the minimum while choosing α too high can result in missing the minimum and actually diverging. The partial derivative of of $J(\mathbf{\Theta})$ w.r.t a certain element of $\mathbf{\Theta}$ is computed as:

$$\frac{\partial}{\partial \Theta_j} = \sum_{i=1}^{m} \left((h_{\mathbf{\Theta}}(\mathbf{X}^{(i)}) - Y_{\text{true}}^{(i)} \right) X_j^{(i)} \tag{2.70}$$

Again, (i) denotes the ith example in the dataset used to train the classifier, which contains m examples in total. $h_{\mathbf{\Theta}}(\mathbf{X}^{(i)})$ is therefore the ith prediction based on the ith feature-input and $Y_{\text{true}}^{(i)}$ is the ith true class label to compare the prediction with. An example plot of minimizing the cost over subsequent iterations of the gradient descent algorithm can be seen in Figure 2.12.

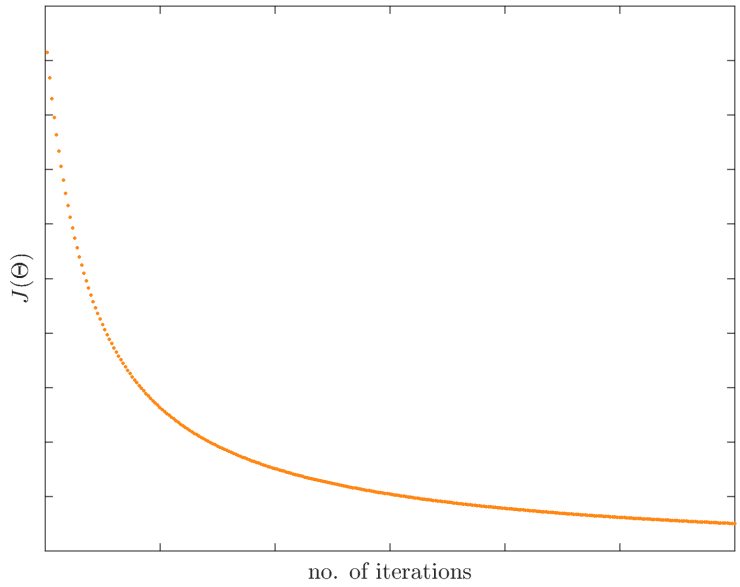


Figure 2.12: Example plot of a converging gradient descent algorithm.

2.4.2 Neural Network

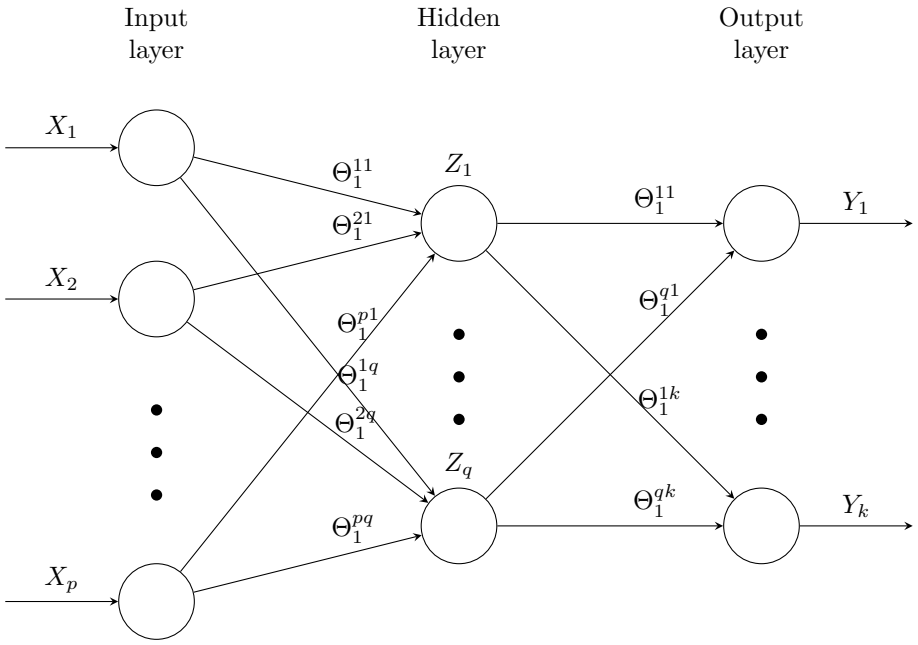
A single hidden layer, fully connected neural network for classification can be described in the following way. [3]

The network is basically a nonlinear function mapping from an input vector of features $\mathbf{X}$ to an output vector of classifications $\mathbf{Y}$. The length of $\mathbf{X}$ is p, with p being the number of features. The length of $\mathbf{Y}$ is k, with k being the number of different classes to be classified.

$$\mathbf{Y} = f(\mathbf{X}) \tag{2.71}$$

Calculation of an output is done by processing the inputs through a neural network structure as depicted in Figure 2.4.2. The structure of the network consists of 3 important building blocks: layers, units and connections. The layers are the vertical groupings seen in Figure 2.4.2. The first is the *input layer*, the second the *hidden layer* and the third the *output layer*. Every layer consists of a number of units (so-called neurons), drawn as the circles in Figure 2.4.2. Each neuron has several input connections from other neurons in the previous layer and several output connections to neurons in the next layer. A connection always has a weight assigned to it.

[3] The reasoning behind using a single hidden layer, fully connected neural network is given in 3.4.



Computation of the output layer is performed in two steps, each advancing one layer forward. As the input layer is given, the first step is to compute the values for the hidden layer. Each unit's value is determined as the linear combination of the respective input neurons' values times the weight of the connection. In order to actually receive a classification value, this value is then scaled to be between 0 and 1 by applying the sigmoid function to it. This function is called the activation function of the neuron.

For example, the value of Z_1 would be computed as

$$Z_1 = g(X_1\Theta_1^{11} + X_2\Theta_1^{21} + ... + X_P\Theta_1^{1P}) \tag{2.72}$$

with Θ_1 being the weight matrix for each connection between the input layer and the hidden layer containing p (number of input units) rows and q (number of hidden units) columns.

This approach can be vectorized to calculate the entire vector of hidden units $\mathbf{Z}$ in the following way.

$$\mathbf{Z} = g(\mathbf{\Theta}_1^T\mathbf{X}) \tag{2.73}$$

The same procedure is repeated from the hidden layer $\mathbf{Z}$ to the output layer $\mathbf{Y}$ to calculate the classification of the inputs.

$$\mathbf{Y} = g(\mathbf{\Theta}_2^T\mathbf{Z}) \tag{2.74}$$

As can be seen from Figure 2.9 the outputs will be values between 0 and 1 and describe the probability of the input to belong to that particular class signified by the respective unit in the output layer.

The entire computation from input to output layer could then be written as follows.

$$\mathbf{Y} = g(\mathbf{\Theta}_2^T g(\mathbf{\Theta}_1^T \mathbf{X})) \tag{2.75}$$

Training the Neural Network

As can be seen from the equations and the principal structure of the network, there are basically 3 ways to influence the mapping from inputs $\mathbf{X}$ to outputs $\mathbf{Y}$:

- number of hidden layers
- number of units in the hidden layers
- connection weights

Backpropagation

With a fixed number of layers and units in the network, the only way to change its output is to adjust the weights of the connections $\mathbf{\Theta}$. Similarly to the gradient descent algorithm described to calculate the optimal values for $\mathbf{\Theta}$ in logistic regression, there is a process to derive the optimal $\mathbf{\Theta}$ configuration for a neural network: backpropagation.

The intuition behind it is similar to gradient descent - the aim is to minimize a cost function. However, instead of one $\mathbf{\Theta}$ vector mapping the inputs $\mathbf{X}$ to the predictions $\mathbf{Y}_{\text{pred}} = h_{\mathbf{\Theta}}(\mathbf{X})$ there are now two $\mathbf{\Theta}$ matrices each describing the weights from one layer to the next. Each of these $\mathbf{\Theta}$ matrices is made up of several vectors, each containing the weights of all units in layer l to a particular unit in layer $l+1$.

Optimizing the entire neural network's $\mathbf{\Theta}$ configuration can therefore be seen as a combination of several local optimizations between the layers. Backpropagation is an iterative approach that computes the partial derivatives of the overall cost function $J_{\mathbf{\Theta}}(\mathbf{X})$ for all elements of $\mathbf{\Theta}$. After that, gradient descent can be used to minimize the cost function like in logistic regression.

The cost function for a neural network is as follows

$$J(\mathbf{\Theta}) = \frac{1}{m}\left[\sum_{i=1}^{m}\sum_{j=1}^{k} -\mathbf{Y}_{\text{true},j}^{(i)} \log((h_{\mathbf{\Theta}}(\mathbf{X}^{(i)}))_j) - (1-\mathbf{Y}_{\text{true},j}^{(i)}) \log(1-(h_{\mathbf{\Theta}}(\mathbf{X}^{(i)}))_j)\right] \tag{2.76}$$

The only difference in comparison with (2.67) is the additional summation over all different classes k.

The algorithm iterates over all training examples from 1 to m and for every input vector it calculates an error term δ for each layer l except for the input layer. For the case of 3 layers the following calculations have to be made to get the error terms for the second and third layers. There is no error term for the first layer, because that is the input layer.

$$\begin{aligned} \boldsymbol{\delta}^{(3)} &= \mathbf{Y}_{\text{pred}} - \mathbf{Y}_{\text{true}} \\ \boldsymbol{\delta}^{(2)} &= \boldsymbol{\Theta}^{(2)}\boldsymbol{\delta}^{(3)} \odot \dot{g}(\mathbf{Z}) \end{aligned} \tag{2.77}$$

with $\dot{g}$ being the derivative of the sigmoid function and $\odot$ denoting element-wise multiplication between two vectors. $\dot{g}$ can be computed as follows

$$\dot{g}(x) = g(x) \odot (1 - g(x)) \tag{2.78}$$

Those error terms for each layer and every input sample are then used to accumulate the share of every connection in the overall error from a node j in layer l to a node i in layer $l+1$, denoted as $\Delta_{ij}^{(l)}$. Those terms are initialized to 0 and then updated on every input as follows

$$\Delta_{ij}^{(l)} = \Delta_{ij}^{(l)} + a_j^{(l)}(\boldsymbol{\delta}^{(l+1)})^T \tag{2.79}$$

where $a_j^{(l)}$ denotes the jth element in the value vector for the lth layer, i.e. $\mathbf{a}^{(l)}$ would be $\mathbf{Z}$ for $l = 2$ and $\mathbf{Y}_{\text{pred}}$ for $l = 3$.

This computation can also be done in a vectorized way directly for each layer.

$$\boldsymbol{\Delta}^{(l)} = \boldsymbol{\Delta}^{(l)} + \boldsymbol{\delta}^{(l+1)}(\mathbf{a}^{(l)})^T \tag{2.80}$$

Once the share of every connection in the error in each of the m data samples is accumulated, this value can be used to derive the gradient of the cost function $J(\boldsymbol{\Theta})$. The following relation holds

$$\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\boldsymbol{\Theta}) = \frac{1}{m}\Delta_{ij}^{(l)} \tag{2.81}$$

Those derivative terms can then be used as described in 2.4.1 to run gradient descent in order to minimize the cost function and find a suitable configuration for $\boldsymbol{\Theta}$.

3

Algorithm

The resulting algorithm implemented in this thesis uses the material found in the previous section as a basis for solving the outlined thesis objectives. How to incorporate the different theoretical frameworks together in a single algorithm, in addition to important steps for making the algorithm suitable for practical use are outlined in this chapter. An overview of how the algorithm works can be seen in Figure 3.1, where a single iteration of the Algorithm is outlined. Each component in the Figure corresponds to a section in this chapter. In addition, the chapter begins with a brief description of relevant sensor properties associated with a lidar.

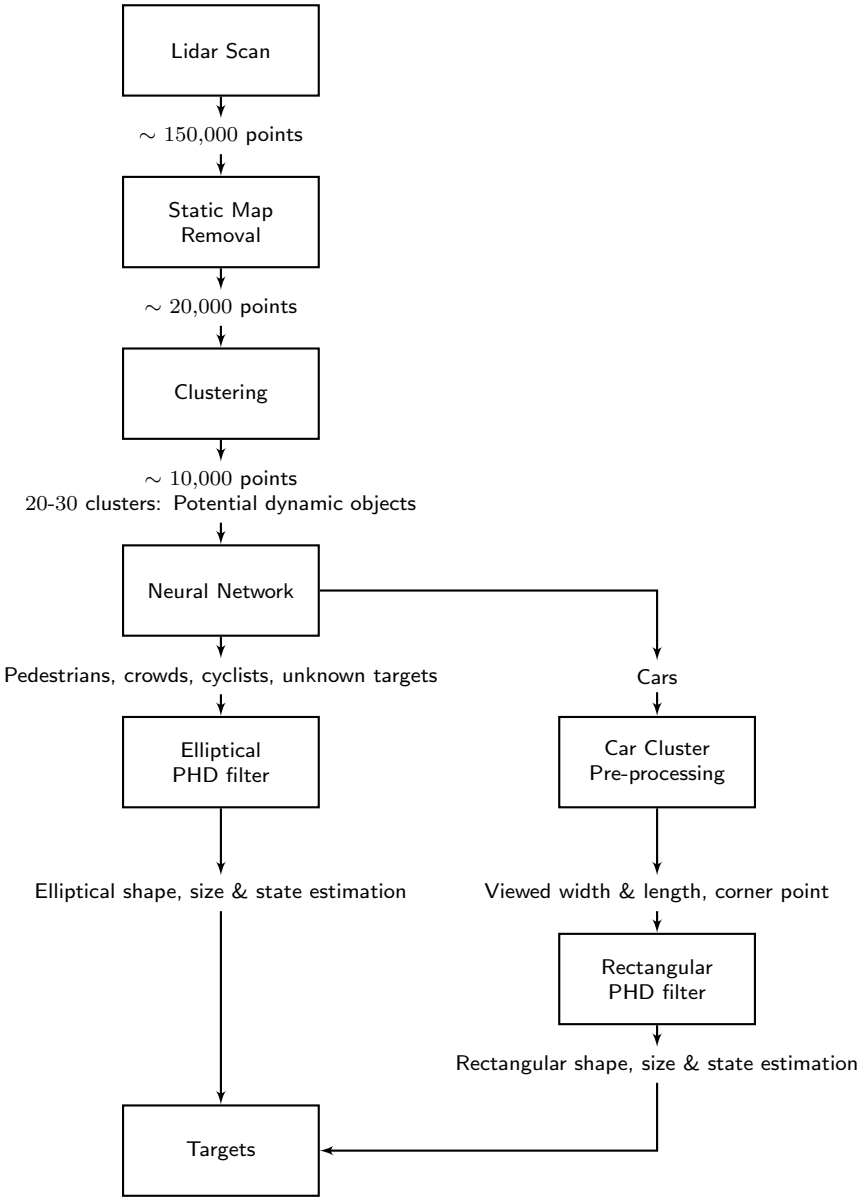


Figure 3.1: A flowchart detailing a single iteration of the algorithm.

3.1 Sensor Properties

Figure 3.2 shows a schematic overview of the lidar sensor. It explains for example why objects further away reflect fewer and fewer beams and therefore consist of fewer and fewer points in the

data. It also shows why there are usually a lot of ground points in the data when the sensor is positioned on top of a car.

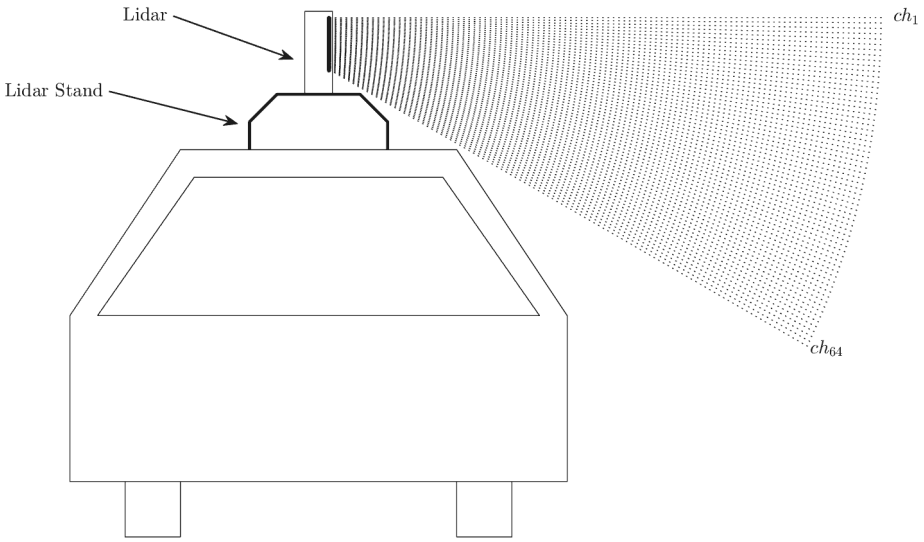


Figure 3.2: Schematic overview of the lidar sensor

The exact sensor model is a *Velodyne HDL-64E*. Some key properties can be seen in Table 3.1. [21]

Table 3.1: Key properties of the Velodyne HDL-64E lidar sensor.

Property	Value
frequency	10Hz[1]
vertical FOV	26.8°
vertical Channels	64
horizontal FOV	360°
horizontal resolution	0.09°
max. distance	$\sim$ 120m

The high resolution results in about 150,000 points per scan, which adds up to 1.5×10^6 points to process every second.

[1]The HDL-64E can be operated in a frequency range from $5 - 15$Hz but the data used in this thesis was gathered with 10Hz.

3.2 Static Point Removal

3.2.1 Ground Removal

In a typical traffic environment, lidar data captured by a sensor mounted on the roof of a vehicle contains a substantial amount of points that stem from the ground. Those usually amount to around 50-70% of the points in one frame (one lidar scan). In this thesis those ground points are not useful for any practical purpose and are therefore discarded.

The employed algorithm lays a grid over the point cloud in the x-y-plane. For every grid cell the points, whose z-value falls within the lower c% of the entire cell, are removed. Figure 3.3 shows the difference between an unprocessed lidar frame and the same frame after the ground removal algorithm.

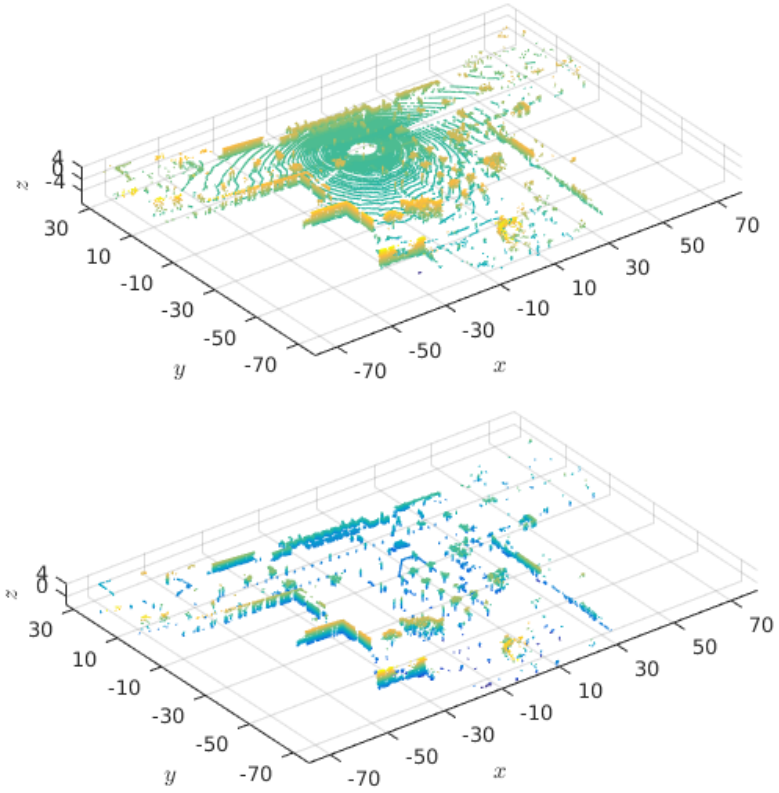


Figure 3.3: An unprocessed frame (top) and the same frame with the ground points removed (bottom).

Algorithm 1 Remove Ground Points

input: Set of points P, number of grid columns/rows n, cut-off c
init: $P_{keep} \leftarrow \emptyset$ as the set of points to keep
  for $i = 1$ to n **do**
    for $j = 1$ to n **do**
      $P_{ij} \leftarrow$ get all points that lie within cell ij
      $z_{ij,max}, z_{ij,min} \leftarrow$ calculate max and min z value for points P_{ij}
      $P_{ij,keep} \leftarrow$ keep only the points that have a z value above $c(z_{ij,max} - z_{ij,min})$
      $P_{keep} \leftarrow (P_{keep} \cup P_{ij,keep})$
    end for
  end for
output: $P \leftarrow P_{keep}$, set of points that are not ground points

The cut-off percentage c and the number of grid columns/rows n are design parameters that influence performance of the algorithm. A higher cut-off will remove ground points more robustly but might impact the quality of processed data by e.g. "cutting off" tyres of cars or lower legs of pedestrians. More grid cells allow for a more fine-tuned result by making it less likely that big and small objects end up in the same cell which usually results in removing substantial parts of the smaller object. However, the smaller the cell size, the longer the computation time of the algorithm.

In this thesis values of $n = 200$ and a cut-off of $c = 0.3$ were found to be a good compromise between performance and speed. The algorithm removes about 80,000 to 110,000 points, emphasizing the large amount of data produced by a lidar sensor. It also provides further proof that pre-processing is needed in order to utilize the high resolution only for the parts of the data that are actually needed.

3.2.2 Static Objects

A core part of this thesis is to determine the eligibility of using a priori information about the environment in order to easily dismiss static parts of the data. The high resolution of a lidar sensor is very useful for gathering detailed information about surrounding targets. Unfortunately this also means that the raw data contains a lot of information about objects that are not useful for the purpose of this thesis. Being able to dismiss static data in a straightforward way could therefore greatly ease the computational load of all following steps of the algorithm.

The static map is made up of simple cuboids that describe walls, buildings and trees. Figure 3.4 shows an example of such a map in comparison with the point cloud data captured by the sensor.

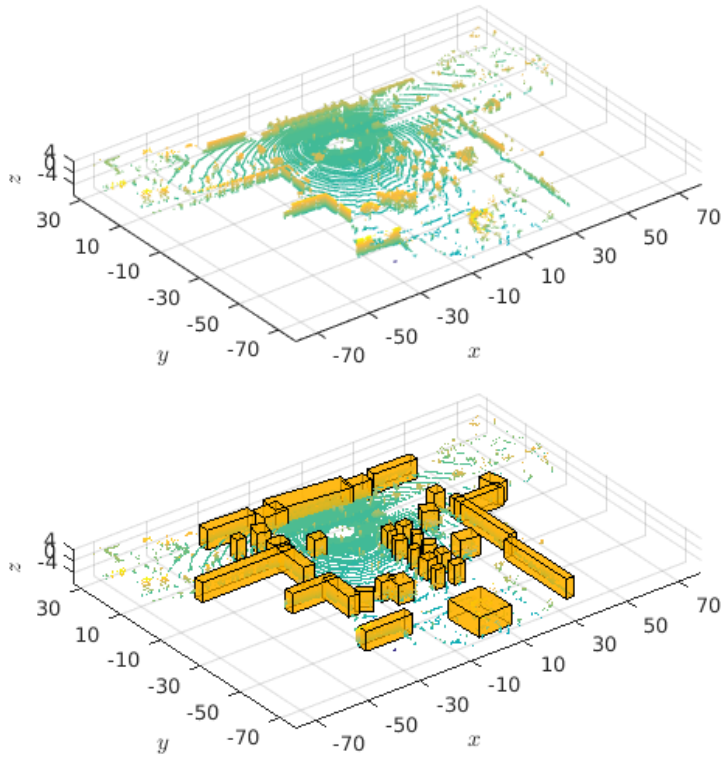


Figure 3.4: Lidar data (top) and the same data overlain by the static map (bottom)[2].

The concept of using prior information about the static environment was suggested by ÅF Technologies Gothenburg were this thesis was written. By measuring the environment with a static lidar sensor, a highly accurate local static map can be built. This point cloud map is then used to derive the cuboids.

Due to scheduling issues, no test data from the official test site (for which the static map existed) was available for this thesis. To still be able to incorporate that concept, the static map over the test data was created manually by placing cuboids over big static objects in the data: buildings and trees. This resembles the original static map very closely.

Cuboids are used instead of the raw point cloud data because a geometrical approach was much faster and more robust than a nearest neighbor algorithm.

Since all corner points p_{cor} and the center of each cuboid p_{cen} are known, a geometrical approach was chosen. First, the normal vector for each of the 6 faces is calculated. Those vectors are then used to determine whether p_{cen} has a positive or negative distance to each of the faces. By checking whether the point to test p_{test} has the same type of distance (negative or positive) towards each face it can be determined whether p_{test} lies within the cuboid or not. The following

[2]N.B. the ground points seen in both images would usually have been removed at this stage of the algorithm but were left in this Figure to make it easier to see where the static map cuboids are positioned within the frame

pseudocode describes the entire process as a nested loop over all points and all cuboids.

In practice it was also found that the algorithm can be sped up significantly by first ordering the set of all cuboids by distance from the current ego position. Since a lidar sensor captures fewer and fewer points the further away an object is, it is reasonable to test against the cuboids in order of increasing distance from the sensor. Any point that is removed at an early stage in the algorithm will not have to be checked against all the other cuboids anymore.

Algorithm 2 Remove Static Points

input: Set of points P, Set of cuboids C
init: Sort C by distance to current ego position (ascending)
  for $j = 1$ to C **do**
    calculate normal vectors for each of the 6 faces of C_j
    $\mathbf{d}_j \leftarrow$ calculate the distance vector of the center c_j
  end for
  for $i = 1$ to P **do**
    for $j = 1$ to C **do**
      $\mathbf{d}_{ij} \leftarrow$ calculate the distance vector of p_i
      if $\mathbf{d}_{ij}./abs(\mathbf{d}_{ij}) - \mathbf{d}_j./abs(\mathbf{d}_j) = \mathbf{0}$ **then**
        remove p_i from P
        break
      end if
    end for
  end for
output: Set of points P that are not within any of the cuboids

The algorithm effectively reduces the data by about 20,000 to 30,000 points and is a key component to speed up the subsequent clustering and filtering.

3.3 Clustering

At this point the pointcloud usually contains somewhere between 10,000 and 20,000 points and looks like the example in Figure 3.5.

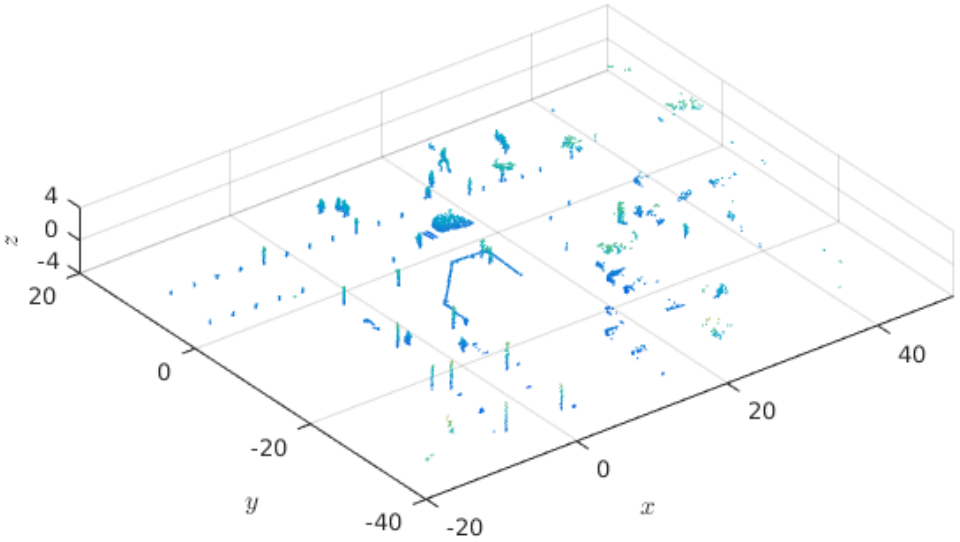


Figure 3.5: A typical pointcloud frame after removing the ground and static objects.

As can be seen, distinct objects appear in the data very clearly. However, the data is still only a set of points with no indication of whether a pair of points might belong to the same object or not. Clustering uses the fact that measurements originating from the same object are assumed to be spatially close.

What kind of clusters are found can be adjusted by two parameters: the minimum number of points in a cluster n_{min} and the maximum distance between any two points in a cluster that are closest to each other d_{max}.

The implemented algorithm builds on creating a k-d-tree structure of the points to be clustered. k-d-trees allow for efficient nearest-neighbour searches (nnsearch) that are important for this application. A k-d-tree is built by continually branching the tree over hyperplanes that divide the entire space into smaller and smaller subspaces. Each point is a node of the tree, with points that are on the same side of any of the dividing hyperplanes ending up in the same branch of the tree. For an introduction to k-d-trees the reader is referred to [22].

Pseudocode for the implemented algorithm can be found in Algorithm 3.

Algorithm 3 Clustering Points

input: Set of points P, m_{min}, d_{max}
init: build a k-d-tree from P, empty set of clusters C
  $i = 0$
  while $P \neq \emptyset$ **do**
    $C_i = \emptyset$, initialize a new empty cluster
    $T \leftarrow P_0$, initialize a new temporary set with the current first point in P
    while $T \neq \emptyset$ **do**
      $X \leftarrow \text{nnsearch}(T_0, d_{max})$, find all neighbors with less than d_{max} distance to T_0
      $T \leftarrow T \cup X$, add all the points X to T
      $C_i \leftarrow C_i \cup T_0$
      $T \leftarrow T \setminus T_0$
    end while
    if $|C_i| \geq m_{min}$ **then**
      $C \leftarrow C \cup C_i$, add the new cluster to the set of clusters
      $P \leftarrow P \setminus C_i$
      $i \leftarrow i + 1$
    else
      $P \leftarrow P \setminus C_i$
      $C_i \leftarrow \emptyset$
    end if
  end while
output: Set of clusters C

In this thesis it was found that values of $n_{min} = 50$ and $d_{max} = 0.7$m delivered good results. Using a fixed threshold is not optimal when using a range sensor like lidar whose data resolution decreases proportionally to the distance from the sensor. A pedestrian that might be defined by ≈ 200 points when 10m away from the sensor, may only end up being composed of ≈ 20 points at 100m distance. However, the mentioned values were found to be robust for all objects of interest within a radius of 50m around the sensor.

Considering Figure 3.5, the goal with clustering is quite diverse. Ideally distinct objects should also be clustered as distinct clusters. This can be quite a challenging task sometimes, e.g. for two pedestrians that walk very close to each other. Also it is beneficial to dismiss tiny objects like tree branches or fence poles that were not discarded by the static point removal and are known to be much smaller than any object this thesis is potentially interested in (i.e. smaller than a pedestrian).

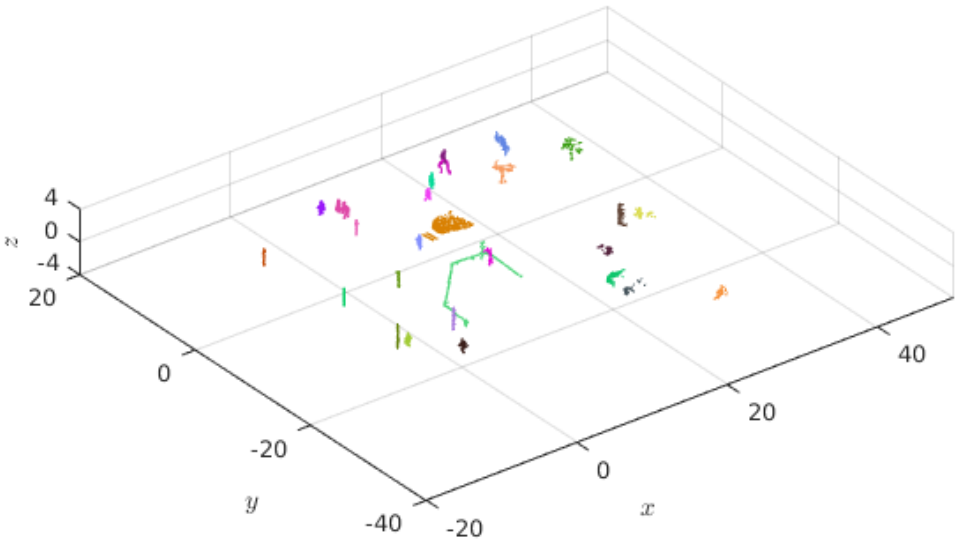


Figure 3.6: A typical pointcloud frame after clustering, different colors indicate different clusters.

Figure 3.6 shows the result of the clustering step performed on the frame in Figure 3.5. It can be seen that a substantial amount of tiny objects could be dismissed and all other major objects are successfully clustered. However, some typical clustering problems can be observed as illustrated in Figure 3.7.

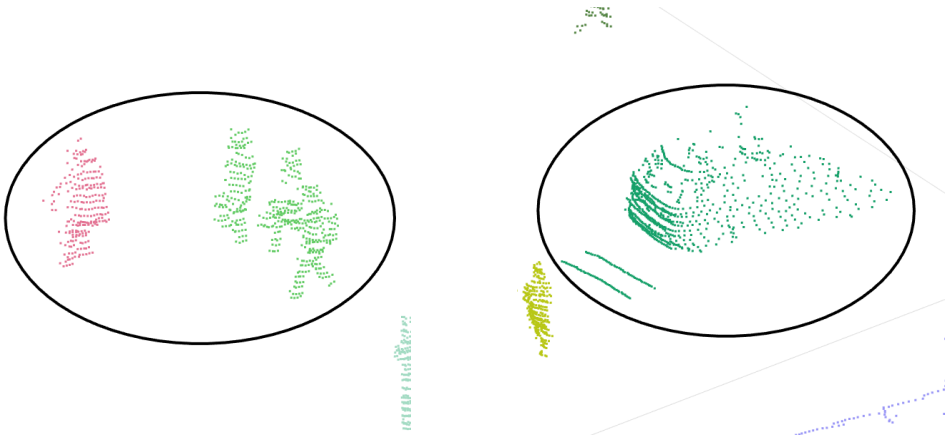


Figure 3.7: Two common clustering problems: a group of 4 pedestrians, 3 of which get clustered into the same cluster (left) and a car that is being clustered together with some nearby ground points (right).

A possible way to remedy this problem is by re-clustering large clusters with a lower distance threshold. However, this also increases computational complexity. As such, performing re-clustering is dependent on performance requirements (accuracy vs. computational complexity).

3.4 Classification

As a first approach a basic single hidden layer, fully connected classification network is implemented for this thesis because it allows to adjust the complexity as needed (up to a certain degree) by adjusting the number of hidden units. It worked so well that a more complex structure was deemed not to be necessary.

Performance of the object-tracking filter can be greatly enhanced by using knowledge about the different objects that will be encountered in typical traffic scenarios. At this point, the pre-processing steps have reduced the large amount of initial points to a small number (usually 10-20) of clusters.

The subsequent filter's task is now to both track all dynamic objects and reject clutter. It is beneficial to the filter's performance to know whether a cluster is an object of interest or clutter (like fragments of walls, trees or poles that were not discarded during the pre-processing). The scope of this thesis limits the former to pedestrians, pedestrian groups, cyclists and cars.

The question is whether the shape of clusters can be used to determine what type of object it is. This is a typical classification problem, which can be solved with a machine learning approach. By looking at lidar data, e.g. Figure 3.7, it becomes apparent that it captures the environment with a clear perception of both depth and shapes. It is easy to detect not only cyclists and cars but also pedestrians with the naked eye, which is usually a meaningful indicator that such a task can indeed be solved by machine learning.

Therefore, it indeed seemed promising to try and classify objects that are within a reasonable range of the sensor. This was found to be about 50m, after that the resolution of objects starts to decrease quite rapidly. Classification also seems like a reasonable idea to pursue in the sense that a Bayesian framework always tries to include every possible prior information available.

Finally, a data-driven algorithm for object classification made sense from a runtime point of view. Initial training might be time-intensive, but the eventual hypothesis function will be a mapping from certain features of a cluster (such as height, length, width or center point) to a classification vector. This function will therefore scale linearly with the number of clusters to classify and will be much faster than any other part of the pipeline. The expected increase in performance is therefore a good trade off.

3.4.1 Features

At the heart of any machine learning algorithm lie the features that are used to quantify each of the available datapoints. In this thesis, those datapoints are the individual lidar clusters. The aim is to choose those features in a way that they have very distinct values for the respective

classes that you expect to appear in the data. Initial guesses for potential features can often be derived from human intuition. Figure 3.8 shows examples of typical lidar representations of the four objects of interest while being approximately equidistant to the sensor position.

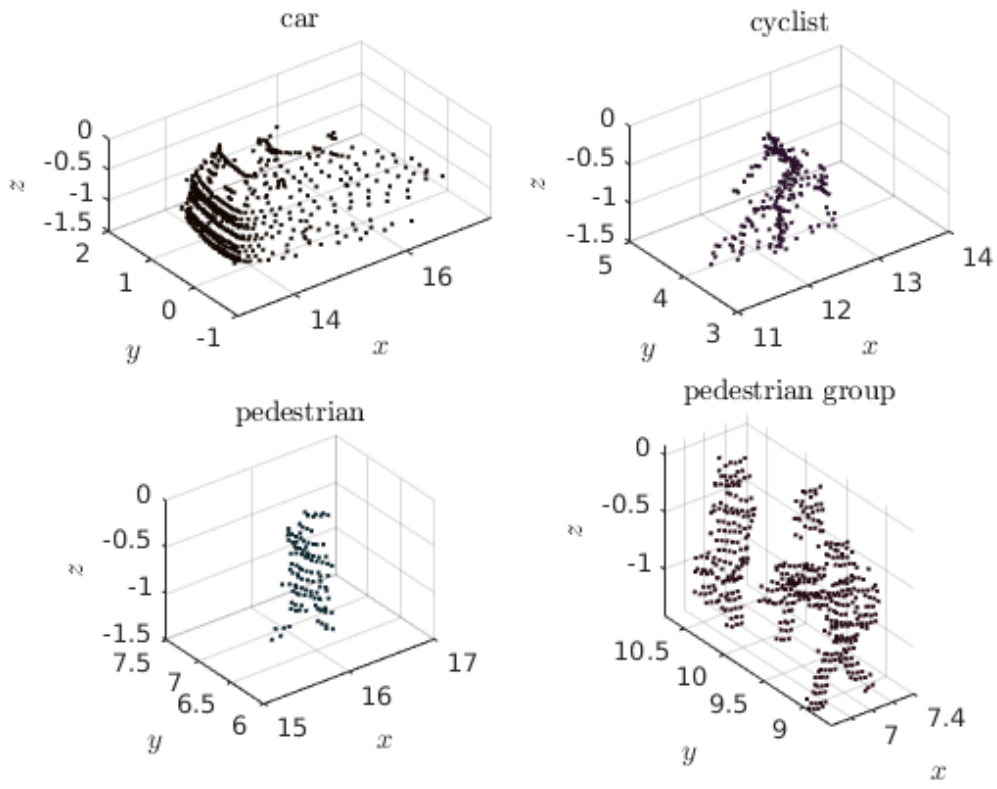


Figure 3.8: Typical lidar representations of the four objects of interest.

As can be seen, there are distinct differences in the dimensions of all four objects especially in x and y direction. Therefore width w, length l and height h of the clusters are considered to be valuable features. Also, while the pedestrian has all it's points spread quite closely around the center, the points of a car or cyclist are more widespread. Thus the point density ρ is viewed as a valuable feature. In addition to that, although not directly visible in Figure 3.8, the total number of points n is different for all four objects. The car consists of $\sim$ 800, the cyclist of $\sim$ 200, the pedestrian of $\sim$ 100 and the pedestrian group of $\sim$ 300 points, which shows that the number of points contains a lot of valuable information. However, since the number of points of an object decrease for an increasing distance to the sensor, the number of points is divided by the distance d of that cluster towards the sensor.

The final feature vector is therefore:

$$f = [w \quad l \quad h \quad \rho \quad \frac{n}{d}]^T \tag{3.1}$$

3.4.2 Training and Testing

The network was trained on 186 frames of manually labeled data from the final example scenario (presented more in detail in the results chapter 4 of this thesis). In total there were almost 5,000 clusters. The dataset was shuffled and then split 60/40 into a training and a test set. As the names say, the training set is solely used for training the network, while the test set is used for subsequent evaluation. By doing so, it is ensures that the algorithm generalizes to new datapoints that it has not seen in the training phase.

However, this training process is slightly flawed. Essentially the training and test set, even though they contain different datapoints, still originate from the exact same scenario. Since labeling data is a very time-consuming process, it was deemed unreasonable for the scope of this thesis to tag several scenarios. Instead, the results of the neural network classification can be regarded as a proof-of-concept that shows that a machine learning approach can successfully and quickly solve the classification task. With an increasing amount of data the algorithm could then be made more general and robust.

The available data samples are quite skewed in the way that there are a lot more clutter clusters than cars, pedestrians or cycles. Table 3.2 shows the composition of the entire dataset.

Table 3.2: Dataset composition showing how many clusters of each class there are.

	Clutter	Cars	Cyclists	Pedestrians	Total
Number	3579	378	164	783	4904
Fraction of total	0.73	0.077	0.033	0.16	1.0

It would therefore be misleading to check performance by computing the accuracy, i.e. the fraction of correct predictions of the network. If the network would only predict ”clutter” on every single input, accuracy would still be about 73%. When working with skewed datasets, precision and recall are more suitable evaluation metrics.

$$\begin{aligned} \text{Precision} &= \frac{\text{True positives}}{\text{true positives} + \text{False positives}} \\ \text{Recall} &= \frac{\text{True positives}}{\text{True positives} + \text{False negatives}} \end{aligned} \tag{3.2}$$

Precision denotes the fraction of e.g. how often the network predicted a car and it actually was a car. Recall denotes the fraction of e.g. how many of all car samples the network correctly predicted as cars. As can be seen, especially recall would have a very low value when only predicting all clusters to be clutter. The performance of the network can be seen in the Results Chapter, Section 4.2.2.

3.5 Elliptical PHD filtering

All clusters that are not classified as cars are used in the elliptical PHD filter. Since classification is never 100% accurate, there is a possibility of an object of interest being classified as clutter. As such, even clutter clusters are used in this filter and will need to be rejected by the filter as targets if necessary. The elliptical PHD filter can be seen as the general purpose extended target tracking filter with good results for all kinds of different objects.

As described in 2.3.2, the elliptical PHD filter follows the general Bayesian recursion of prediction and update. The kinematic state vector is defined as position, velocity and acceleration in both x and y direction.

$$\mathbf{x}_k = [x_k \quad y_k \quad \dot{x}_k \quad \dot{y}_k \quad \ddot{x}_k \quad \ddot{y}_k]^T \tag{3.3}$$

The kinematic motion model with state transition matrix $F_{k|k-1}$ and noise term $Q_{k|k-1}$ are defined according to [14] as follows:

$$\begin{aligned} F_{k|k-1} &= \begin{bmatrix} 1 & T & \frac{1}{2}T^2 \\ 0 & 1 & T \\ 0 & 0 & e^{\frac{-T}{\theta}} \end{bmatrix} \\ Q_{k|k-1} &= \Sigma^2(1 - e^{\frac{-2T}{\theta}}) \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \end{aligned} \tag{3.4}$$

T is sample time, θ is the maneuver correlation time constant and Σ is the scalar acceleration. As you can see, θ is used to scale the negative exponent of an exponential function. This function will therefore result in values between 0 and 1, depending on θ. The state transition $F_{k|k-1}$ therefore scales down the previous time step's acceleration, while the noise term adds a certain value to the acceleration variance.

It is worth noting that both matrices can be written as 3×3. This is possible because the prediction and update steps described in equations (2.56) and (2.59) make use of the Kronecker product to extend the state transition $F_{k|k-1}$ according to the number of desired dimensions, e.g. 2 for tracking x and y or 3 for x, y and z. The noise term $Q_{k|k-1}$ is 3×3 because the covariance term P is also considered to be 3×3 and is then also dynamically extended by using the Kronecker product. This means that no covariance between the states in different dimensions, e.g. x and y, is assumed.

The extent state consists of the two inverse Wishart parameters υ and $\mathbf{V}$ (where $\text{vec}(\mathbf{V})$ is a column vector, the vectorized form of $\mathbf{V}$):

$$\mathbf{X}_e = [\upsilon, \text{vec}(\mathbf{V})^T]^T \tag{3.5}$$

Both the kinematic and the extent state together form the combined Gaussian inverse Wishart (GIW) state component consisting of the following variables:

$$\boldsymbol{\xi} = [x \quad y \quad \dot{x} \quad \dot{y} \quad \ddot{x} \quad \ddot{y} \quad \upsilon \quad \mathrm{vec}(\mathbf{V})^T]^T \tag{3.6}$$

The basic structure of one iteration of the elliptical PHD filter is outlined in the following pseudocode. It can be seen in much more detail in [17].

Algorithm 4 A single PHD iteration for Elliptical Targets.

input: GIW PHD from $k-1$: $v_{k-1}(\xi)$, Birth PHD $\gamma_k(\xi)$, set of m non-car clusters $C_{\neg\mathrm{car}}$
init: calculate center and scatter matrix for all $C_{\neg\mathrm{car}}$ according to equation (2.58)
  for $j = 1, \ldots, |v_{k-1}|$ **do**
    $v_{k|k-1} \leftarrow \mathrm{predict}(v_{k-1})$, according to (2.56), (2.57) and (2.61)
  end for
  $v_{k|k-1} \leftarrow v_{k|k-1} \cup \gamma_k(\xi)$, add birth PHD to the predictions
  update weights for no detection case according to (2.62)
  $v_{k|k} \leftarrow \emptyset$, initialize empty set for all new updated components
  for $i = 1, \ldots, m$ **do**
    for $j = 1, \ldots, |v_{k-1}|$ **do**
      $v_{k|k} \leftarrow v_{k|k} \cup \mathrm{update}(v_{k-1}^j, C^i)$, according to (2.59), (2.60) and (2.63)
    end for
    normalize weights of all components updated with C_i according to (2.64)
  end for
  $v_{k|k} \leftarrow \mathrm{pruning}(v_{k|k}), \quad \mathrm{merging}(v_{k|k})$
output: the updated Gaussian inverse Wishart PHD components $v_{k|k}$

Pruning and Merging are essential steps to reduce the computational complexity of the filter by keeping the number of components in the filter at a minimum. Pruning removes all components whose weight falls below a certain pruning threshold T, while merging aims to join components whose states are below a certain merging distance U to each other. A practical approach to implementing both pruning and merging in a filter with Gaussian inverse Wishart components can be seen in [17] and is therefore not repeated here.

Calculation of the likelihood in (2.63) is susceptible for computational errors since it computes the ratio of very large numbers. This is especially true when working with lidar data, as some values like e.g. $v_{k|k}$ are directly influenced by the number of measurement points in a cluster. Due to the high resolution of lidar data, the number of points per cluster tends to be much higher than with other sensors like e.g. radar. It is therefore essential to use a log likelihood weight update in order to not run into out of bounds errors when calculating the values. This is also suggested in [17].

3.6 Car Cluster Pre-Processing

Each cluster of measurements classified as originating from a car needs several steps of pre-processing before being incorporated into rectangular target tracking. One necessary step is identifying which sides of the car are in view of the lidar sensor. This is especially important, since at any scan either one or two sides of the car will be visible. Distinguishing between the two is important when for example determining viewed length and width, important parameters when generating MGPs. Additionally, measurements in each cluster should be assigned for later use in the rectangular PHD filter.

3.6.1 L-Shape Fitting With Height Based Filtering

A way to identify which measurements belong to which side is by use of Incremental Sub-Matrix Eigen Decomposition (ISED), first proposed in [23] and initially developed for laser scanner data. With ISED an L-shape is efficiently fitted to 2D data in a least-squares optimal way. By fitting an L-shape, the corner point between the two sides can be identified, as well as the two orthogonal lines spanning each side. However, a prerequisite for this method are measurements sorted according to bearing, i.e. clockwise or counter clockwise. This does not hold for lidar sensor data, which necessitates the need to sort measurements according to bearing before using the method.

The simple approach for utilizing ISED is to project all measurements onto the XY-plane, sort them according to bearing (the lidar sensor is located at origin), and then use the ISED method to find the corner point as well as two lines spanning each side. However, such an approach suffers from several drawbacks. A problem with projecting all measurements onto the XY-plane is that several measurements will be from the inside of the car. This is due to the fact that lidar beams can pass through transparent objects (like car windows), and then get reflected back from within the car. Additionally, several clutter measurements can also be present. For example, ground measurements can still be present, or measurements arising from steam emitted from the exhaust-pipe. As a consequence, this will result in poor estimates from the ISED method.

This problem can be alleviated by taking into account how lidar measurements are distributed and by considering the geometric shape of cars. For one lidar scan, the sensor is rotated 360°. Each of the sensor's 64 channels are vertically spaced from each other, each with a different downward tilt. As such, measurements arising from a car are distributed vertically in the form of several horizontal *layers*, where each layer is at a different height (see Figure 3.9 for illustration). Due to the general geometric shape of cars, the further up along the height of the car a layer is, the less measurements it will consist of. This is due to increasing reflectivity from windows and curved surfaces (e.g. the hood of the car). In addition, a car's vertical spatial extent decreases with its height.

With these properties in mind, the mean height μ_z of all measurements should be skewed towards measurement layers not encompassing any of the car's windows or curved surfaces. As such, measurements can first be filtered based on height, where measurements within in a bound around the mean height μ_z are kept.

The remaining measurements are then used in the ISED method. This will not only result in a

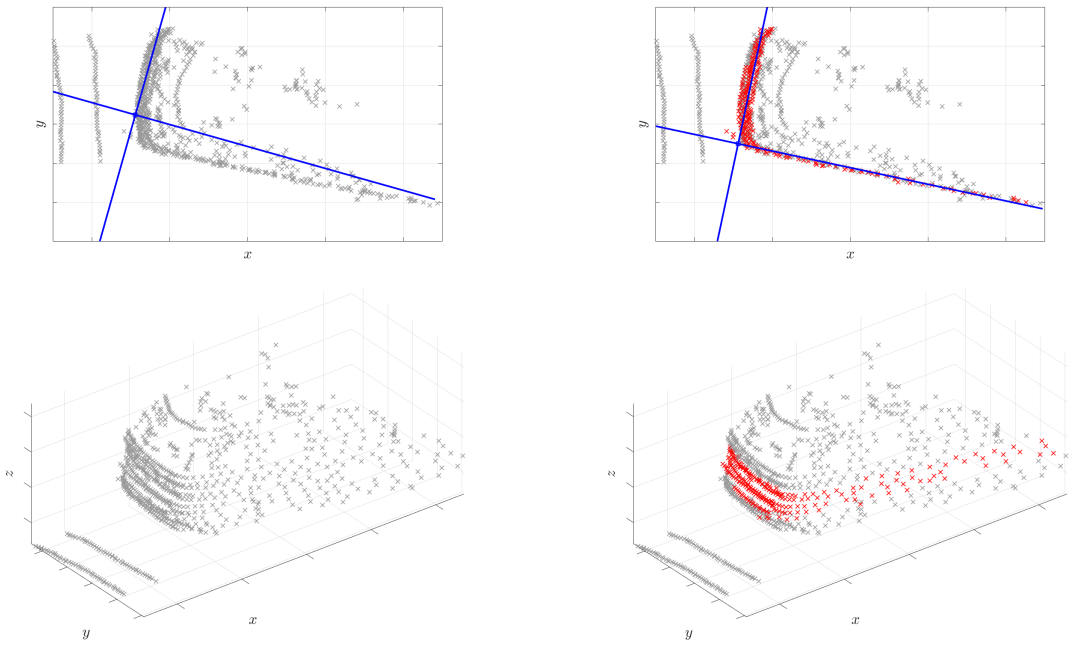


Figure 3.9: Top left corner illustrates the fitted L-shape (in blue), using all clustered measurements. Lower left corner illustrates all clustered measurements in 3D. Top right corner illustrates the fitted L-shape (in blue) using height-filtered measurements (in red). Lower right corner illustrates where height-filtered measurements (in red) are located in the car cluster in 3D.

more accurate estimate from ISED, but it also reduces computational complexity, since less measurements need to be sorted according to bearing beforehand. Additionally, since computational complexity in ISED scales linearly with the amount of measurements, reducing the amount of measurements will also result in more computationally efficient L-shape fitting.

In this work the following bound was used for height-based filtering of car clusters measurements, where σ_z is the standard deviation of the height of all measurements. The bound is skewed more towards lower layers since these layers encompass more surface area of the car, while measurements from layers above the mean risk encompassing windows and curved surfaces. This choice in parameters resulted on average in a 75% decrease of the number of measurements used for L-shape fitting.

$$\mu_z - 0.50\sigma_z < z_k^i < \mu_z + 0.25\sigma_z \tag{3.7}$$

3.6.2 Estimating length of viewed sides

The corner point and orthogonal lines from ISED can be used to estimate the length of each viewed side. The remaining measurements from the filtering step are projected onto each of the orthogonal lines. The two largest distances between the corner point and a measurement from

each set of projected measurements correspond to the viewed lengths.

It is important to note that at this step in the algorithm it is not possible to distinguish between which side corresponds to a car's width or length, and by extension which car corner is in view. This is dependent on the predicted car state, see section 3.7.2 for more information.

3.6.3 Measurement Selection

An important problem when tracking rectangular targets is how to assign measurements to each MGP. One solution to this problem, when dealing with radar measurements sorted by bearing, involves associating an MGP to each measurement present in the car cluster, see [13]. Such a solution has two general drawbacks when dealing with a lidar sensor. Firstly, the large amount of measurements per car will result in increased computational complexity. Secondly, such a solution is based on the assumption that measurements are uniformly distributed along each side of the car. This assumption does not hold when using lidar data, since other objects placed in between the lidar sensor and surrounding cars will result in regions of some cars being subject to occlusion, as in Figure 3.10.

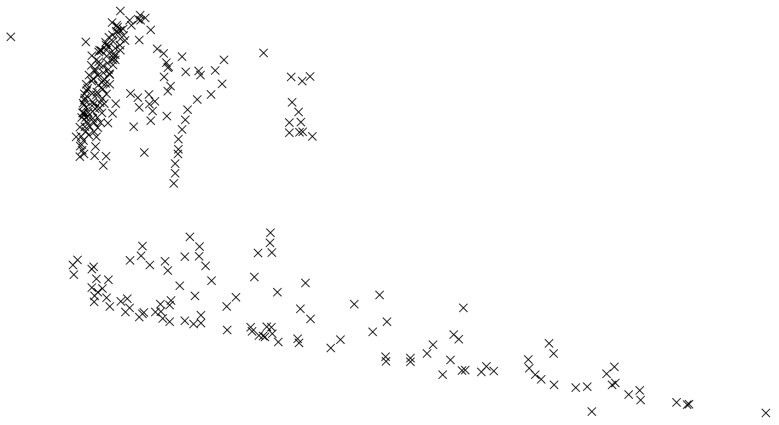


Figure 3.10: Lidar measurements of a car viewed from above. Due to an object in between the car and the sensor, a region of the car is subject to occlusion.

The solution to the measurement assignment problem in this work relies on the estimated corner point and orthogonal lines from the ISED method, while not suffering from the drawbacks listed above. By using the orthogonal lines attained from fitting an L-shape, a 2 dimensional vector $\mathbf{v}$ can be defined for each visible side (i.e. orthogonal line), where the magnitude of each vector is equal to the viewed length ε of its corresponding side (calculated as per Section 3.6.2), see Figure 3.11

$$\mathbf{v}_1, \quad ||\mathbf{v}_1|| = \varepsilon_1 \tag{3.8}$$
$$\mathbf{v}_2, \quad ||\mathbf{v}_2|| = \varepsilon_2 \tag{3.9}$$

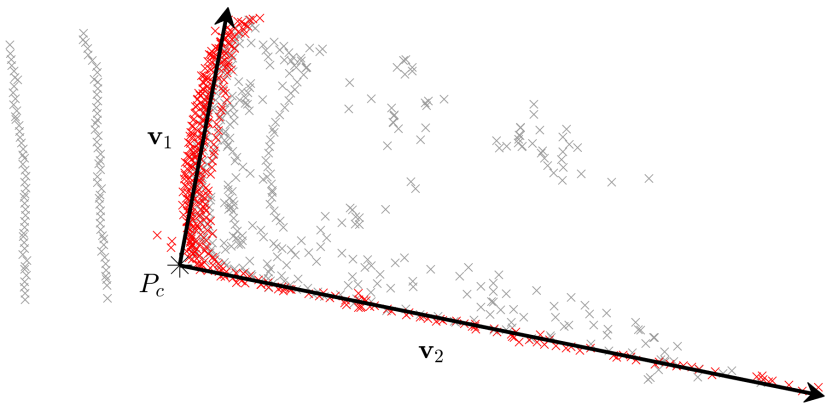


Figure 3.11: By performing L-shape fitting with measurements marked in red, it is possible to locate the corner point $\mathbf{P}_c$, as well as two vectors $\mathbf{v}_1$,$\mathbf{v}_2$, both originating from the corner point and spanning each of the viewed sides.

These vectors can be used to create uniformly distributed *ideal* MGPs $\boldsymbol{\delta}_i$ along each visible side. The ideal MGPs $\boldsymbol{\delta}_i$ represent positions along the car's length and width where MGPs ideally ought to be placed, since by definition a *measurement generating point* is considered as a specific point which at each time step (potentially) generates a sensor measurement. By creating ideal MGPs and assigning them measurements, the same assignment can later be used when constructing MGPs based on the predicted state, $\boldsymbol{\mu}_i(\boldsymbol{\xi}_{k|k-1})$. As such, ideal MGPs can be seen as MGPs constructed from measured data, used to assign measurements and associate them to MGPs constructed from the state $\boldsymbol{\xi}$.

As such, under the presence of no occlusion, the ideal MGPs are uniformly distributed. Consequently, $1 + 2N$ ideal MGPs $\boldsymbol{\delta}_i$ are created in the following way

$$\begin{aligned}
\boldsymbol{\delta}_0 &= \mathbf{P}_c, \\
\boldsymbol{\delta}_i &= \mathbf{G}(\mathbf{P}_c, i, N, \mathbf{v}_1) = \mathbf{P}_c + \frac{i}{1+N}\mathbf{v}_1, && i = 1,2,\ldots,N \\
\boldsymbol{\delta}_{i+N} &= \mathbf{G}(\mathbf{P}_c, i, N, \mathbf{v}_2) = \mathbf{P}_c + \frac{i}{1+N}\mathbf{v}_2, && i = 1,2,\ldots,N
\end{aligned} \tag{3.10}$$

where $\mathbf{P}_c$ is the corner point. The resulting ideal MGPs are uniformly placed along the vectors $\mathbf{v}_1$ and $\mathbf{v}_2$. Each $\boldsymbol{\delta}_i$ needs to be associated with a measurement, which can be performed by conducting a nearest neighbor search. Note that special care needs to be taken in order for

no duplicate measurements to appear among assigned measurements; each measurement can by definition only be associated to one MGP. If the distance between an ideal MGP $\boldsymbol{\delta}_i$ and its associated measurement $\mathbf{z}_i$ is above the threshold d_{max}, then the ideal MGP is not assigned a measurement. Given how closely spaced lidar measurements are, the threshold d_{max} will make sure that ideal MGPs placed in occluded regions are not included for later use in the car PHD filter.

An additional threshold, ε_{min}, is used to determine whether one or both sides of the L-shape are visible enough to be used for state estimation. Consequently, if any of the vectors $\mathbf{v}_1$ and $\mathbf{v}_2$ have a magnitude less than ε_{min} then no MGPs will be created for the associated side, resulting in a one-sided measurement update in Section 3.7.2. By formulating the two vectors $\mathbf{v}_1$ and $\mathbf{v}_2$, this method enables an upper limit of $N_\lambda = 1+2N$ measurements to be associated (and by extension the number of MGPs), thus making it possible to adjust computational complexity, although at the expense of state estimation accuracy. Additionally, the distance threshold d_{max} can be used to take occluded regions into account.

The final algorithm for selecting measurements is detailed in Algorithm 5.

Algorithm 5 Select Measurements

input: P_c, $\mathbf{v}_1$, $\mathbf{v}_2$, d_{max}, ε_{min}, N, Set of measurements Z
  $\delta_0 = P_c$
  if $||\mathbf{v}_1|| > \varepsilon_{min}$ **then**
    for $i = 1$ to N **do**
      $\delta_i \leftarrow G(P_c, i, N, \mathbf{v}_1)$
    end for
  end if
  if $||\mathbf{v}_2|| > \varepsilon_{min}$ **then**
    for $i = 1$ to N **do**
      $\delta_{i+N} \leftarrow G(P_c, i, N, \mathbf{v}_1)$
    end for
  end if
  for $j = 1$ to $|\delta|$ **do**
    $\{z_j, d_j\} \leftarrow$ unique nearest neighbor search for δ_j
    if $d_j > d_{max}$ **then**
      $z_j =$ Null
    end if
    $Z_a \leftarrow$ add z_j to Z_a
  end for
output: Vector of assigned measurements Z_a

After conducting pre-processing for all car cluster measurements, each car cluster $\mathbf{C}^i_{\text{car}}$ passed to the rectangular PHD filter will consist of the following set

$$\mathbf{C}^i_{\text{car}} = \{\mathbf{Z}^i_a, \mathbf{v}^i_1, \mathbf{v}^i_2\} \tag{3.11}$$

where $\mathbf{Z}^i_a$ are assigned measurements and $\mathbf{v}^i_1, \mathbf{v}^i_2$ are the vectors spanning each viewed side.

3.7 Rectangular PHD filter

The rectangular PHD filter incorporates the general recursive equations presented in Section 2.2.2, with the main difference consisting of how each target is represented. In the general PHD recursion, each target consists of a single Gaussian component; this enables use of the basic KF equations found in Section 2.1.1 to calculate all key steps during a PHD recursion: Prediction, Likelihood and Update. For rectangular tracking however, each component is instead represented by an IMM filter. In turn, the IMM filter utilizes either the EKF or UKF equations to perform necessary steps for each mode in the IMM filter.

As such, motion models for use in the IMM filter need to be defined, as well as MGP-functions for each prediction, in order to perform all key steps in the PHD recursion.

3.7.1 Choice of Motion Models

In this work, the IMM framework is used to incorporate two different motion models, one model for when the car is turning and one for when the car is driving straight. In order to incorporate both of these behaviours the following extended state vector is used

$$\boldsymbol{\xi}_k = [x_k \;\; y_k \;\; v_k \;\; \phi_k \;\; \dot{\phi}_k \;\; w_k \;\; l_k]^T \tag{3.12}$$

where x_k and y_k denote the mid-point of the car, v_k is velocity, ϕ_k and $\dot{\phi}_k$ are heading and turning-rate, w_k and l_k are the width and length on the car respectively. The two different motion models, $\mathbf{f}^t(\boldsymbol{\xi})$ and $\mathbf{f}^s(\boldsymbol{\xi})$, where t denotes the model for turning, and s denotes the model for driving straight, are defined in the following way, where $\Delta T = 0.1$ sec is lidar sample time.

$$\boldsymbol{\xi}^t_{k+1} = \mathbf{f}^t(\boldsymbol{\xi}_k) = \begin{bmatrix} x_k + \Delta T v_k \cos(\phi) \\ y_k + \Delta T v_k \sin(\phi) \\ [v_k, \;\; \phi_k + \Delta T \dot{\phi}_k, \;\; \dot{\phi}_k, \;\; w_k, \;\; l_k]^T \end{bmatrix} + \mathbf{q}^t_k, \tag{3.13}$$

$$\boldsymbol{\xi}^s_{k+1} = \mathbf{f}^s(\boldsymbol{\xi}_k) = \begin{bmatrix} x_k + \Delta T v_k \cos(\phi) \\ y_k + \Delta T v_k \sin(\phi) \\ [v_k, \;\; \phi_k, \;\; 0, \;\; w_k, \;\; l_k]^T \end{bmatrix} + \mathbf{q}^s_k \tag{3.14}$$

$$\mathbf{q}_k^t \sim \mathcal{N}\left(\mathbf{0}_{7\times 1}, \mathbf{Q}_t\right), \quad \mathbf{Q}_t = \mathbf{\Gamma}_t^T \begin{bmatrix} \sigma_v^2 & 0 & 0 & 0 \\ 0 & \sigma_{\dot{\phi}}^2 & 0 & 0 \\ 0 & 0 & \sigma_w^2 & 0 \\ 0 & 0 & 0 & \sigma_l^2 \end{bmatrix} \mathbf{\Gamma}_t, \quad \mathbf{\Gamma}_t = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.15}$$

$$\mathbf{q}_k^s \sim \mathcal{N}\left(\mathbf{0}_{7\times 1}, \mathbf{Q}_s\right), \quad \mathbf{Q}_s = \mathbf{\Gamma}_s^T \begin{bmatrix} \sigma_v^2 & 0 & 0 & 0 \\ 0 & \sigma_{\dot{\phi}}^2 & 0 & 0 \\ 0 & 0 & \sigma_w^2 & 0 \\ 0 & 0 & 0 & \sigma_l^2 \end{bmatrix} \mathbf{\Gamma}_s, \quad \mathbf{\Gamma}_s = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.16}$$

With the two motion models defined, calculating each prediction $\boldsymbol{\xi}_{k|k-1}^t, \boldsymbol{\xi}_{k|k-1}^s$ is performed with

$$\begin{aligned} \{\hat{\xi}_{k|k-1}^{0t}, \mathbf{P}_{k|k-1}^{0t}, \hat{\boldsymbol{\xi}}_{k|k-1}^{0s}, \mathbf{P}_{k|k-1}^{0s}\} &= \text{IMM.}\textsc{Mixing}\left(\hat{\boldsymbol{\xi}}_{k-1|k-1}^t, \mathbf{P}_{k-1|k-1}^t, \hat{\boldsymbol{\xi}}_{k-1|k-1}^s, \mathbf{P}_{k-1|k-1}^s\right) \\ \{\hat{\boldsymbol{\xi}}_{k|k-1}^t, \mathbf{P}_{k|k-1}^t\} &= \text{IMM.}\textsc{Predict}\left(\mathbf{f}^t(\cdot),\ \hat{\boldsymbol{\xi}}_{k|k-1}^{0t}, \mathbf{P}_{k|k-1}^{0t}\right) \\ \{\hat{\boldsymbol{\xi}}_{k|k-1}^s, \mathbf{P}_{k|k-1}^s\} &= \text{IMM.}\textsc{Predict}\left(\mathbf{f}^s(\cdot),\ \hat{\boldsymbol{\xi}}_{k|k-1}^{0s}, \mathbf{P}_{k|k-1}^{0s}\right) \end{aligned} \tag{3.17}$$

The IMM block first performs mixing (see Section 2.1.3) in order to get mixed estimates, before the PREDICT operation is called, which can either be performed using the EKF predict or UKF predict equations (see Section 2.1.1-2.1.2).

3.7.2 Generating Measurement Generating Points

In order to calculate the likelihood and update for each estimate, MGPs need to be created for each prediction. From the pre-processing step, each car cluster $\mathbf{C}_{\text{car}}^i$ consists of

$$\mathbf{C}_{\text{car}}^i = \{\mathbf{Z}_a^i, \mathbf{v}_1^i, \mathbf{v}_2^i\} \tag{3.18}$$

where $\mathbf{Z}_a^i$ are assigned measurements and $\mathbf{v}_1^i, \mathbf{v}_2^i$ are vectors spanning each viewed side. As such, in order to generate MGPs the vectors $\mathbf{v}_1^i, \mathbf{v}_2^i$ must be identified by which side they correspond to (i.e. which vector is width and which is length), and by extension from which corner point $\mathbf{P}_c$ they originate from (i.e. $\mathbf{P}_c$ corresponds to either corner 1,2,3,4). This is important, since selection and ordering of the assigned measurements $\mathbf{Z}_a$ is dependent on the viewed corner in the pre-processing step.

The implication of this is that each prediction will act as a form of hypothesis regarding which corner (and by extension which sides) are seen in the data. This results in MGPs-generation which is dependant on predicted state. This can most easily be illustrated by viewing Figure 3.12.

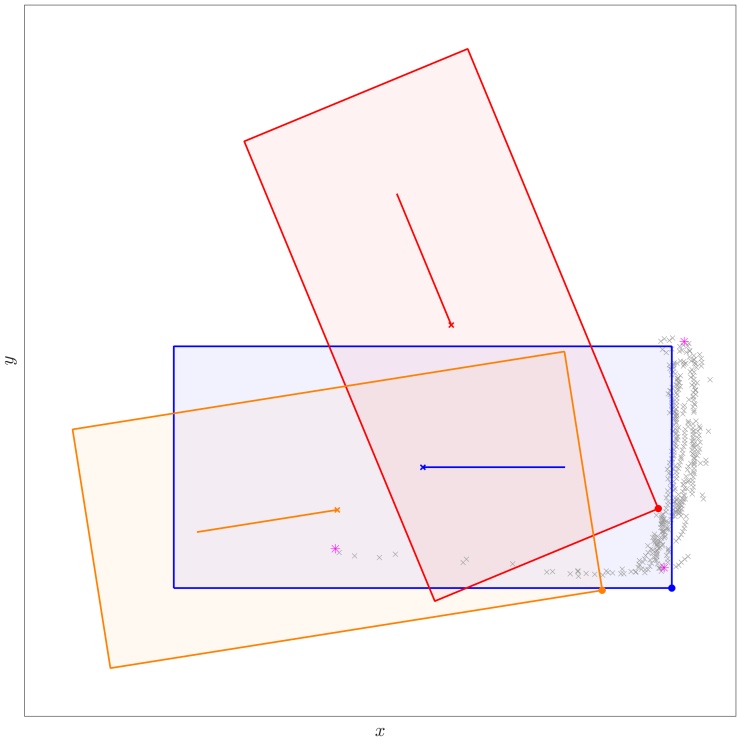


Figure 3.12: Three different predictions, each marked with a different color. Depending on predicted heading, each prediction will act as a hypothesis regarding which corner is seen in the data. Filled colored circles represent each predictions hypothesis regarding viewed corner. Magenta stars mark selected measurements, performed with Algorithm 5. Gray x:s represent measurements remaining after height based filtering.

In the figure measurements remaining after height based filtering are seen, as well as predictions from three different IMM components, each with a different color. For clarity only one prediction is illustrated for each component in the figure.

The two vectors $\mathbf{v}_1^i, \mathbf{v}_2^i$ are known since pre-processing has occurred, and as such a number of measurements have already been assigned. In the figure a total of 3 measurements have been chosen, marked as magenta stars. Since there exists ambiguity in which corner of the car is actually seen, each IMM component will act as a hypothesis on this matter. In the case of the red component, the corner point is seen as $\mathbf{P}_c \rightarrow \text{corner}_1$ according to definition (See Section 2.3.1). For the blue and orange components, the corner will be seen as $\mathbf{P}_c \rightarrow \text{corner}_4$ and $\mathbf{P}_c \rightarrow \text{corner}_2$ respectively. Depending on how the car behaves in the next time step, only one (or none) of these three components will prove to be correct. As such, if one of these predictions proves to be valid, it will result in a higher weight for said component in the PHD filter.

The choice in which corner is seen is solely dependant on predicted heading $\phi_{k|k-1}$, and can be determined by utilizing linear algebra. Consequently, deciding which side the vectors $\mathbf{v}_1^i, \mathbf{v}_2^i$ belong to is also dependant on predicted heading and can also be determined with linear algebra. Knowledge about corner point and which side is seen, are necessary to construct MGPs for each prediction. The necessary steps for deciding corner and which sides are seen can be found in Algorithm 6 and 7.

Algorithm 6 Identify which corner is seen

input: $\phi_{k|k-1}$, $\mathbf{v}_1$, $\mathbf{v}_2$
init: Generate unit vectors of $\mathbf{v}_1$, $\mathbf{v}_2$: $\mathbf{u}_1$, $\mathbf{u}_2$
  Inverse rotate each unit vector with the predicted heading
  $\mathbf{u}_1^0 \leftarrow \text{inv.rot}(\mathbf{u}_1, \phi_{k|k-1})$
  $\mathbf{u}_2^0 \leftarrow \text{inv.rot}(\mathbf{u}_2, \phi_{k|k-1})$
  Calculate angle $0 \leq \psi \leq 2\pi$
  $\psi \leftarrow \arccos(\text{dot}(\mathbf{u}_1^0, \mathbf{u}_2^0))$,
  if $0 < \psi \leq \pi/2$ **then**
    corner $= 1$
  else if $\pi/2 < \psi \leq \pi$ **then**
    corner $= 4$
  else if $\pi < \psi \leq 3\pi/2$ **then**
    corner $= 3$
  else if $3\pi/2 < \psi \leq 2\pi$ **then**
    corner $= 2$
  end if
output: corner

Algorithm 7 Identify viewed width and length

input: $\phi_{k|k-1}$, $\mathbf{v}_1$, $\mathbf{v}_2$
init: Generate unit vectors of $\mathbf{v}_1$, $\mathbf{v}_2$: $\mathbf{u}_1$, $\mathbf{u}_2$
  Construct unit vector representing $\phi_{k|k-1}$
  $\mathbf{u}_\phi = [\cos(\phi), \sin(\phi)]^T$
  if $\text{dot}(\mathbf{u}_\phi, \mathbf{u}_1) > \text{dot}(\mathbf{u}_\phi, \mathbf{u}_2)$ **then**
    $s_1 =$ "length"
    $s_2 =$ "width"
    $l_{view} = ||\mathbf{v}_1||$
    $w_{view} = ||\mathbf{v}_2||$
  else
    $s_1 =$ "width"
    $s_2 =$ "length"
    $w_{view} = ||\mathbf{v}_1||$
    $l_{view} = ||\mathbf{v}_2||$
  end if
output: $s_1, s_2, w_{view}, l_{view}$

By performing the steps outlined in Algorithm 6 and 7, MGP functions can be generated for each prediction and associated measurement. It is important to note that MGPs are not created,

instead MGP-*functions* are created. Revisiting the general MGP equations from Section 2.3.1,

$$\{\mathbf{h}_l^j(\boldsymbol{\xi}_{\min})\}_{j=1}^N = \mathbf{h}_c^i(\boldsymbol{\xi}_{\min}) + \frac{j}{1+N}\rho_l \begin{bmatrix} l\cos(\phi+\eta(i)) \\ l\sin(\phi+\eta(i)) \end{bmatrix}, \qquad \eta(i) \to \{0,0,\pi,\pi\} \tag{3.19}$$

$$\{\mathbf{h}_w^j(\boldsymbol{\xi}_{\min})\}_{j=1}^N = \mathbf{h}_c^i(\boldsymbol{\xi}_{\min}) + \frac{j}{1+N}\rho_w \begin{bmatrix} w\cos(\phi+\eta(i)) \\ w\sin(\phi+\eta(i)) \end{bmatrix}, \qquad \eta(i) \to \left\{\frac{\pi}{2},\frac{3\pi}{2},\frac{3\pi}{2},\frac{\pi}{2}\right\} \tag{3.20}$$

it can be seen that while they do depend on the state $\boldsymbol{\xi}$, they also depend on a number of other variables as well: i, N, ρ_w and ρ_l. The variable i represents the corner point, which at this point is determined with Algorithm 6. The variable N represents the number of MGPs, which is also known since $1+2N$ measurements are assigned at the pre-processing step (the parameter N is user selected). The two ratios, ρ_l and ρ_w are also known, since both predicted state $\boldsymbol{\xi}_{k|k-1}$ as well as w_{view} and l_{view} are known (Algorithm 7). As such, constructing a vector of MGP-functions corresponds to using the general equations together with the state-independent parameters (i, N, ρ_w and ρ_l) to form MGP-functions which solely depend on the state $\boldsymbol{\xi}$. Since assigned measurements $\mathbf{Z}_a^i$ are ordered according to $\mathbf{P}_c$-MGP, $\mathbf{v}_1$-MGPs and $\mathbf{v}_2$-MGPs, the MGP-functions need to be generated in same order. The same threshold for viewed length and width, ε_{min}, is also used when generating MGP-functions.

For example, for $N=2$, $corner=1$, $\rho_w=\rho_l=1$, $\varepsilon_1 > \varepsilon_{min}$, $\varepsilon_2 > \varepsilon_{min}$ and where $\mathbf{v}_1$ corresponds to width and $\mathbf{v}_2$ to length, the MGP-function vector is the following:

$$\mathbf{H}(\boldsymbol{\xi}) = \underbrace{\begin{bmatrix} \mathbf{h}_c^{i=1}(\boldsymbol{\xi}) \\ \mathbf{h}_w^1(\boldsymbol{\xi},\rho_w=1,i=1) \\ \mathbf{h}_w^2(\boldsymbol{\xi},\rho_w=1,i=1) \\ \mathbf{h}_l^1(\boldsymbol{\xi},\rho_l=1,i=1) \\ \mathbf{h}_l^2(\boldsymbol{\xi},\rho_l=1,i=1) \end{bmatrix}}_{2(1+4)\times 1} \tag{3.21}$$

The final step, before using the $\mathbf{H}(\boldsymbol{\xi})$ function vector and the assigned measurements $\mathbf{Z}_a^i$ for UPDATE or LIKELIHOOD, is to remove empty measurements from $\mathbf{Z}_a^i$ and their corresponding MGP-functions from $\mathbf{H}(\boldsymbol{\xi})$. This is since empty measurements fall outside of the threshold d_{max}, indicating that an occluded region is present for the corresponding MGP-functions.

3.7.3 Incorporating Rectangular Targets in PHD recursion

With motion models defined, as well as the capability to generate MGP-functions for each prediction in an IMM component, an entire PHD recursion can be computed, detailed in Algorithm 8.

Algorithm 8 A single PHD recursion for Rectangular Targets.

input: Number of components to keep between recursions N_{PHD}, Birth PHD $\gamma_k(\xi)$, Previous PHD $v_{k-1}(\xi)$, Measurement clusters $\{C^i_{\text{car}} = \{Z^i_a, \mathbf{v}_1, \mathbf{v}_1\}\}_{i=1}^m$

Add birth PHD to predicted set
$v_{k|k-1} \leftarrow \gamma_k(\xi)$
for $j = 1, \ldots, N_{\text{PHD}}$ **do**
  Predict each IMM component in previous PHD $v_{k-1}(\xi)$, add to predicted set
  $v_{k|k-1} \leftarrow \text{IMM}(j).\textsc{Predict}$
end for
Calculate updated PHD, first insert non-detected components from prediction
$v_{k|k}(\xi) \leftarrow (1 - P_D) v_{k|k-1}(\xi)$
for $i = 1, \ldots, m$ **do**
  for $j = 1, \ldots, N_{\text{PHD}}$ **do**
    Algorithm 5
    Algorithm 6
    Algorithm 7
    $H(\xi) \leftarrow$ Generate MGP functions
    $w^j_{k|k} \leftarrow P_D \times w^j_{k|k-1} \times \text{IMM.}\textsc{Likelihood}(Z^i_a, H(\xi), \sigma^2_R)$
  end for
  for $j = 1, \ldots, N_{\text{PHD}}$ **do**
    $w^j_{k|k} \leftarrow$ Normalize with $K + \sum_{j=1}^{N_{\text{PHD}}} w^j_{k|k-1}$
  end for
end for
Perform Merging, Pruning, etc.
Update remaining components
for $j = 1, \ldots, N_{\text{PHD}}$ **do**
  $v_{k|k}(\xi) \leftarrow \text{IMM}(j).\textsc{Update}$
end for

output: $v_{k|k}(\xi)$

4

Results

This chapter shows the results of the proposed algorithm. Behaviour and performance is tested on an example scenario from the KITTI Vision Benchmark dataset [4]. Unfortunately, no ground truth for any of the tracked objects is available, which is why the algorithm is evaluated visually only.

Since there are around 5-10 tracked objects at any given time step, it is impossible to visualize the entire filter performance in this report in a satisfactory way. Instead, examples of characteristic and interesting filter behaviour are given and explained.

The best way to get an overview of the algorithm's performance is to view it in practice. Therefore, a short (22s) video was created and uploaded to a hosting platform. The reader is encouraged to watch it before continuing in this chapter in order to get a better understanding of subsequent explanations [24].

4.1 Scenario Description

The scenario consists of 186 frames of lidar data in an urban area with pedestrians, cyclists and cars moving through the field of view. As outlined in thesis delimitations 1.1.1, ego position is constant, the observed area is assumed to be a flat horizontal plane and a static map of the environment is available.

The area consists of a T-shaped intersection on which two cars and two bicycles are observed during the scenario. There are also several sidewalks around the roads, and a crossing over one of them, on which pedestrians and groups of pedestrians can be seen throughout the entire example. An example frame of the scenario is shown in Figure 4.1.

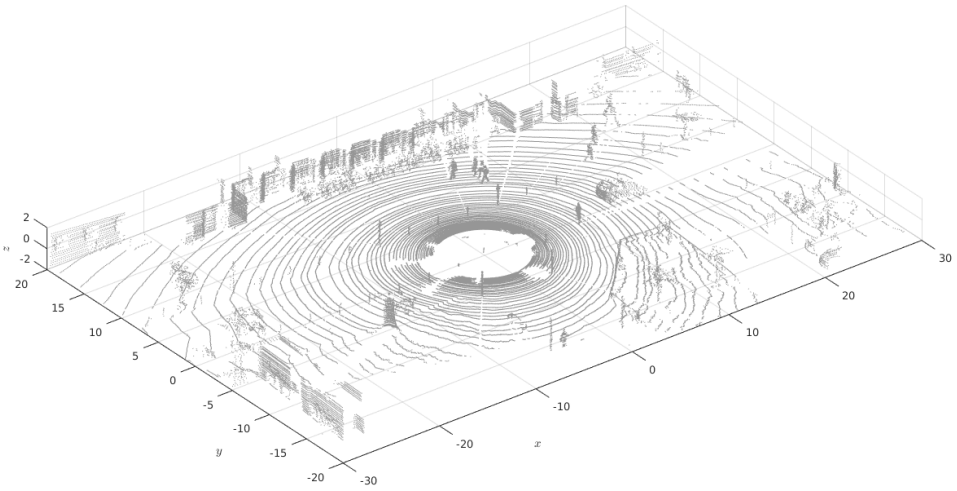


Figure 4.1: An overview of the test scenario area.

4.2 Algorithm Performance

4.2.1 Pre-Processing

The several pre-processing steps - ground removal, static map removal and clustering - all aim to dismiss uninformative data and bundle measurements into objects. As pointed out in the description of these algorithms, all of them can be tuned between reducing the data as much as possible or keeping a high level of detail.

For example, the clustering algorithm is parametrized by a minimum amount of points and a maximum distance between any two points in a cluster. The further an object is away from the lidar sensor, the fewer points it will consist of and the longer the distance between any two of these points will be. Setting a fixed threshold for the clustering parameters thus invariably leads to a decrease in the distance at which objects are tracked and identified by the implemented filter as opposed to the raw data.

An example of this can be seen in Figure 4.2 which compares the raw point data of an example frame with the clusters derived from that data. In the raw data, the point that is furthest away from the sensor lies at a distance of about 100m. In the clustered data this distance is reduced to about 50m.

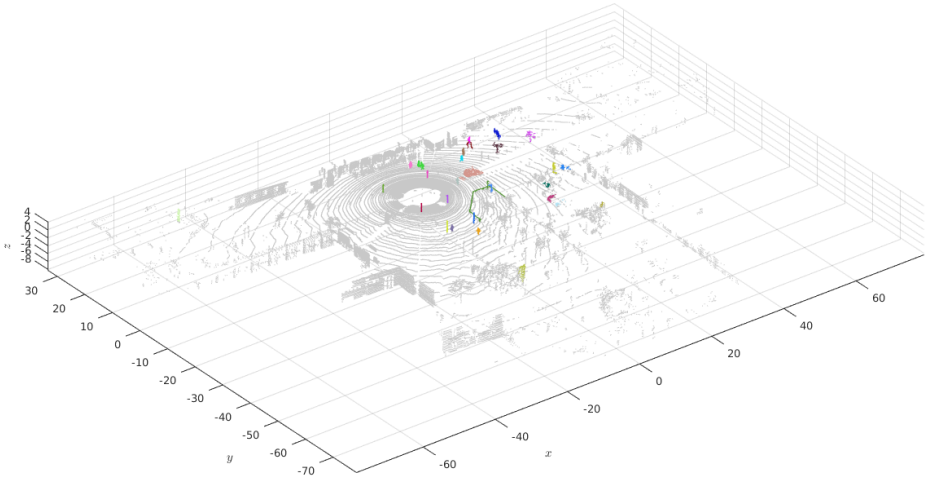


Figure 4.2: The difference in range of perception between raw data (gray) and clusters (colorful).

While this decrease in effective range is unfortunate, it is necessary to achieve a robust filter implementation. Both the neural network and the extended target tracking filters need objects with a certain minimum amount of points in order to derive distinct features (neural network) and extent measurements (tracker). Higher sensor resolution is one way to remedy this problem. Additionally, a dynamic clustering threshold (dependent on distance) can be employed. Distance between measurements increases with range, which can be incorporated when clustering: objects nearby have a comparatively smaller clustering threshold than objects further away.

4.2.2 Neural Network

The features described in the algorithm chapter 3 were added one by one and a ceiling analysis was performed to verify the eligibility of each. Table 4.1 shows the influence of each feature on the network performance.

Feature vector	Precision (%)	Recall (%)
$[w, l, h]$	75.4	73.7
$[w, l, h, \rho]$	81.1	79.9
$[w, l, h, \rho, \frac{n}{d}]$	88.4	83.0

Table 4.1: Ceiling analysis of a feature vector with an increasing amount of features.

As can be seen, the neural network shows promising performance. This makes it possible to incorporate the neural network as a classification layer before filtering, in order to use specific filtering approaches for different classes of objects.

4.2.3 Tracking Filter

The following examples are all taken from the test scenario described at the beginning of this chapter. Elliptical targets are displayed with a pink ellipse under the object cluster. Rectangular targets are shown as cuboid bounding boxes around the cluster. Classification of different kinds of objects is visualized by a specific color value for all points in the cluster of that object. All clusters are drawn as an overlay over the original raw data to make orientation in the scenario easier. The different colors can be seen in Table 4.2.

Class	Raw Data	Clutter	Cars	Cyclists	Pedestrians	Pedestrian Groups
Color						

Table 4.2: Legend showing different colors used to visualize different classes of objects.

Elliptical Filter

The elliptical PHD filter manages to keep track of the multiple targets moving through the area of perception at any given time. Figure 4.3 shows that 8 objects are being tracked as ellipses: 2 cyclists, 3 pedestrians and 3 groups of pedestrians. Although some of those targets are spatially very close, the tracker manages to track the individual objects.

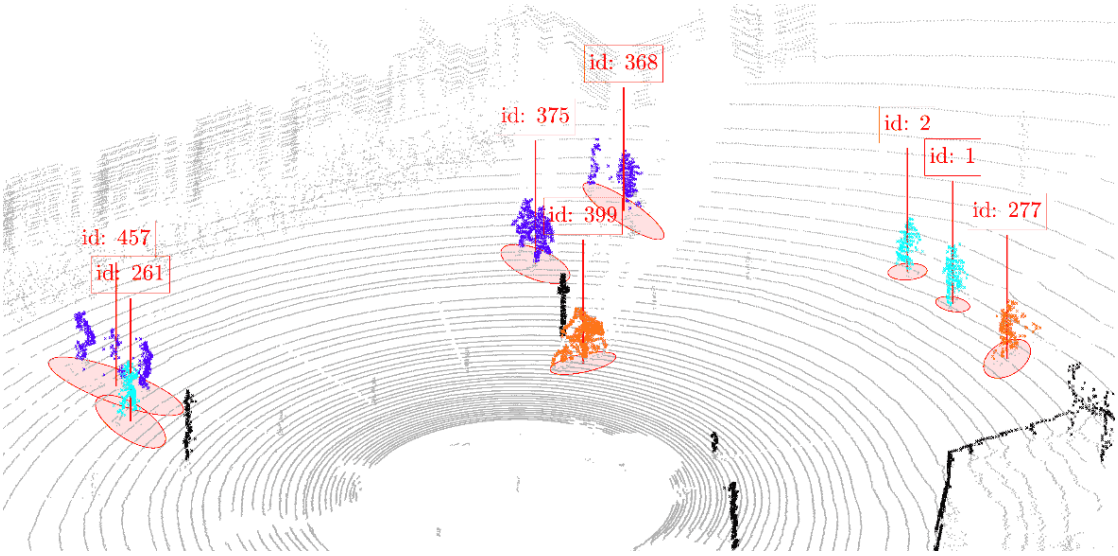
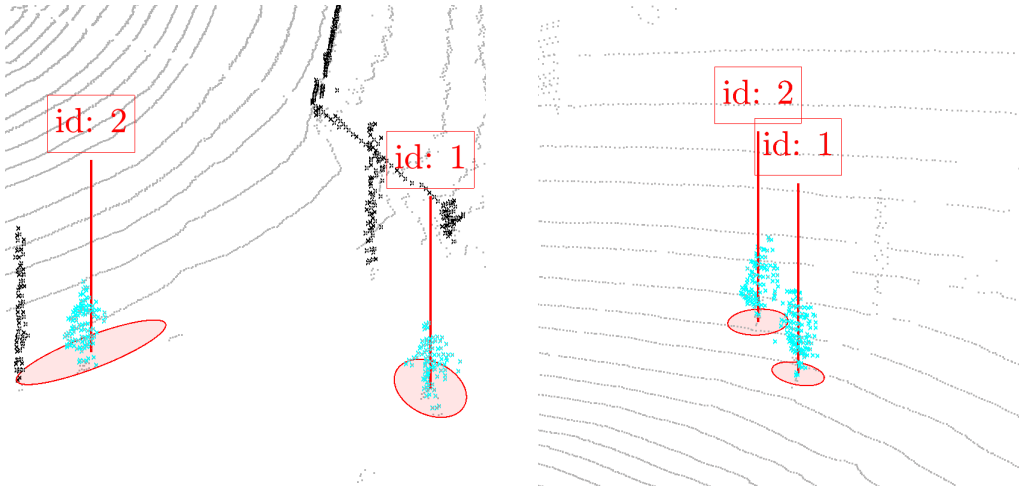


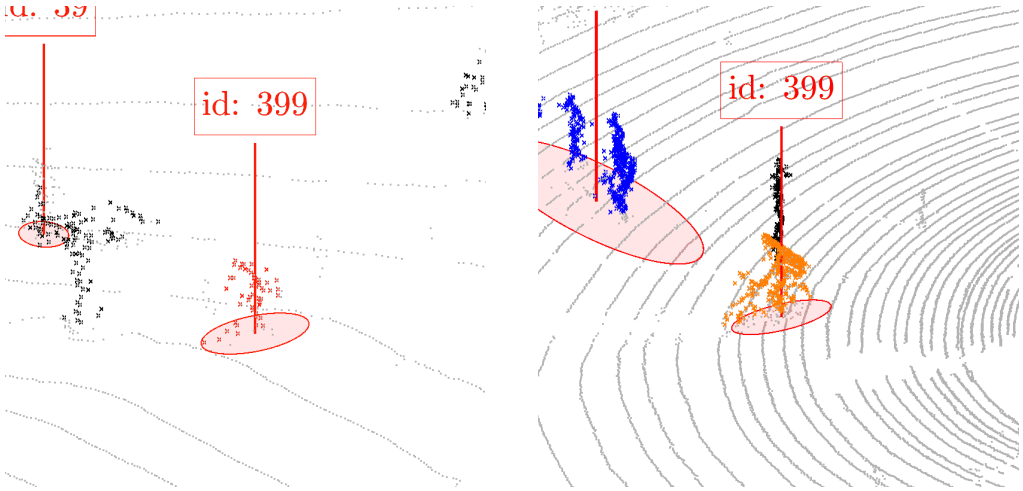
Figure 4.3: Tracking multiple targets, some of them spatially close, $k = 163$.

Internally, each Gaussian inverse Wishart component propagated in the filter, is assigned a unique index. This index can be used to identify the trajectory of any particular target in the multitarget scenario. Figures 4.4 and 4.5 show how the filter effectively keeps track of the index of an example pedestrian and an example cyclist throughout their entire lifetime in the scenario.



(a) $k = 1$ **(b)** $k = 186$

Figure 4.4: Keeping track of the index of a pedestrian target within the multitarget scenario.



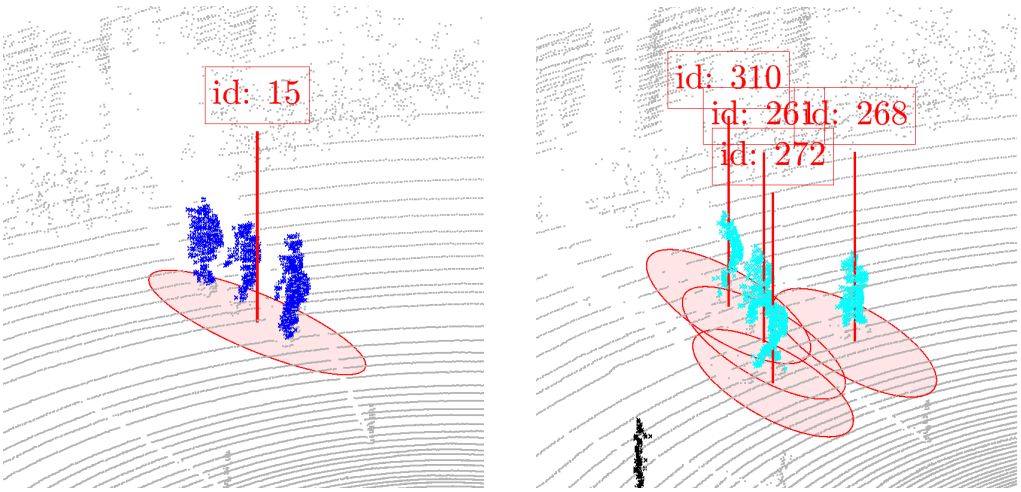
(a) $k = 123$ **(b)** $k = 186$

Figure 4.5: Keeping track of the index of a cyclist target within the multitarget scenario.

The only time a target might lose its index is if it is merged with another target of a higher weight.

Another interesting point is how the filter manages to handle situations where a group of people are suddenly clustered as individual objects, or the other way around. Those situations are complex because the extent of the measurement suddenly changes significantly. The filter has to either split up an existing target and use it as a prior for several measurements, or it has to merge several existing targets into one.

An example of this kind of behaviour in Figure 4.6.

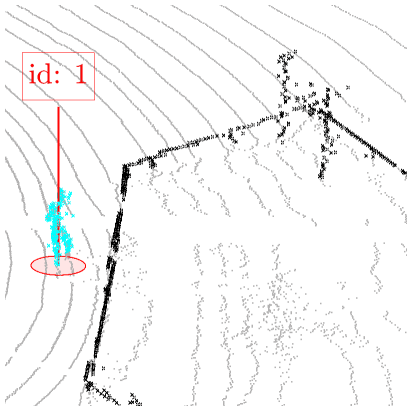


(a) $k = 41$ **(b)** $k = 98$

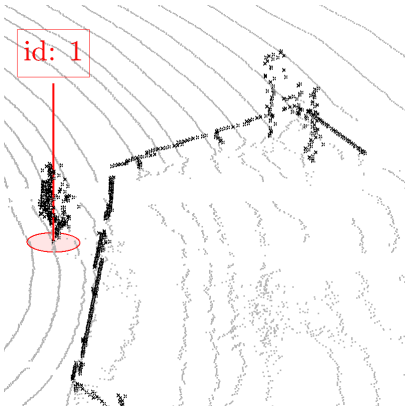
Figure 4.6: Splitting up a pedestrian group target into 4 single pedestrian targets.

The elliptical filter also manages to keep targets for a couple of time steps even if no measurements are received. It does so by propagating the prediction without performing an update step. The weight of the target is lowered for every prediction-only step, so after a certain number of steps without a measurement, the target will eventually be discarded. However, the filter shows that it successfully detects targets again after approximately 10 prediction-only steps as soon as they then reappear.

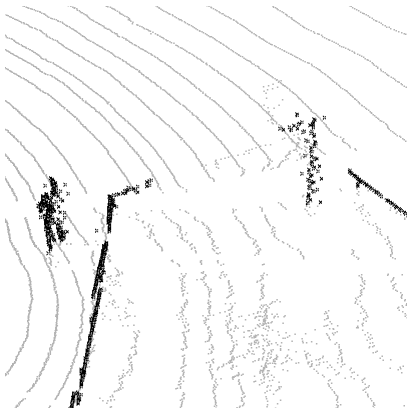
This robustness towards missing measurements is important to be able to handle e.g. occlusion cases, where a target is temporarily hidden behind another object. In the test scenario one pedestrian is moving very close to a fence. In some frames that pedestrian is then clustered into one cluster with the nearby fence, which results in that cluster becoming a lot bigger than a normal pedestrian cluster. Therefore, a matching measurement is effectively not available for several time steps. The described problem can be seen in Figure 4.7.



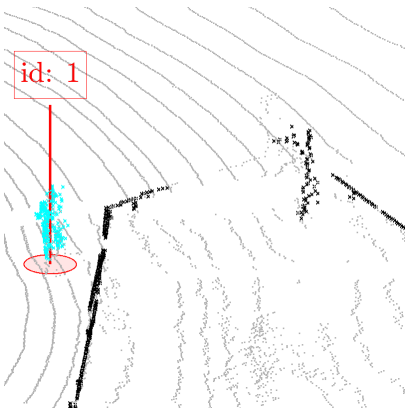
(a) $k = 94$, measurement available, target is being tracked.



(b) $k = 95$, no measurement, pedestrian is clustered together witht the adjacent fence, prediction only.



(c) $k = 98$, no measurement, prediction only, weight has become so low that it is not considered to be a likely target anymore.



(d) $k = 102$, measurement reappears, target is picked up again (as indicated by keeping the same index value as before).

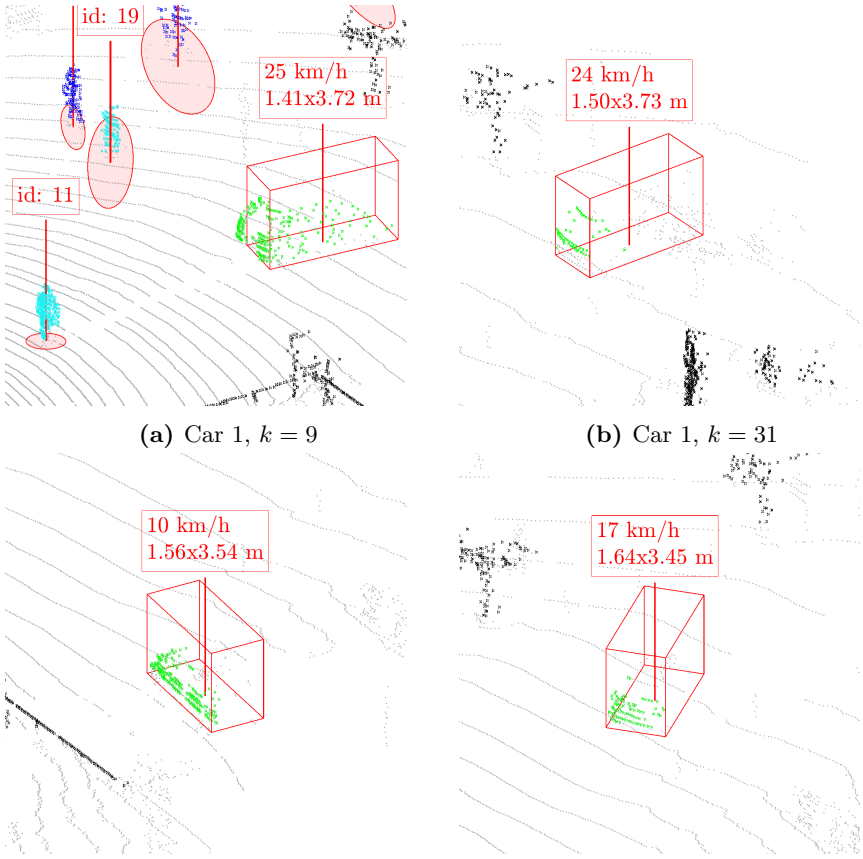
Figure 4.7: A pedestrian target's measurements are not available for several timesteps, still it is tracked again when it reappears.

Rectangular Filter

During the scenario two non-stationary cars are present. One of them, from now on referred to as Car 1, is present at the start of the scenario near the ego position, and drives in a more or less straight path before disappearing from sensor view. In total, Car 1 is present for 35 frames. The other one ,Car 2, appears after approximately 20 frames, drives towards the ego car and later performs a 90° turn, after which it drives away from the ego car and disappears from sensor view. In total, Car 2 is visible for 55 frames.

Both cars are instantly detected by the PHD filter. Estimation of size, as well as position, velocity and heading all appear to be within a reasonable range. Estimation of size changes very little

during the scenario, even though at some instances only one side of each car is seen. In addition to this, estimation appears to be robust even when partial occlusion of the cars occur. This is illustrated in Figure 4.8, where filter estimates are plotted as boxes on top of each car cluster. Velocity as well as estimated width and length are visible above each box.



(a) Car 1, $k = 9$ **(b)** Car 1, $k = 31$

(c) Car 2, $k = 34$ **(d)** Car 2, $k = 69$

Figure 4.8: Car 1 and Car 2 illustrated for selected frames (denoted ky k). Estimation of shape and size varies minimally between successive frames, even though several sides are not visible. In 4.8a a pedestrian is located between Car 1 and ego, resulting in partial occlusion of the car.

Estimation of width and length appear to be slightly underestimated. This can probably be attributed to the combination of only viewing 1-2 sides of the cars and reflectivity around their rounded corners. Before disappearing from sensor view, both cars are subject to missed measurements. This is due to their clusters not containing more than 50 points, resulting in their clusters being discarded. The PHD filter struggles to track them under these instances, most probably due to the choice of survival and detection rate $P_S = P_D = 1$ in the filter. Additionally, before Car 2 disappears from sensor view it is omitted by the PHD filter. This appears to be due to the choice of clutter constant $K = 1$.

The rectangular PHD filter was tested with both an EKF and UKF when performing the standard KF steps of prediction, likelihood and update. Of these two, the UKF outperformed the EKF

in terms of capability to track both cars. The EKF only managed to track each car for $10 - 20$ frames before becoming unstable. The UKF on the other hand did not experience any of these problems. The problems associated with the EKF can either be attributed to a high degree of non-linearity in both motion and measurement model, or due to programming errors present in the EKF implementation[1].

[1]Several tests were performed to see whether the Jacobians were correctly calculated. Both numerical and analytical calculations were tested, with neither resulting in better filter performance.

5

Discussion

Results show that by incorporating elliptical and rectangular extended targets, common road participants can successfully be tracked in lidar data. In addition, the spatial extent of tracked objects is also estimated. This thesis thereby extends similar work on radar data [25] and builds a rigorous framework for the data processing and tracking algorithms needed for a viable solution. By testing the proposed algorithm on real-world data, its potential for practical use is shown.

5.1 Benefits of Neural Network Classifier

Similar solutions for radar data, like [25], make use of a track-before-classify approach. This method initializes one component of each type (i.e. elliptical and rectangular) for each new target. It then propagates both components until the weights of one of them converge in a way that the filter will dismiss it as unlikely. The remaining component is then considered to signify the type of the target. In a way, this can be seen as a classification approach internally in the filter. The downside to this is that until one component can be dismissed, the computational load for that target is essentially at least doubled[1]. If one would like to diversify into even more types of components in the future, this method would not scale well.

In contrast, this thesis proposes a classify-before-track approach. Essentially, the idea is that each cluster of lidar point data holds sufficient information for classification even before subsequent motion of that cluster can be observed. This is mainly due to the much higher resolution of a lidar sensor as compared to e.g. radar. By using statistical tools from machine learning, this information can be exploited to classify clusters before they actually enter the filter. Therefore,

[1] PHD filter complexity is $\mathcal{O}(mn)$, where m is number of measurements and n number of components. Increasing the number of components can also result in more measurements being included, i.e. measurements from both elliptical and rectangular targets. As such, computational complexity will at least double.

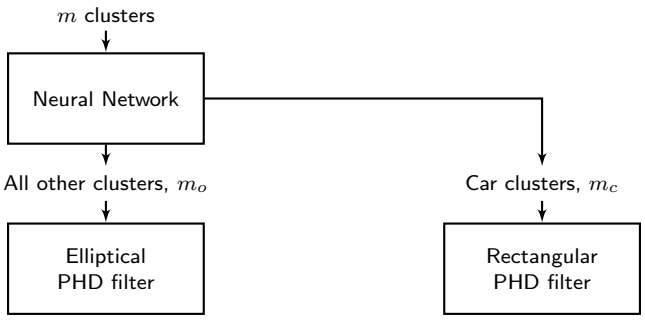


Figure 5.1: The filtering strategy used in this thesis, two dedicated PHD filters, one for cars and the other one for all the rest

any new target can instantly be initialized with the right kind of component. Additionally, any non-stationary target not classified by the neural network will still be tracked, since it will be used in the elliptical tracking filter. As such, elliptical tracking essentially works as a *catch-all* tracking method, capable of tracking any type of non-stationary target.

Interestingly, classify-before-track not only solves the computational overhead of track-before-classify, it actually even results in lower computational complexity for each additional component type. Since e.g. car components will only be updated with car measurements, there are far less combinations to compute likelihoods for. Figures 5.1 and 5.2 exemplify this for using a classify-before-track approach that then utilizes the clusters into different PHD filters that track the objects with suitable components (e.g. ellipses or rectangles) and motion models.

Figure 5.1 shows the approach chosen in this thesis, with one specific PHD filter for cars and another for all other objects. The total number of clusters m is therefore divided into two subsets $m = m_o + m_c$. Existing components in the filters n_o and n_c have to be updated with those measurements. The main complexity of a PHD filter lies in the operations of computing likelihoods and updating existing components with new measurements, so there are generally mn operations to perform. If all components were handled in one filter the overall complexity would thus be $\mathcal{O}((n_o + n_c)m)$, while the divided filter approach results in $\mathcal{O}(n_o m_o + n_c m_c)$. Given that

$$\mathcal{O}(n_o m_o + n_c m_c) < \mathcal{O}((n_o + n_c)m) \tag{5.1}$$

it shows that a PHD filter that has to match fewer measurements against fewer tracked components will have a lower computational complexity than a single PHD tracking all components and matching them against all incoming measurements. [2]

Figure 5.2 shows that the more this approach is diversified to use different PHD filters for different types the less computationally complex the algorithm will be. Figure 5.2 gives an examples of extending the filtering proposed in this thesis with a custom PHD filter for trucks. Thus, the $m = m_o + m_c + m_t$ clusters received as measurements could be divided into smaller subsets. In theory, the main computational complexity in the PHD filters would then look as follows.

[2] In addition, this offers the ability to implement tailor-made components and models for tracking that particular kind of object.

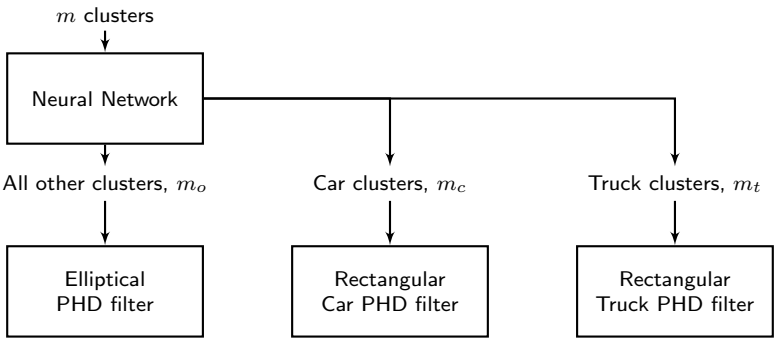


Figure 5.2: A more diversified PHD filtering approach using an additional PHD filter for trucks.

$$\mathcal{O}(n_o m_o + n_c m_c + n_t m_t) < \mathcal{O}((n_o + n_t)(m_o + m_t) + n_c m_c) < \mathcal{O}((n_o + n_c + n_t)m) \tag{5.2}$$

5.2 Comparison of Extended Rectangular vs. Point Target Tracking

Since no reference data is available evaluation of the algorithm's accuracy is limited. However, one way to evaluate the use of extended target tracking for the scenario is to compare filter covariance when incorporating rectangular shape and size in the state-vector, to that when ignoring spatial extent and instead treating each car as a point target. In the case of treating each car as a point target, some type of mean or average is computed for each cluster of measurements, and subsequently treated as a point measurement.

By comparing filter covariance for each kinematic state between the two different approaches to target tracking, it is possible to investigate how uncertain each estimate of state is. As such, this can yield insight into how large the confidence bounds are for each state, an important aspect when incorporating filter output into subsequent sensor fusion or control systems.

In order to perform the comparison an additional rectangular UKF filter is implemented. The additional UKF filter takes the mean of each cluster of measurements and views that as a point measurement. This filter is then used for both cars in the scenario. Filter parameters in terms of initial state (prior), motion noise $\mathbf{Q}_{k-1}$, measurement noise $\mathbf{R}_k$ are identical, the only difference being that terms related to the extended states w_k, l_k are omitted. It should be noted that IMM is not used in this comparison; only the motion model for steering is used. The resulting filter covariance for both approaches, performed on both Car 1 and Car 2, is illustrated in Figure 5.3.

By looking at Figure 5.3 it can clearly be seen that extended rectangular target (ERT) tracking results in significantly lower covariance for all states, compared to point target (PT) tracking. This is most apparent for heading ϕ_k and position x_k,y_k, since these parameters are highly dependant on the shape and size of a car respectively. Calculating the root mean square (RMS) of each filter's standard deviation over the entire sequence for Car 1 and Car 2, seen in Table

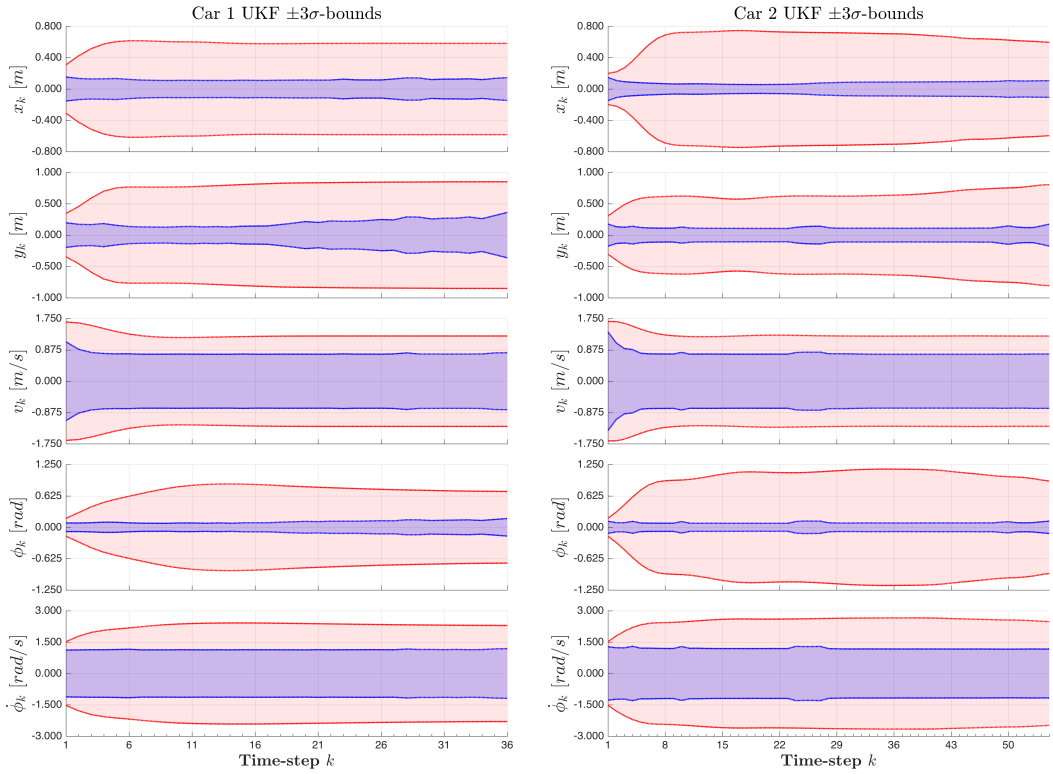


Figure 5.3: Filter covariance, expressed as 3σ bounds, for all kinematic states during the scenario. Red indicates filter covariance when assuming point target (PT), in this case using the mean of all clustered measurements as a point measurement. Blue indicates filter covariance when assuming extended rectangular targets (ERT), i.e. incorporating shape and size in state vector.

5.1, further illustrates the difference between these two different approaches.

For all states a significant gain in confidence is achieved with ERT tracking, when compared to PT tracking. In the case of Car 2, which performs a 90° turn in the scenario, covariance regarding heading σ_ϕ is decreased by a factor of 11.1. The least affected states are velocity and turning-rate. The reason why these states experience a comparatively smaller gain in confidence has to do with how the general MGP function is defined, see Section 2.3.1. Each MGP function incorporates all states except velocity and turning-rate. As such, inference on these states results in higher uncertainty, which is reflected in their corresponding filter covariance.

Table 5.1: Root mean square (RMS) of each state's standard deviation, for each filter and car in the scenario. PT - Point Target assumption, ERT - Extended Rectangular Target assumption.

		RMS				
		σ_{xx}	σ_{yy}	σ_{vv}	$\sigma_{\phi\phi}$	$\sigma_{\dot{\phi}\dot{\phi}}$
Car 1	PT	0.192	0.264	0.433	0.245	0.764
	ERT	0.041	0.070	0.259	0.038	0.380
	Ratio	4.7	3.8	1.7	6.4	2.0
Car 2	PT	0.222	0.214	0.434	0.343	0.846
	ERT	0.028	0.040	0.263	0.031	0.399
	Ratio	7.9	5.4	1.7	11.1	2.2

6

Conclusion

Evaluation shows that the implemented tracker works and fulfills the main goals that were formulated when starting this thesis work.

Sophisticated pre-processing algorithms are necessary in order to reduce the vast amount of data that a lidar sensor provides and in order to make the proposed solution computationally feasible.

The approach of using a combined classifier and tracker is very promising. It allows the usage of specific motion models for certain types of objects in a straightforward way, without having to compute several parallel filters and checking which one of them suits a certain object best. In addition, it makes it possible to update existing targets only against measurements that are deemed to belong to the same type. This reduces computational complexity for the overall solution, since the sum of the number of possible target-measurement combinations for the subsets is much smaller than for the entire set.

Using extended target trackers, as opposed to center point trackers, utilizes the high resolution in lidar data. The added complexity results in a more robust tracker and much less uncertain state estimates.

Unfortunately, the proposed solution could not be tested against ground truth data to verify the estimated state of the tracked objects at this point. While visual results seem very promising, for now the evaluation remains incomplete. Thus, this is one of the most obvious areas in which this thesis could be expanded upon.

The neural network approach could also be improved. The network can be integrated with the rest of the algorithm even more. Currently, its output is only used to decide between two different trackers, one for cars and the other for everything else. However, pedestrians or cyclists also display different behaviour and the filter could therefore benefit from having tailor-made motion models for both of them. This could also be extended to even more road participants like e.g. trucks. Training of the network can also be improved. The current approach of training

and testing on split datasets from the same scenario is not optimal. Ideally the neural network should be trained on a much bigger dataset, preferably coming from many different scenarios, and it should then be checked how well it generalizes to data from a new, previously unseen, scenario.

In addition, the algorithm should be extended beyond the thesis delimitations 1.1.1 present in this work. These mainly include accounting for ego-motion and to track targets not only in x- and y-, but also in the z-direction.

Some implications of using PHD filters for multitarget tracking should also be investigated. In the evaluation, a maximum of around 10 targets at any given point were seen. Since the estimation of the number of targets in a PHD filter is a Poisson distribution, the mean number of targets is also the variance of that value. Therefore, a higher number of estimated targets equals a higher variance in that estimate. Thus, it would be interesting to see how the PHD filters perform in a scenario with a much higher number of targets. A way to eliminate the problem of high variance with many targets is to utilize the Cardinalized Probability Hypothesis Density (CPHD) filter instead, which in addition to the PHD also propagates the cardinality of the set of targets. However, the CPHD has higher computational complexity than the PHD. As such, implementing and comparing the CPHD to the PHD when using a lidar sensor is an area for future research.

Additionally, a more sophisticated method of placing birth PHD ought to be considered. In this thesis the birth PHD was placed in areas where clustered objects were found. While this is sufficient for an early prototype, it does not necessarily extend to a practical implementation of the proposed algorithm.

Regarding the problems arising from using fixed parameters in the clustering algorithm, it should be examined whether a dynamically decreasing minimum number of points and increasing maximum distance between two points could help to augment the region of perception.

Nomenclature

CPHD	Cardinalized Probability Hypothesis Density
EKF	Extended Kalman Filter
ERT	Extended Rectangular Target
FISST	Finite-set statistics
FOV	Field of View
GIW	Gaussian Inverse Wishart
GM	Gaussian Mixture
IMM	Interacting Multiple Models
ISED	Incremental Sub-Matrix Eigen Decomposition
JMLS	Jump Markov Linear Systems
KF	Kalman Filter
MGP	Measurement Generating Point
MHT	Multiple Hypothesis Tracking
PHD	Probability Hypothesis Density
PT	Point Target
RFS	Random Finite Sets
RMS	Root Mean Square
SNR	Signal to Noise Ratio
TPM	Transition Probability Matrix
UKF	Unscented Kalman Filter

Bibliography

[1] The self-driving car in action – drive me.
URL http://www.volvocars.com/intl/about/our-innovation-brands/intellisafe/intellisafe-autopilot/drive-me

[2] Model s software version 7.0.
URL https://www.teslamotors.com/sv_SE/presskit/autopilot

[3] How google's self-driving cars detect and avoid obstacles.
URL http://www.extremetech.com/extreme/189486-how-googles-self-driving-cars-detect-and-avoid-obstacles

[4] A. Geiger, P. Lenz, C. Stiller, R. Urtasun, Vision meets robotics: The kitti dataset, International Journal of Robotics Research (IJRR).

[5] S. Särkää, Bayesian Filtering and Smoothing, Cambridge University Press, 2013.

[6] E. Wan, R. V. D. Merwe, Chapter 7 the unscented kalman filter, in: Kalman Filtering and Neural Networks, Wiley, 2001, pp. 221–280.

[7] S. Julier, J. Uhlmann, H. F. Durrant-whyte, Technical notes and correspondence a new method for the nonlinear transformation of means and covariances in filters and estimators.

[8] T. L. C. K. L. B. Lawrence D. Stone, Roy L. Streit, Bayesian Multiple Target Tracking, Second Edition, Artech House, 2014.

[9] W.-K. M. Ba-Ngu Vo, The Gaussian Mixture Probability Hypothesis Density Filter, IEEE TRANSACTIONS SIGNAL PROCESSING 54 (11) (2006) 4091–4104.

[10] R. P. Mahler, Multitarget bayes filtering via first-order multitarget moments, Aerospace and Electronic Systems, IEEE Transactions on 39 (4) (2003) 1152–1178.

[11] C. Walck, Hand-book on STATISTICAL DISTRIBUTIONS for experimentalists, Particle Physics Group, University of Stockholm, 2007.

[12] D. E. Clark, K. Panta, B.-N. Vo, The gm-phd filter multiple target tracker, in: Information Fusion, 2006 9th International Conference on, IEEE, 2006, pp. 1–8.

[13] D. M. Karl Granström, Stephan Reuter, A. Scheel, A multiple model phd approach to tracking of cars under an assumed rectangular shape, in: Proceedings of the International Conference on Information Fusion, 2014, pp. 1–8.

[14] J. W. Koch, Bayesian approach to extended object and cluster tracking using random matrices, IEEE Transactions on Aerospace and Electronic Systems 44 (3) (2008) 1042–1059.

[15] D. Barber, Bayesian Reasoning and Machine Learning, Cambridge University Press, 2014.

[16] K. Granstrom, U. Orguner, A phd filter for tracking multiple extended targets using random matrices, IEEE Transactions on Signal Processing 60 (11) (2012) 5657–5671.

[17] K. Granstrom, U. Orguner, Implementation of the GIW-PHD filter, "Linkoping University".

[18] G. James, D. Witten, T. Hastie, R. Tibshirani, An Introduction to Statistical Learning: with Applications in R, Springer Texts in Statistics, Springer New York, 2014.

[19] C. Bishop, Pattern Recognition and Machine Learning, Information Science and Statistics, Springer, 2006.

[20] T. Hastie, R. Tibshirani, J. Friedman, The Elements of Statistical Learning: Data Mining, Inference, and Prediction, 2nd Edition, Springer series in statistics, Springer, 2009.

[21] Velodyne, Hdl-64e s2 datasheet_2010, original document from Velodyne (2010).
URL `http://velodynelidar.com/lidar/products/brochure/HDL-64E%20S2%20datasheet_2010_lowres.pdf`

[22] A. Moore, An introductory tutorial on kd-trees, Tech. Rep. Technical Report No. 209, Computer Laboratory, University of Cambridge, Computer Laboratory, Pittsburgh, PA (1991).

[23] X. Shen, S. Pendleton, M. H. Ang, Efficient l-shape fitting of laser scanner data for vehicle pose estimation., in: RAM/CIS, IEEE, 2015, pp. CIS:173–178.
URL `http://dblp.uni-trier.de/db/conf/ram/ram2015.html#ShenPA15`

[24] M. Cotra, M. Leue, Lidar-based methods for tracking and identification.
URL `https://www.youtube.com/watch?v=_Mhgm2BXdFI`

[25] K. Granström, C. Lundquist, U. Orguner, Tracking rectangular and elliptical extended targets using laser measurements, in: Information Fusion (FUSION), 2011 Proceedings of the 14th International Conference on, 2011, pp. 1–8.