Occupancy grid mapping for dynamic object detection using a Particle Filter approach

van Dijk, Maarten H.

Award date:
2023

Occupancy grid mapping for dynamic object detection using a Particle Filter approach

Master Thesis report

M.H. van Dijk
ID 0949952
DC2023.071

Master:	Automotive Technology
Department:	Mechanical Engineering
Research group:	Dynamics & Control
Thesis supervisors:	dr.ir. T.P.J. van der Sande dr. Ö. Aslan

Group of Dynamics & Control
Eindhoven University of Technology

Eindhoven, Tuesday 15th August, 2023

This report was made in accordance with the TU/e Code of Scientific Conduct for the Master thesis

Preface

This master thesis is conducted as part of the Master Automotive Technology at Eindhoven University of Technology. This research marks the end of my studies, a period full of new experiences, both in and outside my studies. Hereby, I would like to thank all the people that supported me along the way.

Above all, I would like to thank my thesis supervisors Tom van der Sande and Ömur Aslan for their feedback and understanding. I always felt I could ask my questions to them, and I enjoyed the productive discussions during the progress meetings. This is very much appreciated.

Next, I would like to thank my close family and friends. For their unconditional support, patience and distractions. Special thanks to Sifa, who always looked after me, supported me and was there during the ups and downs.

Maarten van Dijk, August 2023

Abstract

Mapping the environment of vehicles has been an active research area for decades, for example for autonomous robots [1]. However, automotive applications have some specific challenges which make mapping the environment of vehicles not the easiest task.

This master thesis contributes to existing research by focusing on an environment mapping algorithm that uses an automotive radar as sensor providing measurements. Furthermore, this algorithm focuses on mapping a dynamic environment since automotive applications have to deal with highly dynamic objects, in contrary to the static environments that most research for robotic autonomous applications considers. Therefore, first research contributions are presented including additional background information and preliminaries on particle filter algorithm. Next, an estimation method is presented to localize the dynamic objects in the direct environment of the ego vehicle, and to consider its own movement. This estimation method does not only use positional information, but processes more measurement information in the form of Doppler radar velocity information. Moreover, an approach to deal with occlusion and loss of measurement is introduced.

The evaluation compares the new estimation method to the Bayesian Occupancy Filter used throughout other literature [2]. Different scenarios are defined and metrics to score the performance are implemented. Also, different parameters of the estimation methods are tuned and their effects on performance are analyzed. The new estimation methods seems to only outperform the Bayesian Occupancy Filter in specific scenarios, so further research is needed for optimizing the approach.

Keywords: autonomous vehicles, environment mapping, particle filter algorithm, automotive object detection, Bayesian Occupancy Filter

List of symbols

Below a list of symbols which are used throughout the report is presented. Additionally, a list of subscripts and superscripts, which are used in the particle filter algorithm explanation, is shown.

Symbol	Description
x	Longitudinal direction in vehicle-fixed frame (Cartesian)
y	Lateral direction in vehicle-fixed frame (Cartesian)
r	Range, distance from origin to point (polar)
θ	Azimuth angle, angular direction (polar)
v	Velocity vector
t	Time
μ	Mean of distribution
Σ	Covariance of distribution
p_o	Probability value of being occupied
p_f	Probability value of being free
m_c	Cell c in occupancy grid map m
z_t	Measurement at time t
x_t	Pose estimate at time t
λ^c	Log odds ratio for cell c
γ	Exponential forgetting factor
n	Amount of samples for non-regular sampling
λ_n	Parameter for adjusting the precision of the sampling approach
q	Particle
z	Measurement
w	Weight parameter of particle
N_{total}	Total number of particle in algorithm
ΔT	Difference in time between consecutive algorithm steps
u	Noise
a	Acceleration
λ	Parameter for weight factors
$s_{p,pi}$	Distance from particle p to particle p_i
S_d	Cut-off radius
β	Resampling Wheel step size
ψ	Factor for occupancy grid translation

Subscript & superscripts	Description
x	Longitudinal component
y	Lateral component
θ	Radial component
i	Particle index
k	Time step index
p	Positional factor
v	Velocity vector factor
d	Density factor

Contents

Chapter 1

Introduction

More and more new consumer vehicles include some sort of Automated Driving System (ADS), such as cooperative adaptive cruise control or automated lane keeping systems. These systems aim to improve primarily the safety of the driver, by taking over functions which the driver would normally perform. Automating a certain driving task seems to be an effective way to improve road safety, since a recent survey by the National Highway Traffic Safety Administration (NHTSA) found that 94% of crashes in light vehicles can be related to human driving error [3]. Some companies even aim to develop systems which can take over multiple tasks, or even fully automate the entire driving task. Safety considerations are not the only reason to have more automation in road vehicles. For instance, intelligent communication between vehicles and infrastructure and thereby enabling technologies such as platooning or smart scheduling of traffic lights will also increase the throughput of roads or intersections. To enable these kind of smart road transportation systems, cooperation and communication between vehicles and infrastructure are required for improving the performance of entire road transportation systems [4]. An illustration of how the environment could look like in an automotive application is provided in Figure 1.1.

This project aims to contribute to the development of automated driving, specifically by continuing recent research conducted for the Integrated Cooperative Automated Vehicles (i-CAVE) project [5]. The i-CAVE project addresses current transportation challenges regarding throughput and safety with an integrated approach to automated and cooperative driving. The i-CAVE project contains seven research areas, which each focus on a specific part of the project and help to realize a Cooperative Dual Mode Automated Transport (C-DMAT) system, consisting of dual mode vehicles which can be driven automatically and manually to allow maximum flexibility. Adequate environment perception is one of the requirements to realize this advanced transportation system.

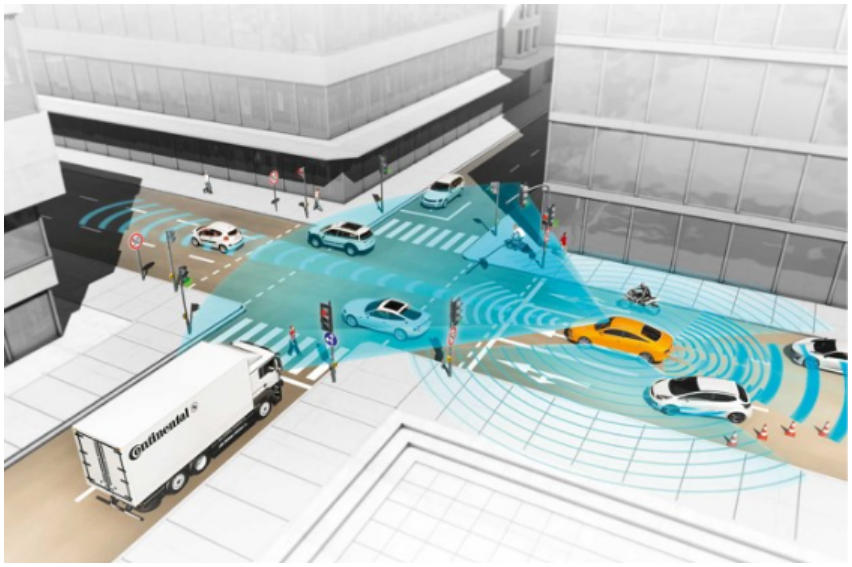

Figure 1.1: Illustration of environment perception in an automotive application

Mapping the environment of vehicles has been an active research area for decades. Lots of research specifically focuses on autonomous robots [1, 6] (see also Figure 1.2). Applications here mostly focus on exploring dangerous or impenetrable area for humans, such as deep underwater landscapes or for example even the surface of the planet Mars [7]. Lately, with the advances in technology and mainly the availability of cheap electronics, drones are an expanding research topic, and adequate perception methods are also necessary for creating aerial unmanned vehicles. This research will focus on automotive applications, which has many similarities to the research conducted on autonomous robots. However, automotive applications have some specific challenges, as will be illustrated.

This study will focus on realizing an environment mapping algorithm using only an automotive radar as sensor input. The algorithm will help in creating a perception method for the vehicle, to eventually be used by further automated driving functions. Important for the automotive application is the correct representation of dynamic objects, next to the static environment. The focus lays on using a single sensor that directly measures position and velocity. Most environmental mapping methods rely on position information only, which is used to map a static environment; this study focuses on the dynamic environment and will use a radar sensor to get velocity information directly from the environment. Keeping in mind that this algorithm is a small part of more extensive automated driving functions, the output will be presented in a non-specific, general form, to allow several applications access to the environment perception information.

Figure 1.2: A cooperative multi-robot system from research of Reid et al.[6].

1.1 Overview of automotive environment mapping

This section is dived into two subsections, respectively Map Representation and Estimation techniques. These subsections will elaborate on the need for these concepts and provide more relevant information for this research.

1.1.1 Map representation

For adequate environment perception, a suitable environment mapping approach is needed. Both static and dynamic entities should be represented in the chosen environment map. The static environment contains entities and objects around the vehicle that have no relative motion with respect to the fixed world, and thus the motion is only defined by the ego vehicle motion. Dynamic environment objects do have a relative motion with respect to the fixed world. For example other traffic is considered part of the dynamic environment, whereas guard rails of traffic lights are considered static.

A common approach to environment mapping in mobile robotics is to represent the environment in a spatial lattice map, containing occupancy states defined for each cell. These kind of maps are called occupancy grid maps, as mentioned in the works of H. Moravec [8] and A. Elfes [9] (see also Figure 1.3). The main advantage of these maps is the ease of incorporating measurement and pose uncertainties in a multi-view and multi-sensor scenario. Occupancy grid maps can also accommodate noise of any distribution, for example Gaussian noise. Moreover, the algorithm knows relatively simple update equations to incorporate new information in the current map.

Occupancy grid maps are one of the most used map representations for autonomous vehicles, but suffer from some problems when it comes to autonomous driving. They have high memory and bandwidth requirements, which might be a problem when it comes to communication. There are some solutions for this, by employing compression techniques such as quad-trees [10]. Another possibility is to create a longitudinal grid, with continuous lateral info, thereby eliminating data of what would otherwise be empty lateral cells [11]. This approach is most suited for scenarios with traffic mainly moving in longitudinal direction, such as highway scenarios where Adaptive Cruise Control (ACC) is used. Apart from the bandwidth requirements, occupancy grids are not directly suitable for representation of dynamic objects,

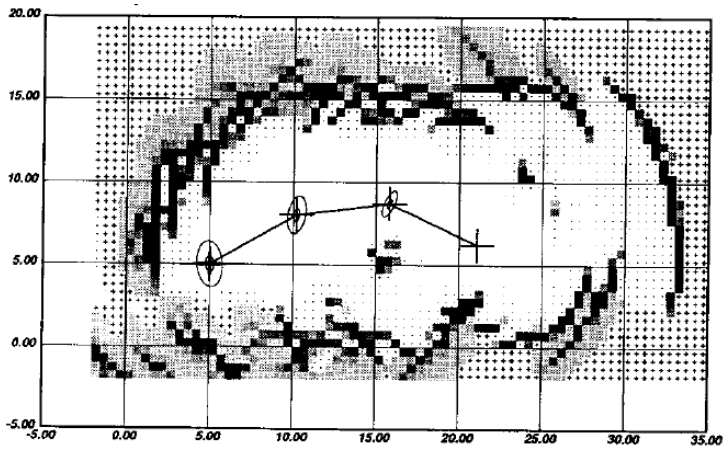

Figure 1.3: Occupancy grid representation from the research of A. Elfes [9]

since the conventional occupancy grid assumes a static world. To deal with this, some models combine the grid-based representation for static objects with a object-based representation for dynamic object [12, 13]. Another approach is to include velocity components in the state space, as shown in the work of C. Coué [14].

Another way to map the environment, and very common in mobile robotics, is Simultaneous Localization and Mapping (SLAM). Here, environment maps are created, while simultaneously trying lo localize the ego vehicle within this unknown environment. These maps are often world-centric, and new local vehicle-centric maps, obtained by the ego vehicle, are added to this global representation. Often, these techniques make use of feature matching to recognize earlier detected shapes or object in the environment. The main difference with respect to many automotive applications, apart from the world-centric approach, is that these models often try to exclude dynamic objects from the environment map, since they are of lesser importance in typical mobile robotics scenarios. For example in [15], a Gaussian Mixture Model (GMM) is used to differentiate static from dynamic objects, and provide lower weight to moving objects, so only static observations are preserved. This is very beneficial for mapping an unknown landscape, but not suitable for automotive applications where detailed information about dynamic objects is very valuable.

Environment representation can also be done as model based object detection, instead of a map based approach. Map based representations assign location-specific information to elements in a space, and give some proximity relation between these elements. In contrast, model based object detection makes assumptions about common object properties, for example the shape and motion of dynamic obstacles, to distinguish them from non-objects or the environment background. Tracking objects can be done on multiple levels, known as macroscopic or microscopic motion tracking [16]. Model based object detection is common for automotive applications, and often makes use of some sort of feature extraction, such as edge template matching [17].

1.1.2 Estimation techniques

A large challenge in automotive environment mapping is predicting the propagation of the dynamic states of the environment. Since the dynamic states can be highly non-linear, an adequate estimation technique needs to be selected, which is able to accurately predict the occupancy of dynamic objects. This prediction can be done through filter methods.

A common filtering method is the Bayesian filter algorithm, which works according to the prediction and update principle. A prediction is done to estimate the future states, which is provided as a distribution. There are broadly two classes in filtering, namely Gaussian and non-parametric filters. The first uses a multivariate normal distribution to represent the posterior, while non-parametric filters do not rely on a fixed functional form, but rather on a finite number of values [18]. Non-parametric approaches can be further divided into histogram filters and Particle Filters (PF).

Kalman filters are the most well known Gaussian filter approaches. Gaussian filters assume that the probability can be represented by a multivariate normal distribution, most commonly described by a mean μ and a covariance Σ. This is common in robotics, where the posterior is focused around one true state with some margin of uncertainty. There are various implementations of Kalman filters, but the standard Kalman filter is an optimal estimator for linear processes, which works using the prediction and update principle. This can be applied to target tracking applications [19]. However, dynamic object in automotive applications do not only show linear behaviour, but also nonlinear movement such as longitudinal accelerations and cornering. Therefore, dynamic objects can be hard to track by the standard Kalman filters. In this applications, the Extended Kalman Filter (EKF) is a better option. The EKF handles non-linear functions by linearising through computing the Jacobian matrix around the current estimates. For highly non-linear functions however, the EKF cannot provide accurate enough estimates, and the Unscented Kalman Filter (UKF) performs better [20, 21]. The UKF gets a set of sample point of the prior distribution and propagates them using non-linear functions to get sample point of the posterior distribution. From those points, a new mean and covariance estimate are formed.

Non-parametric filters such as histogram filters approximate the posterior by a finite number of values instead of a continuous distribution. Histogram filters, and also discrete Bayes filters such as the Bayesian Occupancy Filter (BOF) [14], divide the state space under attention into finitely many regions, which all have their own posterior probability. Particle filters also use a finite number of parameters, however they are different to histogram filters in the way that they populate the state space. The key idea is that the posterior distribution can be represented by a set of random state samples. Instead of using the parametric form, the distribution is represented by a set of samples drawn from this distribution. In this way, it is possible to represent many different distributions. An example of a particle filter on a grid based application is the Sequential Monte Carlo BOF (SMC-BOF), which can also be used to approximate velocities of particles [22].

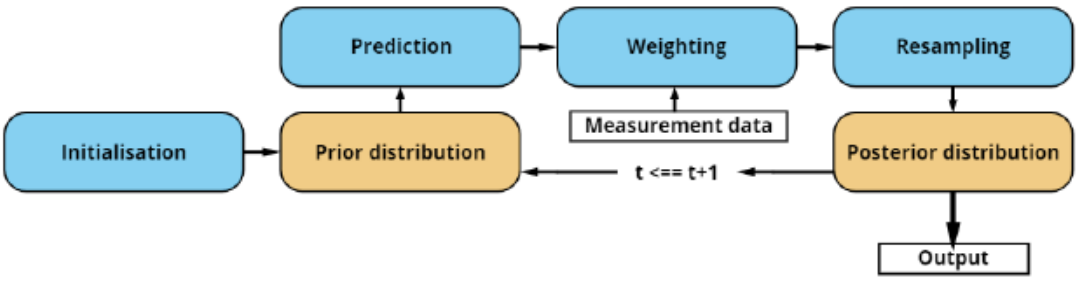

Figure 1.4: Block diagram of the Particle Filter from Teeuwen [23]

Since the particle filter has been used in automotive applications successfully before, a brief explanation will be given. A representation of the algorithm flow is given in Figure 1.4. Initially, a population of particles is generated over the state area. Each particle has a weight and a state vector describing its position and velocity. First, the prediction step updates each particle's states using a Constant Velocity Constant Orientation (CVCO) assumption [24] and random process noise. Then, the particles each receive a weight based on their correspondence with the measurements. Finally, a resampling method is used to duplicate the particles with the most weight to the new set, used for the next time step. In the resulting particle distribution, dense clusters of particles represent a higher likelihood of occupancy, and this can be used to derive an occupancy grid probability.

1.2 Problem statement

The problem at hand focuses on dynamic environment mapping for automotive vehicles. In contrary to most mapping methods developed for robotic autonomous applications, automotive applications have to deal with a highly dynamic environment instead of a static environment. Although there is still a static environment, consisting of lampposts, barriers, parked vehicles etc., the main challenge lays in predicting the dynamic environment. Other road users, such as trucks, cars, cyclists and pedestrians, all make use of the same space as the ego vehicle. The movement of these other actors is unpredictable and often

non-linear. This makes their movement very difficult to capture, especially in an environment where for example sensors can sometimes miss detections, or objects can disappear shortly out of sight.

To tackle this problem, firstly a suitable map representation is necessary. The representation has to accommodate the processing of measurement information and provide space for uncertainties, resulting from e.g. measurement uncertainty from the radar sensor or from vehicle localisation. From the representation it should be clear at which location there is a high likelihood of objects in the environment.

Secondly, an estimation method needs to be found to localize dynamic objects in the environment of the ego vehicle. Measurement information is provided in the form of radar sensor data. Using this data, the estimation method should try to represent as accurately the dynamic objects in the environment. This should also take into account measurement uncertainties, occasional loss of measurements and ideally also occlusion of objects. Moreover, the movement of the ego vehicle itself will need to be taken into account.

Not in scope of this project is the processing of raw sensor data. It is assumed that processed radar data is available, in the form of points in a two-dimensional space. These points result from reflections of the radar beams on surfaces in range of the radar, therefore it is possible to have multiple measurement points from a single object. Also, the used automotive radar sensor provides velocity information through the Doppler effect. This velocity information is provided as a two-dimensional vector for each measurement point, however it only describes velocity in the same direction as the radar beam is emitted, therefore by default not providing the full velocity information of targets. Additional sensor information comes from the inertial navigation system (INS) of the vehicle, which provides noisy velocity and acceleration information.

The majority of the environmental perception approaches uses measurement data containing positional information of possible objects. However, with only a single radar sensor, not only positional information is present, but also velocity measurements are available. Since the goal of this research is to find an estimation method that represents the dynamic objects in the environment as accurately as possible, it seems beneficial to use as much information about the environment as possible. To that end, also the velocity information will be used in the estimation method, and it will be evaluated if the addition of this velocity information improves the prediction of the dynamic environment.

Moreover, loss of sensor data and occlusion of objects still poses a big challenge for conventional filter approaches. Through specific filter design, it might be possible to better account for these occurrences, and more accurately represent an object when measurements on that object are temporarily unavailable. An addition to the estimation method will be proposed to deal with this measurement loss.

1.3 Research contributions

Former research [23] presented a dynamic environment mapping approach for the i-CAVE project. This research made use of an occupancy grid map, in which occupancy of objects in the environment is represented as a probability per grid cell. In earlier occupancy grid research projects, BOF's are used to assign a probability per cell, to eventually create a distribution over the state space [25]. However in more recent research papers, the occupancy grid is more used as a way to represent occupancy, and instead a particle filter is used to represent the distribution. This particle filter is then translated to an occupancy distribution over a two-dimensional grid [22].

Although the particle filter method has been used before in automotive applications, the weighting of particles has still greatly relied on occupancy grid probabilities [22]. In [24], Nuss shows a novel method to incorporate velocity as a means of weighting the particles for the particle filter. However, this method relies on the occupancy grid representation, by combining the particle filter results with a binary Bayes filter. Moreover, this method was developed for fusion of laser and radar data, and therefore has only been evaluated by testing the laser only variant against a laser radar combination. This research will try to decouple the particle filter algorithm from the occupancy grid, gaining more freedom in the application of the particle filter steps, but still retaining a connection to the occupancy mapping approach by translating the particle distribution back to an occupancy map.

Another difficulty for estimating dynamic objects lays in handling missed detections and occlusions. Due to the weighting and resampling in particle filters, already some particles should be able to survive a single missed measurement. However, this is greatly dependent on the chosen weighting and resampling methodology. In [23], a solution has been proposed, where a density weight is added, adding importance to particles which are closer to other particles. This method is also reliant on the underlying grid structure, since it evaluates the amount of particles per cell. And although it has been shown to keep particles alive longer after a measurement disappears, it is not clear if this weight actually improves the overall accuracy of the algorithm. It might cause worse performance in highly dynamic environments, and may even keep particles which should correctly be discarded, instead of keeping them under the assumption that the object they represent is occluded. Therefore, a new weighting approach will be suggested.

The solution in [23] was developed in a simulation environment and simulated for a 1D+ setup, 1D+ here referring to a one-dimensional environment with different levels of height as an extra '+'-dimension. Different simulation and measurement scenarios were defined and the algorithm was tested. The concept was shown to estimate occupancy well in the simulated scenarios. However, the 1D+ approach is still limited in representing realistic, 2D scenarios. Although going from 1D to 2D might seem trivial, some challenges with respect to measurement data and motion compensation will need to be tackled.

Finally, evaluating environment mapping approaches is not straightforward. When running a real world test, often ground truth data is not available, and the evaluation will mainly focus on improving the results within a single algorithm by tuning parameters. This makes it hard to compare different algorithms with each other. This project will simulate driving scenarios, and through the extraction of ground truth data, it should be able to compare different algorithms to each other. Performance evaluation will be done using specific metrics which are relevant for automotive applications.

In conclusion, there are various improvements to be made to the existing particle filter approaches. An extension to two dimensions from Teeuwen [23] (see also Figure 1.4) will not only require some trivial changes to the algorithm and simulation, but will also need to incorporate two-dimensional velocity vectors with velocity information in radial direction. Moreover, the work of Teeuwen has only been tested with a static ego vehicle, whereas in more realistic scenarios, the ego vehicle will have ego movement. Also, the weighting of particles will be reviewed, to find an optimal way to deal with loss of measurement and occlusion. This will be evaluated against a state-of-the-art alternative.

1.4 Overview and report outline

The goal of the algorithm is to track dynamic objects in the direct environment of the vehicle, and produce a map representation which provides region-level information about occupied and free area. To cope with the sparse information provided by the radar sensor, and to deal with the non-linear behavior of dynamic objects in the environment, a particle filter approach is chosen. The output from the particle filter is then translated into an occupancy grid map. This map representation provides a simple 2D representation of objects in the environment, and can later be used for i.e. path planning or fusion of other sensor data. In this research the occupancy grid map resulting from the filter approach will be used to evaluate the performance of the algorithm. Through creating an occupancy grid map, it is also possible to evaluate the particle filter algorithm against a different method for occupancy grid mapping.

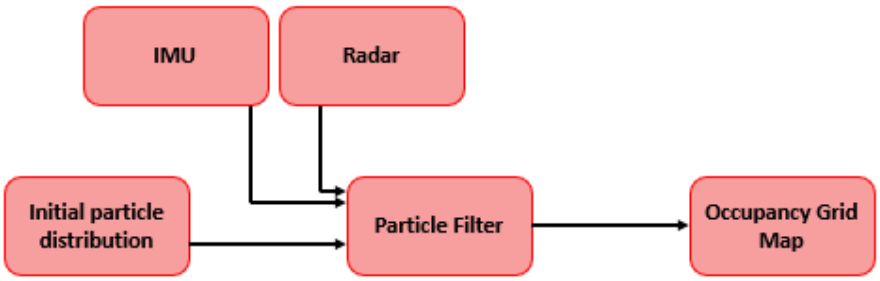

Figure 1.5: Overview of complete algorithm flow

The particle filter algorithm and evaluation method will be created using a simulation environment. An overview of all the different simulation steps is shown in Figure 1.5. Vehicle sensors, such as the inertial navigation system and a radar sensor, will be simulated and will produce the necessary measurements. The developed algorithm starts by creating an initial particle distribution, then continuously evaluates the measurements and the particle distribution, and will produce an occupancy grid map as output.

This report will first continue with a preliminaries chapter, which is necessary to create the framework for the particle filter algorithm. The coordinate system, simulation environment and the Bayesian Occupancy Filter approach will be explained in more detail. Then, in Chapter 3, the proposed particle filter algorithm will be explained in further detail. Here also the solutions for incorporating velocity information and for dealing with measurement loss will be presented, along with the translation to an occupancy grid. Next, the evaluation criteria and simulated scenarios will be shown in Chapter 4, and the produced results will be presented along with a discussion of these results. Finally, a conclusion will be drawn and recommendations for further research will be given in Chapter 5.

Chapter 2

Background and preliminaries

This chapter will introduce several topics relevant to the particle filter algorithm, without going into detail about the proposed approach itself. First, an overview about the used coordinate frameworks is given, which is consistently used throughout the report. Second, more detail will be given about the simulation environment, including the workings of the simulated sensors. Finally, the Bayesian Occupancy Filter will be explained. The Bayesian Occupancy Filter is a commonly used method for occupancy grid mapping, and a detailed explanation will be given about the workings of this algorithm, which will later be used to evaluate the performance of the particle filter approach.

2.1 Coordinate framework and notations

Throughout this work, different coordinate frameworks are used at different stages of the algorithm. This is due to the different simulation environments being used, and therefore translations to a common framework are necessary. Important is that this common framework is chosen to be vehicle fixed instead of world fixed. The reason for this is two-fold; first of all the aim of this research is to map dynamic objects in the direct surroundings of the ego vehicle. There is no interest in mapping the complete environment and its history, or localizing the vehicle in the environment. The second reason is to eliminate a large part of ego vehicle motion compensation. If a world fixed frame were to be chosen, the ego vehicle sensor data would have to be translated to this world fixed frame. Therefore, the ego vehicle position will need to be translated to the world fixed frame. This requires information about vehicle position, velocities and even accelerations. By using a vehicle fixed frame, ego vehicle measurements can be directly placed inside the vehicle fixed coordinate system. This saves a lot of translation and accompanying uncertainties from IMU measurements. However we will still need compensation for acceleration, which will be demonstrated in the next chapter.

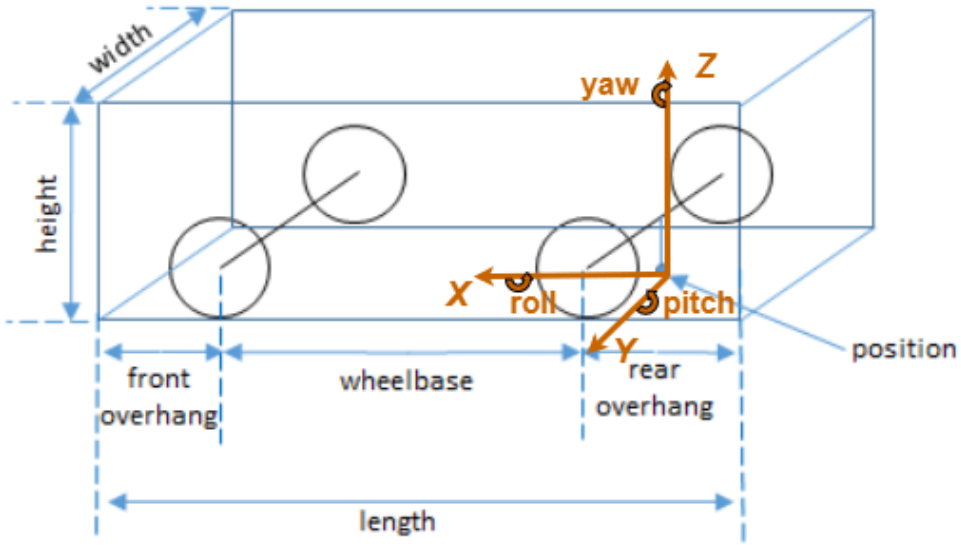

Figure 2.1: Ego vehicle and vehicle fixed coordinates (taken from `mathworks.com`)

The vehicle fixed frame is based on a 2D Cartesian coordinate system. Height (in z-direction) is not taken into account for this study. The origin of the vehicle fixed frame is always centered on the midpoint of the rear axis of the ego vehicle. The lateral direction is called y and is positive towards the left, as seen from a standpoint looking at the rear of the vehicle. The longitudinal direction x extends to the front of the vehicle and is orthogonal to y. It is positive in the vehicle's forwards direction. The yaw angle

is anti-clockwise positive around the z-axis, which extends through the roof of the vehicle orthogonal to both x and y. Since the symbol z is also used in a different context in this research and height is neglected, this symbol will no longer be used to describe coordinates. Roll and pitch are neglected in this study.

The ego vehicle has more properties, such as length, width and height. These are all as defined in Figure 2.1. The length, front overhang and wheelbase can be freely chosen, and the rear overhang will be adapted according to

$$\text{Rear overhang} = (\text{Length}) - (\text{Front overhang} + \text{Wheelbase}) \tag{2.1}$$

Measurements are expressed in ego vehicle fixed coordinates. An example is depicted in Figure 2.2. The measurement point in red has positional data expressed in x and y and two-dimensional velocity data denoted as v_x and v_y. These two velocity vectors are parallel to the x and y directions respectively. Notice that in the illustrated case, the y value of the measurement point position is negative, since it is to the right of the x-axis. Also, the value of v_y is negative, since it points to the negative y direction of the vehicle fixed frame. Important to realize is that the velocities are all relative to the ego vehicle. This is because the radar sensor measures relative velocity by making use of the Doppler effect. Since the algorithm runs in the vehicle fixed frame, it is not necessary to translate the velocity to the world fixed frame, but instead the relative velocity can directly be used.

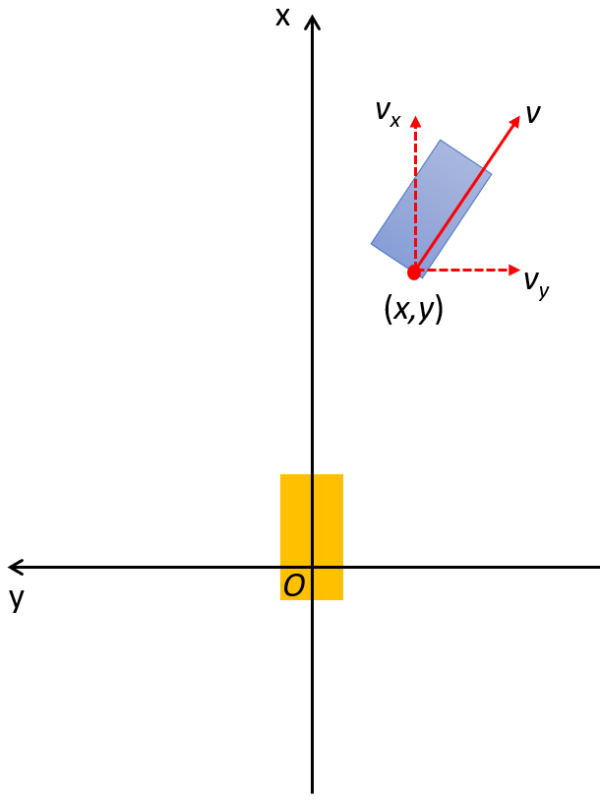

Figure 2.2

For visualization purposes, we will still define a world fixed frame on which the vehicle fixed frame can be projected. This world fixed frame is again a 2D Cartesian frame. Based on the true ego vehicle position, retrieved from simulation data, the vehicle fixed grid can be translated to the world fixed frame. The vehicle position is defined as the origin of the vehicle fixed frame. This translation is only used for visualization purposes, and will not be used in the particle filter algorithm itself.

2.2 Simulation environment

In this section, more details about the simulation environment will be given. The simulations are setup in the Matlab programming language [26]. An overview of all simulation components is given in Figure 2.3. In this figure, all blocks represent a core function of the simulation. The actual code exists of way more functions, but the functionality of several functions combined can be captured by these blocks. The color of the blocks depicts in what category we can classify the functionality; blue for scenario simulation, yellow for the particle filter algorithm, green for the Bayesian Occupancy Filter, and red for evaluation. The solid lines show the flow of the code. This starts at scenario selection and scenario simulation, which includes the sensor models. The scenario simulation runs completely before the rest of the code, and stores all necessary data and time steps in arrays for the rest of the code to access later. After the scenario simulation, the particle filter and Bayesian Occupancy Filter algorithms will run, using the data from the scenario simulation. In parallel to the algorithms, the evaluation code is also executed, using both ground truth data originating from the scenario simulation as well as the output from the respective algorithms.

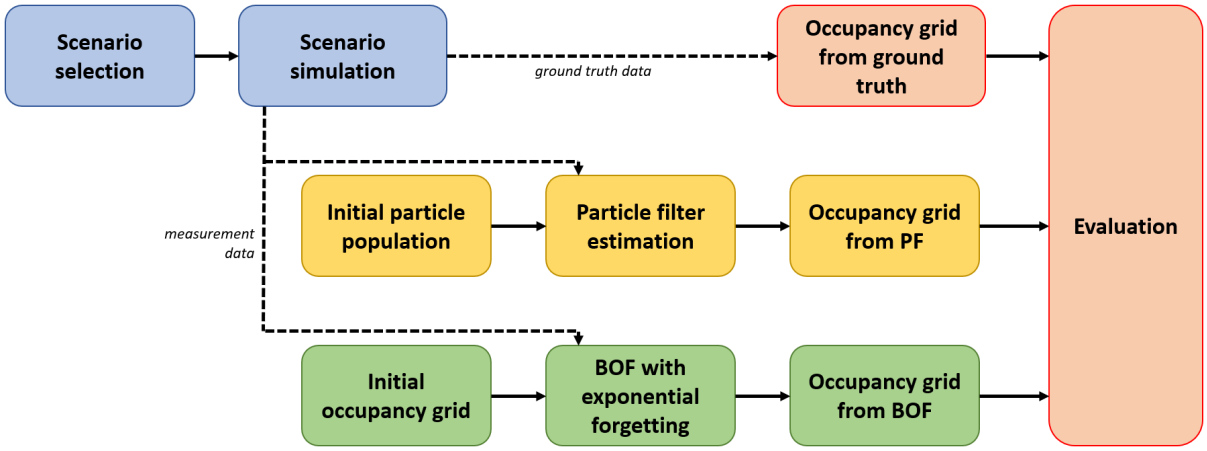

Figure 2.3: Overview of simulation flow. Solid lines show the order in which functions are ran, dashed lines show the transfer of data from the scenario simulation.

The scenario simulation and the creation of a ground truth occupancy grid will be explained in the following sections. The Bayesian Occupancy Filter will be the final topic of the current chapter. The next chapter will explain the particle filter algorithm. In Chapter 4, more details about the evaluation set-up will be given.

2.2.1 Scenario simulation

The scenario simulation has as goal to simulate different driving scenarios, and output realistic, simulated radar and IMU measurements. Next to this, the scenario simulation is also used to extract the ground truth data, which will be used to evaluate the performance of the filter algorithms. This part of the code makes use of the Automated Driving Toolbox from Matlab (see `mathworks.com/products/automated-driving.html` for more information). This toolbox allows for easy creation of a driving scenario. The ego vehicle and target vehicles are created using a Vehicle-object, which can be repeated for more vehicles in the same scenario. Each vehicle will have a starting location. Using the *smoothTrajectory*-function, it is possible to create a jerk-limited trajectory for each vehicle by defining several waypoints and a velocity which should be achieved at each waypoint.

The first block, called 'Scenario selection', has functionality to easily select a scenario from a predefined list of scenarios. The user can input a string which corresponds to a scenario name, and the resulting vehicles and tracks will be generated. This is then passed onto the Scenario simulation, where the scenario is ran at a certain frequency, and measurements are generated. More information on the specific scenarios used at the evaluation can be found in Chapter 4.

The most important part of the scenario simulation is the generation of radar data. The radar sensor is created using the *drivingRadarGenerator* object. This object simulates an automotive radar sensor which generates detections of target objects, with added random noise and false alarm detections. The radar sensor is placed facing forward at the front bumper of the ego vehicle, and has the properties as specified in Table 2.1. Most parameter values were taken from the datasheet of the Cocoon automotive radar from NXP [27]. Every 70 milliseconds, the radar generates new measurements. The range limits and the field of view size determine the area on which it is possible for the radar sensor to generate detections. Similarly, the range rate limits specify the minimum and maximum possible velocities of objects that the radar sensor can measure. Azimuth resolution and range bin size determine the minimum resolution at which the radar can distinguish between two targets, so an object spanning across a multiple of these resolution can generate more than one detection. For velocity, the range rate resolution specifies the minimum separation at which the radar can distinguish between two targets. The probability of detection defines the chance of detecting a target; it is set as the probability of detecting a target with a reference cross-section of 1 m^2 at a reference range of 100 m. The probability of false alarm describes the chance of producing a false alarm for each radar resolution cell, given by the azimuth and range resolution properties. The shown variances describe the Gaussian white noise profile on the measurements.

Table 2.1: Parameters and corresponding values of simulated radar sensor

Parameter	**Value**	**Unit**
Time interval	0.07	s
Range limits	[0.75 69.8]	meter
Range bin size	0.55	meter
Field of view	[-60 60]	degrees
Azimuth resolution	1.0	degrees
Range rates limits	[-69.4 41.7]	m/s
Range rate resolution	0.28	m/s
Probability of detection	0.90	-
Probability of false alarm	$1e^{-7}$	-
Variance range measurement	0.5	m
Variance range rate measurement	2.0	m/s

The resulting data from the radar sensor is in the form of a list of measurement points. Each point has four properties: x-position, y-position, velocity in x direction, velocity in y direction. All these coordinates are given in the vehicle fixed frame, since the radar sensor is attached to the ego vehicle. However, as mentioned in the introduction, the radar sensor can only provide velocity information in radial direction through the use of the Doppler effect. This radial velocity is translated to two Cartesian velocities in x and y direction, but it must not be mistaken that these velocity states therefore describe the full velocity states of an object. How to deal with this will be explained in Chapter 3. Along with the radar measurement, internal navigation system measurements concerning the vehicle acceleration are also created. This is done using the *insSensor* object from the toolbox. This object creates acceleration data and incorporates white noise, which is a realistic noise profile for inertial navigation sensors [28]. The standard deviation for the acceleration noise profile is set to 0.05.

2.2.2 Ground truth occupancy map

Apart from measurements, the simulation part will also output the actual ground truth values of the target vehicle location. These values are used to create a ground truth occupancy map, which will be used in evaluation to compare the filter algorithms against.

The ground truth occupancy map will consist of a two-dimensional Cartesian grid with square cells, to allow for easy comparison against the occupancy grid cells created by the algorithms. Each cell will have a probability value, which correlates to the probability of an object being present at that location. This probability for a cell c in occupancy grid m to be occupied can be written as $p_o(m^c|z_{1:t}, x_{1:t})$, and for a cell to be free as $p_f(m^c|z_{1:t}, x_{1:t})$, where $x_{1:t}$ is the pose estimate for time t and $z_{1:t}$ is the measurement

for time t. Both probabilities are compliments to each other, so

$$p_o(m_c|z_{1:t}, x_{1:t}) = 1 - p_f(m_c|z_{1:t}, x_{1:t}). \tag{2.2}$$

The input to create the ground truth map originates directly from the scenario simulation, and is given as the exact two-dimensional position of the midpoint of every object, along with the objects dimensions. The ground truth positions of objects are all given relative to the vehicle position, therefore the ground truth map is also created in the vehicle fixed frame.

For the ground truth occupancy, there are no other occupancy values necessary other than 0 and 1; it is given if a cell is occupied or free, and if only part of a cell is occupied by an object, the entire cell will be designated as occupied. However, there is some area which is not measurable for the ego vehicle radar sensor, since the radar sensor cannot emit radar beams through solid objects. This area can be seen as the 'shadow' which is being cast by objects. Looking from the radar viewpoint, all area behind an object cannot be detected anymore, since the radar beams are already deflected off the closest object. Therefore, the goal is to eliminate this immeasurable area from the evaluation. The ground truth map will be used to represent this immeasurable area.

Since it cannot be known what the occupancy is in this immeasurable area, these cells have equal probability to be occupied or free. Following Equation 2.2, this leads to the probability of 0.5 for such a cell to be occupied, or free for that matter. Now the problem arises to designate which cells in the two-dimensional Cartesian grid belong to this 'shadow' behind an object. It is possible to trace the area behind an object, however the techniques to do so are rather complex and time-consuming. One of the simplest ways is to find all the intersection points of a line and the Cartesian grid. This can be done using the Bresenham algorithm for line drawing [29]. To find every cell in the Cartesian grid which is behind an object, it will require a very large number of lines to be drawn: to make sure every cell in the shadow is captured, a line will need to be drawn from each grid cell on the outer edge of the state space towards the origin of the state space. This will be very time consuming and not very efficient, since different lines will cross the same cells for cells closer to the origin. Other options are to perform some sort of texture mapping algorithm, however these solutions are computationally expensive and are more suited for implementation on specific hardware (such as Graphical Processing Units)[30, 31].

Since the radar sensor emits a beam at every azimuth angle resolution over the full range of the field of view, the immeasurable area behind objects is easier to describe in a two-dimensional polar grid. A 2D polar grid has a dimension in the azimuth angle and in the range. Here the 'shadow' area is quite straightforward: for each angle, all cells which have a larger range than the range of an occupied cell, are immeasurable. This principle is illustrated in Figure 2.4. Now the immeasurable area can be easily described in polar coordinates. To do this, the ground truth data of the object positions are translated to polar coordinates first. Then, all the polar grid cells which are occupied according to this ground truth data are designated as occupied. Finally, for every angle on the polar grid, the occupied cell with the smallest range at this angle is looked up. All further cells with the same angle but a larger range will be assigned a probability value of 0.5. Finally, the translation back to a 2D Cartesian grid will need to be made. Both the translation from the Cartesian ground truth data to the empty polar grid as well as the translation from the filled in polar grid back to the Cartesian grid pose some challenges.

The first step is to translate the ground truth data to polar coordinates. The ground truth occupancy grid will need to depict all occupied area which is taken up by a vehicle, large or small. Therefore, not only the location of a object is important, but also its size and the resulting area which it covers. The ground truth data from the scenario simulation contains both the rear axle location of each target, as well as the width, length and properties such as rear overhang, wheelbase and front overhang. This is used to calculate each corner point of each vehicle, and then these points are transformed to polar coordinates. To find all polar grid cells which intersect with line, lines are set up between the corner points of the vehicles. The four lines which are set up can be seen as the bounding box lines of each vehicle. Using these four lines between the corner points, intersections with these lines and the grid are found as described by the pseudo-code below. With all these intersections, it is possible to assign a probability of 1 to all cells which are next to an intersection point. This will result in an outline of occupied polar grid cells for each vehicle.

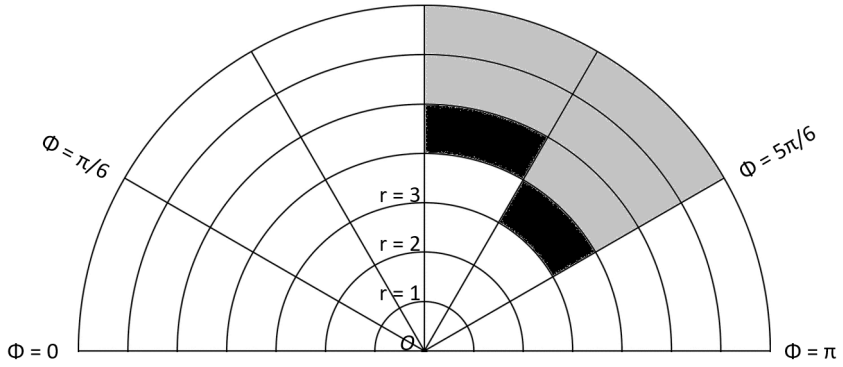

Figure 2.4: Radial occupancy grid. The black area has an occupancy probability of 1, all cells with the same angle but a larger range has an occupancy probability of 0.5.

```
corner point = 1
for corner point < 5
    find line between corner point and corner point + 1
    find theta grid line closest to theta(corner point)
    use line and theta to find range
    if next theta grid line is still on line
        go to next theta grid line
    else
        corner point = corner point + 1
    end
end
```

With the polar grid containing the vehicle bounding boxes, it is straightforward to implement the immeasurable area. The pseudo code for this is written below. Now a polar grid with immeasurable area is created, where the closest vehicle faces are represented as occupied area. This will need to be translated back to a 2D Cartesian grid. This is not trivial, since a polar grid has cells with different sized areas, and multiple cells may overlap with the Cartesian grid depending on the grid resolution and the exact location of the cell. The exact way to do this is to find all polar grid cell surfaces which overlap a grid cell, and then compute the fraction of each polar grid overlap, and summing the results for all overlaps of one 2D grid cell. This is computationally expensive, and there exist less exact ways which are more efficient [31].

```
for i = 1:#columns
    index = find(cell==1,'first')
    if ~isempty(index)
        GT_grid(i,index+1:end) = 0.5
    end
end
```

The chosen method is a sampling approach. In this approach, from every grid cell the midpoint is taken, and the corresponding polar coordinates are found. The value of the polar cell probability is evaluated, and this is taken as the probability value of the Cartesian cell. However, since polar grid cells have a different size area depending on the range, this method is not accurate for Cartesian cells which are close to the origin. To this end, a non-regular sampling is used, where more points are taken if a Cartesian grid cell is close to the origin [31]. The equation for the amount of samples is given by

$$n(x,y) = \lambda_n \frac{dx^2}{((r+\frac{dr}{2})^2 - (r-\frac{dr}{2})^2)d\theta} = \lambda_n \frac{dx^2}{rdrd\theta} \tag{2.3}$$

In Equation 2.3, dx is the length of a Cartesian grid cell, r is the range of the midpoint of the cell, dr is the range resolution of the polar grid and $d\theta$ is the angle resolution of the polar grid. λ_n is a parameter

to create more or less points, making the translation more accurate but slower or less accurate but faster. As can be seen, this equation relates the area of the Cartesian grid cells to the polar grid cells, and therefore results in more sample points at closer range. The minimum of n will be set to 1, so for grid cells further away, at least 1 sample point will be taken in each grid cell.

2.3 Bayesian Occupancy Filter and exponential forgetting

To have a comparison to the performance of the to be developed particle filter algorithm, a commonly used alternative for creating occupancy grid maps will be used. The method selected is the Bayesian Occupancy Filter, which has been widely used in estimating dynamic environments in combination with occupancy grids [12, 32, 33].

The Bayesian Occupancy Filter uses the Bayes rule as an update equation. The Bayes rule describes how to update a belief given some evidence, and in its basic form is

$$P(X|Y) = \frac{P(Y|X) \cdot P(X)}{P(Y)}, \tag{2.4}$$

where the left hand side is called the posterior, which is the updated probability of X given the evidence Y. $P(Y|X)$ is called the likelihood, which is the probability of the evidence given the belief is true, $P(X)$ is the prior probability before the evidence is considered and $P(Y)$ is the probability of the evidence. Ideally this rule is expressed in log-likelihood ratios, which will avoid truncation errors for probabilities close to 0 or 1. Additionally, the multiplication of probabilities will turn out as an addition of log-likelihoods. The goal is to find the joint probability distribution of all cells. This can be done on a per cell basis, as long at the individual cells can be estimated as independent random variables, which is the case for occupancy grids [9]. The derivation follows the work of Joubert [2].

Now the goal is to estimate the probability of each individual cell in the map m_c. Since the occupancy grid is fixed to the vehicle fixed frame, the pose information $x_{1:t}$ does not supply additional information about the environment, and the probability for each cell to be occupied can be written as

$$p_o(m_c|z_{1:t}, x_{1:t}) = p_o(m_c|z_{1:t}), \tag{2.5}$$

which only takes into account measurement information $z_{1:t}$. For the Bayesian Occupancy Filter algorithm, the provided measurement information represents the position states of detections from the radar sensor. Using Bayes' rule from Equation 2.4 while assuming independence between measurements, Equation 2.5 can also be written as

$$p_o(m_c|z_{1:t}) = \frac{p_o(z_t|m_c, z_{1:t-1})p_o(m_c|z_{1:t-1})}{p(z_t|z_{1:t-1})} = \frac{p_o(z_t|m_c)p_o(m_c|z_{1:t-1})}{p(z_t|z_{1:t-1})}. \tag{2.6}$$

From Bayes' rule also follows

$$p_o(z_t|m_c) = \frac{p_o(m_c|z_t)p(z_t)}{p_o(m_c)}, \tag{2.7}$$

and substituting Equation this into 2.6 gives

$$p_o(m_c|z_{1:t}) = \frac{p_o(m_c|z_t)p(z_t)p_o(m_c|z_{1:t-1})}{p_o(m_c)p(z_t|z_{1:t-1})}. \tag{2.8}$$

Since p_o is the compliment of p_f, as noted in Equation 2.2, similarly

$$p_f(m_c|z_{1:t}) = \frac{p_f(m_c|z_t)p(z_t)p_f(m_c|z_{1:t-1})}{p_f(m_c)p(z_t|z_{1:t-1})}. \tag{2.9}$$

The factors $p(z_t)$ and $p(z_t|z_{1:t-1})$ are not easy to compute, since they represent probabilities of the measurement z_t happening under any circumstance or given all the previous measurements. These factors can be eliminated by dividing 2.8 by 2.9, which yields

$$\frac{p_o(m_c|z_{1:t})}{p_f(m_c|z_{1:t})} = \frac{p_o(m_c|z_t)}{p_f(m_c|z_t)} \frac{p_o(m_c|z_{1:t-1})}{p_f(m_c|z_{1:t-1})} \frac{p_f(m_c)}{p_o(m_c)}. \tag{2.10}$$

From here, the natural logarithm is taken on both sides, giving

$$\log\left(\frac{p_o(m_c|z_{1:t-1})}{p_f(m_c|z_{1:t-1})}\right) = \log\left(\frac{p_o(m_c|z_t)}{p_f(m_c|z_t)}\right) + \log\left(\frac{p_o(m_c|z_{1:t-1})}{p_f(m_c|z_{1:t-1})}\right) - \log\left(\frac{p_o(m_c)}{p_f(m_c)}\right). \tag{2.11}$$

This log odds ratio equation is easier to read. On the left hand side, the log odds ratio represents the posterior probability for grid cell m_c. The first term on the right hand side shows the addition of new measurements, the second term shows the posterior probability resulting from previous measurements. The final term depicts the probability of the cell at initialisation. This logarithm will turn out to be 0 if the initial probability is set to 0.5.

This log odds ratio on the left hand side can be converted to a probability, since if

$$\lambda = \log\left(\frac{p_o(m_i|z_{1:t})}{1 - p_o(m_i|z_{z:t})}\right), \tag{2.12}$$

it follows that

$$p_o(m_i|z_{1:t}) = \frac{e^{\lambda}}{1 + e^{\lambda}}. \tag{2.13}$$

This simplifies Equation 2.11 further to

$$\lambda^c_{1:t} = \lambda^c_t + \lambda^c_{1:t-1} - \lambda^c_0, . \tag{2.14}$$

The resulting occupancy values from the log odds ratios are shown in Table 2.2. The threshold is introduced from a practical standpoint, since the ratios would go to plus or minus infinity. Without limitation on the ratios, they will react slow to changes. For each cell, the occupancy state from the table is assigned if the log odd ratio is lower than minus the threshold or higher than plus the threshold.

Table 2.2: Correspondence between log odd ratios, probabilites and the resprective interpretation.

$p_o(m_c)$	$\lambda^c_{1:t}$	**Interpretation**
0	-threshold	free
0.5	0	unknown
1	+threshold	occupied

As mentioned the Bayesian Occupancy Filter has been widely used as an estimation method for environment perception. Some further research on BOFs for dynamic environments have lead to slight adaptation of the standard BOF. In [33], the authors point out the map overconfidence problem. This arises when a sensor observes the same part of the map for a long period of time and the state of this part of the map changes, for example a stationary car starts driving after some time. The standard BOF requires roughly the same number of measurements to change the state of that part of the map, posing a challenge for dynamic environments. The authors of [33] propose an exponential forgetting policy, and adapt Equation 2.14 to

$$\lambda^c_{1:t} = \gamma\lambda^c_t + (1-\gamma)\lambda^c_{1:t-1} - \lambda^c_0, . \tag{2.15}$$

The parameter γ can be used to reduce the impact of the past measurements. Increasing this factor leads to more importance of new sensor readings, and lesser influence of past information, on the resulting occupancy grid.

Chapter 3

A Particle Filter approach using velocity and density information

In this chapter, the developed particle filter algorithm will be explained in more detail. First, the general idea behind the object detection algorithm will be reviewed. Then, the particle filter approach will be discussed. Finally, the translation to an occupancy grid will be explained.

3.1 Particle Filter

Any particle filter works through the prediction and update principle. The filter creates a prediction of the distribution across the state space, and when measurements are available, the filter updates the distribution using the measurement data. The algorithm runs at a certain frequency, and at every time step the filter performs a new prediction to get the latest distribution. Measurements may come in at a different frequency than the algorithm frequency, so only when measurements are available an update step will take place.

The particle filter consists of three main steps, namely prediction, weighting and resampling. Before the algorithm starts to loop through these steps, first an initialisation will take place once. This initialisation creates a particle distribution which is required to start the particle filter loop on. The prediction step propagates the particles according to a specific process model. The weighting step assigns weights to the particles based on their correspondence to the measurement data. The resample step will create a new particle population from the prior population by copying particles from the prior population, where particles with a higher weight have a higher chance of being preserved. Once a prior particle distribution went through all three steps, the particle filter will output the posterior particle distribution, which will later be used to create an occupancy grid map. The particle filter steps from initialisation up to occupancy grid map are visualized in Figure 3.1.

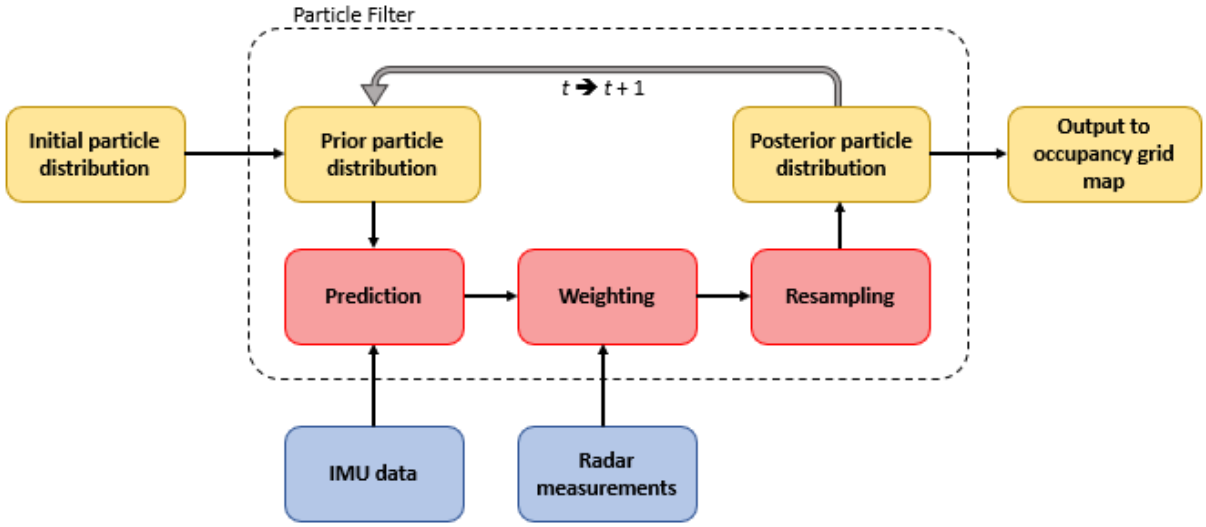

Figure 3.1: Overview of particle filter steps. Yellow represents particle distributions, red represents particle filter operations, blue represents sensor inputs.

Several input signals are used in the particle filter algorithm. In this research, a radar sensor is used to provide information about the surroundings of the ego-vehicle. This sensor will output measurement points with a certain two-dimensional location and a two-dimensional velocity, relative to the sensor.

Radar sensors can measure velocity using the Doppler effect, which means that the two-dimensional velocity is always in radial direction of the sensor. This will be dealt with in the weighting step. It is possible to add multiple radar sensors, for example to cover a larger area around the car. Other sensor types could also be integrated, although some steps might need some adjusting; for example if velocity information is missing from the measurement data, part of the weighting step will be different.

Apart from the single radar sensor, an IMU is used for ego vehicle motion compensation. Currently, lateral acceleration is taken from the IMU and used in the prediction step. Other signals, such as yaw rate could be integrated to perform an even more precise compensation, although it might be able to achieve the same performance without these signals but with additional noise in the prediction step. More on this will be explained at the specific steps, which are explained in more detail below.

3.1.1 Initialisation

In the initialisation phase, the initial particle distribution is generated. The total amount of particles generated at this step is fixed, and is called N_{total}. The exact amount depends on the performance requirements and the computational effort, and is evaluated in the next chapter. Each particle q has an index i, to keep track of individual particles.

$$q^i = \begin{bmatrix} x^i \\ y^i \\ v_x^i \\ v_y^i \end{bmatrix} \tag{3.1}$$

$$w_0^i = \frac{1}{N_{total}} \tag{3.2}$$

Each particles has four dynamic states, two for position (x and y) and two for velocity (v_x and v_y), as shown in Equation 3.1. Additionally, each particle has a weight parameter w. At this step in the filtering process, each particle is placed at a uniform randomly distributed position inside the two-dimensional map, and gets a random 2D-velocity drawn from a uniform distribution over the possible minimum and maximum velocity. These velocities are parameters based on the maximum achievable velocity of possible objects, and can be adjusted if the algorithm will be used in different circumstances. At the initialisation phase, all particles will receive equal weights, as shown in Equation 3.2.

3.1.2 Prediction

In the prediction step, each particle will be kept up-to-date by applying a process model to move the particles through the state space. A process model with constant velocity and constant orientation (CVCO) is used to propagate each particle from time step k to the next time step $k+1$.

$$q_{k+1}^i = \begin{bmatrix} 1 & 0 & \Delta T & 0 \\ 0 & 1 & 0 & \Delta T \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} q_k^i \tag{3.3}$$

This propagates each particle with a constant velocity, namely its two-dimensional velocity state, and a constant orientation, in the direction of the two-dimensional velocity vector. The update is therefore quite straightforward: each particle will keep its prior velocity, and the prior position is updated using the prior velocity and the difference in time ΔT between two time steps of the algorithm. The four dynamic states of the particle are updated according to this process model as shown in Equation 3.3. The resulting states for the next time steps have now received an updated position states and equal velocity states, compared to the previous time step.

This constant velocity and constant orientation (CVCO) model assumes a linear motion through the state space. This will not be likely to represent the motion of non-linear dynamic objects. On the contrary: the main challenge lays in predicting highly non-linear, dynamic objects. To this end, process noise is included.

Since the exact non-linear motion-model of possible targets is unknown, stochastic diffusion is created in this step by implementing noise into the constant velocity and constant orientatinon motion-model [22, 24].

$$u_{k+1}^{i} = \begin{bmatrix} u_{p,x} \\ u_{p,y} \\ u_{v,x} \\ u_{v,y} \end{bmatrix} \tag{3.4}$$

This noise u is added to all the dynamic states of each particle. The exact quantity of noise for both position and velocity is drawn individually for both dimensions from a Gaussian distribution with zero mean and a covariance taken from the sensor properties, as previously given in Table 2.1. By using these values the estimation algorithm aims to capture fast moving objects moving through the state space.

Also in this step, the motion of the ego-vehicle needs to be compensated. Since the algorithm runs in the vehicle fixed frame, there is no need to compensate for vehicle position or velocity, since both are already relative to the ego vehicle. However, acceleration of the ego vehicle is not yet taken into account. Although the impact of acceleration may be small as long as ΔT is small, compensation for longitudinal acceleration is still applied, namely because the longitudinal acceleration of the ego vehicle is already available through measurements. The measured lateral acceleration from the IMU $a_{x,k}$ is used to compensate the position of all particles, by using the time delta between two time steps. If the ego-vehicle has a positive acceleration in x-direction, the particle positions will need to be compensated by subtracting the ego-vehicle distance covered by this acceleration. With this compensation, the total update equation looks like

$$q_{k+1}^{i} = \begin{bmatrix} 1 & 0 & \Delta T & 0 \\ 0 & 1 & 0 & \Delta T \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} q_{k}^{i} + u_{k+1}^{i} - \begin{bmatrix} a_{x,k}\frac{(\Delta T)^2}{2} \\ 0 \\ 0 \\ 0 \end{bmatrix}. \tag{3.5}$$

Compensation for lateral acceleration is not applied, since the influence of lateral acceleration is even smaller in normal, non-slip scenarios. However, although it is negligible, there might be some small lateral acceleration present. Since there is some stochastic diffusion in the position of the particles already due to the added noise, as long as this diffusion is large enough the lateral acceleration will be captured.

3.1.3 Weighting

In the weighting step, the predicted particle population will be assigned new weight parameters for each particle. After this step, the resampling step will provide an updated population with mostly particles with high weights, but this will be further explained in the next section. Each particle has one single weight parameter, which is composed of multiple elements. Below the implemented weights are explained:

- w_p Weight factor based on *correspondence of position* between a particle and a measurement point.
- w_v Weight factor based on *correspondence of velocity vector* between a particle and a measurement point.
- w_d Weight based on *number of other particles in close proximity* of particle under review.

The first weight (w_p) will assign a larger value to particles which are close to measurement points. This is the most intuitive weight, and makes sure that particles which accurately predict the measurement position are preserved. The second weight (w_v) will look at the correspondence between the velocity vectors of particles and measurement points. Although the main interest is in accurately predicting the position of possible objects for representation in the occupancy grid, it is assumed that particles which also accurately represent the velocity vector of a measurement point have even higher correspondence to a measurement point and thus better represent a possible object. Moreover, particles with a velocity vector which highly resembles the velocity vector of a measurement point, are more likely to propagate to a new position with a high correspondence with a measurement point.

The purpose of the final weight (w_d) is to compensate for particle loss due to a missed measurement or occlusion of the measurement behind other objects. When the radar sensor misses a detection of an object, the particle population which was present at this objects location due to convergence after several steps in the algorithm, is likely to disappear. This is because very low weights are given to particles which do not correspond to measurement points. The idea behind this density weight is that groups of closely positioned together particles have converged to this location in previous time steps. Large groups of particles can only have survived several algorithm steps by having high weights and thus high confidence in representing measurement points. Adding a density weight to these groups makes them survive even if no other weight is assigned. However, the density weight alone will not be enough to keep the group alive for a longer period, due to other particles with larger correspondence to measurement points being assigned more weight. Therefore a dense cluster of particles disappears over time, even with the addition of this density weight. This behavior is desired, since if there are multiple time steps without new measurement information, the particle population which was present should disappear; it is not likely to represent an object. The density weight is a trade-off between quick response to changes in measurement information and keeping particle clusters alive in the case of missed detections.

Weight based on position

This weight will assign a larger value to particles which are close in position to the measurement points position. If, at a single time step, there is more than one measurement from the sensor, a weight is assigned for each correspondence of a measurement to the particle under review. This will be explained in further detail when combining the weights, later in this section. The positional weight is assigned according to the probability density function of a two-dimensional normal distribution, with a mean μ_{z_p} corresponding to the current measurement detected position and an variance Σ_p corresponding to the sensor variance on position measurements. This can be seen as the chance of drawing the position vector $q_p^i = [x^i \ y^i]^T$ from the two-dimensional normal distribution around the measured position $z_p = [x \ y]^T = \mu_{z_p}$.

$$w_p^i = \frac{1}{\sqrt{|\lambda_p \Sigma_p|(2\pi)^2}} \exp\left(-\frac{1}{2}(q_p^i - \mu_{z_p})(\lambda_p \Sigma_p)^{-1}(q_p^i - \mu_{z_p})^T\right) \tag{3.6}$$

To tune the filter, the covariance from the sensor is multiplied by a parameter λ_p. This will increase or decrease the chance of nearby particles being drawn from the normal distribution, and will be evaluated in Chapter 4.

Weight based on velocity

The second weight assigns more value to particles with a velocity vector that closely resembles the velocity vector of a measurement point. To this end, the 2D velocity vectors of both the particle and the measurement will be compared. While the velocity states of the particle fully describe the two-dimensional velocity vector of that particle, the velocity measurements resulting from the radar sensor lack complete two-dimensional information. Due to the Doppler effect, it is only possible to extract object velocity in radial direction compared to the sensor. Therefore, it is not possible to directly compare the velocity vector of a particle to the velocity vector of a measurement point. A translation to the radial direction is necessary. The radial velocity of a measurement point is denoted as z_v^ϕ, and to compare the particle velocity vector, a projection of the particle velocity onto the measurement point velocity will be done. This projection of the particle's velocity will be called q_v^ϕ. This is done by decomposing the particle velocity vector into two vectors, which are orthogonal to each other. One of these vectors, q_v^ϕ, is parallel to the radial velocity vector of the measurement points and will be used for the correspondence. An illustration is given in 3.2.

This weight will again use a normal distribution to find the correspondence. The weight will be based on the likelihood of drawing the particle velocity vector in radial direction from the normal distribution around the measurement point velocity μ_{z_v}, now with variance σ_v taken from the variance on radial velocity measurements of the radar sensor. Since this is now reduced to a one-dimensional problem, the equation will be

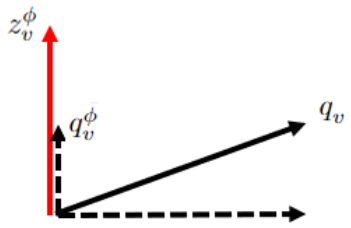

Figure 3.2: Projection of particle velocity onto measurement point velocity

$$w_v^i = \frac{1}{\sqrt{(\lambda_v \sigma_v)^2 \cdot 2\pi}} \exp\left(-\frac{1}{2}|q_v^\phi - z_v^\phi|^2 (\lambda_v \sigma_v)^{-2}\right). \tag{3.7}$$

In this case as well a parameter λ_v will be introduced, which will be multiplied with σ_v, to tune this weight in the evaluation chapter. Similarly to the position weight, there will be a separate velocity weight for each measurement point per particle. The combination of these weights will be shown below.

Density weight

The third weight is based on the density of clusters of particles. The reasoning behind this weight is that dense clusters are likely to correctly represent measurement points, and when a detection is missed or occluded at the current timestamp, still some weight is added to dense clusters from previous time steps. This weight is based on the amount of particles within a radius to the particle under review. To this end, for every particle, the distance to all other particles is calculated. The distance from particle p to an other particle in the population p_i is denoted as $s_{p,pi}$.

$$w_d^i = \lambda_d \left(\frac{\#\text{particles where } s_{p,pi} < S_d}{N_{total}}\right) \tag{3.8}$$

The parameter S_d is a radius in meters which will act as the cut-off; within this radius the amount of particles is counted and divided by the total number of particles in the population. λ_d is a parameter to tune the contribution of the density weight.

Combination of weights

The three different weighting factors are combined into a single weighting parameter per particle. Since the particle weighting for both position and velocity are measures of the likelihood of resemblance, inspiration is taken from probability theory to combine these weights. The combined probability of both event A and B can be written as the product of the probability of event A and the probability of event B if and only if the two events are statistically independent [34]. This can be written as Equation 3.9.

$$P\{AB\} = P\{A\}P\{B\} \tag{3.9}$$

This independence can be seen in the two weighting factors as well, since the probability of a particle's correspondence to a certain position of a measurement point does not influence the probability of the particle's velocity correspondence to that measurement points velocity, and vice versa. There is no relation between the probability of the velocity correspondence and position correspondence. In other words, whatever the probability of velocity correspondence, any positions correspondence can be taken independently of the velocity correspondence. Therefore we combine the first two weights using

$$w_{int}^i = w_p^i w_v^i \tag{3.10}$$

Here, w_{int}^i is an intermediate weight for particle i. Since the two individual weighting factors were both probabilities between 0 and 1, the multiplication, i.e. the intermediate weight, also has a value between 0 and 1.

This combination of weights might be easier to understand by an example. Assume a random particle has a large distance to a measurement point, but the same measurement point has a velocity vector which resembles the particle velocity vector quite accurately. The weight based on velocity will be high, whereas the weight on position is negligible. Therefore, by the above combination, this particle will receive negligible weight. Vice verse, a particle might have a very high weight based on position, but the velocity vector in no way matches the measurement point. Also in this case, the particle is not a good representation of the measurement point and will receive a low weight due to the velocity weight being combined with the positional weight. This will aid the prediction for the following particle distribution, since the update step will propagate particles depending on both the position and velocity of a particle.

In case of multiple measurements in one time step, both the position and velocity weights have multiple values per particle, since each weight is evaluated per measurement-particle pair. This needs be combined into a single intermediate weight per particle. Consider a scenario where there are five measurement points: every particle receives five positional weights and five velocity weights. First, following Equation 3.10, the weights are element-wise combined into five intermediate weights per particle. This will then need to be combined into a single intermediate weight. A simple summation of each intermediate weight is not successful; consider that, of the five measurement points, four measurements are originating from a single large object and are thus close together, while the fifth measurement has a different location and originates from a different, smaller object. If all intermediate weight per particle were summed, than the particles which resemble the larger object will have a considerable higher summed intermediate weight than the particles resembling the smaller object, since there are four measurement points with high resemblance to the particles close by the larger object. In contrast, a particle which has high resemblance to the smaller object, will have only one measurement point which it resembles, and therefore through summation of intermediate weights will have relatively low weight compared to particles closely resembling multiple measurements. To this end, the intermediate weights are first summed, but then divided by the number of intermediate weights which are nonzero. In this example, this will mean that a particle near the smaller object will have one nonzero intermediate weight, which will be its resulting intermediate weight. However, a particle which resemble the larger object will have four different nonzero intermediate weights, which will be summed and divided by four. In this way, particles resembling small objects will be kept alive and each particle will still end up with an intermediate weight between 0 and 1.

Now, only the third weighting factor for keeping particle clusters alive needs to be added. This factor cannot be multiplied with the intermediate weight, since this weight should have a relatively high contribution in cases where the intermediate weight is very small or even zero. Therefore, this factor is summed with the intermediate weight. Note that the value of this final combined weight will therefore no longer have a maximum value of 1, but instead of $1 + w_d^i$. The final weight parameter for particle i then looks like

$$w^i = w_{int}^i + w_d^i \tag{3.11}$$

3.1.4 Resampling

After the weighting of all particles, the population will need to be resampled. This step will randomly select particles, where particles with a high weight are more likely to be selected. The selected particle will be copied to the posterior population. Particles may be selected multiple times or not at all. Some randomness is included, as to avoid the problem of degeneracy of particles over time.

The resampling is done using the Resample Wheel method [35] (see also Figure 3.3). A random particle index i is chosen, and a random value β between 0 and twice the largest weight is generated. The index i is incremented if w_i is smaller than β, and also w_i is subtracted from β to make a new β. When the algorithm encounters a w_i which is larger than the current β, the particle with that index is duplicated to the next set. This is done N times, for the total amount of particles chosen at initialisation of the algorithm.

Now a new, resampled posterior particle distribution is created. If the algorithm has ran for a while, it is likely that dense clusters of particles will form around areas where consecutively measurement points

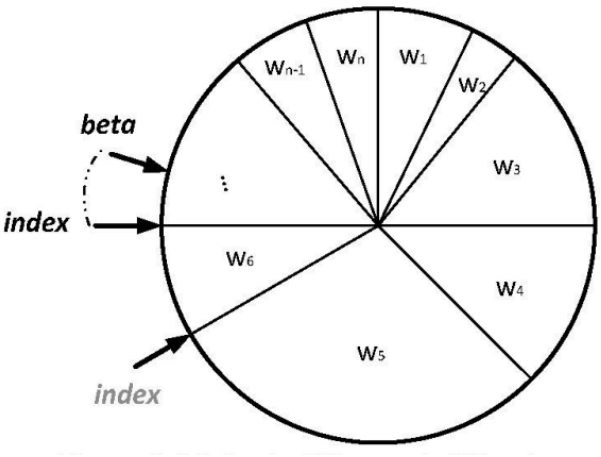

Figure 3.3: Resampling Wheel method

have appeared. If then a new measurement point at a completely different location appears, it might happen that no particles are close enough in the vicinity to assign weight to, therefore leading to no particles being able to resemble this new measurement point. To make sure new measurements at various locations can be captured, particles will need to be added over the entire state space. Therefore, after the Resample Wheel algorithm, a fraction of the population of the posterior distribution will be chosen to be replaced with newly initialised particles, which will therefore be uniformly distributed over the state space. This fraction should not be too high to replace too many high value particles, but should also not be too low to make sure enough particles are distributed over the state space to cover new measurements. The size of this fraction of re-distributed particles will also be evaluated later on.

3.2 From particle filter to occupancy grid

The final step of the algorithm is to create an occupancy grid out of the posterior particle distribution. The particle distribution is assumed to represent a state space distribution of likelihood of occupancy, i.e. regions with dense particle clusters are more likely to represent an occupied area. Occupancy grid cells also depict the likelihood of the area of that cell to be occupied. The gap to bridge is how to assign the particle distribution to grid cells, since the particle filter algorithm itself was independent of any grid representation.

To this end, for every grid cell it is evaluated how many particles are present in each cell. This will lead to every cell having an integer value in the range of $[0\ N_{total}]$. This will need then need to be reshaped into the occupancy grid representation. To do this, some inspiration is taken from probability theory. In the initialisation, the particles are spread uniformly over the map. Also at the resampling step, again a fraction of the particles is spread uniformly. Assuming a uniform distribution, the average number of particles for any cell on the map is

$$n_{average} = \frac{N_{total}}{c_{total}} \tag{3.12}$$

where $n_{average}$ is the average number of particles in a single cell, and c_{total} is the total number of grid cells on the map. If the particle filter algorithm works as expected, dense clusters of particles will form closely around measurements. The assumption here is that these clusters are considerably denser than the initial, uniform distribution of particles. Therefore, any grid cell with which contains a factor ψ more particles than the average amount, will be considered occupied. The remaining cells can only be classified as free space. It is not possible to make a distinction between free and unknown are using this method; grid cells with zero particles cannot be classified as unknown, as there always is the probability of not having any particles present at a certain time step, depending on the shape of the particle distribution.

Therefore, the resulting occupancy of a grid cell c will be binary, following

$$O(c) = \begin{cases} \text{occupied} & \text{if } n_c > \psi \cdot n_{average} \\ \text{free} & \text{otherwise} \end{cases} \tag{3.13}$$

where n_c is the counted number of particles in cell c. The value of parameter ψ will be discussed at the evaluation.

Chapter 4

Evaluation & results

The performance of the particle filter approach is evaluated in this chapter. This is done by comparing it to a different, well-known occupancy grid mapping approach: the Bayesian Occupancy Filter. This method has been explained in detail in Chapter 2. In this explanation, briefly also the addition of exponential smoothing has been shown in Equation 2.3, as proposed by [33]. Although this method should be better suited for tracking dynamic objects according to [33], it does have an extra parameter in the form of the exponential forgetting factor γ, which will need to be chosen. During the preparation of this evaluation, it was found that there is no optimal choice for γ, which leads to better results than using a standard Bayesian Occupancy Filter in the scenarios under review in this evaluation. Therefore, the standard Bayesian Occupancy Filter is chosen to compare the performance of the proposed particle filter approach against.

First, the scenarios which have been simulated are presented. Second, the different metrics for scoring the performance of the two filters are explained. Then, the results of the scenario simulation for the two algorithms is shown, and a detailed performance comparison is made. After this, results from changing the algorithm parameters are shown, in order to see the influence of different parameters.

4.1 Evaluation scenarios

To properly test the performance of the proposed particle filter algorithm, several different evaluation scenarios have been set up. For each scenario, both the particle filter and the Bayesian Occupancy Filter (BOF) will be simulated. The scenarios are set up to simulate different cases arising in dynamic object detection for automotive applications. These scenarios pose different challenges to the estimation algorithms, which include non-linear behavior of objects, occlusion and ego movement.

Below, the evaluation scenarios are shown in Table 4.1. A short description is given, along with the main challenge which motivates the selection of the given scenario. More details about each individual scenario are given in Section 4.3 along with the results.

Table 4.1: Evaluation scenarios

Scenario	**Challenges**
1 - Ego movement This scenario will have a single target moving through the state space The ego vehicle will be accelerating in the longitudinal direction, the target vehicle will have a constant velocity and a constant orientation.	Compensation for ego acceleration
2 - Cornering This scenario will have two target vehicles in front of the ego vehicle. The road has a small straight section, followed by a constant radius curve. One target moves faster than the ego vehicle, the other slower, all with constant forward velocity.	Multiple objects Non-linear motion Occlusion Compensation for ego cornering
3 - Occlusion This scenario will have two targets, where target 1 occludes target 2 at some point in time. After some more time, target 2 becomes visible again. The ego vehicle and target 1 will have identical constant velocity, target 2 will have a higher constant velocity while changing lanes.	Multiple objects Non-linear motion Occlusion

4.2 Evaluation metrics

Currently, three evaluation metrics are chosen and are calculated within the simulation environment. Below, a description of each metric is given, along with an explanation of the implementation and notes

about its importance to automotive environment mapping.

The first metric which will be used is the Map Error (ME). Similarly to the Map Score (MS) [36], the map error evaluates occupancy probabilities per cell over the entire grid. Both metrics provide the same insight on the performance of an occupancy grid, with the difference that a higher MS means a better performance, whereas a higher ME means a worse performance. Since the MS may include truncation errors, the ME is chosen as the preferred metric. The map error gives the absolute difference between probability values of two occupancy grids on a per cell basis. To be able to evaluate the ME between different sized grids, the ME will be divided by the total number of grid cells c_{total}. The ME will then be

$$ME = \frac{\sum_c |p_o(m_c) - p_o(m_{c,GT})|}{c_{total}}. \tag{4.1}$$

Notice that the comparison is done between the occupancy grid under review m and the ground truth occupancy grid m_{GT}. Also, only the probability for a cell to be occupied is used, since the probability for a cell to be free is just the compliment. The occupancy grids of both the particle filter approach as well as the Bayesian Occupancy Filter will be evaluated at every time step of the algorithm against the ground truth occupancy grid map. For more insight into the detailed workings of each approach, plots are made of the metric scores over time. However, for easy comparison of the different methods, a single score per simulation for each metric is created, which is equal to the mean of the metric over all the time steps in the simulation. Since the Bayesian Occupancy Filter starts off at the first time step with an initial grid with probability of 0.5 everywhere, the first time step is excluded from this mean score.

As mentioned in section 2.2.2, the immeasurable area behind vehicles has a probability of 0.5 on the ground truth map. All cells with this exact probability in the ground truth map will not be taken into account for the Map Error calculation, since this area is impossible to observe for the radar sensor. However, cells with a probability of 1 of being occupied on the ground truth map will always be included in the metric, even if they are behind other vehicles. This is to evaluate the performance of the estimation methods, when objects that were first visible to the sensor are occluded from sight.

In automotive applications, safety is of the utmost importance, since an error could lead to a fatal accident. This is no different for object detection, where a missed detection could have severe consequences, if for example path planning algorithms depend on the environment mapping created by the particle filter algorithm. To quantify missed detections, a confusion matrix is used. The matrix is depicted in Table 4.2. Especially important are the False Negatives (FN), since this means an occupied cell is classified as free.

Table 4.2: Confusion matrix for comparing predicted values to ground truth values

Confusion matrix		**Prediction**	
		Occupied	Free
Truth	Occupied	True Positive	False Negative
	Free	False Positive	True Negative

To use the confusion matrix, the occupancy grids are converted to binary occupancy grids. To this end, a cell with a probability lower than 0.5 is considered free, and a cell with a probability of 0.5 or higher is considered occupied. The two metrics under review are the False Negative Rate (FNR) and False Positive Rate (FPR). The FNR is the ratio between cells classified as false negatives and all ground truth occupied cells.

$$FNR = \frac{c_{FN}}{c_{TP} + c_{FN}} \tag{4.2}$$

The FPR is the ratio between cells classified as false positives and all ground truth free cells.

$$FPR = \frac{c_{FP}}{c_{FP} + c_{TN}} \tag{4.3}$$

The FNR is deemed very important, since the higher this ratio, the more cells are falsely identified as free, whereas they actually are occupied. The FPR will give an indication of the overestimation of the

generated occupancy map. This is not as safety critical as the FNR, but the higher this ratio, the more actual free space is falsely classified as occupied. This limits the perceived free space and can be an issue for automotive applications.

4.3 Results from scenario simulations

Firstly, the results of the particle filter algorithm versus a Bayesian Occupancy Filter approach will be shown for the three scenarios. The particle filter approach has the following parameters as shown in Table 4.3.

Table 4.3: Parameter values for particle filter algorithm

Parameter	Value	Description
N_{total}	5000	Total number of particles in algorithm
ξ	0.5	Fraction of the particle population which gets re-distributed at resampling step
ψ	10	Factor for translation to occupancy grid

These parameters describe some of the settings for the particle filter algorithm. Later on they will be changed, but they have negligible influence on the weighting factors of the particle filter approach. The three individual weighting factors have their own set of parameters, which also can be used to tune the filter. Moreover, these factors will alter be changed to see the influence of the new approach, evaluating the impact of the newly added weighting methods. The weighting factors below are the ones which have been developed during the initial testing of the particle filter algorithm. The weighting parameter values for the particle filter algorithm are provided in Table 4.4.

Table 4.4: Weighting parameter values for particle filter algorithm

Parameter	Value	Description
λ_p	1	Influence of positional weight
λ_v	1	Influence of velocity weight
λ_d	0.1	Influence of density weight
S_d	0.5	Cut-off radius in meters for density weight

For both methods, the simulated measurement information is equal, as shown in 2.3. The radar sensor properties for each scenario are equal to those previously presented in Table 2.1.

4.3.1 Scenario 1 - Ego movement

This scenario is created to test the basic concept of estimating a single vehicle moving in the same direction as the ego vehicle. Both vehicles drive on the same lane with the same longitudinal direction. The target vehicle will have a constant velocity of 20 meters per second. The ego vehicle will have an initial velocity of 10 meters per second, but will accelerate up to 15 meters per second and then decelerate back to 10 meters per second. This introduces longitudinal ego motion, which the particle filter algorithm should compensate for. This is done by using INS sensor data. The simulated time will be 6 seconds.

Snapshots of the scenario simulation of Scenario 1 can be seen in Figure 4.1. The blue square is the ego vehicle, the orange square is the target vehicle. The grey area shows the radar sensor coverage area, whereas the red circles show the location of radar measurements at that time step. As can be seen, the rear of the car generates multiple different radar measurements simultaneously. It can also be seen that sometimes faulty detections happen, when the radar sensor provides a measurement where in reality there is no object. In the two rightmost snapshots of Figure 4.1, a faulty detection can be seen as a red circle not located near the orange target vehicle.

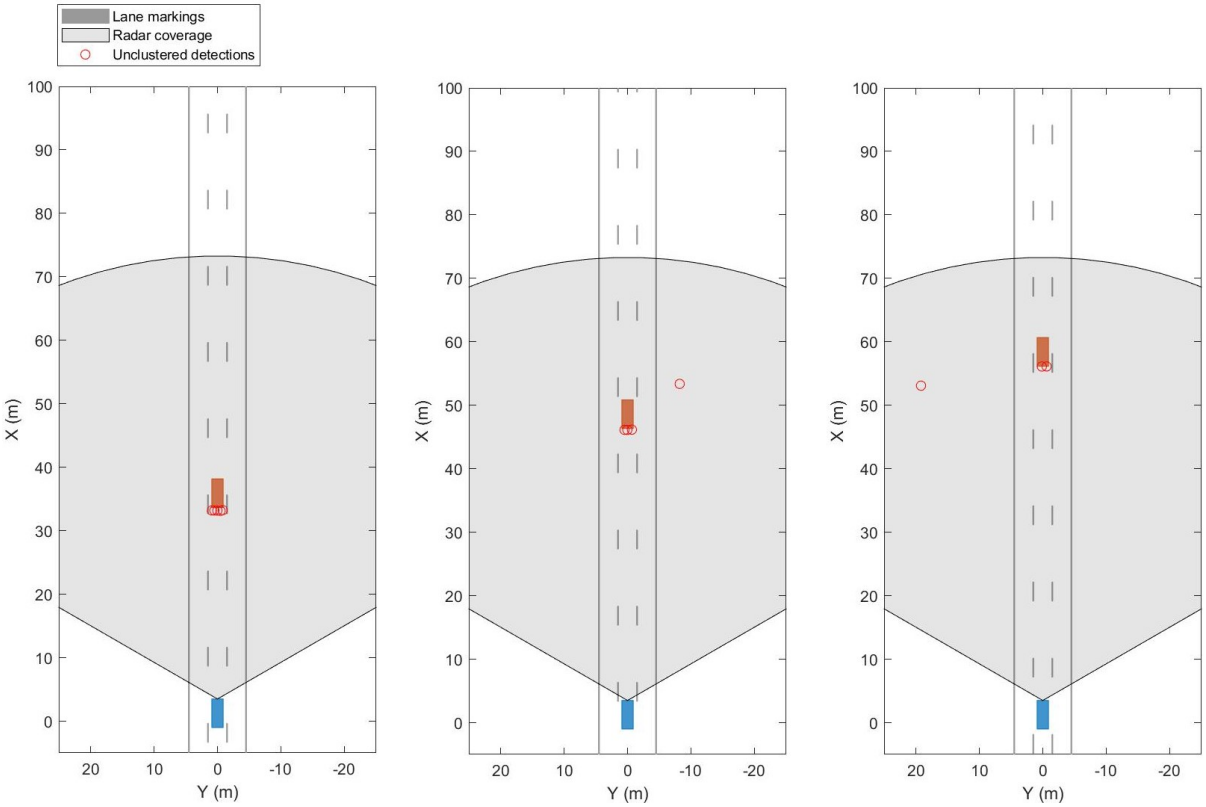

Figure 4.1: Screenshot of scenario simulation for scenario 1 at times t = 1.5 s, t = 3.0 s and t = 4.5 s.

The scores of the three different metrics for both the particle filter algorithm as well as the Bayesian Occupancy Filter are shown in Table 4.5. As can be seen, both the Mean Error (ME) and the False Positive Ratio (FPR) are small numbers. This is due to the large amount of cells present in the two-dimensional grid. The ME is divided by the total number of cells in the grid, whereas the FPR is divided by the total number of ground truth free cells. For the FNR, the score is larger, since this is divided by the total number of ground truth occupied cells, which is considerably lower in a scenario with a small area occupied by objects relative to the total area of the occupancy map.

As can be seen, the Bayesian Occupancy Filter performs better when looking at the Map Error and False Positive Rate. The Particle Filter in this scenario classifies more cells as occupied compared to the ground truth map, and relatively more so than the Bayesian Occupancy Filter, leading to the worse score. However, it scores better in the False Negative Rate metric. The Bayesian Occupancy Filter classifies relatively more cells as free, whereas in reality these cells are occupied.

Table 4.5: Average scores for Scenario 1 - Ego movement

Scenario 1	**ME**	**FPR**	**FNR**
Particle Filter	0.00238	0.00220	0.201
Bayesian Occupancy Filter	0.00133	0.000991	0.263

4.3.2 Scenario 2 - Cornering

The next scenario introduces non-linear motion in the form of cornering. The road starts with a straight section, followed by a corner with a constant radius of 50 meters. The ego vehicle starts on the straight section with a constant forward speed of 10 meters per second. 40 meters ahead of the ego vehicle, and a lane to the left, the first target vehicle drives with a constant forward speed of 15 meters per second. Further up the road, already in the corner and one lane to the right, drives a second target vehicle with a

constant speed of 5 meters per second. Although the ego vehicle is driving faster than the second target vehicle, it does not catch up to it. After 4.5 seconds or 45 meters, the ego vehicle reaches the corner itself, and up till the 6 second ending of the scenario follows the curve of the road. The first, fastest target vehicle overtakes the second target vehicle while cornering. It disappears from the radar sensor view. It does reappear again inside the radar view after overtaking the second target vehicle, but this is only briefly at the end of the scenario. Snapshots of the scenario can be seen in Figure 4.6.

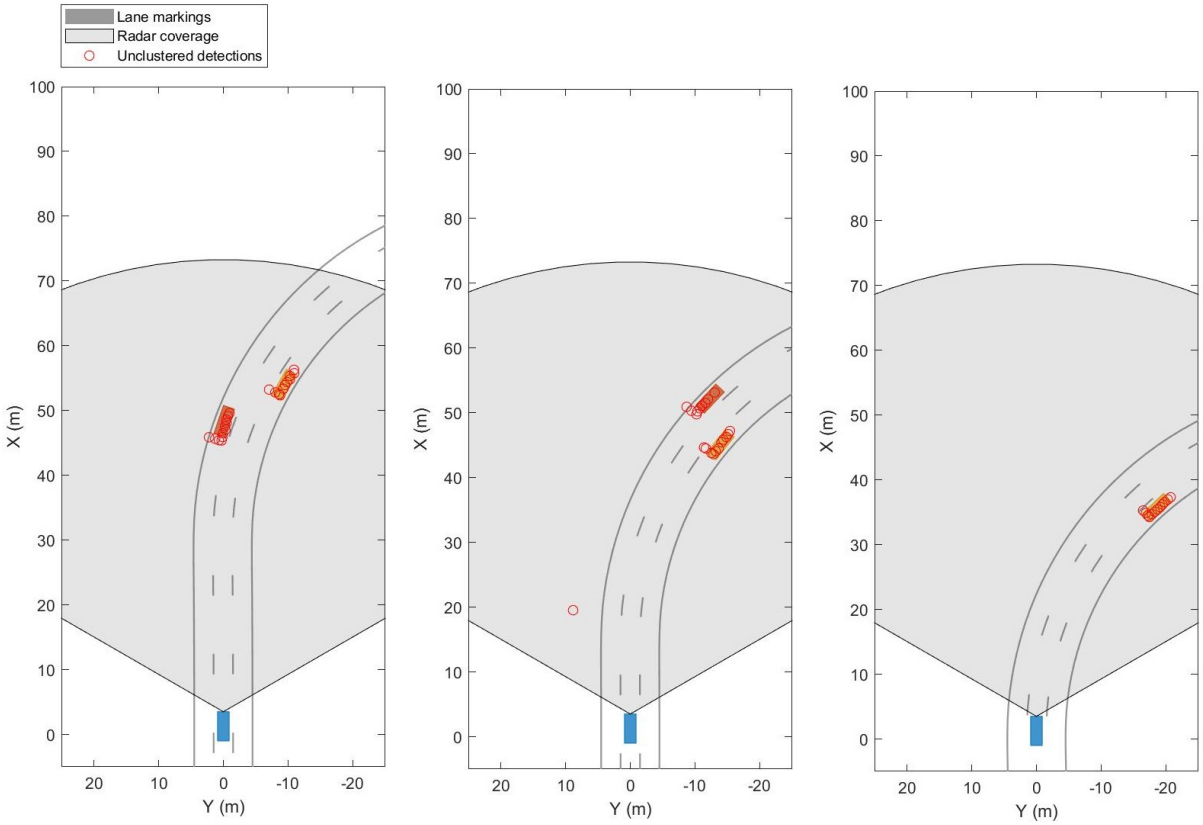

Figure 4.2: Screenshot of scenario simulation for scenario 2

The results are given in Table 4.6. The results show that the particle filter approach performs slightly less compared to the Bayesian Occupancy Filter in the Mean Error and the False Positive Rate metrics. For the False Negative Rate, the difference is larger, again in favor of the Bayesian Occupancy Filter.

Table 4.6: Average scores for Scenario 2 - Cornering

Scenario 2	**ME**	**FPR**	**FNR**
Particle Filter	0.00323	0.00196	0.384
Bayesian Occupancy Filter	0.00283	0.00185	0.241

Looking in more detail at the results, the scores of the three metrics are plotted over time for this scenario in Figure 4.3. Specifically looking at the False Negative Rate, since the difference is largest here, the biggest discrepancy between the algorithms can be seen between 1.5 and 3 seconds. To better zoom in on what is happening, a closer look at the particle population is needed.

Plots of the particle population are shown for two time frames in Figures 4.4 and 4.5. In this figure, the red dots depict the individual particles, with a larger dot showing a particle with relatively more weight than a smaller dot. The blue crosses are measurements from the radar sensor, and each measurement point has a blue arrow showing the direction and relative magnitude of the measured velocity. As can be seen in the first figure, the blue arrows of the central cluster of measurements points towards the north, since these measurements belong to the car slowly moving away from the ego vehicle. The other cluster

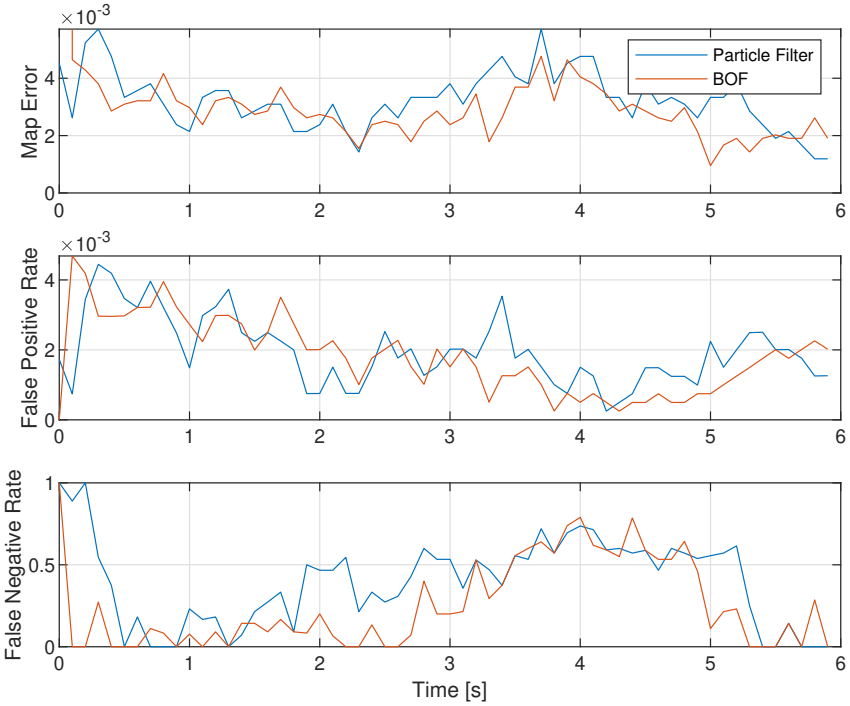

Figure 4.3: Evaluation metrics over time for scenario 2

of measurements has vectors pointing towards the south, since the ego vehicle is approaching this target and thus this target has a negative relative velocity, pointing towards the ego vehicle. In the first frame, the particle populations are centered around the measurement clusters. However, the particle population is not very dense around the faster car, located at x = 60 m and y = 0 m. In the next frame, half a second later, almost all particles have disappeared from the measurement cluster around the faster car. In this scenario, it appears that there are not enough particles to sustain two different clusters of particles. Over time, the cluster around the faster car slowly decreased in size. This might be due to the random redistribution of particles, in combination with insufficient weights for those particles. Every time step, half of the particle population is redistributed over the area under review. If a cluster is somewhat less dense, it could happen, due to the randomness in the redistribution, that particles with high weights get redistributed and given a new location. This leads to thinner clusters, which should be dealt with by large enough clusters; if there are many particles in a cluster, half of them can be relocated without creating a slow decrease over time. In this scenario, that was not the case. Adding more total particles might be a solution to improve performance, as well as a smaller fraction of redistributed particles. This will be investigated when varying the parameters later on.

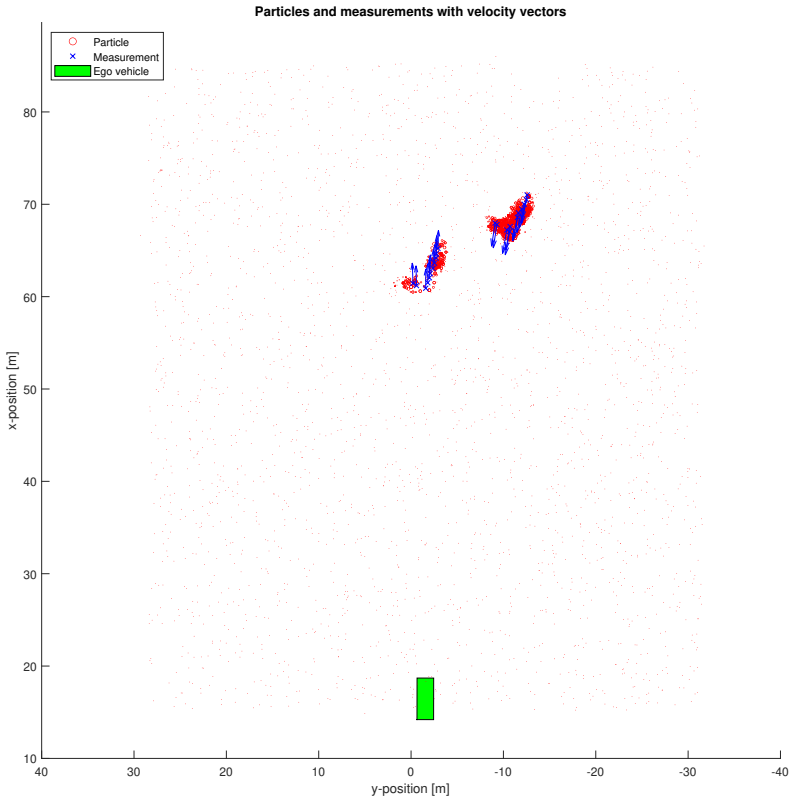

Figure 4.4: Particle filter distribution at time t = 1.5 s

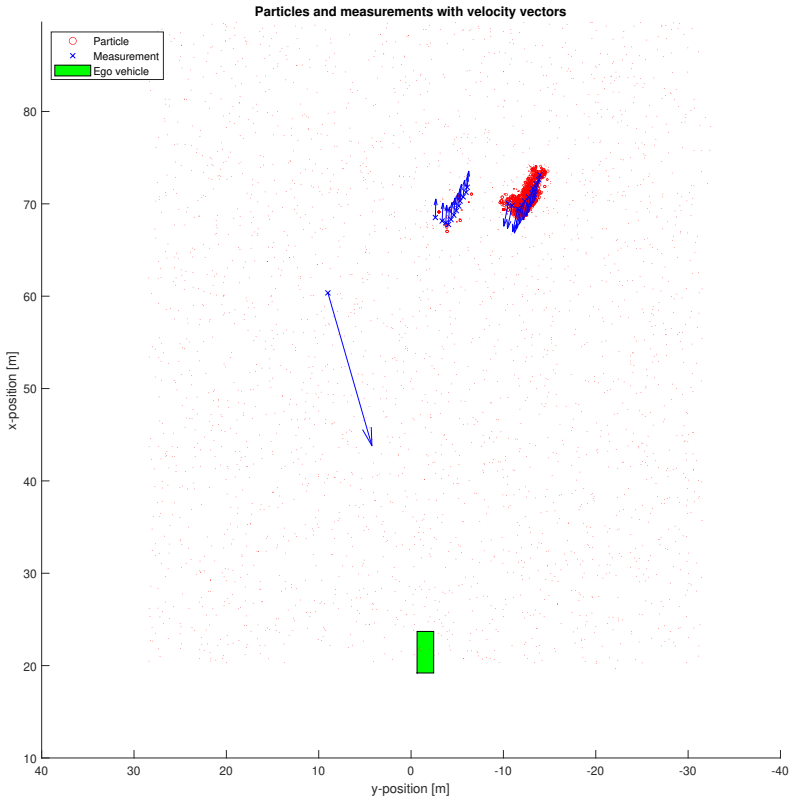

Figure 4.5: Particle filter distribution at time t = 2.0 s

4.3.3 Scenario 3 - Occlusion

This next scenario has two target vehicles and one blue ego vehicle. Both the blue ego vehicle and the orange target vehicle drive at the same constant speed of 10 meters per second. They both start in the same lane and have the same longitudinal direction, there is no lateral acceleration for these two vehicles. The yellow target vehicle drives at a speed of 25 meters per second, and maneuvers from the right hand lane all the way to the left hand lane. This gives the yellow vehicle some lateral acceleration, and moreover the vehicle passes in between the ego vehicle and the orange target vehicle. This leads to the radar sensor not detecting the orange vehicle for a small period of time. After the sideways motion of the yellow vehicle, the orange target vehicle does reappear before the radar sensor.

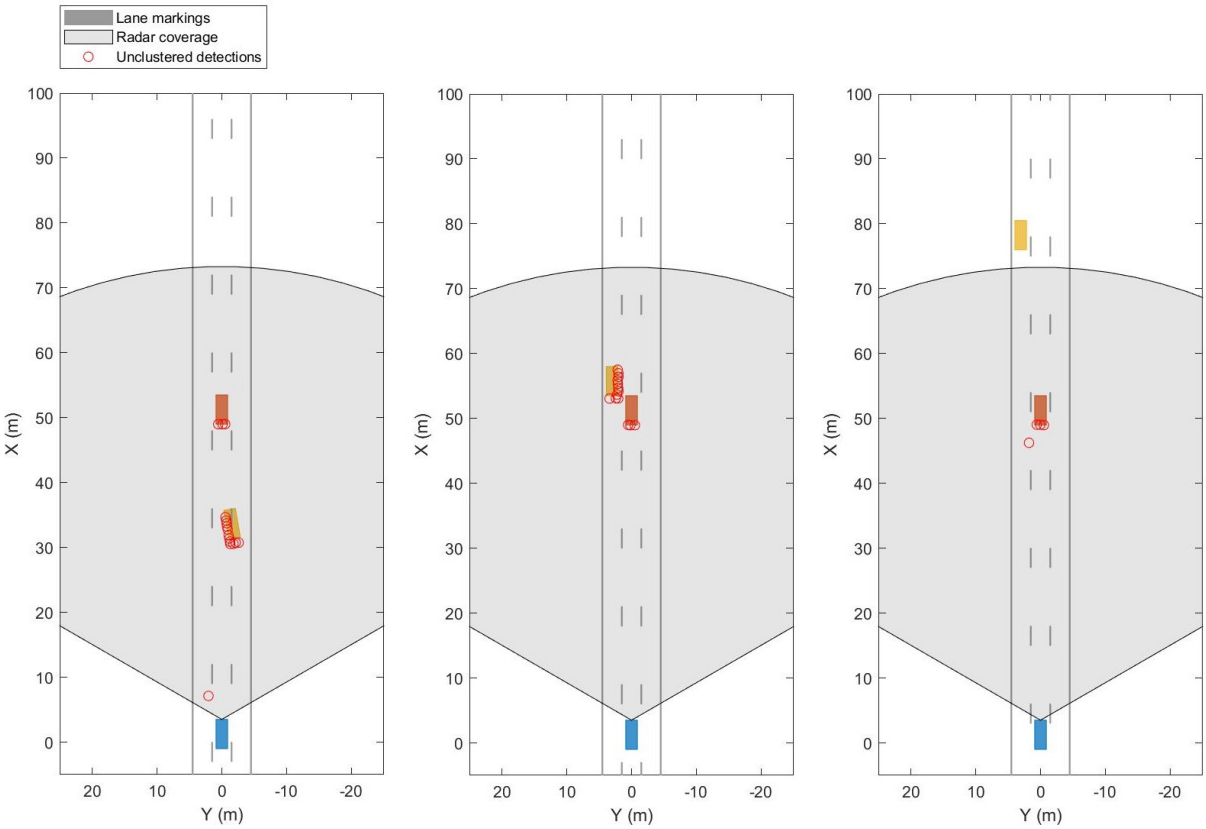

Figure 4.6: Screenshot of scenario simulation for scenario 3

Snapshots of the scenario simulation of scenario three can be seen in Figure 4.6. Interesting to note is that the yellow car causes way more radar measurement points than the orange car, even though they have the same dimensions. This is due to two things: firstly, the yellow car is closer, and secondly the frontal area of the yellow car observed from the ego vehicle is considerably larger. Since the yellow vehicle has a higher velocity, it disappears beyond the radar range after roughly 4 seconds. Again here, sometimes faulty detections happen as can be seen in the rightmost snapshot.

The results are shown in Table 4.7. The results for the Mean Error and False Positive Rate are very close, with a slight advantage for the Bayesian Occupancy Filter. However, in the case of the False Negative Rate, the Bayesian Occupancy Filter outscores the Particle Filter algorithm considerably.

Table 4.7: Average scores for Scenario 3 - Occlusion

Scenario 3	**ME**	**FPR**	**FNR**
Particle Filter	0.00267	0.00255	0.121
Bayesian Occupancy Filter	0.00246	0.00227	0.0757

To evaluate where this difference comes from, a plot over time of the three metrics is produced. This can

be seen in Figure 4.7. The main difference in the False Negative Rate metric is occurring at the first second of the simulation. To further understand what is happening, a snapshot off the three occupancy grids at two time steps (0.6 and 1.4 seconds) is taken.

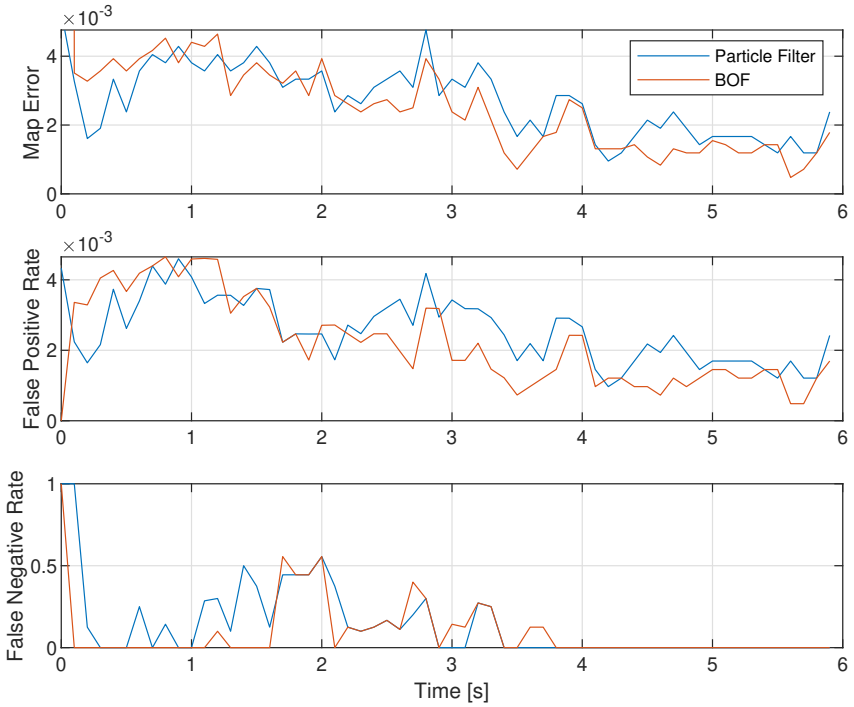

Figure 4.7: Evaluation metrics over time for scenario 3

In Figures 4.8 and 4.9, the three occupancy grids are shown for the two time steps. At the shown time steps, it is known that the Particle Filter algorithm scores lower on False Negative Ratio than the Bayesian Occupancy Filter approach. The figure shows that primarily at the location 50 meters in front of the ego vehicle, less grid cells are receive a high occupancy value in the particle filter approach, compared to the ground truth. Also the Bayesian Occupancy Filter has more grid cells with high occupancy at this location at this time, and in fact does not falsely identify any positive ground truth occupancy grid cell; the False Negative Ratio is 0 at this time for the Bayesian Occupancy Filter. The Particle Filter approach turns out to have too little particles at this location, to represent a large enough area as occupied. This can be attributed to the fact that the other vehicle in this scenario, the one closest to the ego vehicle, generates many more detections and thus measurements from the radar sensor at this time step. Therefore, particles around this area receive more weight from being associated with multiple measurement points, compared to the sparse amount of measurements for the vehicle further away. Later, it is investigated if this can be solved, for example by adding more particles in the population.

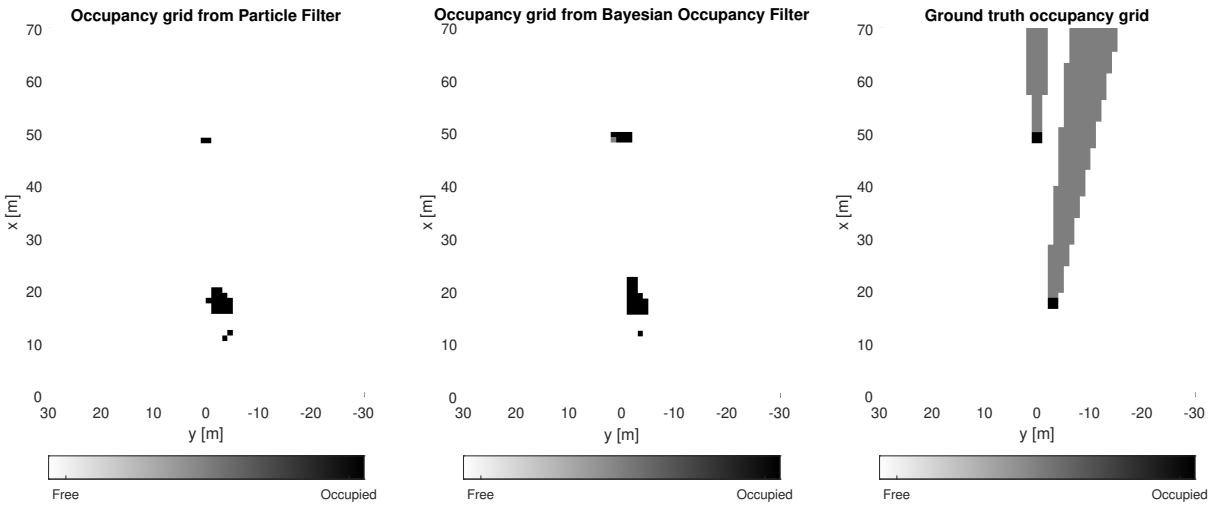

Figure 4.8: Occupancy grids at t = 0.6 s for scenario 3

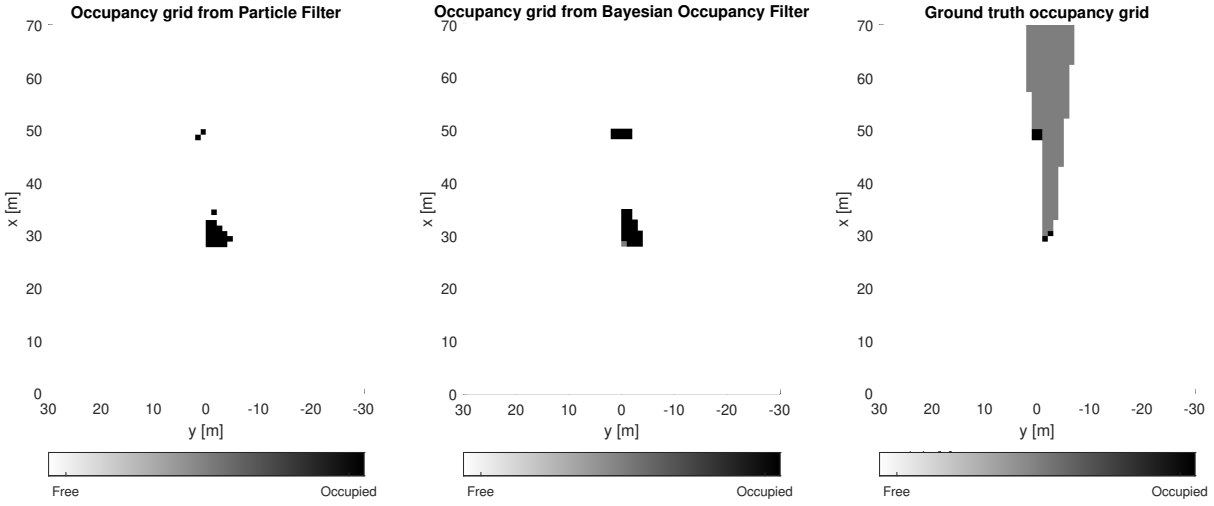

Figure 4.9: Occupancy grids at t = 1.4 s for scenario 3

4.4 Parameter variations

There are several parameters, which have been introduced in Chapter 3, which allow for changing the behavior of the particle filter algorithm. Especially interesting here are the parameters which are tied to the different weighting factors; these allow for an analysis of the workings of each weighting factor. Changing the magnitude or eliminating a factor from the filter altogether influences the final results. This section will describe the effect of different parameters on the algorithm results.

The weighting parameters will show the effect of the three weighting factors on the final result, and will be reviewed first one by one. Apart from these weighting factor parameters, there are other algorithm parameters which will effect the performance. After the review of the weighting parameters, a closer review of these algorithm parameters will be conducted.

4.4.1 Weighting factors

The weighting parameters have been introduced in Chapter 3 as λ_p, λ_v and λ_d. Next to these, the parameter S_d also influences the behavior of the density weight, and therefore will also be evaluated.

Positional weight parameter

Firstly, a check is done to see the results if the positional weight is excluded totally. This is done by removing the positional weighting factor from the calculation of the intermediate weight. Equation 3.10 then becomes

$$w_{int}^{i} = w_{v}^{i}. \tag{4.4}$$

The resulting particle filter algorithm does not provide usable results. No convergence of the particles is achieved in this case, since the weight for each particle is only composed of a velocity vector resemblance weight and a density weight. Since all particles are initialised with random velocity vectors, there are multiple particles distributed over the entire map with high correspondence to the measurement velocity vector. There is no convergence on location, but also the distribution of particles with higher weights remains randomly uniform over the map. This is because half of the particles get re-initialised after the resampling step. Also, since the velocity vectors of particles are only compared in the radial direction, as shown in Figure 3.2, a wide variety of velocity vectors receive weight and therefore particles with different velocity vector directions survive, preventing convergence in velocity vector direction.

The influence of the positional weight can be tuned using parameter λ_p, as can be seen in Equation 3.6. Using this parameter, it is possible to increase or decrease the variance of the normal distribution around a measurement position. Increasing the variance leads to particles with a larger distance from the measurement receiving a positional weight, and particles with a similar distance receiving a larger positional weight. The resulting metrics for scenario 1 for different values of λ_p are shown in Figure 4.10. For reference, the results from the Bayesian Occupancy Filter are also shown, but this is a single value since the Bayesian Occupancy Filter is not influenced by changing the particle filter algorithm parameters.

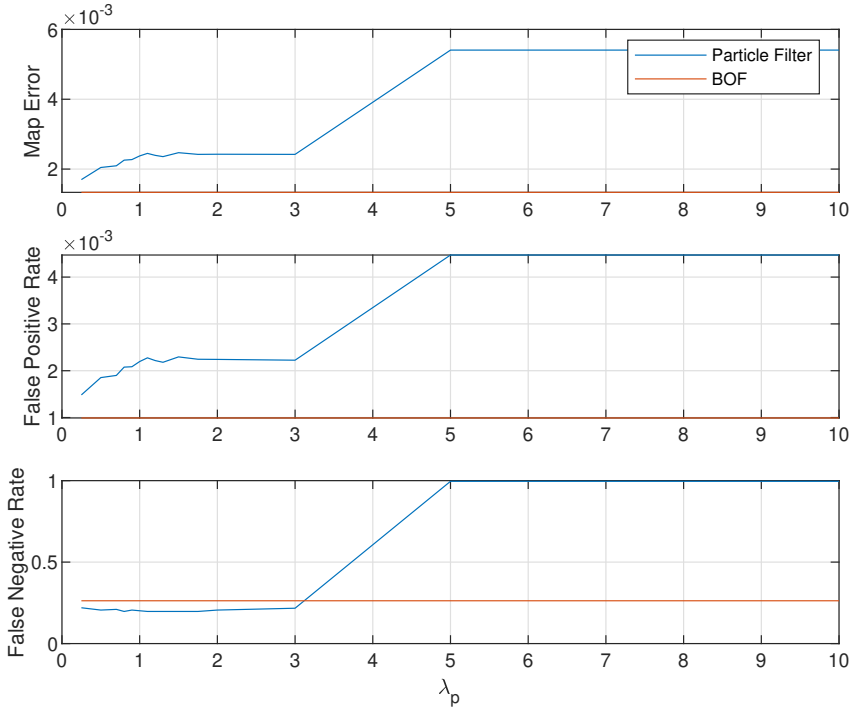

Figure 4.10: Scores for simulation of scenario 1 for different values of λ_p.

From Figure 4.10 it can be seen that both the Map Error and the False Positive Rate are lowest for a small λ_p, and increase a lot for a very high λ_p. For the False Negative Rate, there seems to be a small range with similar scores, and then increasing again for high λ_p values. It seems the particle filter algorithm does not respond well to a large λ_p, resulting in now convergence for high values above 5. This is because too many particles receive some weight from the positional weight, and these particles can be further away from measurement points. Eventually, this is so spread out that also particles which do not reflect

the measurements are kept alive. On the contrary, a smaller positional weight decreases the area around a measurement in which particles will receive some positional weight, therefore leading to a smaller, more concentrated distribution around a measurement point. This leads to less false positive grid cells because of the more concentrated particle distribution. False negative occur when ground truth occupied cells have been missed; a lower λ_p cannot prevent the missed detection of some cells, and a higher λ_p will at some point (roughly around 5) lead to no convergence at all and thus a false negative rate of 1.

Similar behavior can be seen for simulations of scenario 2 and 3 in Figure 4.11. The Mean Error and False Positive Rate are low for small values of λ_p, since the particle population is very concentrated around measurement due to the low area around a measurement where particles will receive a positional weight. However, the False Negative Rate now increases for very small values. This is due to the occlusion happening in this scenario. Because the area around a measurement for which particles receive some positional weight is very small, the algorithm has difficulty in capturing 'new' measurements, i.e. measurements where no particle was nearby at the previous time step. The chance for the randomly distributed particles to be close enough to the measurement is so small, that new detections are not captured properly.

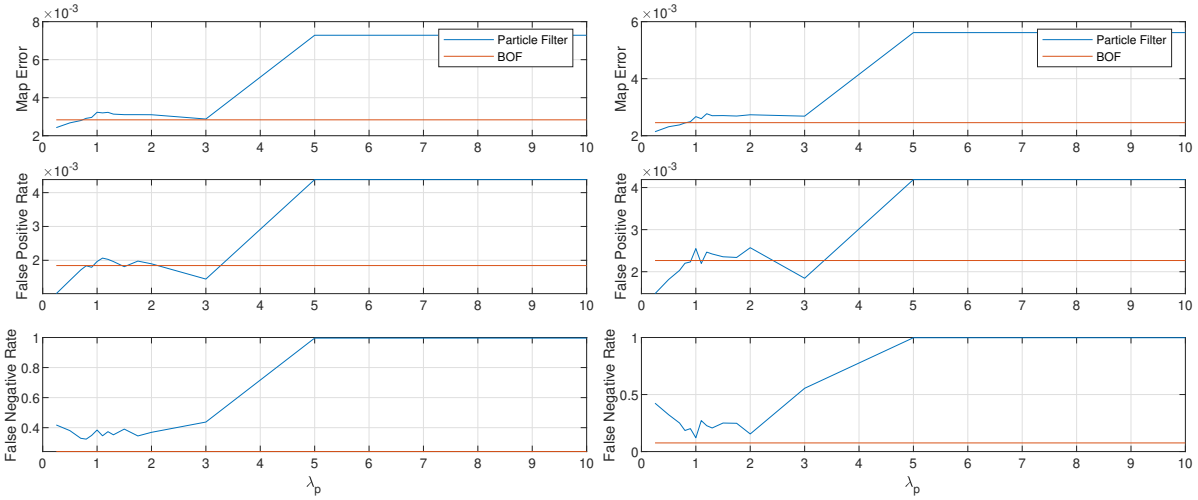

Figure 4.11: Scores for simulation of scenario 2 and 3 for different values of λ_p.

Velocity weight parameter

To test the influence of the velocity weighting factor, the parameter λ_v can be changed, as shown in Equation 3.7. Increasing λ_v leads to more particles receiving a velocity weight, since the variance increases. A particle with a higher discrepancy between its velocity vector and that of a measurement will now have a chance to be drawn from the distribution. Moreover, particles with similar discrepancy will receive more weight compared to a lower λ_v value.

First, a check is done to see if the algorithm still functions without the velocity vector weighting factor. To this end, Equation 3.10 is changed to

$$w_{int}^i = w_p^i. \tag{4.5}$$

It is expected that this will still work, since most of the estimation algorithms discussed earlier use only weighting based on position, and disregard velocity measurements. Indeed, looking at the results in Table 4.8, the algorithm still works without a weighting factor. In this table also the previously shown results for the standard set of parameters are shown, and the Bayesian Occupancy Filter results. With colors, the improvement or reduction in performance metric compared to the standard particle filter is shown.

Table 4.8: Average scores for all scenarios, including particle filter without velocity weight

Scenario 1	**ME**	**FPR**	**FNR**
Particle Filter	0.00238	0.00220	0.201
Particle Filter - no w_v	0.00262	0.00236	0.282
Bayesian Occupancy Filter	0.00133	0.000991	0.263
Scenario 2	**ME**	**FPR**	**FNR**
Particle Filter	0.00323	0.00196	0.384
Particle Filter - no w_v	0.00305	0.00220	0.252
Bayesian Occupancy Filter	0.00283	0.00185	0.241
Scenario 3	**ME**	**FPR**	**FNR**
Particle Filter	0.00267	0.00255	0.121
Particle Filter - no w_v	0.00279	0.00275	0.0808
Bayesian Occupancy Filter	0.00246	0.00227	0.0757

For scenario 1, leaving out the weighting factor from the particle filter leads to worse performance across all three metrics. However for the second scenario, there is a slight increase in Mean Error performance, and a significant increase in False Noise Ratio, getting closer to the Bayesian Occupancy Filter Score. For the third scenario, the False Negative Rate metric scores better if the velocity weight is not taken into account, while the other metrics are slightly worse.

In Figures 4.12, the results of the average metrics are shown for different values of λ_v. It can be seen that both the Map Error and the False Positive Rate increase as the parameter gets larger. When λ_v is smaller than 1, the results are not very clear for these metrics, maybe a small improvement can be seen. For the False Negative Rate, there is a slight improvement as the velocity weight parameter gets larger. As the parameter reduces in size, the performance of the False Negative Rate metric gets worse. Concluding, it seems that increasing the λ_v leads to particle clusters being more widespread, since more particles with some resemblance to the measurement velocity vector receive a velocity weight. This leads to more false positive grid cells and less false negative cells, since the area which is designated as occupied is larger due to the larger area on which the velocity weight now has effect due to the larger parameter. Also as shown in the table above, leaving out the weighting factor altogether leads to a lower score for scenario 1 for all metrics.

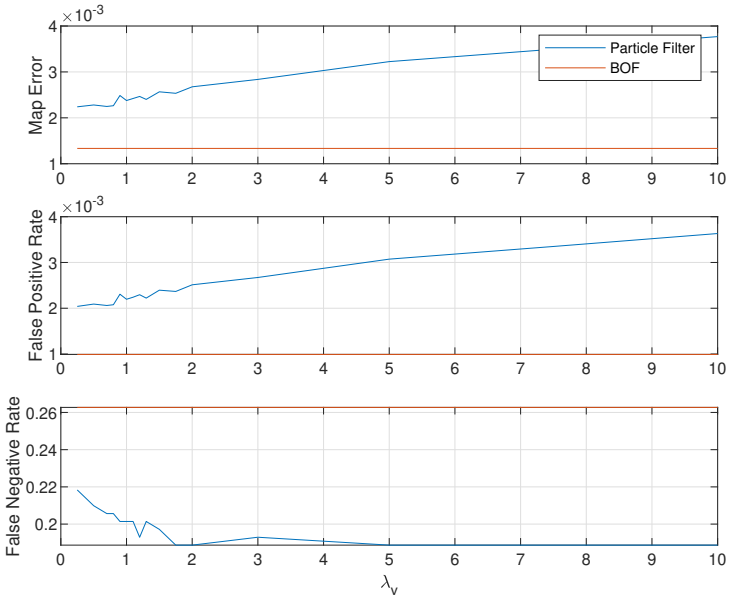

Figure 4.12: Scores for simulation of scenario 1 for different values of λ_v.

For scenario 2 and 3, the results are shown in Figure 4.13. The same tendencies as with scenario 1 can be seen; the larger the velocity weight parameter, the more particles with some resemblance will receive weight, leading to a wider cluster of particles and thus more cells being shown as occupied. This leads to more false positives and a less performing Map Error, but simultaneously to a lower False Negative Rate. However, leaving out the velocity weight altogether leads to slightly better results for scenario 2 and 3 when looking at the False Negative Rate. To analyze this, a plot is made of the metrics over time for a particle filter without velocity weight, which can be seen in Figure 4.14. Comparing this to Figure 4.7, the biggest difference is in the first two seconds of the False Negative Rate. As explained earlier, the particle filter cannot hold a large enough cluster of particles at one of the targets. It appears this problem disappears when disregarding velocity measurements in the estimation method. An explanation for this is that there are barely enough particles to keep two dense clusters, and with comparing both the position and velocity vector, particles have a combined lower weight than only comparing the positional vector, due to the combination of the two factors. Therefore, the cluster is slightly lost when using two weighting factors, and better kept alive when disregarding the weighting factor for velocity altogether.

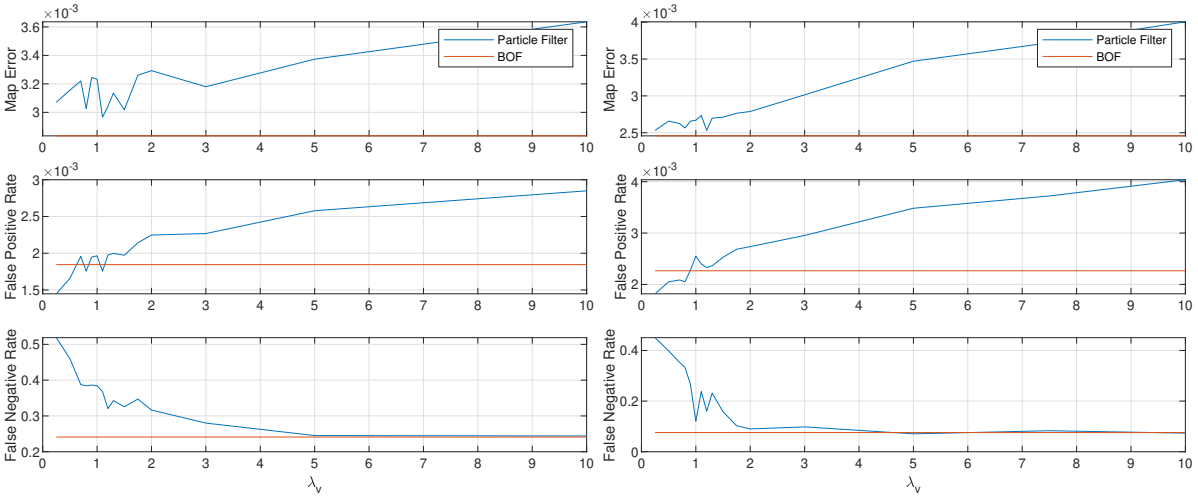

Figure 4.13: Scores for simulation of scenario 2 and 3 for different values of λ_v.

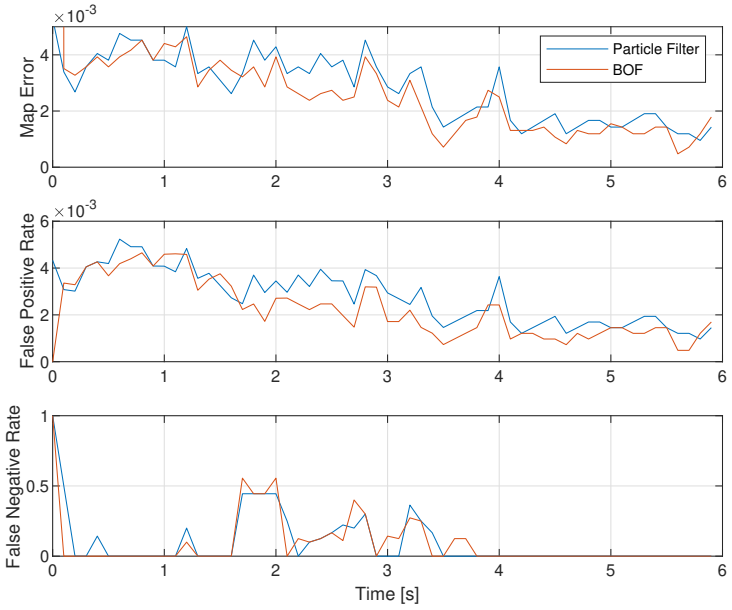

Figure 4.14: Results of error metrics over time for scenario 3 without velocity weight

Density weight parameters

The density weight is dependent on two different parameters, as shown in Equation 3.8. The number of particles within a radius S_d is counted, and divided by the total number of particles in the simulation. This number is then multiplied by a factor λ_d. λ_d increases or decreases the relative importance of the density weight, compared to the intermediate weight factor from Equation 3.10. The parameter S_d increases or decreases the radius in which particles are counted. Increasing this leads to more area around the particle being considered, potentially counting more particles.

First, the density weight is completely discarded by setting λ_d to zero. The results for this are shown in Table 4.9. Almost all metrics improve at all scenarios, except for the False Negative Rate in scenario 2 and 3.

Table 4.9: Average scores for all scenarios, including particle filter without density weight

Scenario 1	**ME**	**FPR**	**FNR**
Particle Filter	0.00238	0.00220	0.201
Particle Filter - no w_d	0.00183	0.00164	0.193
Bayesian Occupancy Filter	0.00133	0.000991	0.263
Scenario 2	**ME**	**FPR**	**FNR**
Particle Filter	0.00323	0.00196	0.384
Particle Filter - no w_d	0.00278	0.00142	0.418
Bayesian Occupancy Filter	0.00283	0.00185	0.241
Scenario 3	**ME**	**FPR**	**FNR**
Particle Filter	0.00267	0.00255	0.121
Particle Filter - no w_d	0.00205	0.00171	0.228
Bayesian Occupancy Filter	0.00246	0.00227	0.0757

Figures 4.17 shows the results of the average metrics when changing the value of λ_d for scenario 1. The False Negative Rate shows almost no change resulting from changes in this parameter. The Map Error and False Positive Rate slightly increase when λ_d increases.

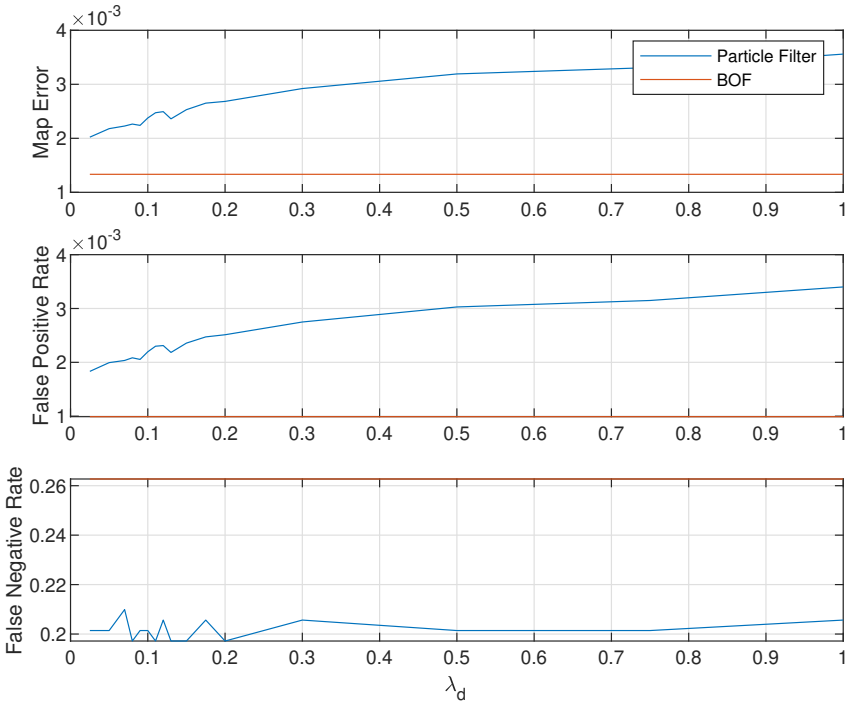

Figure 4.15: Scores for simulation of scenario 1 for different values of λ_d.

The results for scenario 2 and 3 are shown in Figure 4.16. For both the Map Error and the False Positive Rate, the tendencies are similar to the results of scenario 1. For the False Negative Rate, the results are not very consistent at low values of λ_d, although performance here seems to decrease. The swingin in results here is partly because of the small number of ground truth positive cells, meaning that a few cells wrongly identified lead to a very different ratio. If λ_d increases to larger values than 0.2, also the performance of the False Negative Rate metric decreases. Here the contribution of the density weight is too high, leading to too many particles receiving a high weight through density, making the contribution of the position and velocity weight relatively lower.

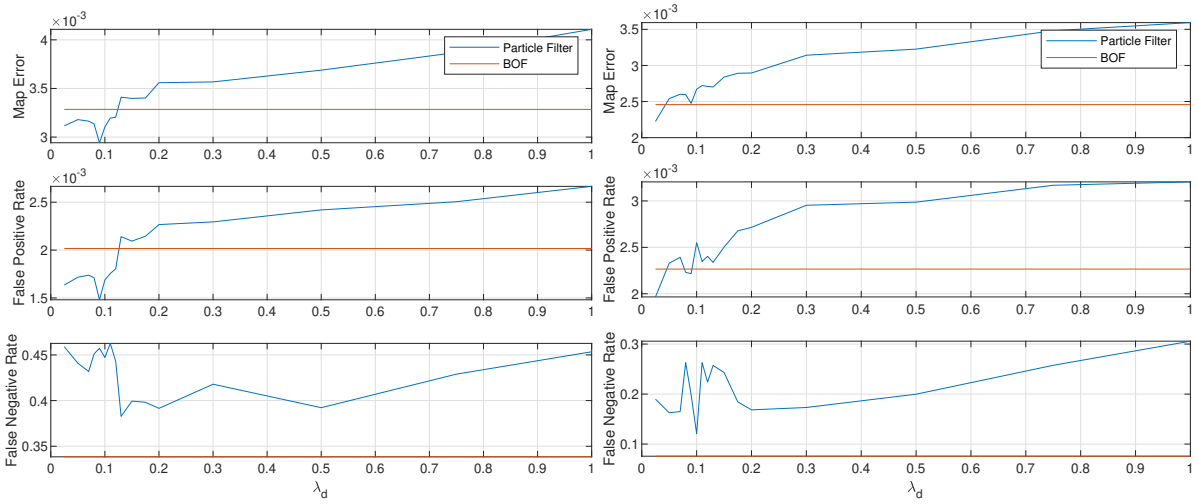

Figure 4.16: Scores for simulation of scenario 2 and 3 for different values of λ_d.

In figure 4.17, the results are shown for a changing S_d parameter. This increases the radius where particles are counted. Again, the effect on the False Negative Rate is negligible. For the Map Error and the False Positive Rate, the performance worsens as the radius gets bigger. This is because the clusters get wider, since also particles which are further away from clusters will receive weight due to the increased radius. Therefore, more area will be designated as occupied. Since the False Negative Rate was already very low, with better performance than the Bayesian Occupancy Filter, there is no improvement here. A larger cluster of particles is not enough to improve the performance. A slightly increased λ_v, as shown in Figure 4.12, seems the only way to improve the False Negative Ratio for this scenario.

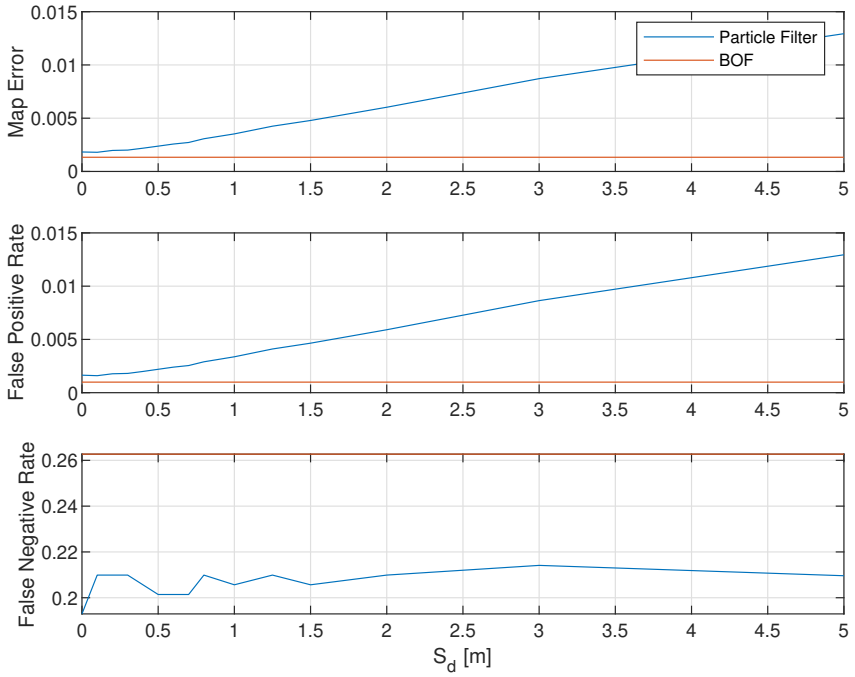

Figure 4.17: Scores for simulation of scenario 1 for different values of S_d.

The results from scenario 2 and 3 are shown in Figure 4.18 for a changing radius S_d. Similarly to scenario 1, the Map Error and False Positive Rate increase with an increased radius, for the same reason as explained at scenario 1. The False Negative Rate is impacted by a changing radius, but there is not a clear trend visible.

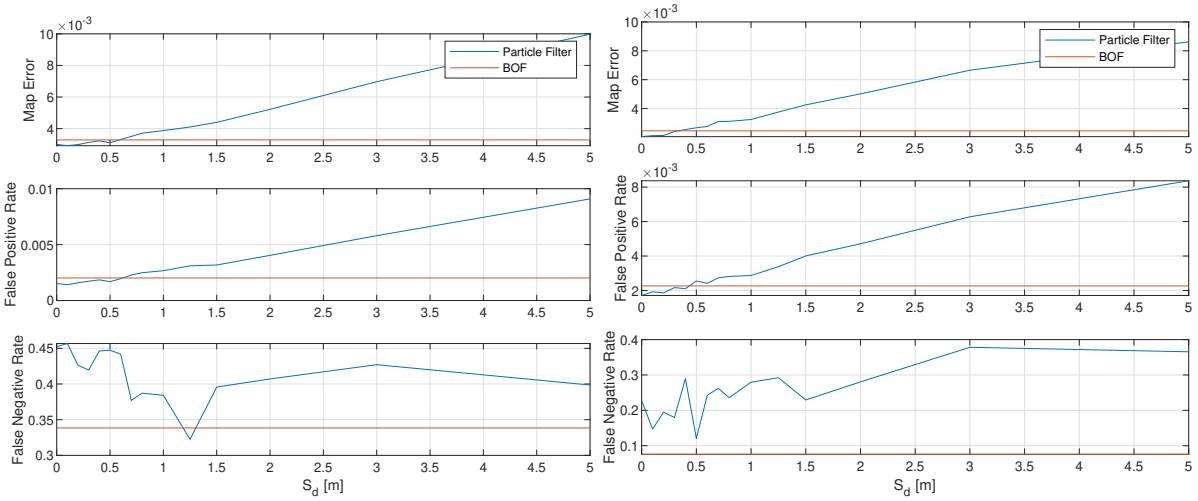

Figure 4.18: Scores for simulation of scenario 2 and 3 for different values of S_d.

4.4.2 Algorithm parameters

In this section, three parameters which influence the algorithm performance other than the weighting factors are discussed. First, a look is taken at the total number of particles in the simulation, then at the fraction of particles which get redistributed at the resampling step, and finally at the cut-off ratio which is used for translating from a particle distribution to an occupancy grid. The analysis is done on scenario 3 only; the effect of these factors will is not heavily dependent on the scenario such as the weighting

parameters, and scenario 3 has the most identified challenges, which are only partly present in the other two scenarios.

The number of total particles can influence the performance of the three metrics, but it also has a big effect on the time it takes to run the simulation. The calculations for the weights are done on a per particle basis, comparing every particle one by one to the available measurements. Therefore, this is also a trade-off for simulation time. In Figure 4.19, the results can be seen for the three metrics when changing the total number of particles. As can be expected, a very low number of particles leads to worse metrics across the board; there are just not enough particles to capture measurements, and at some point no particle clusters can be formed at all. Increasing the number of particles past the standard 5000 leads to a slightly worse Mean Error and False Positive Rate, but a better performing False Negative Rate. This is due to the fact that the particle clusters will have a higher number of particles. This in itself does not lead to more area being designated as occupied, since from Equation 3.13 it follows that for a cell to be designated as occupied, it should be ψ times higher than the average number of cells. The reason for the particle clusters consisting of more particles nonetheless, which is the reason of the increasing False Positive Rate and decreasing False Negative Rate, is because there is a larger variety of particles, with all slightly different states. Since the population is larger, these particles are spread slightly wider because of the variety in the cluster.

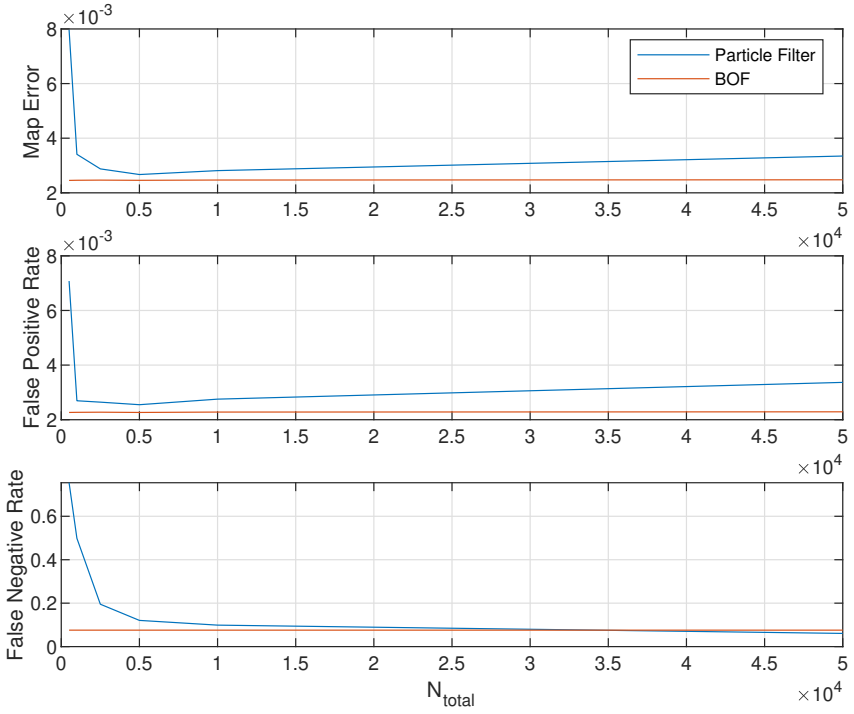

Figure 4.19: Scores for simulation of scenario 2 for different total number of particles.

In Figure 4.20, the results can be seen for a changing faction of the particle population to redistributed. This redistribution needs to take place, to populate the entire area of the map with some particles to capture new measurements. However, this leads to part of the particles from dense cluster to be spread over the map. A clear trend is hard to see in these figures. It appears that a very high fraction leads to a lower Map Error and False Positive Rate; this can be explained since in this case a very large portion of the particles is uniformly distributed over the map, leading to clusters of particles with only a small portion of the total number of particles. These clusters are therefore very narrow, leading to almost no false positives. This does lead to more false negatives, and an increase in False Negative Rate can be seen. For very low fraction, the False Negative Rate increases, with almost no change to the other metrics. This is due to the fact that new detections, for example one of the target vehicles reappearing after being occluded from view, are not well captured, leading to an increase in False Negative Rate, without affecting the other parameters much.

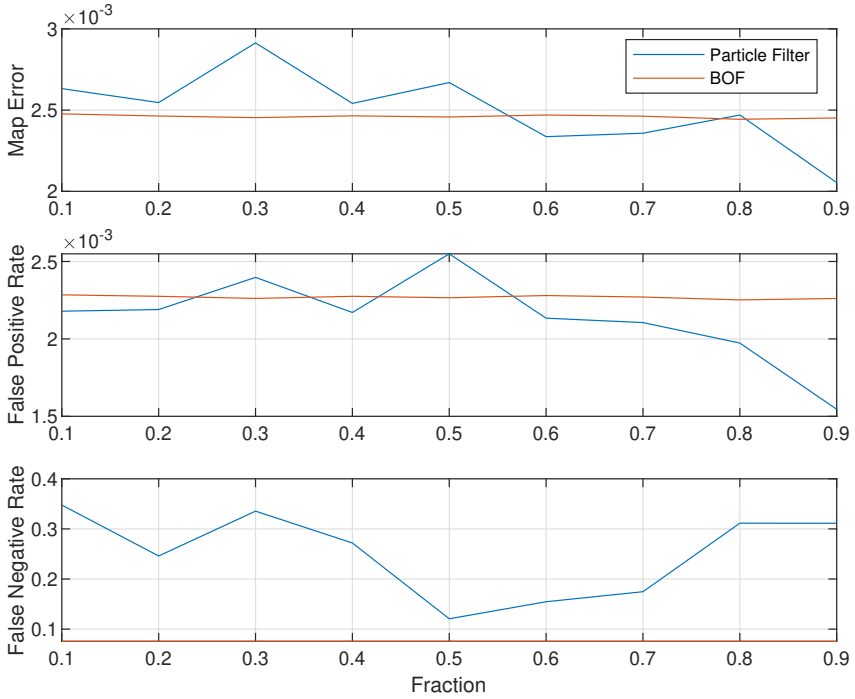

Figure 4.20: Scores for simulation of scenario 2 for different fractions of repopulated particles.

In Figure 4.21, the results can be seen for a change in ψ, also known as the cut-off ratio, as shown in Equation 3.13. This ratio determines the translation from particle distribution to occupancy grid; every cell has a certain amount of particles in its area, and when this amount is ψ times higher than the average that can be expected with a uniform distribution, the cell is designated as occupied. It can be seen that with a very low cut-off ratio, the False Positive Rate and Map Error increase significantly. More cells will be designated as occupied because of the lower threshold, up to a point where even cells with no correspondence to measurement points have enough particles to be designated occupied. This is due to the redistributing of cells. A lower ψ therefore leads to more false positives. When increasing the cut-off ratio, the Map Error and False Positive Rate decrease slightly, since less false errors are made. At the same time, the False Negative Rate increases for a higher cut-off ratio. Less and less cells will be eligible to be designated as occupied, and only the cells with a very large particle count, which are the cells at the heart of particle clusters, will be shown as occupied. This leads to more detections being missed at the edges of particle clusters, therefore an increased cut-off ratio leads to a worse False Negative Rate.

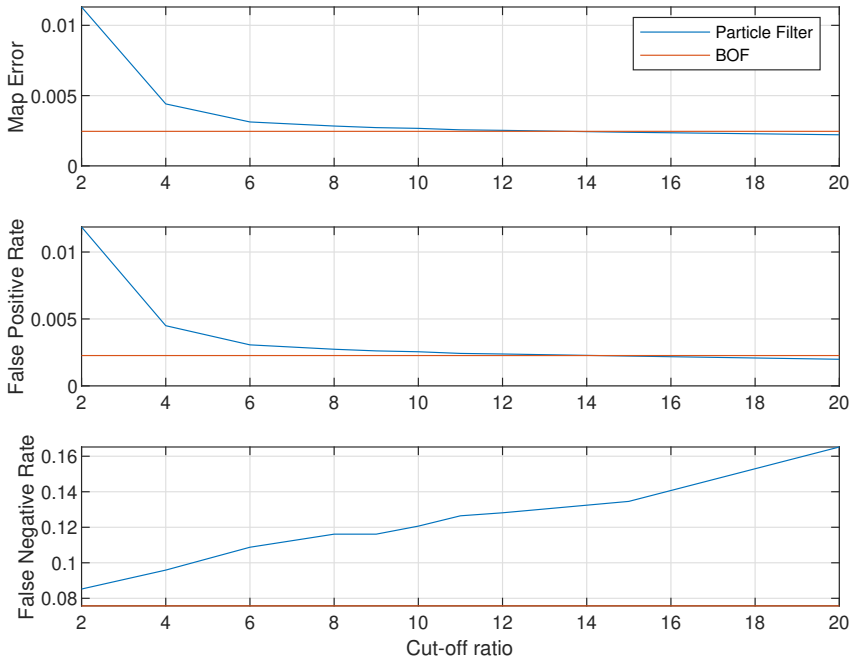

Figure 4.21: Scores for simulation of scenario 2 for different cut-off ratios.

Chapter 5

Conclusion and recommendations

This report has illustrated the current state of automotive occupancy grid mapping within dynamic environments. With inspiration taken from relevant literature, a particle filter approach is developed, using not only positional weight, but with the addition of velocity measurements for estimating dynamic object locations. Also, a density weight is proposed, as a way to deal with sporadic measurement loss and occlusion. Next to the particle filter method, a way to translate from the particle population towards an occupancy grid map is made. The Bayesian Occupancy Filter, a widely used approach in environment mapping, is also implemented, for evaluation purposes. To evaluate the results and enable a comparison between the new particle filter approach and the Bayesian Occupancy Filter method, a way to designate unknown area on a ground truth occupancy map is implemented and three different metrics are proposed. Multiple different parameter variations have lead to more insight into the workings of the particle filter.

From the results of the three metrics in the evaluation, the particle filter algorithm has not proven to be consistently better performing over the standard Bayesian Occupancy Filter. The differences are small but not negligible, however not one of the methods is always better performing than the other. The Bayesian Occupancy Filter performs in almost all metrics and scenario better than the standard particle filter, except for a straight driving scenario with ego vehicle acceleration, where the particle filter approach has less false negatives.

When looking at the introduced weighting factors, the velocity weight contributes to a slightly better performing particle filter approach, and changing the velocity weight parameter can increase or decrease specific performance metrics. The introduced density weight seems to have less of a positive impact, even leading to better results when it is removed from the particle filter. Its contribution to keeping occluded objects visible in the occupancy grid is negligible, not creating the desired effect.

There might be specific use cases where the particle filter approach turns out to be better performing compared to the Bayesian Occupancy Filter. There are also certain metrics where either the particle filter or the Bayesian Occupancy Filter performs better, depending on the chosen scenario. For example, if a low amount of cells which are falsely identified as free is desired, a low False Negative Ratio should be given priority over the other metrics. In the first scenario, where there is only longitudinal motion, the False Negative Ratio is better for the particle filter. And for the other two scenarios, some tuning of the velocity weight might also lead to less false negatives than the Bayesian Occupancy Filter.

However, this means choosing an estimation method, and the accompanying parameters, is based on which very specific use case is under review. Moreover, it may require tuning the parameters of the particle filter approach to suit only that specific use case. Since there are multiple parameters in the particle filter method, tuning is not straightforward, and a specific combination of parameters might prove hard to find. This is considered a disadvantage of the particle filter method, especially compared to the simplicity of the BOF. However, if a method needs to be chosen for a very specific scenario with well defined behavior, this effort can be taken to try and create a particle filter with higher performance than the standard Bayesian Occupancy Filter.

Although the particle filter approach has not shown to be a reliable alternative to the Bayesian Occupancy Filter in its current form, the additional weighting factor for velocity vector resemblance which has been shown does improve the classical particle filter approach, which is only based on positional information. An additional form of measurement information therefore does lead to a better estimation of occupancy. Moreover, the particle filter can be further improved when looking at a specific use case. To this end, recommendations will be given in the next section.

5.1 Recommendations

There are a few aspects of the current particle filter approach which prevent it from being effective in the presented scenarios, or for that matter to be implemented in a real world scenario. Suggestions to improve the particle filter are presented, as well as different areas of research for which the developed approach could be interesting.

5.1.1 Improvements for particle filter

A downside of the addition of multiple weighting factors is the introduction of accompanying parameters to tweak the influence of each weight, depending on the required behavior in a specific scenario. This should be more generalized, to be applicable to a wide variety of scenarios without the need to change parameter values. An interesting option might be to choose the parameters depending on the filter settings. For example, the filter will be more precise with more particles on the same area, since denser particle clusters can be created and the chance of capturing new measurements is higher. Instead of choosing the number of particles depending on the scenario, it is more convenient to create a relation between the area on which the particle is working and the number of particles. Then, the parameter can be reduced to an amount of particles per square meter for example.

Another coupling could be made between the fraction of particles which is redistributed, and the cut-off ratio for determining the occupancy values for the grid map. A higher fraction of redistributed particles causes the clusters of particles around measurements points to get more reduced; part of the dense clusters gets selected to be initialised again with a random position on the map. If the fraction of redistributed particles is very low, the clusters stay very dense, meaning the threshold for determining an occupied cell could be increased. Vice versa, redistributing many particles leads less dense clusters, which may cause the need for a lower cut-off ratio. Special care needs to be taken here, since a very high redistribution fraction means that the particle population over the entire map gets more evened out, up till a point where clusters will not survive.

For further tuning of the parameters, different combinations of the multiple parameters should be considered. In this study, the proposed parameters have been varied one by one, which allows for a detailed insight into the workings of each parameter. However, a better combination of parameters might be found by tuning multiple parameters at once. To this end, programmatic parameter tuning is highly advised. In order to speed up this process, it will be beneficial to reduce the simulation time. Also if the particle filter approach were to be implemented in a real world scenario, the simulation time will need to be reduced (or the number of particles, but that would lead to a lower performance score). Parallel execution of code might greatly reduce the simulation time. Parallel execution of the algorithm, for instance on a multi-core CPU or even a GPU, will cause repetitive sections of code to be executed simultaneously. For example the code to determine the positional and velocity vector weight factors are repetitive calculations between each measurement and particle. MATLAB offers solutions for both programmatic tuning of parameters as well as parallelization [37, 38].

Other improvements to the particle filter can be made for the density weight. This weight did not perform as expected, with negative impact on the particle filter method. A different way of combining the three weighting factors might lead to better performance. Currently, the density weight is added to the intermediate weight, consisting of both the particle and velocity weight. This might not be the best option, since it adds extra weight to particles which are close to measurement cluster, which already have high weight due to the other factors. The clusters of particles which do not have measurements, for example due to occlusion, do receive some density weight, but this is too small to help them survive. A better option might be to only add a density weight factor to particles which did not receive any weight from the positional or velocity vector weights. In this way, the weight of already high weighted particles is not increased, and particles which are in dense clusters without any measurements close by do receive only a density weight. Properly tuning this weight to make sure these clusters survive the resampling step might lead to better performance in occlusion, although still attention needs to be taken to make sure these clusters will die out afer a longer period without measurements.

Another suggestion for improving the resulting occupancy grid is the addition of unknown area. Currently, only occupied or free cells are provided in the resulting occupancy grid. However, a method has been

developed to determine the unknown area behind objects, namely for the ground truth occupancy grid. This method could also be used for the particle filter resulting occupancy grid, to increase the information this grid holds. It will show the final user which area is not measurable by its sensors, which can be of importance to for example path planning applications. The implementation can be taken from the ground truth occupancy grid implementation, but for real life testing the approach will need to be more efficient, to decrease the overhead in computation that the sampling method introduces. Of course, specialized hardware such as a GPU (with ray-tracing capability) could drastically improve this.

5.1.2 Future research topics

An interesting topic to research further is the translation from particle filter to occupancy grid. This research chose to step away from the occupancy grid for the estimation method, thereby providing more freedom to develop weighting factors for the particle filter independent of the grid representation. However, this does lead to difficulties when translating the particle population back to an occupancy grid representation, as there is no real relation between the two. The currently implemented method does work, but is very crude; it only provides a binary occupancy grid as result. Ideally, the occupancy grid should have continuous values, to have more detailed information about the probabilities of occupancy in the environment. A different translation is necessary here. In an early stage of this research this has been tried by normalizing the count of particles per grid cell to a range between 0 and 1, but this was deemed unsuccessful, especially in multi-target scenarios.

Another interesting topic is cooperated driving of automated vehicles. If vehicles which are near to each other can work together in mapping the environment, more information is available through the communication of sensor information. It could also be possible to not share sensor information, but instead share the resulting occupancy of a vehicle's direct region with another vehicle. This introduces different challenges, such as communication bandwidth constraints and localizing the other actor in the ego vehicle frame, and translating the two different vehicle frames to each other. Sensors such as GPS can be used for this, but might not be accurate enough. Further research can focus on the problems of localization and sharing of information.

Stepping away from the occupancy grid as output, there are other use cases for the particle filter approach. For example in target tracking application, where the goal is not to create a map of the environment, but identify an object from measurements, and accurately follow the object while keeping information about the traversed track. To this end, the translation to an occupancy grid is not necessary, and the particle population can be directly used. There is the need for a grouping the particles however, since currently the individual particles are independent of each other and do not contain information identifying them to a certain object. Specific clustering algorithms can be to achieve this, such as introduced in [39]. Moreover, since the particles each posses a velocity state, this velocity information can be used to more accurately determine the velocity of tracked objects. This has been shown in a one-dimensional scenario by [23], and shows promise for the particle filter as part of a multi-target tracking method.

Bibliography

[1] J. Leonard, H. Durrant-Whyte, and I. Cox, "Dynamic map building for autonomous mobile robot," in *EEE International Workshop on Intelligent Robots and Systems, Towards a New Frontier of Applications*, pp. 89–96 vol.1, 1990.

[2] D. Joubert, *Adaptive occupancy grid mapping with measurement and pose uncertainty*. PhD thesis, Stellenbosch: Stellenbosch University, 2012.

[3] S. Singh, "Critical reasons for crashes investigated in the national motor vehicle crash causation survey," March 2018. Traffic Safety Facts Crash Stats. Report No. DOT HS 812 506.

[4] S. E. Shladover, "Opportunities and challenges in cooperative road vehicle automation," *IEEE Open Journal of Intelligent Transportation Systems*, vol. 2, pp. 216–224, 2021.

[5] i CAVE, "I-Cave - Integrated cooperative automated vehicles," 2018. Accessed on 27 July 2022 at `https://i-cave.nl/`.

[6] R. Reid, A. Cann, C. Meiklejohn, L. Poli, A. Boeing, and T. Braunl, "Cooperative multi-robot navigation, exploration, mapping and object detection with ros," in *2013 IEEE Intelligent Vehicles Symposium (IV)*, pp. 1083–1088, 2013.

[7] S. Goldberg, M. Maimone, and L. Matthies, "Stereo vision and rover navigation software for planetary exploration," in *Proceedings, IEEE Aerospace Conference*, vol. 5, pp. 5–5, 2002.

[8] H. P. Moravec, "Sensor fusion in certainty grids for mobile robots," *AI Magazine*, vol. 9, p. 61, Jun. 1988.

[9] A. Elfes, "Using occupancy grids for mobile robot perception and navigation," *Computer*, vol. 22, no. 6, pp. 46–57, 1989.

[10] G. K. Kraetzschmar, G. P. Gassull, and K. Uhl, "Probabilistic quadtrees for variable-resolution mapping of large environments," *IFAC Proceedings Volumes*, vol. 37, no. 8, pp. 675–680, 2004.

[11] T. Weiherer, E. Bouzouraa, and U. Hofmann, "A generic map based environment representation for driver assistance systems applied to detect convoy tracks," in *2012 15th International IEEE Conference on Intelligent Transportation Systems*, pp. 691–696, IEEE, 2012.

[12] M. E. Bouzouraa and U. Hofmann, "Fusion of occupancy grid mapping and model based object tracking for driver assistance systems using laser and radar sensors," in *2010 IEEE Intelligent Vehicles Symposium*, pp. 294–300, 2010.

[13] M. Schreier, "Bayesian environment representation, prediction, and criticality assessment for driver assistance systems," *at-Automatisierungstechnik*, vol. 65, no. 2, pp. 151–152, 2017.

[14] C. Coué, C. Pradalier, C. Laugier, T. Fraichard, and P. Bessière, "Bayesian occupancy filtering for multitarget tracking: an automotive application," *The International Journal of Robotics Research*, vol. 25, no. 1, pp. 19–30, 2006.

[15] H. Wang, Z. Hou, and M. Tan, "Mapping dynamic environment using gaussian mixture model," in *6th IEEE International Conference on Cognitive Informatics*, pp. 424–429, IEEE, 2007.

[16] P. Held, D. Steinhauser, A. Koch, T. Brandmeier, and U. T. Schwarz, "A novel approach for model-based pedestrian tracking using automotive radar," *IEEE Transactions on Intelligent Transportation Systems*, 2021.

[17] P. C. D. Hirvonen and T. C. B. Southall, "Stereo-based object detection, classification, and quantitative evaluation with automotive applications," in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)-Workshops, San Diego, CA, USA*, pp. 20–26, 2005.

[18] S. Thrun, "Probabilistic robotics," *Communications of the ACM*, vol. 45, no. 3, pp. 52–57, 2002.

[19] J. Choi, S. Ulbrich, B. Lichte, and M. Maurer, "Multi-target tracking using a 3d-lidar sensor for autonomous vehicles," in *16th International IEEE Conference on Intelligent Transportation Systems (ITSC 2013)*, pp. 881–886, IEEE, 2013.

[20] S. J. Julier and J. K. Uhlmann, "Unscented filtering and nonlinear estimation," *Proceedings of the IEEE*, vol. 92, no. 3, pp. 401–422, 2004.

[21] E. A. Wan and R. Van Der Merwe, "The unscented kalman filter for nonlinear estimation," in *Proceedings of the IEEE 2000 Adaptive Systems for Signal Processing, Communications, and Control Symposium (Cat. No. 00EX373)*, pp. 153–158, IEEE, 2000.

[22] R. Danescu, F. Oniga, and S. Nedevschi, "Modeling and tracking the driving environment with a particle-based occupancy grid," *IEEE Transactions on Intelligent Transportation Systems*, vol. 12, no. 4, pp. 1331–1342, 2011.

[23] D. H. E. Teeuwen, "Dynamic occupancy grid mapping with velocity measurements by a particle filter approach," 2021.

[24] D. Nuss, T. Yuan, G. Krehl, M. Stuebler, S. Reuter, and K. Dietmayer, "Fusion of laser and radar sensor data with a sequential monte carlo bayesian occupancy filter," in *2015 IEEE intelligent vehicles symposium (IV)*, pp. 1074–1081, IEEE, 2015.

[25] C. Coue, T. Fraichard, P. Bessiere, and E. Mazer, "Using bayesian programming for multi-sensor multi-target tracking in automotive applications," in *2003 IEEE International Conference on Robotics and Automation (Cat. No.03CH37422)*, vol. 2, pp. 2104–2109 vol.2, 2003.

[26] The MathWorks Inc., "Matlab version: 9.11.0 (r2021b)," 2021. `https://www.mathworks.com` Accessed on 8 August 2023.

[27] NXP, "Cocoon user manual," 2017.

[28] O. J. Woodman, "Technical report: An introduction to inertial navigation," 2007.

[29] F. L. Gaol, "Bresenham algorithm: Implementation and analysis in raster shape.," *J. Comput.*, vol. 8, no. 1, pp. 69–78, 2013.

[30] P. S. Heckbert, "Survey of texture mapping," *IEEE computer graphics and applications*, vol. 6, no. 11, pp. 56–67, 1986.

[31] M. Yguel, O. Aycard, and C. Laugier, "Efficient gpu-based construction of occupancy girds using several laser range-finders," in *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 105–110, 2006.

[32] G. Ferri, M. V. Jakuba, A. Mondini, V. Mattoli, B. Mazzolai, D. R. Yoerger, and P. Dario, "Mapping multiple gas/odor sources in an uncontrolled indoor environment using a bayesian occupancy grid mapping based method," *Robotics and Autonomous Systems*, vol. 59, no. 11, pp. 988–1000, 2011.

[33] M. Yguel, O. Aycard, and C. Laugier, "Update policy of dense maps: Efficient algorithms and sparse representation," in *Field and service robotics*, vol. 42, pp. 23–33, Springer, 2008.

[34] R. G. Gallager, *Stochastic Processes: Theory for Applications*. Cambridge University Press, 2013.

[35] K. Li, G. Lin, L. Lee, and J. Juang, "Application of particle filter tracking algorithm in autonomous vehicle navigation," in *2013 CACS International Automatic Control Conference (CACS)*, pp. 250–255, IEEE, 2013.

[36] R. Grewe, M. Komar, A. Hohm, S. Lueke, and H. Winner, "Evaluation method and results for the accuracy of an automotive occupancy grid," in *2012 IEEE International Conference on Vehicular Electronics and Safety (ICVES 2012)*, pp. 19–24, 2012.

[37] The MathWorks Inc., "Tune parameter structures by using matlab language," 2023. https://nl.mathworks.com/help/slrealtime/ug/tune-parameter-structures-with-matlab-language.html Accessed on 14 August 2023.

[38] The MathWorks Inc., "Parallel computing toolbox," 2023. https://nl.mathworks.com/products/parallel-computing.html Accessed on 14 August 2023.

[39] D. Frijters, "Adaptive recursive clustering and target tracking," 2022.