Dempster Shafer Sensor Fusion for Autonomously Driving Vehicles

Association Free Tracking of Dynamic Objects

ANDREAS HÖGGER
Abstract

Autonomous driving vehicles introduce challenging research areas combining different disciplines. One challenge is the detection of obstacles with different sensors and the combination of information to generate a comprehensive representation of the environment, which can be used for path planning and decision making.

The sensor fusion is demonstrated using two Velodyne multi beam laser scanners, but it is possible to extend the proposed sensor fusion framework for different sensor types. Sensor fusion methods are highly dependent on an accurate pose estimate, which can not be guaranteed in any case. A fault tolerant sensor fusion based on the Dempster Shafer theory to take the uncertainty of the pose estimate into account is discussed and compared using an example, although not implemented on the test vehicle.

Based on the fused occupancy grid map, dynamic obstacles are tracked to give a velocity estimate without the need of any object or track association methods. Experiments are carried out on real world data as well as on simulated measurements, for which a ground truth reference is provided.

The occupancy grid mapping algorithm runs on central- and graphical-processing units, which allows to give a comparison between the two approaches and to stress out which approach is preferably used, depending on the application.

Sammanfattning

Självkörande bilar har lett till flera intressanta forskningsområden som kombinerar många olika discipliner. En utmaning är att ge fordonet en sorts ögon. Genom att använda ytterligare sensorer och kombinera data från samtliga så kan man detektera hinder i fordonets väg. Detta kan naturligtvis användas för att förbättra fordonets planerade rutt och därmed också minska klimatpåverkan.

Här används tvåsammankopplade Velodyne laserstrålsensorer för att undersöka detta Närmare, men det går också att utöka antalet sensorer ytterligare. Sammanlänkningen av sensorer är mycket känslig och kräver därför exakta koordinater, vilket inte alltid kan garanteras. Därför utreds istället om en sensor baserad på Dempster Shaferteorin kan användas för att hantera fel och osäkerheter. Denna används dock inte i testfordonet.

Baserat på sammanvägda kartbild över upptagna och fria områden (occupancy grid mapping) kan objekt och hinder i rörelse följas för att uppskatta fordonets hastighet utan att metoder för objekt- eller banidentifiering behöver användas. Experiment har utförts påverklig data. Dessutom används simulerade mätningar där en sann grundreferens används.

Algoritmen som används för occupancy-kartan använder sig av central- och grafik-processorenheter, vilket ger oss möjlighet att jämföra tvåmetoder och finna den bäst fungerande metoden för olika applikationer.
To my parents and grandparents.
Contents

1 Introduction 5
   1.1 Problem Statement ........................................... 7
       1.1.1 Goals ................................................. 9
       1.1.2 Assumptions ........................................... 9
       1.1.3 Implementation Limitations ............................ 9
       1.1.4 Outcome .............................................. 10
   1.2 Related Work .............................................. 11
       1.2.1 Sensor ............................................... 11
       1.2.2 Ground Removal ...................................... 15
       1.2.3 Sensor Fusion ........................................ 16
       1.2.4 Safety Analysis ...................................... 20
       1.2.5 Static Map Integration ................................. 21
       1.2.6 Parallel Computing ................................... 22
   1.3 Major Contribution ...................................... 23
   1.4 System Overview ..................................... 23

2 Occupancy Grid Mapping 25
   2.1 Dempster Shafer Sensor Fusion .............................. 26
   2.2 Sensor Model ........................................... 27
      2.2.1 Mass Assignment for Accurate Position Estimation 28
      2.2.2 Parameter Derivation ................................ 29
      2.2.3 Fault Tolerant Sensor Model .......................... 29
   2.3 Occupancy Grid Update .................................. 32
   2.4 Obstacle Map ........................................ 32

3 Dynamic Object Tracking 33
   3.1 Dynamics Identification ................................ 34
   3.2 Dynamic Object Filtering on Cell Level .................. 34
      3.2.1 Problem Definition ................................ 36
      3.2.2 Filtering ........................................... 36
   3.3 Particle Management ................................... 38
      3.3.1 Draw Particles .................................... 38
      3.3.2 Normalization of Particle Weights ................ 39
CONTENTS

3.3.3 Unknown Velocities ........................................... 40
3.4 Colliding Obstacles ............................................. 40
3.4.1 Proof of Collision Free Rigid Objects ..................... 40
3.4.2 Downgrading of Crossing Particles ......................... 41

4 Evaluation ......................................................... 43
4.1 Simulated Static Environment ................................. 43
4.1.1 Experimental Setup ........................................ 44
4.1.2 Results ..................................................... 46
4.2 Static Real World Experiments ............................... 47
4.2.1 Driving scenario ............................................ 48
4.2.2 Enhanced Knowledge through Movement .................. 49
4.2.3 Ground separation methods ................................ 49
4.3 Dynamic Obstacle Tracking .................................. 50
4.3.1 Simulated Velocity Comparison ......................... 51
4.3.2 Real World Pedestrian Tracking ......................... 52
4.3.3 Step Response .............................................. 53
4.4 Central Processing Unit (CPU) vs. Graphics Processing Unit (GPU) 54
4.4.1 Timing Overview .......................................... 55
4.4.2 Results .................................................... 57

5 Conclusion ......................................................... 59
5.1 Improvements .................................................. 60
5.2 Drawbacks ....................................................... 60
5.3 Ethical Aspects of Autonomous Driving .................... 61
5.4 Future Work .................................................... 61

List of Figures ...................................................... 63
List of Tables ........................................................ 65
List of Algorithms .................................................. 65
Bibliography .......................................................... 69
A Further Experiments ............................................. 73
B Sensor alignment ................................................... 77
C Velodyne VLP-16 .................................................. 81
D Implementation Details ......................................... 83
Chapter 1

Introduction

Intelligent transportation systems are key factors of our century. More and more people and goods are required to be at different locations within less time. At the same time safety standards must not be decreased but should even be improved by new developments. A big part of the transportation capabilities is provided by human operated vehicles on the road, which is a very flexible way of mobility. For example the International Council on Clean Transportation (ICCT) expects an increased number of heavy duty vehicles from one billion in 2010 to 1.7 billion in 2030 [21]. Nevertheless human drivers are not perfect and make errors. Humans get tired, type messages on a phone and read the newspaper while driving or simply are not able to oversee the whole traffic situation. In the Netherlands, the use of mobile phones while driving was responsible for 8.3% of the total number of dead and injured victims in 2004 [44]. A solution is to use appropriate sensors and algorithms running on reliable hardware to assist or replace human drivers. In addition this would free the driver to do other things or to relax whilst traveling. But on the other hand the human perception is brilliant. The eye brain combination can recognize all different kind of objects, unexpected behaviors of other drivers and different weather conditions immediately. Even though it has never seen the exact same situation before. Classical computers or machines fail if they are not specially programmed for given situations.

In particular uncertainties cause problems in the decision making of self driving vehicles. Such uncertainties are always introduced by all kind of sensors sensing the surroundings of a car, as their reliability varies depending on the situation and the conditions. To get the best possible knowledge about the area surrounding the ego vehicle, it is important to include all information accessible, but also to know how reliable the gathered information is. This is the reason for using a probabilistic representation of the sensed information in self driving vehicles and addressed within this work using the Dempster Shafer (DS) sensor fusion theory.

Despite the unsolved problems a lot of engineers are addressing this challenge nowadays. There are many examples of companies and research institutes with very different approaches trying to make vehicles self-driving. Additional motivation is
given through various kinds of competitions regarding autonomously driving vehicles. The Defense Advanced Research Projects Agency (DARPA) Grand Challenge was the first of this kind of events. In the beginning cars had to navigate through a desert without any other participants on the street. The last Grand Challenge, also called the DARPA Urban Challenge [8], was held in a simulated urban environment. Therefore cars had to follow street rules and react to other vehicles’ behaviors. Figure 1.1 shows a reformed autonomously driving research vehicle based on experiences from the challenge (left) besides the experimental research vehicle used within this thesis (right).

![Figure 1.1: Autonomous research vehicle based on experiences in the DARPA challenge © 2011 IEEE (left) and the RCV (right) used to carry out experiments throughout this thesis.](image)

To successfully overcome the task of autonomous driving various engineering areas have to work together. Sensors need to gather information about the vehicle’s surrounding environment. All the sensor information is fused to get an overall picture of the world around the car, also called perception. Another important aspect is to have precise position estimation. Localization can either be provided by positioning systems like Global Navigation Satellite System (GNSS) and Inertial Measurement Unit (IMU) only or by taking the perception sensor information into account as well. Supported by a map the decision making is done based on all of the information provided to the car.

This thesis is carried out in collaboration with two other master thesis students, giving the opportunity to cover all the tasks in a simplified way. Simplified means to state limitations on the complexity of covered scenarios and environmental conditions. Nevertheless the goal after combining the three thesis projects is to show a research vehicle driving autonomously on a test track. It becomes very challenging as besides the physical challenges integration on the RCV has to be done from scratch. Details about the system configuration and the required changes are given in a later section 1.4. In this paper focus is given on the perception part covering sensor fusion, which results in a grid map of different kind of information sources. The other two degree projects are about providing a very accurate position estimate and doing the path planning. The position estimate combines data from GNSS, IMU and odometry sensors. To do path planning the grid map provided by the
perception system is used. As the car follows the path static obstacles are avoided autonomously. More details about the goals, assumptions and limitations for this specific work are given in the following section 1.1.

1.1 Problem Statement

Figure 1.2: Example for detection of different static and dynamic obstacles

The problem statement section clearly outlines goals for this specific thesis work. As mentioned before, the work depends on another final degree project and is a predecessor for a following thesis. Figure 1.2 shows an intersection in a city with a few other cars besides the red autonomous car. This example gives an idea of what the perception system of self-driving cars should be capable of. By detecting the opponents the car performs a path planning to avoid collisions. For static scenarios
it is sufficient to detect drivable space on a street, which is the whole space without obstacles. However in urban environments many moving obstacles are present and in order to avoid collision, tracking of these dynamical obstacles becomes necessary. A lot of research has already been carried out and is conducted towards detection of drivable space, tracking other vehicles and many other extensions of the autonomous driving problem. Results are satisfying for limited cases. The biggest limitation is the real time capability, which is challenging to reach for very accurate algorithms. In this thesis focus is given to explore sensor fusion using Dempster Shafer theory and to give a solution running in real time on low cost hardware. Based on that fusion framework tracking of dynamic objects is done. The velocity of dynamic objects is estimated in a probabilistic manner as soon as the obstacles are moving. Therefore no classification is required for dynamic object detection. To get a compromise between execution time and accuracy a continuous velocity description overlays tracking on grid resolution.

An overview of the sensor fusion framework is given in Figure 1.3. Different sensor information is represented by grid maps. Some sensors can generate different information sources. For example a stereo camera could detect classes of vehicles as well as their position. The grid fusion is done by using the Dempster Shafer grid fusion framework. Like the commonly used Bayesian sensor fusion approach, Dempster Shafer uses sensor models to generate a conversion from the sensor readings to a probabilistic value called the belief mass. Dempster Shafer fusion is chosen due to further freedom and extended opportunities in conflict resolution compared to the Bayesian approach. Finally the developed framework provides two results. One is

![Figure 1.3: Sensor Fusion Overview](image-url)
the definition of drivable space and the second one is a probabilistic velocity esti-
mation for each grid cell. In addition to the research work carried out, the concept
is proofed by implementation of the algorithm on the RCV. Due to time limitations
only the parts in Figure 1.3 drawn with solid lines are actually implemented on the
test hardware. The presented framework shall be expandable to cover all sensors
and cases in a future version of the implementation running on the RCV.
According to the algorithm implementation a simulation framework is set up. This
gives the opportunity for further testing of varying scenarios and makes the thesis
results independent of the test hardware availability and weather conditions. How-
ever not all effects can be simulated efficiently. In the simulation vehicle dynamics or
skewing of rotating sensors are neglected as well as different optical conditions and
reflection coefficients of materials. The simulator provides snapshot point clouds and
ground truth position estimates at every time. A summary about goals, assumptions
and limitations is given as follows.

1.1.1 Goals

- Fuse sensor information using Dempster Shafer framework
- Provide map with drivable space
- Real time, model-free tracking of dynamic obstacles
- Example implementation using two Velodyne VLP-16 Light Detection and
  Ranging (LiDAR)
- Map update rate of 10 Hz for implementation

1.1.2 Assumptions

- Short term position drift is negligible
- No disturbance by hazardous weather conditions

1.1.3 Implementation Limitations

- Flat, even street
- Dynamic objects speed of $1 \text{ m s}^{-1}$ to $15 \text{ m s}^{-1}$
- Obstacles detected within 0.15 m to 2 m above ground
- Grid size 0.1 m $\times$ 0.1 m
- No simulation of vehicle dynamics, skewing of sensors, weather conditions and
different reflection indexes
- Ground truth position estimate provided by simulation
1.1.4 Outcome

Finally the outcome map where the path planning takes place is further explained. The first outcome is a grid map providing information about drivable space. Figure 1.4 shows how the grid looks like. The grid shows a locally fixed occupancy grid map, with cell colors indicating the occupancy state of a given cell. For the outcome grid, previous sensor information is mapped as well. There is no further information provided, apart from the cell being occupied, free or unknown in the first part (left hand side in Figure 1.4).

Additionally a second grid layer describes the velocity estimates of dynamic objects (right hand side in Figure 1.4). This grid describes velocity estimates as probability distributions over different velocities for each grid cell (in the figure only mean velocity values and directions are indicated by the arrows). With probability distributions over the velocities, including different directions, it is possible to predict positions of obstacles and use that information in a path planner to reduce the risk of collision.

In the next section a literature study is carried out including a larger area of topics related to the autonomous driving field than used within the thesis. The scientific contribution is explicitly mentioned (section 1.3) afterwards, which refers to the related work to point out the differences to other works.
1.2 Related Work

The related work section covers all areas throughout the sensor fusion, including sensor calibration and modeling, sensor fusion approaches including dynamics and the safety analysis. Note that only static and dynamic obstacle mapping is implemented but for future work more literature is covered, for example the safety analysis. Sensors are given as two Velodyne VLP-16 LiDAR and therefore this section focuses on papers for this particular kind of sensor type. For a more general overview about different sensor types it is referred to another student thesis [12], which also gives a good idea of the topic in general. A critique is given on literature reviewed taking the goals of the thesis into account. This critique gives an idea how these publications can improve the solutions developed throughout the thesis.

1.2.1 Sensor

The Velodyne VLP-16 sensor is a multiple beam laser scanner, which outputs three dimensional point clouds including an intensity value according to each reflected laser beam. More information about the sensor is found in appendix C. Velodyne LiDARs cover a 360° horizontal Field of View (FOV) as the laser beams are rotating around the vertical axis. This rotation needs to be considered when using these sensors.

Calibration

Before using the sensors they need to be calibrated. Calibration is divided into intrinsic and extrinsic calibration. The former calibrates the laser range measurements and the laser reflection indexes to a sensor reference, while the latter one calibrates the sensor coordinate frame with respect to a common frame on the vehicle. An intrinsic calibration file is already provided by the manufacturer but improvements are still possible [17]. Extrinsic calibration of rotational LiDAR is mostly done with reference to another vision sensor like cameras [16] and a static calibration setup (camera to range sensor calibration). During this work the extrinsic calibration between two LiDARs is required. Typically point cloud registration is done by using an Iterative Closest Point (ICP) algorithm, which tries to align the two sets of points by minimizing the distance between the individual points. Figure 1.5 shows point clouds recorded by two VLP-16 sensors mounted on a car parked indoors. Many points are reflected from the ground plane which leads to the circular reflections pointed out in Figure 1.5. Matching these points leads to an erroneous alignment of both sensors, as the spacing of the different lines on the ground floor does not contain information about the translation in x- and y-axis.

To overcome this problem, known shapes can be placed in front of the different sensors. Such an approach is already used for camera to range sensor calibration. While it is usually sufficient to do the calibration once in a static way other calibration routines correct the parameters during the vehicle’s operation [25]. Adapting
the calibration parameters during operation requires a high algorithmic complexity and a reliable motion estimate of the robot.

**Rotation**

Beams of the laser scanner are rotating around the vertical axis with a configurable rotation frequency of 5 Hz to 20 Hz. The number of points generated per second is independent of the rotation frequency, in other words the horizontal resolution decreases accordingly. This frequency range is not sufficient to treat the point clouds as static pictures when the sensor is mounted on a car. An accurate pose of the vehicle is required to do the transformations at every time instance. Overcoming these effects is further on called deskewing. If a smooth and reliable position estimate is not available or the position and LiDAR data are not synchronized, one could calculate the differential position change between each frame [29] and use it for deskewing. By using the proposed method an accumulated position error of 2.3 m and 4.1 m is reached after driving test loops of 1.3 km and 1.1 km in distance.

Figure 1.5: Top down view on indoor point clouds used for calibration - orange frames mark the ground sections and green and white points are laser reflections on obstacles, walls and the ground
1.2. RELATED WORK

respectively.

Model

A sensor model provides a relation between given measurements and the likelihood of a grid cell being occupied, empty or unknown. Previously LiDARs had one beam only and were scanning the environment in a plane parallel to the ground. Therefore the occupancy state of grid cells was determined by using the method [31] shown in Figure 1.6. The top half of the figure shows the situation in Cartesian coordinates,

![Figure 1.6: Occupancy grid state using a single beam, planar LiDAR [31] © 2011 IEEE](image)

while the bottom part shows the measurements in the polar sensor grid. Every cell a laser beam hits an obstacle is set occupied and all the cells a laser pass through are determined free. A three dimensional occupancy grid mapping using a Velodyne LiDAR is presented in [3]. Both approaches are idealized in the sense that the distance measurement of the sensors is taken as ground truth. Probabilistic sensor models are given in [1] and [11], whereas the latter one gives a good idea of how these models are derived and modeled as combined Gaussian functions. Multi beam laser scanners cover all directions. For ground and obstacle detection the lower half of the beams is important. The basic laser beam model stays the same when using lasers pointing towards the ground. An improved information gathering from sensor measurement is described in [7] assuming the ground level is detectable at all points of the occupancy grid. Although it is using a very similar Velodyne sensor, the model can not be applied directly, as the calculations assume the sensor to be mounted in parallel to the surface. This assumption does not necessarily hold as the sensor positions and orientations on the RCV vary to experiment
with different setups. The idea for the improvement shown in Figure 1.7 could be adapted to consider varying sensor poses. By setting a threshold \((H)\) for detecting obstacles multiple cells can be assigned with the free state. A comparison of the extracted information with and without backward extrapolation \([7]\) for a threshold height of 20 cm is given in Figure 1.8. The inverse sensor model provided is specifically adapted for being used within the Dempster Shafer fusion framework. As an accurate ground extraction is required subsection 1.2.2 studies approaches solving that task.

Figure 1.7: Backward extrapolation of laser beams \([7]\) © 2014 IEEE

![Figure 1.7: Backward extrapolation of laser beams](image)

(a) Point Cloud without backward extrapolation  (b) Point Cloud with backward extrapolation

Figure 1.8: Comparison of occupancy grids with and without backward extrapolation and a threshold height of 20 cm \([7]\) © 2014 IEEE, where red indicates obstacles and green the ground space known to be free.
1.2. RELATED WORK

1.2.2 Ground Removal

Detecting the ground and excluding it from possible obstacle data points is an important task and can even increase processing speed for following algorithms. The problem description (section 1.1) mentions the limitation to a flat, even surface. In this case ground removal becomes straight forward by ignoring all points with z-coordinates lower than a specific value. If the car rolls because of turning, it is important to detect the roll angle, which could be done by a plane detection using Random Sample Consensus (RANSAC) [13].

One very popular method during the DARPA challenges is described in [19]. This approach calculates the maximum absolute difference in z-coordinates (z-axis orthogonal to the ground plane) for every grid cell. Obstacles are identified if the difference is above a specified threshold value. The main drawback of this fast and easy to implement algorithm is that it would fail for obstacles with flat slopes. An expansion to the previous decision rule is given [4] by using the derivation of the mesh connecting points. It is not limited to a given grid size and could additionally detect curbed area along the street as well. The algorithm [4] is developed to run on a GPU, which makes it suitable for real time applications. A comparison of processing time between different GPUs and CPUs is evaluated.

Another simple (computationally efficient) approach is described in [10]. The ground partition is found by clustering together adjacent voxels based on vertical means and variances. If the difference between means is less than a given threshold, and the difference between variances is less than a separate threshold, the voxels are grouped. The largest partition found by this method is chosen as the ground. As the ground extraction is a typical classification problem one could also train a classifier with labeled data sets. Labeling point cloud data manually is almost impossible as the LiDAR typically generates hundreds of thousand points each second. If the labeling is done automatically huge data sets can be used for training a constant-time decision tree implementation [38]. The advantage is that the labeling algorithms can be more sophisticated and time consuming than the trained decision maker running on the vehicle. For smaller numbers of grid cells the labeling method based on following parameters could also run in real time:

- Average height of ground hits in global coordinates
- Variance of ground hit heights
- Average height of obstacle hit in global coordinates
- Variance of obstacle hit heights
- Ground height as measured by known tire position
- Variance of tire hit heights
Often it is required to separate all potential obstacles and the ground from point cloud data. With a general separation algorithm [28] potential obstacles are merged automatically. A local convexity criterion is introduced to decide about membership of point clusters. The method is used to separate ground and potential objects and successfully track dynamic or static objects without any model assumptions [30].

It is worth mentioning that the problem exists in other applications, for example in extracting the ground [27] from airborne LiDAR data. The results are really satisfying although there is no focus on real time performance of such algorithms. In contrast to all previously mentioned methods the scenarios seem much more challenging as according to the other example’s point clouds their ground was fairly flat.

1.2.3 Sensor Fusion

Goal of the sensor fusion is to combine different sensor information and represent the environment in a condensed way. It should be easy to use the fused data for higher level applications.

Architecture

Data from different sensors can differ in their coordinate frame, resolution or update rate. A framework [11] to merge different sensor information is shown in Figure 1.9. The sensor fusion takes place in Sensor Integration and Map Updating. A hierarchical sensor fusion architecture is presented in Figure 1.10. This architecture [33] focuses on a modular approach, which is manageable for more complex perception systems. Tracking of dynamic objects is included in the framework as well. As the raw data from any sensor may provide it’s information to all modules, perception sensors could support the Ego Motion Estimation as well as the the Grid Mapping.

Dempster Shafer Fusion

First the basic difference between Dempster Shafer and Bayesian sensor fusion approach are explained in [7] and [12], which are compact resources with clear examples and closely related to this thesis. The Dempster Shafer combination rules are not perfectly suited for every application. A counter example exists in [46], where it obviously leads to results, which humans interpret as wrong. In this example two sources of information give zero belief to the event which is given a very high belief by the other source respectively. All the spare belief is close to zero but not equal to and given to the same third event by both information sources. Now the normalization of Dempster Shafer (DS) sensor fusion spreads all the belief to the third event, because both information sources provide a belief bigger than zero, even though these believes are neglectable small. One possible reason for a misleading combination result is ignoring some assumptions included in the Dempster Shafer
theory, namely that given beliefs are considered as reliable values. Different adhoc improvements to the theory exist to overcome the particular problem, for example in [43]. Another approach is to take the *open world* into account. In an *open world* the models for calculating evidence never perfectly cover all situations, especially not while operating in an unknown environment. The Transferable Belief Model [40] explicitly considers that fact and builds on the Dempster Shafer theory. Dempster Shafer’s advantage is, that a measure of conflict can be generated. It is based on the *consistency assumption* saying a point in space can either be occupied or free. Different reasons can cause conflicting believe values for occupied and free [5]:

- Sensor noise
- Use of inappropriate sensors
- Inaccurate *a priori* models
- Use of a flawed internal representation

Figure 1.9: Framework for occupancy grid based sensor fusion [11] © 1989 IEEE
Experiments showed the possibility of detecting inappropriate sensors and decrease their influence on the result, when fusing different sensor types. When sensor information is highly conflicting the Dempster Shafer theory outperforms the Bayesian filter for position estimation [5]. Results show smaller errors by a factor of 20 using an adapted Dempster Shafer filter. Another use of a conflict measurement is to detect dynamic obstacles [22]. In this paper a very simple approach for filtering dynamic obstacles on a second occupancy grid layer is introduced. The tracking is done using a grid based particle filter.

**Dynamic Environment**

Dynamic obstacles are challenging for the perception system. If such obstacles are not considered noise is added to the static environment which results in less accurate obstacle maps or ego vehicle localization. Detection of dynamic obstacles could be done by using single frames only, or by comparing frames over time. When using a single frame only, classifiers need to be trained beforehand. In real world scenery it becomes difficult to cover all cases. A more general way is to extract non static information by comparing consecutive sensor readings. Focus is given to the latter one as LiDAR data is not well suited for doing steady frame classification as cameras are for example. Dynamic object tracking is done on different levels of abstraction. Starting from an easy to handle 2D occupancy grid up to tracking of raw point cloud data. It is a tradeoff between accuracy and complexity. A more abstract representation can cover multiple sensor types, while point clouds
for example only fit for 3D range measurement sensors. This gives a good reason for trying to use occupancy grid maps as a start point for tracking applications. Of course some of the available informations of LiDARs are ignored by using this abstraction. In Bayesian framework Bayesian Occupancy Filter (BOF) is very popular to describe both, occupancy and speed of grid cells. A real time BOF implementation [6] based on 2D grid representation can track dynamic obstacles without any shape or complex dynamic model of the obstacle. Only constant linear velocities are considered, which holds if the frame rate of sensors is high compared to the velocity changes. Data association is postponed by tracking each cell independently. Later it is possible to combine grid cells with similar velocities to extract objects for high level usage.

The comparable method for evidential grids [22] uses particle filters to track dynamic objects. All knowledge about the dynamics is modeled in a second layer evidential grid. The identification of dynamics is supported by the Dempster Shafer conflict theory. As the free of model approach results in clusters of particles, the footprint shapes of objects are available after successful tracking. Successful tracking is shown as examples at 50 km h$^{-1}$ and 160 km h$^{-1}$ in an urban environment and on a highway respectively. However in each test case only one moving car is present in the scenery. Using a 2D representation of 3D point clouds summarizes the information and many features are neglected. To overcome the problem one could simply use 3D grid maps. The computational complexity for 3D occupancy grids increases by the power of three with increasing resolution of the grid. A lot of cells would be empty in autonomous driving scenarios. It is more efficient to use dynamically adaptable octrees [3] to represent the environment. By keeping the 3D information it is possible to classify dynamic obstacles using the dimensions of the bounding boxes. Of course this only gives a rough estimation. Nevertheless the classification whether it is a car or a pedestrian is used to use different dynamic models for the tracked objects.

In [30] a very general approach is presented to simultaneously separate the scene, track dynamic objects and keep all the point cloud information. As all the point cloud data is kept related to the objects it is possible to reconstruct the whole objects when passing by (Figure 1.11). The results are truly satisfying and the source code is provided. Concluding from the comments within the source code, it is impossible to achieve real time performance on current consumer CPUs. One could take the algorithm and explore potential parallelism in order to port the code running on a GPU.
1.2.4 Safety Analysis

Safety analysis provides a framework for path planning algorithms to double check their collision risk using all information extractable from sensor fusion. A good understanding of the environment including potential obstacles is inevitable. For this full understanding a lot of facts have to be taken into account. An example of how such a safety analysis framework [24] looks like, is shown in Figure 1.12. Examples for additional information are Light Indicators, the Road Geometry and other Objects. Taking all information and the ego vehicle path into account a risk assessment is carried out. The risk estimation is done by calculating all possible future paths using Gaussian processes [24]. For this thesis the focus is given to collisions with obstacles.

The basic idea stays the same for all risk or safety analysis tasks. In some way possible future paths of dynamic obstacles have to be estimated. It is impossible to consider all possible paths of potential dynamic objects. Instead a Monte Carlo simulation could be carried out [2] to give infinite future time probability of collision estimation. Monte Carlo method is applied for possible ego vehicle trajectories. Although ego vehicle trajectories are limited to the suggestion of the path planner, it is useful for handling trajectories of other vehicles or persons on the street. A different approach is to use probabilistic linear object velocities [14] to detect potential collisions. The method is well suited for a grid based environment representation.

Figure 1.12: Architecture of the risk assessment module [24] © 2011 IEEE
1.2. RELATED WORK

1.2.5 Static Map Integration

Maps for autonomous driving purposes are highly detailed. These maps contain information about static obstacles and landmarks for usage with localization algorithms. Automotive industry forces to have uniform representations [23] for interfaces between sensors, mapping and functionality. One representation is using fences surrounding the free space of sensor readings, as this is the only area of interest. Maps would be stored in a similar way. Correlating these maps with every fence given by sensor information leads to a position estimate of the vehicle.

In a similar way landmarks are used [45]. Therefore the landmarks are labeled in advance. Possible landmarks are poles, trees, edges of houses or any other objects giving distinct measurements with the used sensor types. A test road with detected landmarks is shown (Figure 1.13) as top down view.

![Figure 1.13: The test route along with the mapped landmarks. The red arrow indicates the starting position and the driving direction. The numbers indicate situations which are discussed in [45]. © 2014 IEEE](image)

Some LiDAR sensors provide an intensity feedback which gives an additional information for doing localization. A very promising method [26] first generates continuous images of the road surface. The images represent the laser beam reflection intensity with a resolution of 5 cm × 5 cm per pixel where road markings are clearly visible due to their high reflection coefficient. Refinement of the map is done after recording. For localization the current sensor readings are correlated with the provided map. Position is estimated with a precision of 10 cm to 30 cm in absence
of GNSS. Another advantage is that lane markers are already considered in the position estimation. When using different sensors the intensity values must be well calibrated, which is a challenging task. After all it is difficult to keep map data or other landmarks up to date at all time due to a continuously changing environment.

1.2.6 Parallel Computing

Occupancy grid maps have multiple thousand cells, depending on their resolution. During sensor fusion the same mathematical operations have to be applied on each cell independently. These operations may be performed in parallel. Nvidia provides a general purpose parallel computation framework (CUDA [34]), which makes it easier to perform calculation on a GPU. For operations on point clouds an average GPU with around 400 cores can improve the speed compared to a Intel i7 Dual Core by a factor of 15 [4]. The experiments are carried out using the Robot Operating System (ROS) [35] and a Velodyne LiDAR, which is the setup used in this thesis. A comparison of the consumed power shows minor differences when using CPU or GPU.

Using a GPU for occupancy grid mapping [20] leads to grid update times of a few milliseconds. The authors give a summary of all the computational intensive parts in occupancy grid mapping and provide solutions or point out other literature to overcome the issues. One specific problem occurs when mapping polar grids to Cartesian grids. As in Figure 1.14 a direct mapping is impossible as the projection is non linear. Interestingly this problem is well known as texture mapping in the computer graphics domain. For that reason GPUs are designed to accelerate texture mapping.

![Texture T in Polar space and its target polygon P in Cartesian space.](image)

Figure 1.14: Texture T in Polar space and its target polygon P in Cartesian space. The beam width is considered to be 10° [20] © 2010 IEEE
1.3 Major Contribution

- Whole sensor fusion process (calibration, visual odometry and fusion) can be done with freely mounted Velodyne sensors. This increases the freedom for experimenting with different field of views for the sensors. Usually authors use one high resolution Velodyne sensor mounted horizontally on top of the car.

- Real time model free dynamic object tracking using 3D laser scanners and use of Dempster Shafer conflict measure based on a Hybrid BOF [32].

- Improving quality and robustness of tracking result, introducing a way of downgrading unreasonable solutions, which would cause collisions. As the goal of autonomous driving is to prevent collisions it could be a valid assumption in the future.

1.4 System Overview

All experiments are carried out on the RCV built by the Integrated Transport Research Lab (ITRL) at KTH Royal Institute of Technology (KTH). The RCV realizes a research-based fully electric drive-by-wire concept vehicle, that is used to validate and demonstrate current and future research results. Initially it was built to perform different vehicle dynamics tests, as it allows to test different geometry and drive train configurations. It uses four electrical in-wheel motors, which allow to choose between front or rear two wheel drive and four wheel drive mode. Every wheel’s steering angle can be defined by software, which allows different steering modes such as front/rear wheel steering and crawling. As the design is modular to include research from several disciplines it is possible to integrate sensors required for autonomous driving tasks. The integration is not described in detail, as it is not the main focus of this thesis. Figure 1.15 shows the vehicle and indicates the positions of different sensor types. Global Positioning System (GPS), the acceleration sensor (IMU) and odometry sensors (wheel speed and steering angle) in all four wheels are used by the position estimation, while two LiDARs are used to sense the surrounding. The Velodyne VLP-16 LiDARs can be mounted freely on the rails of the vehicle. Other than shown in Figure 1.15, both LiDARs are mounted on the front top corners of the roll cage during the experiments throughout this thesis. Specifications for the vehicle are found in Table 1.1.
Figure 1.15: The Research Concept Vehicle of the Integrated Transport Research Lab (KTH Royal Institute of Technology) with the sensors or mounts for the sensors indicated.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>In-wheel electric motors</td>
<td>4</td>
</tr>
<tr>
<td>Individual steering actuators</td>
<td>4</td>
</tr>
<tr>
<td>Weight</td>
<td>400 kg</td>
</tr>
<tr>
<td>Drivetrain</td>
<td>17 kW</td>
</tr>
<tr>
<td>Lithium battery(^1)</td>
<td>52 V / 42 Ah</td>
</tr>
<tr>
<td>Full dynamic driving range(^1)</td>
<td>30 min</td>
</tr>
<tr>
<td>Top speed</td>
<td>70 km h(^{-1})</td>
</tr>
<tr>
<td>Steering angle</td>
<td>±25°</td>
</tr>
<tr>
<td>Camber angle</td>
<td>−15° to 10°</td>
</tr>
</tbody>
</table>

Table 1.1: Research Concept Vehicle specifications

\(^1\)The battery pack was replaced by a more powerful one right after the presented experiments were conducted. The future battery pack has a capacity of 15 kWh.
Chapter 2

Occupancy Grid Mapping

Occupancy grid mapping approach introduced by [11] becomes a very popular method for doing mapping and localization in robotics field. It reduces the complexity to a two dimensional problem (or three dimensional problem for aerial applications) of fixed resolution. With the given resolution one can calculate the computational complexity for operations on the grid, as soon as the size and the mathematical operations are known for the application. In this way a proof for a certain real time condition is given. Another advantage is the unified way of representing obstacles in a model free way and independent of the sensor types. For the current sensor fusion involving two identical LiDARs it is not important to have a sensor independent representation, however it becomes a crucial feature for future use of the sensor fusion framework when adding additional sensors. When using two identical sensors to show a unified framework for any kind of obstacle detecting sensor type it is important to keep a strict border between the sensor model and the sensor independent fusion framework.

Probabilistic statements about occupancy are obtained from the sensor model, which should be fused in a uniform way. Two popular frameworks, the Bayesian and the Dempster Shafer sensor fusion framework exist, where the Bayesian approach maintains one probability value and the Dempster Shafer framework could be extended by additional states for which beliefs are considered. Dempster Shafer is chosen instead of the commonly used Bayesian framework because it gives more freedom due to the possibility to add as many states as required and as seen in subsection 1.2.3 could be useful to detect conflicts between different sensors. Again when using the exact same sensor type it is unlikely that the measurements differ a lot. First the basic Dempster Shafer sensor fusion framework is introduced, followed by the sensor model for the Velodyne VLP-16 LiDAR and a decay factor. In the end an interpretation is given for the resulting DS grid, which can be used by the path planner.
2.1 Dempster Shafer Sensor Fusion

In this introduction to the Dempster Shafer theory the focus is given to the basic rules of combination and belief assignment used for the particular sensor fusion application. Dempster Shafer is one method to give a sample implementation on the RCV and provides obstacle maps on real world data sets. For a deeper insight to the theory itself the reader is referred to the original publications [9] [39] as a comparison of different sensor fusion algorithms is not the purpose of this work.

The theory is based on a mutually exclusive set of $N$ world states

$$\Theta = \{w_1, w_2, \ldots, w_N\}, \quad (2.1)$$

which is called a Frame of Discernment (FOD). Instead of assigning belief values to world states of $\Theta$, every possible subset of $\Theta$ (power set $2^\Theta$) has it’s own belief value. Take $\Theta = \{w_1, w_2, w_3\}$ as an example which leads to a power set

$$2^\Theta = \{\emptyset, w_1, w_2, w_3, \{w_1, w_2\}, \{w_1, w_3\}, \{w_2, w_3\}, \{w_1, w_2, w_3\}\}, \quad (2.2)$$

where $\emptyset$ is the empty set. Belief values or mass functions are assigned to each member of the power set $2^\Theta$ and are comparable with the Basic Probability Assignment (BPA) in Bayesian theory. However DS implicitly includes mass functions

$$m : 2^\Theta \rightarrow [0, 1] \quad (2.3)$$

for the Unknown state ($\{w_1, w_2, w_3\}$ in the given example) and every other combination of exclusive states. Following conditions apply for the mass function $m(A)$ with $A \in 2^\Theta$ being any subset of the power set:

$$\sum_{A \in 2^\Theta} m(A) = 1 \quad m(\emptyset) = 0 \quad (2.4)$$

Dempster Shafer theory uses additional measures to state the knowledge as a mass of a subset is not necessarily related to a classical probability measure. Three measures Belief ($Bel(X)$), Plausibility ($Pl(X)$) and Conflict ($Con(X, m_1, m_2)$) are defined, where $X$ is the subset to be evaluated:

$$Bel(X) = \sum_{A \in 2^\Theta | A \subseteq X} m(A) \quad (2.5)$$

$$Pl(X) = 1 - \sum_{A \in 2^\Theta | A \cap X = \emptyset} m(A) \quad (2.6)$$

$$Con(X, m_1, m_2) = \log \left( \frac{1}{1 - (m_1 \oplus m_2)(\emptyset)} \right) \quad (2.7)$$
2.2 Sensor Model

The Velodyne LiDARs provide a point cloud of the scanned environment. At this step the point clouds of the two Velodynes are already transformed into a common global reference frame. This is achieved using a pose estimator on the vehicle [15], which is integrated according to the system architecture in appendix D. Dempster Shafer sensor fusion requires belief masses which represent the current sensor measurement. Dempster Shafer cedes the user to define the FOD. It is straight forward to define the FOD with OCCUPIED (OCC) and EMPTY (EMP), which leads to a FOD, $2^\Theta$ of

$$2^\Theta = \{\emptyset, OCC, EMP, \{OCC, EMP\}\}, \quad (2.11)$$

where $\emptyset$ denotes the empty set. State UNKNOWN is implicitly referred to the set $\{OCC, EMP\}$, as it says it is occupied or empty.
2.2.1 Mass Assignment for Accurate Position Estimation

All further calculations assume grid cells as statistically independent of other cells like in [7], which is not true for any case. For example cells behind an obstacle are never hit and therefore dependent on the occupied cell [31]. In this model the dependency is not used to get more information, instead information is gathered in a conservative way, so that the assumption holds. Therefore Dempster Shafer parameters like mass belong to one grid cell and every calculation step has to be executed for all grid cells. In each grid cell four cases occur:

1. No points in grid cell
2. All points in grid cell belong to the ground
3. All points in grid cell belong to obstacles
4. Points in grid cell belong to ground and obstacles

Decisions between obstacles and ground points are made using a threshold height above ground. Ground level can be determined using methods mentioned in subsection 1.2.2. Case four, that both ground points and obstacle points exist in one grid cell, is treated equally to case three, which follows the conservative rule as it is better to detect a half occupied cell as OCCUPIED rather than EMPTY.

For every case a BPA should be defined, depending on the number of ground points $n_G$ and the number of obstacle points $n_O$ per grid cell. The proposed model [7] calculates the BPAs by first defining a false alarm probability $\alpha_{FA} = P(C = F|z_1)$, where probability of cell $C$ is free ($F$) is estimated given one obstacle impact $z_1$. Supposing that errors are independent, the total false alarm probability in one cell given $n_O$ obstacle points are detected in this cell should be $P(C = F|z_1, z_2, \ldots, z_{n_O}) = \alpha_{FA}^{n_O}$. Thus the probability of OCCUPIED ($O$) can be represented as $P(C = O|z_1, z_2, \ldots, z_{n_O}) = 1 - \alpha_{FA}^{n_O}$. Based on the same methodology, for FREE cells, the missed detection probability $\alpha_{MD} = P(C = O|\Delta)$, where $\Delta$ represents no above ground impact is returned to the sensor. Assuming $n_G$ ground points are detected in this cell, the total missed detection probability is $\alpha_{MD}^{n_G}$. Thus the probability of FREE is represented as $1 - \alpha_{MD}^{n_G}$. Based on the principle, the BPA assignment for the first three cases:

1. For an **UNKNOWN** cell:
   $$m(O) = 0, m(F) = 0, m(\Omega) = 1, m(\emptyset) = 0$$

2. For a **FREE** cell:
   $$m(O) = 0, m(F) = 1 - \alpha_{MD}^{n_G}, m(\Omega) = 1 - m(F), m(\emptyset) = 0$$

3. For an **OCCUPIED** cell:
   $$m(O) = 1 - \alpha_{FA}^{n_O}, m(F) = 0, m(\Omega) = 1 - m(O), m(\emptyset) = 0$$
2.2. SENSOR MODEL

2.2.2 Parameter Derivation

In the proposed sensor model [7] the false alarm - and missed detection probabilities $\alpha_{FA} = 0.15$ and $\alpha_{MD} = 0.66$ are given. The false alarm probability is estimated empirically as it is depending on sensor noise and highly related to the environment and the way of mounting the sensor. False alarm is hardly ever noticed in experiments carried out, which means that a value of 15% seems very high under clear weather conditions. For the missed detection probability a polar grid is used for the BPA and the sensor is assumed to be mounted parallel to the ground. Then the beamwidth of the laser ($0.17^\circ$) is compared to the angular resolution of the polar grid ($0.5^\circ$), which leads to $\alpha_{MD} = 1 - \frac{0.17^\circ}{0.5^\circ}$. In this work a polar grid is not applicable in the same way, as the sensors are potentially mounted slightly tilted and shifted. The solution is to transform the measurement points directly to a global Cartesian grid, making the estimation of the missed detection probability more complicated. A similar approach is comparing the area covered by the laser beam on the ground with the area of the grid cell

$$\alpha_{MD} = 1 - \frac{A_{Beam}(l)}{A_{Cell}},$$

(2.12)

where $A_{Beam}$ is the area covered by laser beam on the ground, $A_{Cell}$ is the area of one grid cell and $l$ the distance from laser origin to ground.

The beam area is an ellipse with half axis length $a$ and width $b$ and requires a given sensor height $h$ above ground:

$$A_{Beam} = \pi \cdot a(l) \cdot b(l)$$

$$a(l) = \frac{1}{2} \cdot l \cdot \sin(\phi(l) + 0.17^\circ) - \sin(\phi(l) - 0.17^\circ))$$

$$b(l) = l \cdot \sin(0.17^\circ)$$

$$\phi(l) = \arccos\left(\frac{h}{l}\right)$$

(2.13)

Figure 2.2 shows the missed detection probability $\alpha_{MD}$ over the distance from the sensor to any ground point measured along the ground plane. Interestingly this means the further away the more reliable ground is detected. However for closer distances more points reach the same cell which compensates the effect. For the region of interest (distances 2 m - 30 m) a linear function approximates the coefficient $\alpha_{MD}$ accurately. In the implementation a mean value is chosen due to difficulties to access the distance at this software part.

2.2.3 Fault Tolerant Sensor Model

In many cases, especially when using highly accurate sensors like LiDARs, the position estimation introduces a magnitude bigger errors than the sensor itself. Therefore the uncertainty of the position estimation should be considered when fusing
the sensor measurements. Position estimation algorithms often generate probability distributions [15] and forward the mean value. The general case is to get a probability distribution \( P(x, \mu) \), where \( x \) is some position, \( \mu \) the mean value of the position estimate given in map frame coordinates and \( \int \int P(x, \mu)dx = 1 \) must hold. \( P(x, \mu) \) gives the probability of a point detected at position \( \mu \) being at position \( x \).

Some further definitions are required:

With these definitions the sensor model from subsection 2.2.1 can be redefined. The number of obstacle or ground points per cell will change. Instead of counting points within one cell, theoretically all points influence the number of points in a given grid cell. Equation 2.14 gives the continuous values for number of obstacle or ground points per grid cell \( C \).

\[
n_{O/G} = \int \int \sum_{z_i \in z_{O/G}} P(x, z_i)dx
\] (2.14)

This equation has to be evaluated for every grid cell, which is not practicable on a real system. Instead discrete integration of distributions from a specific number of neighboring cells would be used to limit the calculation time.

Matlab simulations give insight to the resulting effects such an extension to the proposed sensor model has. For visualization a one dimensional evidential grid is used, where the Occupied belief mass is color coded for each cell and plotted over time frames (Figure 2.3). To simulate an erroneous position estimation, the generated points are biased with a random normal distributed offset around the true position of the obstacle. After applying a threshold the estimated obstacle is visualized in Figure 2.4. First all calculations are done without the fault tolerant
method and then the fault tolerant integration over the probability distribution is applied for the same input data.

![Belief mass comparison over time between normal and fault tolerant evidential sensor model](image1)

**Figure 2.3:** Belief mass comparison over time between normal and fault tolerant evidential sensor model

![Occupied cell comparison over time between normal and fault tolerant evidential sensor model](image2)

**Figure 2.4:** Occupied cell comparison over time between normal and fault tolerant evidential sensor model, where yellow area is detected as occupied (ground truth is at $x = 2.5\, \text{m}$ and covers an area of 10 cm)

The advantage of the fault tolerant solution is a more uniform distribution of obstacles detected compared to the standard sensor model. But the problem is the introduced delay making the obstacle invisible for the first 20 frames. In the example the variance for the position drift was set to 0.5 m, which is a very high value for the short term position drift. The position drift for the given limitation of measuring distances up to 30 m stays in the range of 10 cm and for such a scenario the improvement is not worth the effort anymore. Only for very large measurement distances the fault tolerant sensor model could be useful, as small heading errors result in position errors in the magnitude of multiple grid cell dimensions.
2.3 Occupancy Grid Update

In the static world the information gathered by the sensor fusion is theoretically valid forever. However, even if there is no dynamic object the position estimation still introduces an error over time. A decay factor $\beta_{\text{Decay}}$ is introduced to let the knowledge of the grid going towards the unknown state. Therefore Equation 2.15 is applied after each sensor fusion cycle.

\[
\begin{align*}
  m(O) &= m(O) \cdot \beta_{\text{Decay}} \\
  m(F) &= m(F) \cdot \beta_{\text{Decay}} \\
  m(\Omega) &= 1 - \beta_{\text{Decay}} - \beta_{\text{Decay}} \cdot m(\Omega)
\end{align*}
\]  

The decay factor is empirically set to $\beta_{\text{Decay}} = 0.98$, but could be derived if the position drift is known. For a given position drift per time instance, the grid should get the unknown state as soon as the error reaches the dimension of the cells.

2.4 Obstacle Map

After combining previous section 2.1 to section 2.3, a global DS grid is available and through Equation 2.5 and Equation 2.6 the belief $\text{Bel}(X)$ and the plausibility $\text{Pl}(X)$ can be accessed. A framework to access all grid values is provided to other software parts like the path planner.

The specific path planner used, requires a binary grid saying occupied or not occupied for each cell. Therefore a rule is necessary, which evaluates each cell and assigns a binary occupancy state. It is important to understand how the path planner behaves on certain situations presented by the grid. In this case it tries to avoid every obstacle present in the current map and assumes all the other space to be free. As there is no information if the space is free or unknown, it assumes unknown space becomes known space when approaching it and in case of obstacles appearing the vehicles reaction time is fast enough to avoid them.

For the binary obstacle grid map it is desired that obstacles only appear if OCCUPIED is more likely than UNKNOWN or FREE. Otherwise it could prevent the path planner from exploring unknown space. The cell state $C$ either being OCCUPIED ($O$) or NOT OCCUPIED ($\neg O$) can be expressed by Equation 2.16.

\[
C = \begin{cases} 
  O, & \text{if } OCC = \arg\max_{X \in 2^\Theta} \text{Bel}(X) \\
  \neg O, & \text{otherwise}
\end{cases}
\]

The solution, caused by the specific path planner used, has the drawback of not being able to increase the velocity as there is no possibility to know that space is empty for sure. Instead the UNKNOWN and EMPTY beliefs are both treated as $\neg O$. Driving this way assumes the sensor range being high enough and that obstacles are detected reliably and fast enough to react safely, limiting the maximum velocity.
Chapter 3
Dynamic Object Tracking

In the previous chapter 2 a sensor fusion technique is described, which assumes the environment to be static. This assumption is abolished now, to keep track of dynamic objects and simultaneously distinguish between static and dynamic objects. It is desirable to avoid specialization on any sensor type, in order to allow the integration of future sensor types without changing the dynamic object tracker. One solution is the previously introduced occupancy grid mapping scheme, which allows adding additional sensor types, by replacing the sensor model from section 2.2. The idea is to assign a velocity distribution to each grid cell independently. Bayesian Occupancy Filter (BOF) solves that task, but usually requires discretization in both space and velocity, resulting in a four dimensional grid representation [6]. The number of cells $n$ is therefore given in Equation 3.1, which is proportional to the computational complexity or the memory usage and can become unmanageable quickly.

$$n = \frac{h \times w \times 4 \times vMax_x \times vMax_y}{s_{Cell}^2 \times \Delta v^2}$$ (3.1)

$h$ ... Grid height in m

$w$ ... Grid width in m

$s_{Cell}$ ... Cell dimension in m

$vMax_i$ ... Maximum i-velocity in m s$^{-1}$

$\Delta v$ ... Velocity resolution in m s$^{-1}$

This challenging problem is tackled by the hybrid version of the BOF introduced by [32] as the Hybrid Bayesian Occupancy Filter (HBOF), where the strict discretization of velocity is replaced by continuous velocities. A single velocity vector can not describe a velocity distribution within one cell. The idea of HBOF is to assign many velocity vectors to dynamic cells, representing the velocity distribution. In the HBOF the whole grid’s overall number of velocity vectors, also called particles, stays the same to guarantee the real time capability. These particles are dynamically allocated to dynamic cells to avoid unnecessary tracking of static cells. In [32] the authors of HBOF claim to reduce the average number of velocity samples from...
900 to 2 per cell in typical scenarios. A typical scenario is considered to include obstacles with velocities of up to 130 km h$^{-1}$.

The HBOF is divided into dynamics identification (section 3.1), the Bayesian filter on cell level (section 3.2) and the particle management (section 3.3). Afterwards an improvement regarding the convergence time is introduced based on a collision free environment assumption in section 3.4.

### 3.1 Dynamics Identification

Potential dynamic cells have to be identified first, for being able to assign dynamic belief to a cell. Otherwise a lot of unnecessary particle assignments would occur. Dempster Shafer (DS) theory introduces a logarithmic conflict measurement given in Equation 2.7. The non log conflict mass $(m_1 \cap m_2)(\emptyset) = K_{\text{App}} + K_{\text{Disapp}}$ can be divided in an appearing conflict mass $K_{\text{App}}$ and a disappearing mass $K_{\text{Disapp}}$. Using the *conjunction* formula from Equation 2.9 and the assumption that the mutually exclusive set $\Theta$ only contains *free* ($F$) and *occupied* ($O$), the equation [31] leads to equations

\[
K_{\text{App}} = m_t(O) \cdot m_{t-1}(F) \quad \text{and} \\
K_{\text{Disapp}} = m_t(F) \cdot m_{t-1}(O),
\]

where mass values $m_t(X)$ represent the DS belief mass at time $t$ of cell state $X$. The part of the conflict value, where some obstacle is disappearing $K_{\text{Disapp}}$ is ignored, because empty spaces can not have a velocity estimate. It is not possible to give the dynamic belief to the neighboring cells in the direction of movement, as cells might be skipped for high velocities due to the limited sensor frame rate. On the other hand the appearing beliefs $K_{\text{App}}$ only occur on the cells where the object is currently located. Later on in section 3.2 a belief of the cell being dynamic $m(v \neq 0)$ is introduced and whenever $K_{\text{App}} > m(v \neq 0)$ new dynamic probabilities with unknown velocities are assigned to the cell.

### 3.2 Dynamic Object Filtering on Cell Level

In this section the main part of the Hybrid Bayesian Occupancy Filter (HBOF) [32] is derived to do dynamic object tracking on a cell basis, which allows to track objects of any shape and postpones the association of measurements and objects to a later stage. Postponing the object association allows to use the velocity estimation for more robust object clustering. As the name of the Hybrid Bayesian Occupancy Filter algorithm indicates, it is based on the Bayesian theory and uses a probability distribution $P(X)$ for random variable $X$. The filter implementation is done using the Bayesian theory as described in [32], but it turns out (section 3.2.2) that the Dempster Shafer theory is more suitable to the filtering algorithm. All used variables
3.2. DYNAMIC OBJECT FILTERING ON CELL LEVEL

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$C$</td>
<td>Index, that identifies each 2D cell.</td>
</tr>
<tr>
<td>$A$</td>
<td>Index, that identifies each possible antecedent cell.</td>
</tr>
<tr>
<td>$O$</td>
<td>Occupancy state of the cell at the current time. It’s possible values are {EMP, OCC}.</td>
</tr>
<tr>
<td>$O^{-1}$</td>
<td>Occupancy state of the antecedent cell at the previous time.</td>
</tr>
<tr>
<td>$V$</td>
<td>Speed of the cell at the current time. It’s possible value is a 2D vector in $\mathbb{R}^2$.</td>
</tr>
<tr>
<td>$V^{-1}$</td>
<td>Speed of the antecedent cell at the previous time.</td>
</tr>
<tr>
<td>$Z$</td>
<td>Sensor measurement.</td>
</tr>
<tr>
<td>$P(A)$</td>
<td>Is the distribution over all possible antecedent of the cell. It is chosen to be uniform as the cell is considered reachable from all the antecedents with equal probability.</td>
</tr>
<tr>
<td>$P(O^{-1}, V^{-1}</td>
<td>A)$</td>
</tr>
<tr>
<td>$P(O, V</td>
<td>O^{-1}, V^{-1})$</td>
</tr>
<tr>
<td>$P(O</td>
<td>O^{-1})$</td>
</tr>
<tr>
<td>$P(V</td>
<td>V^{-1}, O)$</td>
</tr>
<tr>
<td>$P(C</td>
<td>A, V)$</td>
</tr>
<tr>
<td>$P(Z)$</td>
<td>Is the distribution over the sensor measurement value.</td>
</tr>
</tbody>
</table>

Table 3.1: Variable definitions for HBOF [32]

and probabilities are defined in Table 3.1 as proposed by [32]. With these variables, the goal shall be described, which is the probability distribution of occupancy state and velocity with respect to the sensor observations and a certain cell $P(O, V|Z, C)$. 
3.2.1 Problem Definition

The goal is to find $P(O, V | Z, C)$ as a combination of accessible, discrete probability distributions. A joint distribution can directly be written as in Equation 3.3.

$$P(C, A, O, O^{-1}, V, V^{-1}, Z) = P(A) \cdot P(O^{-1}, V^{-1} | A) \cdot P(O, V | O^{-1}, V^{-1}) \cdot P(C | A, V) \cdot P(Z | O, V, C)$$

(3.3)

The goal probability distribution $P(O, V | Z, C)$ can be derived from the joint distribution using the definition of conditional densities $P(X | Y) = \frac{P(X \cap Y)}{P(Y)}$ in the discrete form and leads to

$$P(O, V | Z, C) = \sum_{A, O^{-1}, V^{-1}} P(C, A, O, O^{-1}, V, V^{-1}, Z) \cdot \sum_{A, O^{-1}, V^{-1}} P(C, A, O, O^{-1}, V, V^{-1}, Z),$$

(3.4)

where the sum covers the whole set of combinations between the given random variable indexes. In Equation 3.4 the denominator is a scale factor and not considered for further derivations. By using Equation 3.3 it can be rewritten as follows:

$$P(O, V | Z, C) \propto P(Z | O, V, C) \cdot \sum_{A, O^{-1}, V^{-1}} P(A) \cdot P(O^{-1}, V^{-1} | A) \cdot P(O | O^{-1}) \cdot P(V | O^{-1}) \cdot P(C | A, V)$$

(3.5)

Now consider the probability of one specific cell $c$, to figure out the filtering process.

3.2.2 Filtering

$P(A)$ has no effect on the proportion, as a uniform distribution is assumed, because there is no way to access such an information in our scenario. A map of different road areas could provide such a probability of having an antecedent cell.

The term $P(C | A, V)$ results in a binary value, which is one if $x_a + v \cdot \delta t \in c$ is fulfilled and zero otherwise, where $x_a$ is the antecedent position and $v$ the velocity.

In the actual implementation it is taken into account by the data structure implicitly, because all the antecedent velocities are assigned to the future cell objects. This reallocation of the antecedent cells in every cycle is done right after the cell particles are propagated with the respective velocities $v$.

In order to keep the equations in a compact form, define

$$\alpha(o, v) = \sum_{A, O^{-1}, V^{-1}} P(O^{-1}, V^{-1} | A) \cdot P(o | O^{-1}) \cdot P(v | o, V^{-1})$$

(3.6)

which is the sum of Equation 3.5 considering the simplifications described before.
In the static case the velocity is zero and the antecedent cell is equal to the current cell by definition. The probability \( P(v|a, V^{-1}) = 1 \) for the relevant cases of the static cell definition \( (v = 0 \text{ and } V^{-1} = 0) \), because otherwise the probability distribution \( P(O^{-1}, V^{-1}|A) = 0 \) and therefore \( P(v|a, V^{-1}) \) can be ignored. For alpha occupied \( (\alpha(OCC, 0)) \) and alpha empty \( (\alpha(EMP, 0)) \) in the static case following Equation 3.7 describes the update equation for the current time step given the values from the previous time.

\[
\begin{align*}
\alpha(OCC, 0) &= p^{-1}_{OCC,0} \cdot (1 - \epsilon) + p^{-1}_{EMP,0} \cdot \epsilon \\
\alpha(EMP, 0) &= p^{-1}_{OCC,0} \cdot \epsilon + p^{-1}_{EMP,0} \cdot (1 - \epsilon)
\end{align*}
\] (3.7)

The probability a cell being occupied or empty depending on the previous state \( P(o|O^{-1}) \) is considered through the constant values \( (1 - \epsilon) \) for the case the cell keeps the occupancy state or \( \epsilon \) the (usually more unlikely case) that it changes the state. See Table 3.1, where the transition matrix \( T \) is defined, taking into account the approximation errors. These factors are comparable to the decay factor introduced by the DS theory in section 2.3.

In the dynamic case only cells with occupancy state OCCUPIED are considered, as an EMPTY cell is not allowed to have any velocity. The probability of a certain velocity \( v \), given the previous velocity and that the cell is OCCUPIED \( P(v|OCC, V^{-1}) \) is implicitly considered, by the way velocity particles are propagated. For each velocity particle \( v_i \), the velocity vector \( v_i \) and the position vector \( x_i \) are updated as described by Equation 3.8

\[
\begin{align*}
v_i &= v_i^{-1} + \sigma \\
x_i &= x_i^{-1} + v_i,
\end{align*}
\] (3.8)

where \( \sigma \sim \mathcal{N}(0, \Sigma) \) is the zero mean normal distribution of the acceleration and \( \Sigma \) is the two dimensional covariance matrix. As every velocity particle is added with one discrete value of the acceleration distribution, the particle filter has to make sure that the number of particles with a certain velocity is according to the likelihood of a cell having that velocity. For a theoretical infinite number of particles, the exact velocity distribution would be propagated. The Particle Management introduced in section 3.3 takes care that the overall particle number stays constant and particles are distributed along the grid according to their likelihoods and weighted with a weight \( w_i \).

For each particle \( v_i \), which falls in cell \( c (x_i \in c) \), the weight of the particle at the previous time is \( w_i^{-1} = P(O^{-1} = OCC, V = v_i^{-1}|A = a_i) \), where \( a_i \) is the previous cell of the particle \( (x_i^{-1} \in a_i) \). Similarly to the static case the

\[
\alpha(OCC, v_i) = w_i^{-1} \cdot (1 - \epsilon),
\] (3.9)
where $P(O = OCC|O^{-1} = OCC) = (1 - \epsilon)$.

The final step is to include the new sensor measurement, given as the distribution $P(Z|O, V, C)$, which completes Equation 3.5, which is the initially introduced goal distribution $P(O, V|Z, C)$. An intermediate step is necessary to do normalization and assure, that the sum of static and dynamic probabilities per cell equals one. The not normalized goal distribution is given by the following $\beta(o, v)$ in Equation 3.10:

$$
\begin{align*}
\beta(OCC, 0) &= P(Z|O = OCC, V = 0, C = c) \cdot \alpha(OCC, 0) \\
\beta(EMP, 0) &= P(Z|O = EMP, V = 0, C = c) \cdot \alpha(EMP, 0) \\
\beta(OCC, v_k) &= P(Z|O = OCC, V = v_k, C = c) \cdot \alpha(OCC, v_k)
\end{align*}
$$

With the normalization factor $d = \beta(OCC, 0) + \beta(EMP, 0) + \sum_k \beta(OCC, v_k)$ the final result is given in Equation 3.11.

$$
\begin{align*}
P(O = OCC, V = 0|Z, C) &= \frac{\beta(OCC, 0)}{d} \\
P(O = OCC, V = v_i|Z, C) &= \frac{\beta(OCC, v_i)}{d}
\end{align*}
$$

The normalization is necessary, because the denominator of Equation 3.5 has been neglected so far, because it does not affect the ratios between the betas. Interestingly the fusion process tends to behave like the DS update rule where different beliefs are always redistributed, if the overall sum is not equal to one.

### 3.3 Particle Management

The particle management assures that the fixed number of particles is distributed over the grid according to the probability distributions as mentioned in subsection 3.2.2 before. Algorithm 1 gives an overview of the HBOF algorithm and in which order the different functions are called, where `normalizeParticleWeights()` and `drawParticles()` are the main parts of the particle management.

This algorithm runs with a fixed frequency, which should be greater or equal to the highest sensor update rate. In case of the used Velodyne sensors, the algorithm runs with 10 Hz.

#### 3.3.1 Draw Particles

The desired result of the Draw Particles function is a new set of particles, which represents a velocity distribution for each dynamic cells. A straightforward approach would be to assign a certain number of particles to each cell with a certain dynamic threshold. However, the real-time capability can not be guaranteed in that way. Instead the overall number of particles must stay fixed. To achieve a constant number of particles, the Draw Particles function first chooses a cell and afterwards
Algorithm 1 HBOF Main Functions

\begin{algorithmic}
  \State C is the array of grid cells
  \State P is the array of particles
  \State P_i is the array of particles belonging to cell \( c_i \)
  \Procedure{applyMotionModel}{P}
  \State \Procedure{reallocateParticles}{C, P}
  \For{c_i in C}
  \State computeBetaStatic(c_i)
  \For{v_k in P_i}
  \State computeBetaDynamic(c_i, v_k)
  \EndFor
  \State normalizeDistributions(c_i)
  \EndFor
  \State \Procedure{drawParticles}{C}
  \State \Procedure{normalizeParticleWeights}{C}
\end{algorithmic}

picks a particle within this cell. When choosing a cell \( c_i \), the dynamic probability \( P(O = OCC, V \neq 0|Z, C) \) of the cell is considered, which means cells with a higher probability of being dynamic are allocated with more particles, than other ones. In the second step, particles within the selected cell are duplicated according to the assigned probability \( P(O = OCC, V = v|Z, c_i) \) that the chosen cell \( c_i \) is moving with the velocity \( v \) of the given particle. Particles drawing is repeated until the maximum number of particles is reached.

3.3.2 Normalization of Particle Weights

Like in the previous subsection 3.2.2, the sum of all probabilities describing one cell state must be equal to one. Following Equation 3.12 must hold with the particle weights \( w_k \).

\[ P(O = OCC, V = 0|Z, C) + P(O = EMP, V = 0|Z, C) + \sum_k w_k \frac{1}{n_{ParticlesMax}} = 1 \]  

(3.12)

It is simple to normalize the probabilities and the weights accordingly, but sum of all weights within a cell after drawing the new particles, is highly depending on the number of particles available per cell. That number of particles is dependent on the maximum number of particles for the whole grid cell. Therefore the weights are scaled with a tuning parameter before normalization, which is dependent on the maximum particle number (for the scenarios evaluated, \( n_{ParticlesMax} \) is used).
3.3.3 Unknown Velocities

When new particles are added by the detection of dynamic cells, the velocity is unknown, which is marked with a special flag. In case such a velocity particle is drawn, the initial velocity has to be guessed. The velocity vector is randomly set according to a uniform distribution between \( \mathbf{v} = \left[-[v_{\text{Max}_x}, v_{\text{Max}_y}]^T; [v_{\text{Max}_x}, v_{\text{Max}_y}]^T\right]. \)

3.4 Colliding Obstacles

So far no restriction is given which suppresses non-meaningful tracking results. Two colliding obstacles are a non-meaningful situation, especially in future scenarios, when the majority of vehicles on the streets are moving autonomously and have to be safe. The goal is to use this knowledge to achieve a faster convergence time towards the true velocity. Colliding obstacles are crossing velocity particles of the HBOF having an intersection within a given time span. A straightforward approach is to punish particles, which cross many other particle velocity vectors. But at the same time valid configurations, for example particles belonging to the same object, should not be punished by a downgrading factor. These particles within one object are very close to each other and therefore it is most likely to get crossing particles there. In the next subsection 3.4.1 it is proofed, that for any two particles within a rigid object, their velocity vectors do not cross each other at the same time.

3.4.1 Proof of Collision Free Rigid Objects

Rigid non-holonomic objects like vehicles on the street, can in general turn around a specific point with turn rate \( \omega \) and move towards a certain direction with absolute velocity \( v \). Set the origin to the rotation point of the car and choose two points \( (P^{(1)} \) and \( P^{(2)} \)) within the vehicle like shown in Figure 3.1.

\[
\begin{align*}
\mathbf{v}^{(i)} = & \begin{bmatrix} v - \omega \cdot r_i \cdot \sin(\phi_i) \\ \omega \cdot r_i \cdot \cos(\phi_i) \end{bmatrix} 
\end{align*}
\] (3.13)
For two points \( x^{(i)} \) with given velocity \( v^{(i)} \) the equation

\[
x^{(1)} + v^{(1)} \cdot t = x^{(2)} + v^{(2)} \cdot t \tag{3.14}
\]
describes the colliding point at time \( t \). A few constraints are necessary before solving the equation:

- \( r_1, r_2, \phi_1, \phi_2, v, \omega \in \mathbb{R} \)
- \( r_1, r_2 > 0 \)

Solving Equation 3.14 for \( t \) using the Matlab symbolic solver gives following result:

\[
t = \begin{cases} 
\mathbb{C} & \text{if } r_1 \cdot \cos(\phi_1) = r_2 \cdot \cos(\phi_2) \land \\
& r_1 \cdot \sin(\phi_1) = r_2 \cdot \sin(\phi_2) \\
\left\{ \frac{r_1 \cdot \cos(\phi_1) - r_2 \cdot \cos(\phi_2)}{r_1 \cdot \omega \cdot \sin(\phi_1) - r_2 \cdot \omega \cdot \sin(\phi_2)} \right\} & \text{if } \omega \cdot (r_1 \cdot \cos(\phi_1) - r_2 \cdot \cos(\phi_2))^2 + \\
& \omega \cdot (r_1 \cdot \sin(\phi_1) - r_2 \cdot \sin(\phi_2))^2 = 0 \land \\
& r_1 \cdot \cos(\phi_1) \neq r_2 \cdot \cos(\phi_2) \land \\
& r_1 \cdot \sin(\phi_1) \neq r_2 \cdot \sin(\phi_2) \land \\
& \omega \neq 0 \\
\emptyset & \text{else}
\end{cases}
\tag{3.15}
\]

The conditions of the second solution are not fulfilled, as at least one condition is always wrong and the first solution says a collision occurs for any time \( t \), which is obviously the case for the given condition \( x^{(1)} = x^{(2)} \). Summarizing this result tells that a collision only occurs if both points describe the same point within the object.

### 3.4.2 Downgrading of Crossing Particles

The goal is to apply a downgrading on colliding particles of the HBOF with the knowledge of the previous proof. A drawback of the proof is the theoretical approach using points of infinitesimal small width, which should be replaced by a probabilistic field around the velocity vectors or an area. It is an open question if one can give upper bounds for yaw rate and velocity of vehicles with such areas instead of points, so that the assumption of no collision within objects still holds. This is left for further research and first a practical, unverified downgrading method is introduced, to see if the idea itself gives some improvement on the convergence time. The proposed downgrading factor is calculated using a linear fading having the maximum downgrade factor if the particles are colliding within the smallest time step and the minimum punishment if the intersection is at the maximum time difference \( T_{\text{Max}} \) from the current time. If \( t = 0 \) is the current time, the downgrade factor is calculated via

\[
\gamma = 1.0 - (1 - \gamma_{\text{Max}}) \cdot \frac{T_{\text{Max}} - |t|}{T_{\text{Max}}} \tag{3.16}
\]

and multiplied with every particle, which is crossing another particle within \([- T_{\text{Max}}, T_{\text{Max}}]\).
Chapter 4

Evaluation

The evaluation chapter describes the experiments for different parts of the thesis. Every topic requires different experiments to proof the quality of the proposed solutions or to show improvements. Usually sensor fusion results in an abstract representation of the environment which is compared to either previously known or measured ground truth values. The Velodyne sensors are already very accurate and often serve as the ground truth source for sensor fusion evaluations based on other sensor types like cameras and radars. In real world experiments there is no other sensor source available and therefore no ground truth information accessible. Instead it is shown that the functionality works qualitatively on real data and quantitatively evaluated in a simulation environment. The simulation of the sensors is computationally expensive, hence the horizontal resolution of the simulated sensors is limited to a third of the real sensors’ number of points and the frame rate is set to 4 Hz instead of 10 Hz used for the real experiments.

4.1 Simulated Static Environment

First, static situations are evaluated with the ROS Gazebo [36] simulation environment. Gazebo provides pre built models to create an own world where a robot equipped with custom sensors explores the simulated world and provides the exact same type of ROS messages as the Velodyne sensors on the RCV would do. Using the ROS environment makes it very easy to exchange the real sensors by the simulated ones, as the driver nodes are replaced by the simulation node. A sample world with the drawn drive path of the robot is shown in Figure 4.1.
Figure 4.1: Gazebo [36] simulation environment with the robot’s sample track drawn in green

The sensor shapes are not visible in this simulation because not necessary needed, but have the same translations and orientations as on the RCV for the static tests. As the Gazebo simulator uses the same syntax for pose description as the Unified Robot Description Format (URDF) language used by ROS, different sensor locations are seamlessly integrated in the simulation environment.

4.1.1 Experimental Setup

Goal of the experiment is to show how sensor fusion improves results compared to a straight forward object detection algorithm using Velodyne sensors. This is a counter intuitive behavior as one would expect two identical, accurate sensors not to differ a lot compared to a direct obstacle detection. But there are still important differences:

- Sensor fusion takes history into account
- Straight forward obstacle detection can not distinguish empty and unknown areas
- Sensor fusion introduces (comparably small) delays

In the following DempsterShafer is called the proposed DS sensor fusion network and Z-difference the obstacle detection algorithm to compare with. The Z-difference algorithm is described in Algorithm 2.
Algorithm 2 Z-difference obstacle detection

$C$ is the array of grid cells
$\lambda$ is a threshold for the algorithm

1: for $c_i$ in $C$ do
2: $z_{\text{min}} \leftarrow \text{getMinZ}(c_i, \text{points})$
3: $z_{\text{max}} \leftarrow \text{getMaxZ}(c_i, \text{points})$
4: if $z_{\text{max}} - z_{\text{min}} > \lambda$ then
5: $c_i \leftarrow \text{OCCUPIED}$
6: else
7: $c_i \leftarrow \text{UNKNOW}$
8: end if
9: end for

Figure 4.2 shows the signal flow when using the Gazebo simulator.

Figure 4.2: Gazebo evaluation architecture with data flows and Dempster Shafer (DS) and Z-difference (Z-diff) algorithms to be evaluated in parallel

First the ground truth map has to be created. For this purpose the robot is driven around in Gazebo to explore the whole space covered by the map and an ideal sensor fusion is performed, where the map never becomes outdated. In the end the map is saved and reused by the map server to provide it to the evaluation node. When recording the ground truth map the simulated sensor noise is set to zero. A given scenario is evaluated in Gazebo [36] by loading a world and steering the robot around, while point cloud sensor data is forwarded from both simulated Velodyne sensors. The two algorithms to compare with get the same data in parallel and output a map which is aligned with the ground truth map. Finally the Evaluator counts all combinations of correct values and false positives/negatives for both algorithms and stores it as a .csv file. As the set of possible map states is \{Occupied, Empty, Unknown\} nine different combinations from what the ground truth map’s cells tell and what the algorithm outputs. For every combination the number of cells is stored and chosen ones are plotted in the following subsection 4.1.2.
### CHAPTER 4. EVALUATION

#### 4.1.2 Results

All presented values are recorded during one run along the path shown in Figure 4.1, which is comparable with a typical static urban environment. For all results the number of cells can be converted to an area in m² by dividing with a factor of 100. The titles of each plot in Figure 4.3 represent the evaluated occupancy states for a combination of values in the ground truth map and detected values by the tested algorithms. Only the cells for which both parts of the title match are counted.

![Comparison of algorithms](image)

Figure 4.3: Comparison of *Dempster Shafer* and *Z-difference* algorithms. The four diagrams show the relevant combinations of ground truth and algorithm output, which are all combinations of OCCUPIED (OCC) and EMPTY (EMP). The overall cell number is $3 \cdot 10^4$.

Figure 4.3 shows the evaluation for combinations of OCCUPIED and EMPTY ground truth cells matched by the algorithm (top left and right diagram, respectively) and the number of wrong detected cells (bottom two diagrams). In the top two diagrams a higher cell number means better result, whereas the cell counts of the bottom diagrams should show values as close to zero as possible. As the cell dimension is set to 0.1 m, 100 cells are equal to the area of 1 m². At every time step during the drive the *Dempster Shafer* algorithm outperforms the *Z-difference* algorithm.
4.2 Static Real World Experiments

After pointing out the advantages and disadvantages of the Dempster Shafer sensor fusion in a simulated environment, this section qualitatively shows, that it is actually working on the car. Experimental vehicles like the RCV are not allowed to drive on public roads, thus the experiments are carried out on the Arlanda Test Track. Figure 1.15 gives an impression of the test track, which has a flat, paved area and very few obstacles surrounding this area. The lack of such clear landmarks in the closer area around the vehicle makes it difficult to use them for ego position estimation. Position estimation and the synchronization of position and obstacle sensor data is the challenge during these real world experiments.

For the tests, the position estimation from a previous master’s thesis [15] is used. This position estimation is using the integrated IMU, the wheel speed encoders, the steering angle sensors and a single receiver GPS and fuses all sensor readings using an Unscented Kalman Filter (UKF). As the IMU and the GPS receiver provide processed signals, the position estimation result is not able to compensate the actual error sources, which makes the position estimation drifting some times. Nevertheless it is a sufficient source for doing occupancy grid mapping as the results show. The
The final obstacle map for path planners is shown in Figure A.1. For all scenarios colors have the following meaning:

- Green: LiDAR reflection points
- Orange-Yellow: Belief of empty space
- Blue-Red-Turquoise-Yellow: Belief of occupied space
- Dark Grey: Unknown space
- Light Grey: Grid lines with 1 m spacing

### 4.2.1 Driving scenario

In the first scenario the test vehicles passes a car followed by a single person and two persons next to each other. The ego vehicle is moving from bottom to top passing the car and the two persons on the left hand side and the single person on the right hand side in a small slalom.
4.2. STATIC REAL WORLD EXPERIMENTS

Figure 4.4 shows three frames while the research vehicle is driving through the scene. For a better imagination of the test case, there are additional views attached as Figure A.2 and Figure A.3. The orange/yellow areas clearly represent the knowledge of the empty area, while the obstacles, including some small cones are visible, even when they get off the view of the LiDAR sensors. Similar experiments are attached showing the sensor fusion approach working with different position estimations (highly accurate GPS: Figure A.4, ICP: Figure A.5). As laser scanners provide discrete points, this knowledge is gathered because of the movement of the ego vehicle and shown in subsection 4.2.2.

4.2.2 Enhanced Knowledge through Movement

To show the enhancement of the knowledge caused by driving around, a different static scenario (again with a car and two people) is used, where the test vehicle stopped in front of the obstacles for around 10s. Figure 4.5 shows the vehicle standing still (left) and after passing the obstacles (right). The visualization of the occupied area is disabled to make the difference between unknown and empty area more clear. It is clearly visible (left), that after some time the knowledge of the system decays and only the areas with the thin laser rings (green reflection points) contain information. The 8 small cones are still visible as obstacles on the right image of Figure 4.5.

4.2.3 Ground separation methods

The simplest method to separate the obstacles from the ground is to split the points at elevation equal to zero. This works for the low dynamic situations and because the test vehicle has very stiff springs. But when turning quickly in the end of the test
track, the roll of the vehicle could cause ground points being detected as obstacles. In figure Figure 4.6 this problem is marked with the red box. The black/white map

![Figure 4.6: Comparison of the robustness for static ground extraction method (left) and RANSAC based robust ground extraction method (right). The red rectangle points out the false alarm obstacle detection due to the roll angle of the car while turning.](image)

is the final output forwarded to the path planner, where black cells are obstacles. Most of the area covered by these images show the empty test track while in the top area some bushes and poles are along the test track. Clearly the simple elevation separation technique shows false positive obstacles (left) while the RANSAC ground separation method (right) is robust against such errors. The robust ground separation method uses RANSAC to fit a plane model to the ground plane. Afterwards the point clouds are separated by using the distance along the normal axis of the fitted ground plane. This way roll and pitch angles are compensated for obstacle detection. Of course this only works as long as enough ground area is visible to the sensors. Another drawback of this solution is the additionally introduced delay of about 20 ms as evaluated in section 4.4.

### 4.3 Dynamic Obstacle Tracking

Dynamic obstacle tracking is done using the hybrid form of the Bayesian Occupancy Filter (BOF). As this tracking method is based on an occupancy grid, the resolution is limited to the cell width, which for a 10 cm grid resolution means a wasteful use of the Velodyne accuracy. Therefore one does not expect very low latency or very high precise tracking. There are other advantages resulting from the proposed method as:

- Common probabilistic interface for different sensor types
- No assumptions about the shape
- Object association postponed
4.3. DYNAMIC OBSTACLE TRACKING

While all these advantages implicitly given by the method, this section proofs that the tracking converges to the correct velocities and works with real Velodyne sensor data.

4.3.1 Simulated Velocity Comparison

First a simulated object is used to determine the tracking behavior for a $5 \times 5$ cell object. The original filter was developed to work with horizontal laser beams, which usually result in a thin line reflected from an object. With 3D LiDARs it often happens, that multiple cells ”inside” an object are detected as well. On a car facing towards the sensor, these additional returns could be originated from the engine bonnet and the windscreen instead from the front bumper only. The experiment is carried out at very low speeds, as this velocity region is the difficult part for the tracker. At higher velocities, cells from subsequent frames do not overlap anymore, which leads to more velocity information and therefore faster convergence times. The reference velocity trajectories are generated using sine and cosine with different cyclic duration and amplitudes.

In figure Figure 4.7 the references are plotted with the tracked velocities on top. There are problems whenever the zero line is crossed because there the least velocity information is available and the cells tend to take the static state. The remaining velocity particles have more influence on the resulting object velocity, but aliasing effects are dominant and leading to noise in this region. A small delay occurs because of the filtering, the smaller the delay, the more noise occurs. Especially when it comes to faster velocities the algorithm is able to track the correct velocities smoothly and gains from the tracking for future velocity estimation. As soon as the noise caused by the zero-crossing disappears it follows the constantly changing velocity reliable.

Figure 4.7: Velocity tracking of dynamic object in simulation. The object is of $5 \times 5$ cells size.
4.3.2 Real World Pedestrian Tracking

With some real world experiments the applicability shall be proven. Real world data is less steady and aliasing issues introduce a lot of appearing and disappearing cells. Such appearance and disappearance of objects result in similar measurements to what would be caused by very quick direction changes. As for the static case no real ground truth data is accessible. For the experiments three people are crossing the lane with a more or less constant speed in front of the car. Figure 4.8 shows the scenario with the three people visible as the red marked laser reflections and also the velocity distribution as green arrows. To get some reference velocity, the start and the end point of the people are remembered and the time it takes them to cross the lane is stopped manually. This rough estimation leads to reference velocities, which consist of constant absolute velocities and direction angles. Those reference values are used for ground truth comparison in Figure 4.9, where Front, Back and Middle refer to the persons starting points. In Figure 4.8 the Front person is the right one and the Back person the most left one.

The slowest person in the Front is not detected properly within the first 2 s. This person causes the biggest problem in tracking the direction, as the average velocity is lower than for the other two persons. For the other two persons it really makes sense, when comparing to the recorded video, as the references drawn in the figures are averaged and ignore small turns of the pedestrians. The Back person for example walks along a curvature, which result in a continuously decreasing angle, until around time 8 s, when this person already changes direction and starts turning to move backwards.

Figure 4.8: Pedestrians walking at around 0.75 m s$^{-1}$ with velocity distribution shown as green arrows. The length of the arrows indicates the velocity, if the dimension of one grid cell corresponds to 1 m s$^{-1}$.
4.3. DYNAMIC OBSTACLE TRACKING

Single velocity arrows are generated by clustering the velocity distributions according to their weights and shown in Figure 4.10. For the two people who are not yet at their final position, the clustered velocity is very close to the estimated reference value. Summarizing these results shows that the tracking approach is able to work in real world environment, but it is still a very simple scenario, where for example the association of the objects would not have been a problem with much simpler techniques. Additionally the algorithm requires a higher number of particles, if more dynamic objects become present in the scene, which can cause real time issues. The demonstrated version runs with more than 10 Hz on a single CPU thread.

4.3.3 Step Response

In this coherence a step response is used to measure the convergence time of the dynamic velocity estimation through HBOF [32] and compared to the improved version introduced in section 3.4. At time zero, a simulated object of dimension
0.5 m × 0.5 m starts to move towards the x-direction with 5 m s⁻¹. The standard deviation is measured during the experiments with and without improvement and compared. Because the randomness introduced by the HBOF the experiment is carried out 100 times each, which gives the results in Figure 4.11.

![Graph](image)

**Figure 4.11:** Convergence behavior of the velocity estimation (experiment repeated 100 times, to average the deviation)

It shows an improved convergence time to the true velocity value when applying the *cross-punishment* improvement. There are no values for the first 200 ms, because the algorithm takes two frames (at 10 Hz) to generate the first velocity estimation after initialization of new particles. A drawback is the higher computational cost, but as the implementation is not optimized in that part, no execution time comparison can be provided.

### 4.4 CPU vs. GPU

As seen in subsection 1.2.6 about ideas for parallel programming, this topic gained a lot of interest in the area of occupancy grid mapping. The DS grid mapping algorithm given in section 2.1 introduces the assumption of containing statistically independent grid cells. This number of grid cells \( n \) increases by the power of two with decreasing cell size \( s_{\text{Cell}} \), potentially resulting in huge numbers as in Equation 4.1 of grid cells in the map with dimensions \( h \times w \).

\[
 n = \frac{h \cdot w}{s_{\text{Cell}}^2} \quad \text{(4.1)}
\]

The map dimensions are highly dependent on the way of implementation, namely if the map follows the car’s position or if it is aligned with a locally fixed world coordinate frame. A moving map requires to cover the area of the maximum sensor range, whereas a locally fixed world map needs to cover all the space the vehicle moves. Referring to the used Velodyne sensor specifications given in Table C.1, the
maximum range of the laser beams is 100 m. However Figure C.1 shows that the vertical spacing between the laser beams is in the range of 1 m at 30 m distance already. By defining this range of 30 m as the maximum useful range and assuming the sensor is not occluded in any direction, the maximum grid size is $60 \times 60$ m. As an example for a locally fixed map, the whole area of the Arlanda Test Track is estimated, which by looking into satellite images is $1 \text{ km} \times 200 \text{ m}$.

By further setting lower and upper limits of the cell size from $0.1 \text{ m}$ to $1 \text{ m}$, the overall range of cell numbers equals to $n = 3.6 \cdot 10^3 \ldots 20 \cdot 10^6$. Within this range of cell numbers the DS fusion algorithm is performed on real world scenarios described in section 4.2 and the computation time is measured on CPU and GPU.

There are different parts of software involved, where for some parts a GPU implementation is not done. All software parts and their influence on the timing are explained in the following subsection 4.4.1.

### 4.4.1 Timing Overview

The timing evaluation focuses on the sensor fusion parts mainly consisting of ground extraction and sensor fusion. For the ground extraction three different ways, a plain cubic filter, the $Z$-difference Algorithm 2 and a RANSAC [13] plane fitting based method are implemented. These methods increase the delay between an arriving measurement frame and the updated map output. Figure 4.12 shows all the main steps required in one sensor fusion iteration.

![Sensor fusion tasks](image)

Figure 4.12: Sensor fusion tasks (colors in accordance to Figure 4.13 Figure 4.14 and Figure 4.15)

The *Ground Separation*, the *Sensor Model* and the *Get Grid* blocks are implemented on the CPU only. Obviously that is not the optimal way to use the GPU when it comes to performance, but there are other arguments playing a role as following examples. The allocation between CPU and GPU results from the goal, to create a sensor fusion architecture, which can be used for different sensor types. As different sensor types require different *Sensor Models* and possible *Ground Separation* blocks, it is not clear, what kind of information these sensors require. Therefore the interface between the CPU and the GPU is defined as Dempster Shafer belief values. Such probabilistic values can represent different occupancy sensors in
a common framework. After updating the occupancy grid, it has to be forwarded to some subscribers. The Get Grid functionality converts the inner grid DS belief values into a message type to forward the occupancy map (Occ Map) to following modules, for example a path planner. This block is not implemented on the GPU, to keep the processing structure the same as for the CPU only solution, as it is still possible to deactivate the GPU via a compiler flag to run on devices without dedicated Nvidia GPUs. In the end it is always a compromise between performance and maintainability of the software as it grows.

For the timing evaluation it is important to understand the dependencies consequently resulting in the processing order. There are two main sequences running for the sensor fusion process, which are shown in the timelines of Figure 4.13.

![Figure 4.13: The timeline shows sensor fusion tasks for CPU only (top) processing and use of GPU support (bottom).](image)

The first (top) sequence containing Decay Grid and Get Grid is executed once a cycle period, whereas the frequency for the other blocks Ground Separation, Sensor Model and Fuse Cells is depending on the used sensor frame rate. New sensor readings can be processed in parallel to the first sequence as shown in Figure 4.13. In fact this relaxes the requirement if a certain frame rate is required, because the possible maximum update rate is only depending on the first sequence execution time, as long as all sensor’s individual processing can be done within the update frequency. Because there is only one sensor type configured to output with a frame rate of 10 Hz in the test case, both sequences are required to be executed within 100 ms.

When comparing the GPU supported processing scheme with the first CPU schedule, two additional memory copy MEM CPY operations are required. The one between Sensor Model and Fuse Cells is transferring the probabilistic information from the Sensor Model to the device memory on the GPU, while the other memory copy operation reads the processed (Decay Grid) grid back to the host system executed
4.4. CPU VS. GPU

by the CPU. All memory copy operations are blocking, which means that the calling CPU thread can not process other things in parallel and as the results in subsection 4.4.2 show, this can cause the GPU supported implementation to perform worse compared to a CPU only implementation.

In general the goal using GPU is to get a timing diagram similar to the one sketched in Figure 4.13, where the performance gain from parallel processing overcompensates the additional time required for memory copying operations. The duration needed for copying data from and to the GPU grows approximately linearly with the data length. Therefore the data length should be small, while the operations on the data should still be executed in a highly parallel fashion.

4.4.2 Results

As mentioned in subsection 4.4.1 before, it is important to have the proper scenario for using the GPU support. This is not given in the used fusion architecture, as the Decay Grid always changes values in every grid cell, implying that the whole grid has to be copied in every time step. Otherwise the changing information would not be accessible instantly for the subscribers to the occupancy map.

To compare the two implementation methods, experiments with real data is executed for 60 s and the average of each execution time is plotted. The size of the locally fixed grid is kept at 1 km × 200 m, while the cell resolution is varied to cover the range of \( n = 3.6 \cdot 10^3 \ldots 20 \cdot 10^6 \) grid cells.

The hardware used is a notebook with Intel Core i7 quad core CPU running at 2.2 GHz, a Nvidia GeForce GT 555M GPU and 4 GB of host memory. Especially the GPU is comparably weak compared to the latest devices, as by 2016 GPUs with 50 times the number of parallel processing units are available.

Figure 4.14 shows the problematic situation, where the memory copy operation takes almost as much time as the duration of Get grid and Decay. Although it is difficult to see, but there is a small performance gain in processing the decaying running on the GPU of around 10%. However it can not compensate the additional time needed for copying the data from the graphics card’s memory.

Figure 4.14 also shows from which resolution on the 10 Hz update rate is achieved, namely at 0.25 m or \( n = 3.2 \cdot 10^6 \) grid cells. The execution time for the cyclic tasks only depends on the cell count \( n \).

For the sensor dependent processing, the duration is mostly dependent on the amount of data the sensor produces per frame and in our case it slightly depends on the grid resolution. In Figure 4.15 the execution times are shown, where one can immediately see, that these are of magnitudes lower than for the cyclic tasks.
Figure 4.14: Duration for cyclic tasks with the 100 ms threshold marked, which is the upper limit to fulfill the real time requirement of 10 Hz.

Figure 4.15: Duration of sensor tasks with the average time for a ground separation using static (left) and dynamic RANSAC (right) ground separation.

To get an impression of the ratios between the ground separation and the actual sensor fusion, a fast static ground extraction is used in the left part and a very robust RANSAC [13] based separation for the right hand side plot. It is much more important that these execution times are low, as the final self driving vehicle might have many sensors producing data which accumulates the processing time as soon as all cores are used. The cyclic execution time will not be increased by adding more sensors, as long as the overall limit of the processing capabilities are reached.
Chapter 5

Conclusion

The basic goal of the thesis is to provide a path planner with an occupancy map for the purpose of static obstacle avoidance. This requirement is fulfilled, as it has already been integrated with the path planner and tested on recorded real-world data, showing that both software parts interface properly and run in real time with a fixed frequency of 10 Hz.

All sensor fusion tasks are carried out using the Dempster Shafer theory, which turns out to have no advantage compared to the commonly used Bayesian theory in log-odds form for the dual state (occupied/empty) case and the use of identical sensor types. On the other hand the DS theory requires more mathematical operations and is less intuitive to people. A DS belief value is hard to imagine whereas the meaning of a Bayesian probability is a direct measure of how likely an event is. According to these facts, the Bayesian occupancy grid mapping approach should be preferred, for the static occupancy grid mapping, when no additional states are required. As DS is a more generalized form of the Bayesian theory, one can still argue that there might be some application in future, but so far no example is known. However, the extension to track dynamics using the Hybrid Bayesian Occupancy Filter introduces new states, like dynamic belief of cells which is more suitable to handle with DS theory.

Afterwords the existing grid mapping framework for static scenarios and the developed sensor model is used to extend the solution for dynamic object identification. As long as the sensor readings can be transferred to DS belief values using an appropriate sensor model, the dynamic estimator immediately gains from additional sensor information. This is a great advantage, as the dynamic obstacle identification is not reliant on sensor specific properties. In contrast to the classical tracking approach, where objects are identified and associated to existing tracks, this association is postponed. It is furthermore not required to assume specific shapes for the dynamic estimator to work. By simulation it is shown, that velocities of grid cells are estimated properly for velocities greater than 1 m s\(^{-1}\). For lower velocities the finite grid resolution becomes the major impact and it cannot be distinguished between dynamic and static cases anymore. Besides the simulation, the velocity
estimator works on a simple real world scenario, in which three pedestrians are crossing a street. This example is too simple to actually see an improvement due to the postponing of object association, as the objects could easily be separated by using single frames and are visible most of the time.

5.1 Improvements

While the two main principles of the DS occupancy grid mapping and the HBOF are directly used, the sensor configuration is different from all other publications reviewed. Traditional laser range detection devices make use of one horizontally scanning laser beam. This makes it easy to determine the whole area before hitting an obstacle as free space. With 3D laser scanners like the Velodyne used this is not possible anymore. Some later publications have already used rotating 3D laser scanners, but mounted in a horizontal way. A horizontally mounted scanner simplifies the grid mapping, as circular polar coordinate frames work for each layer of beams, which can be used, to identify the free space. Whereas a tilted sensor provides elliptical and partly occluded occupancy information in horizontal directions. The proposed sensor model approach works for all ways of mounting the sensors, which makes it very flexible to test new configurations.

Another improvement suggests the use of information about colliding objects, as such situations have a low probability of representing the real world. The used Hybrid Bayesian Occupancy Filter for velocity estimation shows faster convergence when such information is considered. This is a very low level application for this idea, but it is possible to use it on object level as well. It is probably something humans do intuitively, to resolve uncertainties.

5.2 Drawbacks

- One main drawback is the lack of a continuously moving grid, as the vehicle is moving. Instead a locally fixed grid is maintained, which simplifies some of the transformation processes and avoids aliasing effects.

- The algorithm can not be compared to ground truth data on real world scenarios, because a more accurate sensor than the ones used is not available. Generating real world ground truth data would therefore be highly demanding, as the environment has to be measured (manually) in some way.

- A comparison between CPU and GPU shows results which does not meet the expected values. It should be redone on a newer GPU to exclude, that the old mobile graphics card is simply not comparable to recent ones (latest devices have 50 times the amount of parallel processing cores). Apart from that, the tested static occupancy grid mapping application requires a lot of memory.
copy, which makes it almost impossible for any GPU to outperform the CPU in this application. For the dynamic estimation a big improvement using the GPU is expected, as only few data needs to be copied, but on the other hand more calculations can be executed on the GPU. Unfortunately the GPU implementation for the dynamic tracking has not been carried out.

5.3 Ethical Aspects of Autonomous Driving

After all self driving cars only make sense if the society accepts these computer driven cars in a shared road area. The acceptance mostly depends on whether people trust in the safety of such cars for passengers and other participants on the street. Especially for pedestrians on the street it is important that autonomously driving vehicles avoid collisions with the weakest participants on the road. Assuming, that self-driving cars will not make any mistakes eventually, there can still be situations where incidents are inevitable due to misbehaviour of humans.

A worst case example is someone running in front of the car by intention. In most situations an autonomous car can still compensate the error of others, because of the magnitudes faster reaction time compared to humans. However there will occur situations, where the algorithm of the car has to decide between injuring one or the other, which raises an old ethical question. The so called Trolly Problem [41] raises the problem to people who have to decide between letting kill many people or kill one single person to rescue the others. In this study, the authors figure out, that humans do not decide rational at all times. For example most people would rather protect a child or a relative person than others, even though that means injuring or killing more people. These kind of decisions will be handled by the control unit of self driving cars and therefore a decision making strategy is required. It is always important to remember, that self-driving cars should cause less incidents than human drivers and therefore it is desirable to get as many self-driving cars as possible on the street. If cars would rather kill the driver no one would buy these cars, preventing self-driving cars to increase the overall street safety.

5.4 Future Work

• The current grid mapping approach is working based on a locally fixed global map, which can only cover a small area around the vehicle. When the vehicle starts moving longer distances, it is necessary to move the map accordingly. As the car potentially changes the direction of movement in every time step, aliasing effects occur and have to be solved for every map update.

• It would be very interesting to see, if it makes any difference, to use the DS fusion method for the dynamic tracking approach. Furthermore the dynamic tracking approach should be evaluated with focus on the converging delay, as
this might be the biggest drawback compared to a sensor specialized approach, especially when the shapes of the objects cover a big connected area.

- For the research platform RCV in general, it is desirable to improve the position estimation, as errors in the position estimation result in magnitudes of bigger error than any change in the sensor fusion approach. So far the visual sensors are not used to improve the position estimation accuracy and robustness, although these sensors like laser scanners can be used to compensate errors of other position estimation sources. With the current IMU and GPS receiver, the raw measurement signals are not accessible and it is therefore difficult to compensate the actual error sources by adding additional sensors and combining them. Therefore it should be considered to replace those sensors, for example with a setup suggested in [18].
List of Figures

1.1 Autonomous research vehicles ........................................... 6
1.2 Obstacle Detection ............................................................. 7
1.3 Sensor Fusion Overview ...................................................... 8
1.4 Resulting occupancy map .................................................... 10
1.5 Indoor point clouds ......................................................... 12
1.6 Occupancy grid state single beam laser scanner ....................... 13
1.7 Backward extrapolation of laser beams .................................. 14
1.8 Comparison backward extrapolation ..................................... 14
1.9 Sensor fusion framework .................................................... 17
1.10 Hierarchical sensor fusion architecture ................................. 18
1.11 Moving object mapping ..................................................... 19
1.12 Risk assessment architecture ............................................. 20
1.13 Landmarks along a test route ............................................ 21
1.14 Polar to Cartesian grid mapping ......................................... 22
1.15 Research Concept Vehicle ................................................ 24
2.1 Coherence Belief and Plausibility ....................................... 27
2.2 Missed detection probability over distance .............................. 30
2.3 Belief mass comparison fault tolerant model .......................... 31
2.4 Occupancy state comparison fault tolerant model .................... 31
4.1 Gazebo sample world ......................................................... 44
4.2 Gazebo evaluation architecture .......................................... 45
4.3 Algorithm comparison occupied and empty ............................ 46
4.4 Static real world scenario ................................................... 48
4.5 Knowledge of unknown space comparison .............................. 49
4.6 Robustness comparison of ground extraction methods ............... 50
4.7 Velocity tracking of dynamic object ..................................... 51
4.8 Pedestrian tracking with velocity distribution ......................... 52
4.9 Dynamic tracking real world velocity comparison .................... 53
4.10 Clustered velocity estimation ............................................. 53
4.11 Dynamic tracking convergence time ................................... 54
4.12 Sensor fusion tasks ......................................................... 55
4.13 Timeline for sensor fusion tasks ........................................ 56
4.14 Duration for cyclic tasks .................................................. 58
4.15 Duration of sensor tasks .................................................... 58

A.1 Resulting outcome map for path planner .............................. 73
A.2 3D view on sample scenario .............................................. 74
A.3 Front view on sample scenario ............................................ 74
A.4 Static Trimble real world scenario ....................................... 75
A.5 Static ICP real world scenario ............................................ 76

B.1 Calibration setting ............................................................ 78

C.1 Vertical beam spacing over distance .................................... 82

D.1 Software architecture ....................................................... 83
D.2 Class diagram of occupancy grid mapping ............................ 84
D.3 Software architecture ICP extension .................................... 85
List of Tables

1.1 Research Concept Vehicle specifications . . . . . . . . . . . . . . . . . . 24
3.1 Variable definitions for HBOF . . . . . . . . . . . . . . . . . . . . . . 35
C.1 Velodyne VLP-16 specifications . . . . . . . . . . . . . . . . . . . . 81

List of Algorithms

1  Hybrid Bayesian Occupancy Filter . . . . . . . . . . . . . . . . . . . . . . 39
2  Z-difference obstacle detection . . . . . . . . . . . . . . . . . . . . . . . 45
Abbreviations

**BOF** Bayesian Occupancy Filter
**BPA** Basic Probability Assignment
**CPU** Central Processing Unit
**DARPA** Defense Advanced Research Projects Agency
**DOF** Degree of Freedom
**DS** Dempster Shafer
**FOD** Frame of Discernment
**FOV** Field of View
**GNSS** Global Navigation Satellite System
**GPS** Global Positioning System
**GPU** Graphics Processing Unit
**HBOF** Hybrid Bayesian Occupancy Filter
**ICCT** International Council on Clean Transportation
**ICP** Iterative Closest Point
**IMU** Inertial Measurement Unit
**ITRL** Integrated Transport Research Lab
**KTH** KTH Royal Institute of Technology
**LiDAR** Light Detection and Ranging
**PCL** Pointcloud
**RANSAC** Random Sample Consensus
RCV  Research Concept Vehicle

ROS  Robot Operating System

UDP  User Datagram Protocol

UKF  Unscented Kalman Filter

URDF  Unified Robot Description Format
Bibliography


Appendix A

Further Experiments

Figure A.1: The resulting binary occupancy grid map used by the path planner. There are two persons, a car, a truck, some cones and in the top left corner some bushes visible.
Some more experiments show that the solution of static obstacle mapping also works with alternative position estimation sources (Figure A.4) or with position estimation using the LiDAR data itself (Figure A.5) and a point cloud matching algorithm. Additionally a top-down view on the outcome forwarded to the path planner is shown (Figure A.1), where all obstacles, even around 20 cm tall cones, are visible after the car passing by once. For better understanding of the experiment used within the thesis some more views on the scenario are given in Figure A.2 and Figure A.3.

Figure A.2: 3D view on sample scenario used in subsection 4.2.1

Figure A.3: Front view on sample scenario used in subsection 4.2.1
Figure A.4: A frame by frame situation of a static real world scenario. The ego vehicle starts at the top goes towards the bottom of the images and then does a U-turn around the real car standing there. The position estimation is done using the Trimble GPS and used directly.
Figure A.5: A frame by frame situation of a static real world scenario. The position estimation is done using the ICP node, which is based on the ICP algorithm from [37]. In this approach no position estimation apart from the LiDAR data itself is used.
Appendix B

Sensor alignment

The Velodyne LiDAR scanners are mounted on top of the upper front rail of the RCV. Both sensors are slightly tilted towards the front. As the mounts are fairly flexible and the sensors are potentially dismounted and attached frequently, a simple and fast way of aligning them according to each other is needed. This section describes how the alignment of the LiDAR sensors is done semi-automatically.

It is assumed that the car is standing still on a flat surface and has at least one other plane of different pose in range. The Velodyne sensors (VLP-16) provide a point cloud with 360° horizontal and 30° vertical FOV each. To bring the two frames of the sensors together we need to estimate a 6 Degree of Freedom (DOF) geometric transformation (three DOF for translation and three DOF for rotation) between them.

First the calibration process tries to find the transformation matrices from a temporary frame (TMP) to both sensors. Temporary frame is defined as x- and y-axis laying within the ground plane and z is pointing up. The z-axis crosses the first Velodyne sensor and x-axis of first sensor frame ($V_1$) and temporary frame are aligned.

Normal vectors of planes are detected by using the RANSAC [13] algorithm to do plane fitting of the Pointcloud (PCL) library.

Define $p_{V1}^1$ to be the normal plane vector presented in the first Velodyne sensor frame and $p_{TMP}^1$ the same vector presented within the temporary frame. The main measurement vectors are drawn in Figure B.1, where $p_1$ is defined as the normal vector of the ground plane.

Normal vector $p_{TMP}^1$ is defined as $p_{TMP}^1 = [0, 0, 1]^T$ to meet the previously mentioned convention. Then one can calculate the rotation matrix $T_{TMP}R_{V1}$ from Velodyne sensor frame one to temporary frame by applying Equation B.1 to Equation B.5.

$$v = p_{TMP}^1 \times p_{V1}^1$$  \hspace{1cm} (B.1)

$$s = \|v\|_2$$  \hspace{1cm} (B.2)

$$c = p_{TMP}^T \cdot p_{V1}^1$$  \hspace{1cm} (B.3)
Figure B.1: Calibration setting, where $p_1$ is defined as the ground plane.

\[
V = \begin{bmatrix}
0 & -v_3 & v_2 \\
v_3 & 0 & -v_1 \\
-v_2 & v_1 & 0
\end{bmatrix}
\] (B.4)

\[
TMP_R V^1 = I + V + V^2 \cdot \frac{1 - c}{s^2}
\] (B.5)

The translation is simply determined by the distance of the ground plane to the sensor, which is also returned by the RANSAC. As the z-axis of the ground’s normal vector is crossing the first sensor, x- and y-components are zero. The translation vector is given as

\[
TMP_t V^1 = [0, 0, d_1]^T,
\] (B.6)

where $d_1$ is the height of the place where the sensor is mounted relatively to the ground.

In a similar way the rotation matrix $TMP'R^{V^2}$ and the translation vector $TMP't^{V^2}$ are determined. The subscript $TMP'$ indicates that neither the rotation and nor the translation are completely done. It is important to ensure that the same ground plane is detected by the two sensors. As both sensors are mounted on the front top of the car with sensor frames z-axis point towards a very similar direction, a filter is used to extract only negative z-values in the sensor frames. For most scenarios this is sufficient as the ground is the largest plane for both sensors and therefore will be detected as the ground plane.

At this point both sensors are aligned along the temporary frame z-axis and the $z$-translation values are known for both of them. The $x$- and $y$-translations and the rotation around the z-axis between the two sensors are still unknown. First the rotation angle $\phi_z$ along the temporary frame’s z-axis is determined. Afterwards a
simple translation in the temporary frame is enough to align the two sensor point clouds.

To determine the rotation angle and the translation a second plane appearing in both sensor views is needed. The second plane is detected by RANSAC as well using a different filter in advance. Therefore we get the two normal vectors $\mathbf{p}_1^V$ and $\mathbf{p}_2^V$ from the second plane and the two closest distances to each sensor $d_2^V_1$ and $d_2^V_2$. For the rotation calculation the temporary frame is used. Equation B.7 shows how the second plane vectors are related to the coordinate frames.

$$\mathbf{p}_2^{TMP} = \mathbf{R}^{TMP'} \cdot \mathbf{R}^V \cdot \mathbf{p}_2^V$$  \hspace{1cm} (B.7)

$\mathbf{R}^{TMP}$ is given by the three dimensional rotation matrix around the z-axis $\mathbf{R}_z (\phi_z)$ by the angle $\phi_z$. For arbitrary vectors $\mathbf{v}_1$ and $\mathbf{v}_2$ the angle $\theta (\mathbf{v}_1, \mathbf{v}_2)$ in between is calculated as in Equation B.8.

$$\theta (\mathbf{v}_1, \mathbf{v}_2) = \arccos \left( \frac{\mathbf{v}_1 \cdot \mathbf{v}_2}{\left\| \mathbf{v}_1 \right\| \left\| \mathbf{v}_2 \right\|} \right)$$  \hspace{1cm} (B.8)

As the second plane does not need to be orthogonal to the ground plane, there might be a z-component present in $\mathbf{p}_1^{TMP}$ and $\mathbf{p}_2^{TMP}$, where $\mathbf{p}_2^{TMP} = \mathbf{R}^V \cdot \mathbf{p}_2^V$. The z-components of $\mathbf{p}_2^{TMP}$ and $\mathbf{p}_2^{TMP'}$ are set to zero thus giving $\mathbf{p}_{2z}^{TMP}$ and $\mathbf{p}_{2z}^{TMP'}$ respectively. Finally the rotation matrix from sensor frame two to the temporary frame through sensor one is given by

$$\mathbf{R}^{TMP} \cdot \mathbf{R}^V = \mathbf{R}_z \left( \theta (\mathbf{p}_{2z}^{TMP}, \mathbf{p}_{2z}^{TMP'}) \right) \cdot \mathbf{R}^V. \hspace{1cm} (B.9)$$

Translation from temporary frame to sensor frame two $\mathbf{t}^{V2}$ is given by Equation B.10 to Equation B.13 due to basic trigonometry. The distance between the sensors is measured and given as $l$.

$$\Theta_1 = \arctan \left( -\mathbf{p}_{2x}^{TMP}, -\mathbf{p}_{2x}^{TMP} \right) \hspace{1cm} (B.10)$$

$$\Theta_2 = \arcsin \left( \frac{d_2^V - d_2^V_2}{l} \right) \hspace{1cm} (B.11)$$

$$\Psi = \frac{\pi}{2} - \Theta_1 - \Theta_2 \hspace{1cm} (B.12)$$

$$\mathbf{t}^{V2} = \mathbf{t}^V + l \cdot [\cos (\Psi), \sin (\Psi), 0]^T \hspace{1cm} (B.13)$$

In the special configuration of the car, where both sensors are mounted on the upper front corners of the frame, it is easy to determine the forward direction. To take that information into account an additional frame is introduced. The frame is defined the coordinate frame of a GPS antenna, which is mounted half way in between both Velodyne sensors. The axis are pointing in following directions:

- X-axis: Forward
- Y-axis: Left
- Z-axis: Up (orthogonal to the ground plane)

The question is to determine the rotation matrix \( T_{\text{MP}}R_{\text{GPS}} \) and the translation vector \( T_{\text{MP}}t_{\text{GPS}} \) between the temporary frame and the gps frame. By using the symbols introduced before following Equation B.14 to Equation B.15 leads to the result.

\[
T_{\text{MP}}R_{\text{GPS}} = \mathbf{R}_z \left( \frac{\theta \left( p_{2xy}^{\text{TMP}}, p_{2xy}^{\text{TMP}}' \right)}{2} \right) \quad \text{(B.14)}
\]

\[
T_{\text{MP}}t_{\text{GPS}} = \frac{T_{\text{MP}}t_{V1} + T_{\text{MP}}t_{V2}}{2} \quad \text{(B.15)}
\]

For later transformations between the vehicle and earth fixed coordinate frames a base link frame is introduced aligned with the gps frame, but within the ground plane. The rotation matrix for base link (BL) is equal to the gps matrix \( T_{\text{MP}}R_{\text{BL}} = T_{\text{MP}}R_{\text{GPS}} \). As the support frame TMP is aligned with the ground, the z-value of the translation vector is set to zero \( T_{\text{MP}}t_{\text{BL}} = \left[ T_{\text{MP}}t_{\text{GPS}}^x, T_{\text{MP}}t_{\text{GPS}}^y, 0 \right]^T \).
Appendix C

Velodyne VLP-16

The Velodyne VLP-16 multiple beam LiDAR is a time of flight distance measurement sensor with calibrated reflectivities. All specifications given in Table C.1 are based on the data sheet [42] provided by Velodyne Acoustics, Inc.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Channels</td>
<td>16</td>
</tr>
<tr>
<td>Range</td>
<td>up to 100 m</td>
</tr>
<tr>
<td>Accuracy</td>
<td>±3 cm (typical)</td>
</tr>
<tr>
<td>Vertical FOV</td>
<td>−15° to 15°</td>
</tr>
<tr>
<td>Horizontal FOV</td>
<td>360°</td>
</tr>
<tr>
<td>Angular resolution (vertical)</td>
<td>2°</td>
</tr>
<tr>
<td>Angular resolution (horizontal)</td>
<td>0.1° to 0.4°</td>
</tr>
<tr>
<td>Rotation rate</td>
<td>5 Hz to 20 Hz</td>
</tr>
<tr>
<td>Wavelength</td>
<td>905 nm</td>
</tr>
<tr>
<td>Power consumption</td>
<td>8 W (typical)</td>
</tr>
<tr>
<td>Weight</td>
<td>830 g (without cabling)</td>
</tr>
</tbody>
</table>

Table C.1: Velodyne VLP-16 specifications

Within the range of up to 100 m each single distance measurement has got an accuracy of ±3 cm in "typical" scenarios. The question is if one beam to hit a target is enough for the application. In our case, we want multiple laser beams being hitting one target like humans or cars. Figure C.1 gives an overview of the vertical spacing over the obstacle distance, assuming the sensor being mounted horizontally leveled.
Figure C.1: Vertical laser beam spacing over distance for 16 laser beams. Top to bottom lines represent spacing between central to most outer laser beams, respectively.
Appendix D

Implementation Details

The implementation of all parts is done within the ROS framework and according to the belonging guidelines. In the ROS framework, packages combine different software parts for similar purposes. Information between the different software parts is exchanged via a publish/subscribe mechanism, where many useful topics for robotic application are already predefined. The given implementation has neither required any new topic to transfer the data within the sensor fusion part nor for forwarding the resulting occupancy map to the path planner.

An overview of the software architecture is given in Figure D.1, with the main parts the low level sensor drivers, position estimation and occupancy grid mapping including the sensor model. The whole low level part is setup in the ROS Gazebo [36] simulation environment to run simulated experiments, while the sensor fusion part stays exactly the same (the simulator outputs the same data format and position information as the real car). All blocks are different ROS nodes, which exchange

![Software Architecture Diagram]

Figure D.1: Software architecture sensor fusion with two Velodyne LiDAR sensors (nodes as blocks and topics as arrows)
the data through the topics labeled at the connecting arrows. Following sections describe the different parts in more detail.

**Sensor Fusion**

Part of the sensor fusion is excluded from the *DS Fusion* block in Figure D.1, as the functionality *Ground Separation* can be is necessary for the used sensor model, but could be reused in other applications as well. An additional advantage is that the *Ground Separation* method is easily exchangeable by using one of three different ROS Nodes provided:

1. **Cubic Filter**: For simple and fast filtering in x-, y- and z-coordinates
2. **Z-Difference Filter**: For robust obstacle detection, but raw point data is summarized to one point per cell (Algorithm 2)
3. **RANSAC Filter**: Robust ground detection and roll/pitch angle compensation for filtering the raw point cloud data (drawback is the delay introduced)

The other part of the sensor model as well as the occupancy grid mapping are held within the *DS Fusion* block, which internally is structured as shown in Figure D.2. All grid related parameters and functionality is covered by the grid class, while a *Sensor Model Base* class provides an interface for the sensor dependent class. The grid can hold infinitely many objects of sensor models. Adding a new sensor type is done by adding a new model class like the *Velodyne Model* in the current setup. Through the *passNewCellInfos* function the base class provides the interface needed to fuse sensor data. This base class also provides all functions required to do operations on the grid, for example convert coordinates to grid indexes or check if a point is laying in the grid and therefore being relevant.
Position Estimation

Three different sources for the position estimation are available or implemented to use during the tests. Two use position data from either the accurate Trimble GPS device or the position estimation system using all sensors of the RCV [15]. For both position sources ROS nodes are implemented to read and synchronize the recorded position information and output the proper ROS transforms in the locally fixed frame.

A third solution is to use the point cloud data for position estimation only. For this solution the software architecture has to be adapted in the following (Figure D.3) way. The *Iterative Closest Point* node matches consecutive frames of point cloud data and accumulates the position offset to generate a continuous locally fixed position estimate. Errors introduced by the used ICP matching tool [37] accumulate with the given solution leading to a global position drift. It is necessary to do the point cloud transformation in the end, to get the latest position update.

![Figure D.3: Software architecture with Iterative Closest Point (ICP) [37] extension.](image)

Velodyne Drivers

There is a bundle of useful ROS nodes available within the ”ROS velodyne” package for working with the Velodyne LiDAR sensors. The main ones used are:

1. *Velodyne Driver*: Reading the raw User Datagram Protocol (UDP) packages and converting them to the Velodyne *Packets*. The driver is able to emulate a Velodyne sensor by reading recorded network data in the .pcap format.

2. *Velodyne Transform*: Converts the raw measurements contained in the *Packets* to point cloud data and immediately transforms the data to a desired coordinate frame, if the transformations are available in the ROS environment.

3. *Velodyne Cloud*: Is similar to *Velodyne Transform*, but skips the transformation, which saves time. It is used during the ICP position estimation, as in the first step a point cloud is required to compare with, but the correct transformation is not yet available.
Additional Components

Some additional nodes are implemented for different purposes:

1. **Evaluation**: A node to compare different occupancy grid mapping algorithms to a ground truth map, which is used in section 4.1.

2. **Calibration**: Provides a node to support the accurate extrinsic calibration of different LiDAR sensors relatively to each other.

3. **Car Counter**: Is a clustering tool based on the occupancy map, which allows object counting, given that the spacing between object is bigger than the gaps which occur within single objects.