Masterarbeit am Institut für Informatik der Freien Universität Berlin,

Arbeitsgruppe Autonomous Cars

Object detection in 3D point clouds

Christian Damm
Matrikelnummer: 4356891
c.damm@fu-berlin.de

Erstgutachter: Prof. Dr. Daniel Göhring
Zweitgutachter: Prof. Dr. Raúl Rojas
Betreuer: Fritz Ulbrich

Berlin, 28.09.2016

Abstract

With the ongoing spread of autonomous vehicles, challenges like obstacle avoidance get more important. To realize obstacle avoidance, a reliable obstacle detection is one of the preconditions. While common autonomous vehicles mainly use camera and radar sensors for this purpose, currently laser range sensors are enforcing as alternatives. Due to its high accuracy, this kind of sensor establishes in different industries. In general, the sensor data is used as point clouds. Within this master's thesis, an approach for obstacle detection based on these point clouds is presented. Therefore, several subtasks, e.g. downsampling and plane segmentation, of a reliable obstacle detection are carried out. Finally, an algorithm for obstacle tracking, based on a linear Kalman filter, is implemented. The received results are evaluated within several test drives of the autonomous vehicle MadeInGermany (MIG).

Statement of Academic Integrity

Hereby, I declare that I have composed the presented paper independently on my own and without any other resources than the ones indicated. All thoughts taken directly or indirectly from external sources are properly denoted as such. This paper has neither been previously submitted to another authority nor has it been published yet.

28.09.2016

Christian Damm

Contents

List of Figures

Acronym

ADAS Advanced driver assistance systems

DARPA Defense Advanced Research Project Agency

DMI Distance Measuring Indicator

FOV Field of view

GNSS Global Navigation Satellite System

GPS Global Positioning System

IMU Inertial Measurement Unit

LiDAR Light Detection and Ranging

MAV micro aerial vehicles

NASA National Aeronautics and Space Administration

PCL Point Cloud Library

RANSAC random sample consensus

ROS Robot Operating System

SPAWAR Space and Naval Warfare Systems Command

TBB Threading Building Blocks

USV unmanned surface vehicle

Chapter 1

Introduction

Within this chapter, the reader receives an outline of the general context which surrounds this thesis. Starting with the research motivation and the aspire goal, a summary of the thesis' structure follows. At last, an overview of related work is presented.

1.1 Motivation

Even though semi-autonomous driving on highways with support of advanced driver assistance systems (ADAS) is almost common, we are still far away from driving autonomous everywhere. Due to the same moving direction of vehicles, the absence of traffic lights and other traffic participants like pedestrians, the complexity of the autonomous driving task on highways is decreased. Whereas, in urban areas the amount of possible traffic situations is vast. Although the vehicle speed is much lower as on highways, the visual range is reduced as well. Often blocked by objects like cars, trucks, fences or houses, traffic participants can not be sure to detect obstacles early enough. Based on this, autonomous vehicles are equipped with different kinds of sensors for various tasks, including the obstacle detection. Each of the sensors has its own pros and cons, given by physical law. Some of them are good to measure near distance areas and some are better used for far distances. The suitability of the sensors regarding factors like daytime or night has to be considered, too.

The main kind of sensors, which the thesis is dealing with, are multi beam **li**ght **d**etection **a**nd **r**anging sensors (LiDAR). This LiDAR sensors can create a three dimensional model of the surrounding world. A great advantage of a 3D model is the ability to determine additional information of obstacles

like contours. Due to their high prices, LiDAR sensors are not distributed as standard cameras. Therefore the quantity of existing algorithm is limited. Moreover, until a few years ago, consumer computer were not powerful enough to validate a three dimensional model of our environment. But today, good precondition for research in this area exits.

This thesis was developed within the workgroup "Autonomous cars" at Freie Universität Berlin. With the received results, more complex tasks like dynamic path planning will be realised.

1.2 Goal

In this work, a multi obstacle detection algorithm for the multi beam LiDAR sensor Velodyne HDL-64E of the autonomous car MIG of Freie Universität Berlin is presented. The algorithm receives one sensor scan, uses different kinds of preprocessing steps and finally determines several obstacles in this scan. Afterwards, the information of consecutive scans will be used to track the obstacles. Another goal is to add new tracks automatically; and also remove tracks, which are lost. The received results will be compared with an already existing obstacle tracking algorithm of several combined IBEO Lux sensors, which are also part of the autonomous car MIG.

1.3 Thesis structure

Following this first chapter, fundamental information for the subsequent research will be provided. The second chapter explains aspects like the used sensor technology and the fundamental software framework. Successive by chapter 3, which elucidates the several steps for creating an obstacle detection, as well as an evaluation of the implemented algorithm with a similar obstacle tracker. Finally, chapter 5 summarizes the achieved results and prospects possible optimizations.

1.4 Related work

In 2002 the Defense Advanced Research Project Agency (DARPA) announced a competition for autonomous vehicles, called DARPA Grand Challenge. Within this competition, the aim was to spur the development of a full functionality autonomous vehicle. The first challenge occurred on March 13, 2004. To win the challenge, the participating vehicles had to drive an 140

miles outdoor route autonomously. None of the 15 finalists reached the goal. The longest reached distance only was 7.4 miles [IB06]. Nevertheless, the DARPA Grand Challenge 2004 was a good possibility to collect experience in a more practical environment. One year later the next competition occurred already (DARPA Grand Challenge 2005). This challenge contained an 132 miles long off-road route. Unlike the year before, five of the 23 finalists reached the finish line. The team of the Stanford University finished in first place with an overall time of 6 h, 53 min [TMD+06].
The last DARPA Grand Challenge take place in 2007. Different to the previous challenges, the third challenge occurred in an urban environment. This time the participants had not to drive long distance. Instead the vehicles had to obey the traffic laws and avoid other traffic participants. With regard to this thesis, the urban challenge is the most interesting one. To finish the challenge the eleven participants had to complete multiple missions in a timeframe of six hours. The missions contained simple tasks like lane changes or brake at a stop sign. But even more complex tasks like three-point turning or pulling into specific parking space were demanded [dar07]. Six participants finished this challenge. Each of them had its own technology for obstacle detection and avoidance. Looking at their technical specification, made clear, that all finishers used combinations of different sensor types for the detection. The main reason for the sensor combinations is the aim to reduce the shortcoming of each sensor type.
One of these sensor types are laser range sensors, which are mostly used for determining obstacle positions [MCH+08]. Especially the laser range sensor Velodyne HDL-64E , which is also part of this thesis, was used by most of the participants. Further more, this sensor was used as primary sensor for obstacle detection in some cases [MBB+08] [BFK+08].

Chapter 2

Fundamentals

This chapter aims for imparting the basic knowledge for the following content. At first, the project context of this thesis is introduced, followed by the used sensor technology and some algorithmic fundamentals. In the end, the framework ROS and the mainly used library gets introduced.

2.1 MIG

Regarding the implementation, the autonomous car MIG is used. It is part of the project AutoNOMOS of the group Autonomous cars of Prof. Daniel Göhring. Before it was used for researches of the group Artificial Intelligence of Prof. Dr. Raúl Rojas. Since 2010 the car is under ongoing development and due to an exceptional permission, the group is allowed to test functionalities of the car in real traffic situations in Berlin, Germany. The car is based on a VW Passat Variant 3c equipped with Drive-by-Wire and Steer-by-Wire technology. During the last years various publication, which used MIG as test platform, were put out. Their topics are widely spread, e.g traffic awareness [LUG16], sensor fusion [GWSG11] [TG15] and car control using brain computer interface [GLWR13].
For the autonomous driving additional sensors were attached. Their positions are illustrated in figure 2.1. The sensors can be divided in the following base groups:

1. positioning system
2. camera system
3. laser scanner
4. radar system

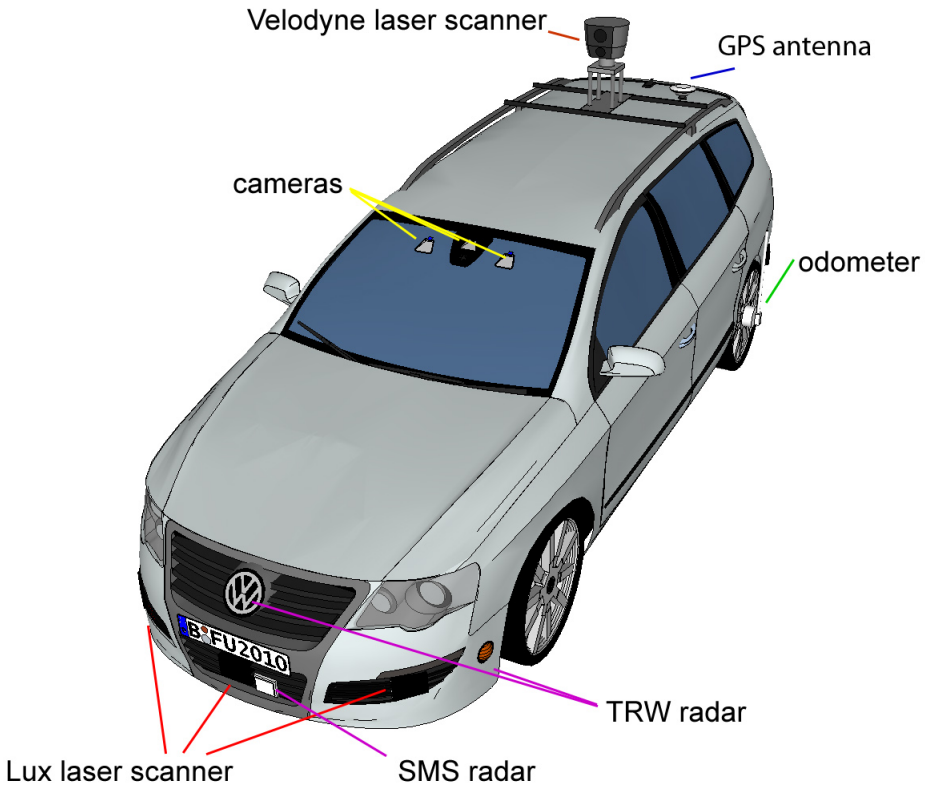

Figure 2.1: Autonomous car MIG of Freie Universität Berlin with contained sensors

For the positioning system an Applanix POS LV 510 is used. This system combines the information of GPS, IMU, DMI and further sensors to provide a high accurate position and orientation measurement. With an existing global navigation satellite system (GNSS) connection it has precision up to 0.02 m[app]. For the camera system two INKA cameras of Hella Aglaia and one camera of Continental are mounted in the front window. They are used for visual recognition, e.g. lane detection and traffic light detection. The group of laser scanners consists of multi IBEO Lux sensor and one Velodyne HDL-64E. In the present research, these sensors are the major group. How this kind of sensor works, especially the Velodyne HDL-64E, will be explained afterwards. Finally, the base group "radar system", contains two short range radars and four long range radars. Their tasks are Adaptive Cruise Control and Blind Spot Detection.

Feature	
Application range	up to 120 m
Horizontal FOV	360 °
Vertical FOV	26.8 °
Multi layer	64
Data update rate	5-15 Hz
Distance accuracy	0.02 m
Angular Resolution horizontal	0.09 °
Angular Resolution vertical	approximately 0.4 °

Figure 2.2: Technical data of Velodyne HDL-64ES2

2.1.1 Velodyne HDL-64E

For this thesis a HDL-64ES2 (manufactured by Velodyne LiDAR) is used as data source. As seen in figure 2.1, the sensor was mounted on the roof of the car and rotates with a speed between 5 and 20 Hz. Within one second the sensor generates over 1.3 million data points. With its weight of almost 14 kg, it was designed for larger vehicles. In horizontal direction, it has a 360° Field of View (FOV) and in vertical direction 26.8° FOV. Additional technical data can be found in table 2.2.

To generate a 3D model of the surrounding environment, the sensor emits 64 laser beams at different vertical angles. The beams are separated in one upper and one lower group containing 32 beams each. The figure 2.3 shows the grouping. Each group involves one receiver in the middle and two emitters next to them, which emits 16 beams each.

Figure 2.3: The Velodyne HDL-64E is separated in one lower and on upper group of laser beams. Each group involves one receiver and two emitters.

Looking at figure 2.4 shows, that the groups have different vertical FOV. The upper group has a vertical FOV of only 10.5° and the lower group of 16°. As a result, the average angle distance of the upper beams is smaller (approximately 0.3°) than of the lower beams (approximately 0.5°).

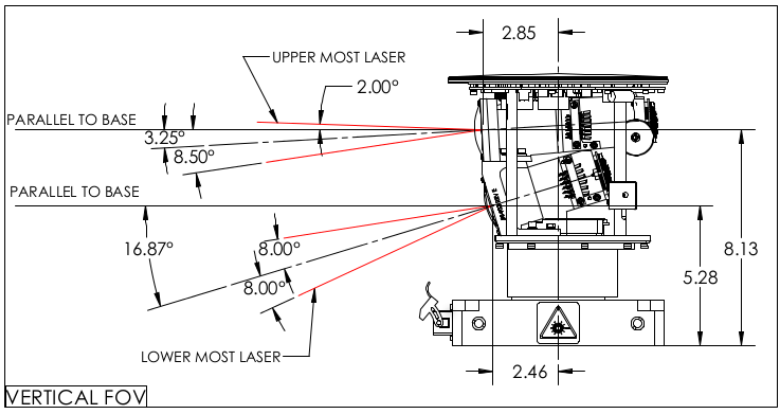

Figure 2.4: Draw of Velodnye HDL-64E, showing different size of FOV [velb]

Due to divergence in the manufacturing step of the sensor, each sensor will be shipped with a specific configuration file. This file contains precise information about the alignment of each sensor. Nevertheless, some participants of the DARPA Grand Challenge 2007 reported about still existing calibration errors. They suggested two different methods to reduce this error rate: In addition to the vertical displacement of the beams, [MBB+08] compared the shift of horizontal adjacent beams. Whereas [BFK+08] made an own individual calibration in combination with a SICK LMS-291 sensors. Still existing outliers were rejected.

One important aspect is the proper motion of the car on which the sensor is mounted. Due to the minor spinning speed of the sensor, the 360° are not perfectly circular, but rather spiral. Even with a moving speed of 30 $\frac{km}{h}$ ($\approx$ 8,3 $\frac{m}{s}$) and sensor spinning speed of 5 Hz, there is a shift of $\approx$ 1,667 m between one revolution. As seen in figure 2.5, this can also cause duplicate obstacles. Towards nearby obstacles, the vehicle itself has a great relative proper motion. Therefore, the sensor scans the obstacle twice in one rotation, which can not be recognized by the sensor itself.

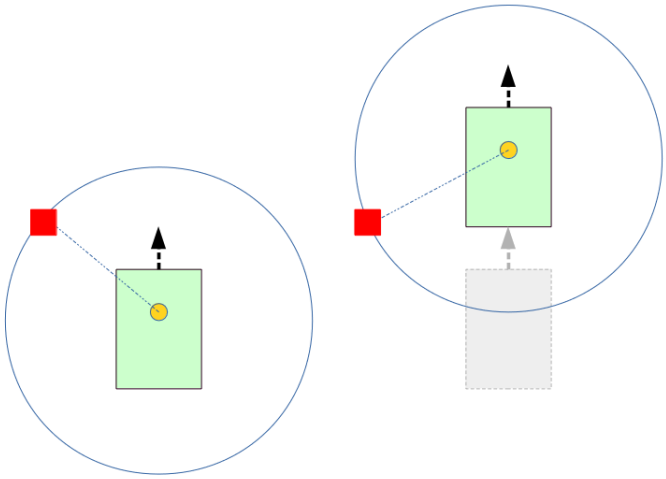

Figure 2.5: Problem with the proper motion of the sensor. Green rectangle is the own vehicle (seen from topview) and the red cube is an obstacle. The sensor is spinning clockwise. Due the proper motion the obstacle is detected twice in one rotation.

Beside the usage in autonomous ground vehicles, the HDL-64E sensor is also used in maritime area. The Space and Naval Warfare Systems Command (SPAWAR) utilized it for some of their unmanned surface vehicles (USVs). Based on little up to no signal returns for a calm sea surface, the sensor can detect smaller obstacles up to 50 meters and larger obstacles even up to 100-120 meters[NLK+09].

2.1.2 IBEO Lux Sensor

As mentioned before, the results are compared with an obstacle tracking algorithm of the IBEO Lux sensor. Similar to the HDL-64E, it is also a LiDAR sensor. But it differs in several aspects from the HDL-64E. The technical data of the IBEO Lux sensor in table 2.6 shows, that the sensor only contains four vertical stacked beams, which leads to a smaller vertical FOV (3.2° compared to 26.8°). Based on the longer application range and the higher update rate, the IBEO Lux sensor has a temporal advantage compared to the HDL-64E. It can detect obstacles earlier and also react faster on occurring changes. To overcome the limited horizontal FOV of only 110° , up to six IBEO Lux sensors can be fused. Thus, a horizontal FOV of 360 °can be achieved.

Feature	
Application range	up to 200 m
Horizontal FOV	110 °
Vertical FOV	3.2 °
Multi layer	4
Data update rate	12.5 / 25.0/ 50 Hz
Distance accuracy	0.1 m
Angular Resolution horizontal	up to 0.125 °
Angular Resolution vertical	0.8 °

Figure 2.6: Technical data for a single IBEO Lux sensor.

2.2 LiDAR

In the last decades **Li**ght **D**etection **a**nd **R**anging has become a common distance measurement technology in a lot of different domains. It is used in archaeology to find possible places for excavation [MO14], in astronomy for surface measuring of new planets [SZS+99] and, in robotics for obstacle avoidance [RP15].

For the distance measurement, the sensor emits a signal and measures the time (t) until the signal returns. Based on this information, it is possible to determine the travel distance d of the signal with the formula 2.1, where c is the speed of light.

$$d = \frac{c \cdot \Delta t}{2} \tag{2.1}$$

But this is just a single measurement. To generate a map of the environment, the signal must be emitted in a 360° rotation. Different approaches can be used to achieve this intentions. One solution is to rotate the sensor in circular motion. An example is shown in figure 2.7. The upper row shows the sensor, whose signal refracts on a contra clockwise circulating mirror (gray object). The middle row is a topview of a fenced area and visualizes the direction of the signal. As shown, the area only contains the blue sensor and a green obstacle in the right part of the area. The lower part shows the generated environment map. The sensor receives no information about the area covered by the obstacle (see column b). Due to that, there are situations, where no information about the depth of an obstacle can be received, the obstacle is just seen from the front.

Another solution to generate an all around environment map is to fuse the data of several similar sensors. Therefore multi sensors are aligned in differ-

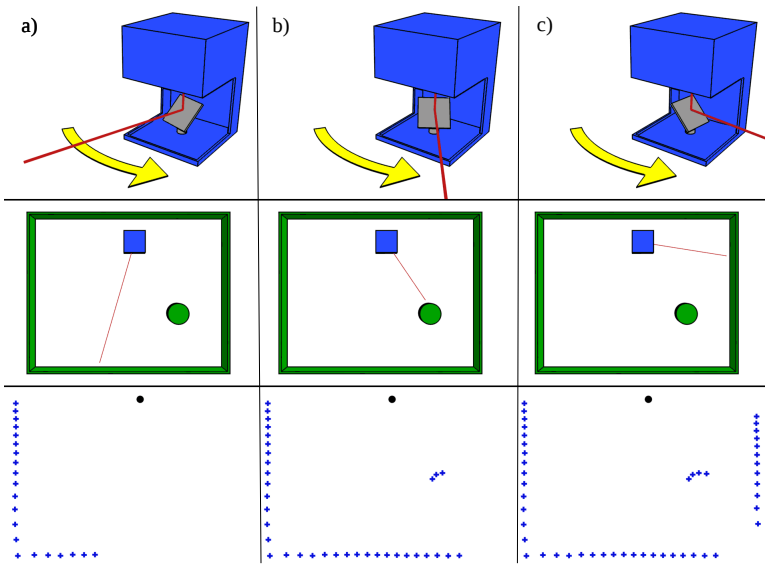

Figure 2.7: Rotation of a LiDAR sensor for three different moments. a) first measurements b) hitting an obstacle c) after rotation of 180 °[Com08]

ent direction, which receive only parts of the full circle. In a subsequent step, these individual scans get fused to create a horizontal surround view. This procedure is also used on the autonomous vehicle MIG. The car contains six IBEO Lux sensors, which were mounted on different positions. Each of them has a horizontal FOV of 110 °. An illustration of the covered areas can be seen in figure 2.8.

Both described solutions (rotating sensor and fusion of several sensors) afford a horizontal surround view. However, in vertical direction, the view range would be still flat. Using just one laser, would produce only a two dimensional scan. In other words, the vertical FOV is 0°. Therefore, multiple beams with different vertical angles are emitted. As mentioned before, the HDL-64E uses 64 vertical stacked beams, which strike the obstacles on different heights (see figure 2.9). In contrast the IBEO Lux sensor only uses four stacked beams.

2.3 Point cloud

The received measurement data from the input sensors gets converted in a more generic structure, the so called point cloud. In general, a point cloud is a collection of data points in a specific coordinate system.

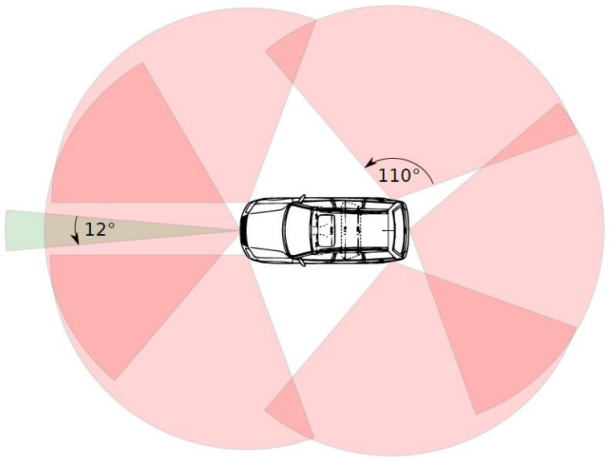

Figure 2.8: Alignment of the six IBEO Lux sensors. Each of them has a vertical FOV of 110 °. By combining their individual data, a 360 ° scan is received. [LUG16]

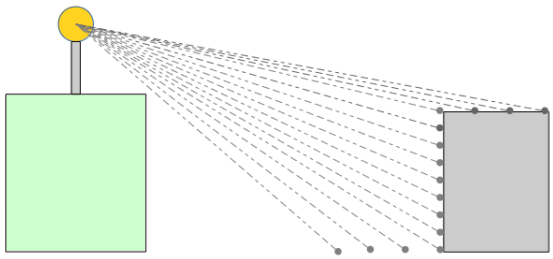

Figure 2.9: Drawing of the vertical stacked beams of the Velodyne sensor. Due to the different vertical angles, they hit the nearby obstacle different heights.

For the present purpose, a three dimensional coordinate system is sufficient. Besides the spatial coordinates, the data of point clouds contain additional parameter like rgb-color or intensity of the received laser beam for particular points. But these information are more sensor specific. As mentioned before, the generated point cloud is based on sensor information. This causes the following restrictions for the point cloud:

1. The sensor range is finite. As a result the point cloud only represents the nearby area.

2. Due to the circular spread of the sensor beams, the density of the data points gets reduced with ongoing distance.

3. The sensor data is always noisy, which leads to small deviation. The distance accuracy of the Velodyne sensor is ±2 cm [vela].

2.4 K-d Tree

Since the point cloud data is received as unsorted list, a ordering algorithm for subsequent process steps is needed. Invented 1975 by Jon L. Bentley [Ben75], k-d trees are a widespread technique to organize data points in a k-dimensional space. The main idea is to split the data points at one specific dimension per each level of the tree.

In the figure 2.10, the node D is used as root node r. It divides along the x-dimension. Each node n with $n_x \leq r_x$ is added to the left subtree and each node with $n_x > r_x$ is added to the right subtree. To avoid unbalance, the median value of all nodes in a subtree is used as splitting node (actually the $\lceil \frac{n}{2} \rceil$-th smallest value).

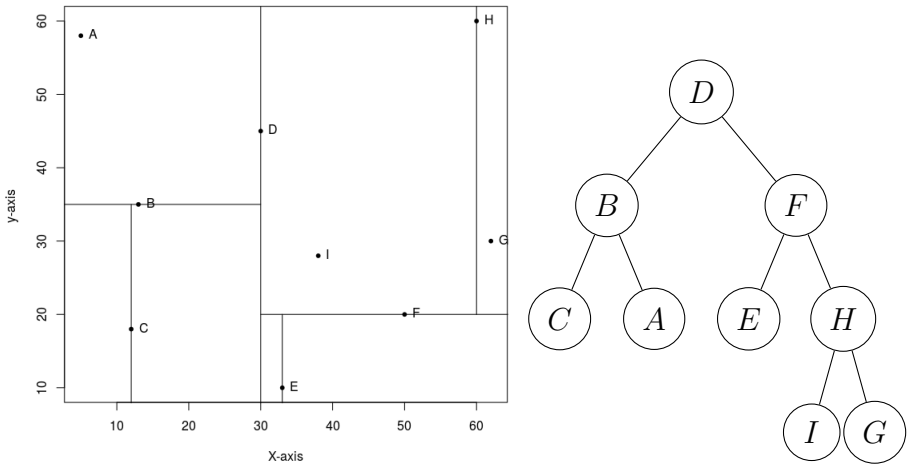

Figure 2.10: Example of a k-d tree. The left part shows some points in a two dimensional space and the subdivision. The right part shows the result in a k-d tree.

A pseudo algorithm to build a k-d tree can be found in algorithm 1 (based on [DBVKOS00]). The parameter d defines the dimension of the splitting (e.g. x-dimension, y-dimension). For each recursive call, d is increased (line

7, 8).

Algorithm 1: *Algorithm to build a k-d tree*

Input : Set of points P and current dimension d
Output: Root of a kd-tree storing P

```
1  begin
2      if P contains only one point then
3          return a leaf containing this point
4      else
5          Split P into two subsets. Subset P_1 contains all points with
           values ≤ median value on dimension d. Subset P_2 contains all
           points with values > median value on dimension d
6      end
7      v_left ← (P_1, (d + 1) mod k);
8      v_right ← (P_2, (d + 1) mod k);
9      Create node v, which stores the median value and contains v_left as
       left child and v_right as right child;
10     return v
11 end
```

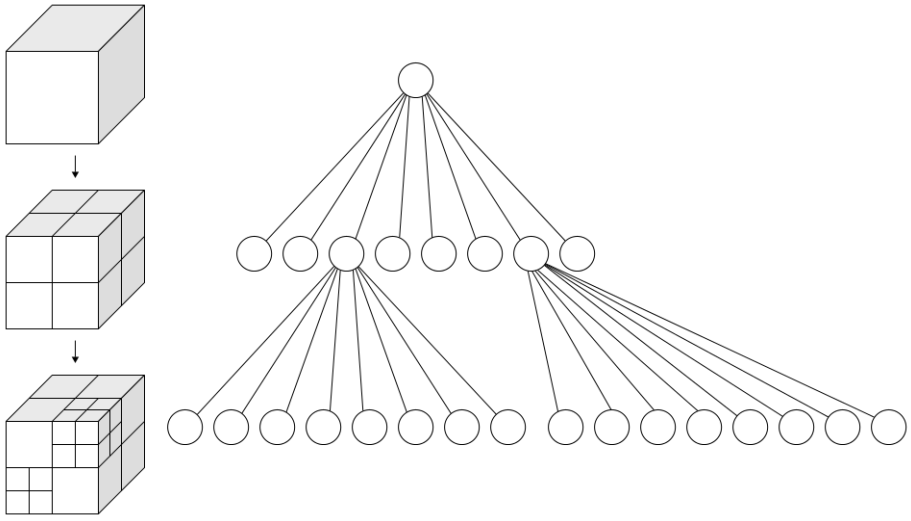

Figure 2.11: Schema of an octree. [Com07]

A modified version of k-d trees are **octrees**, which are specially used for three dimensional spaces [Mea82]. In contrast to normal k-d trees, each node represents a splitting point instead of a splitting dimension. In other words, each node stores three values (one for each dimension) in place of just one.

So each node has eight instead of two children. In the schema 2.11, one splitting point subdivides in eight smaller cubes. Using the normal k-d tree algorithm would require three steps (rather than one) for a similar partition. As an outcome of the subdivision, the root node represents one of the corner points of each child node.

2.5 ROS

The Robot Operating System (ROS) is a common software framework used in several robotic projects. It offers various services like hardware abstraction, package management and inter-process communication. Due to an active community, plenty software packages exist for different kinds of tasks. ROS has its origin within the project STAIR at Stanford University. During this project a predecessor of the ROS framework was invented [QBN+07]. Until today, ROS is used for all kind of robots, e.g. several robot assistants [BKM+11] [SFH+10], autonomous vehicles [BOG+08] as well as micro aerial vehicles (MAV) [MTH+12]

Figure 2.12: Communication between ROS nodes

ROS provides a modular based structure. Each component (sensor, visualizer, etc.) exists in a separated **node**. Communication between the components occurs by passing **messages**. Each node can publish messages to a given **topic**. Other nodes, which are interested in this topic, can subscribe to it. Figure 2.12 shows a simple publish-subscriber structure. The node */turtlesim* is listening to the topic */turtle1/command_velocity*, which is published by */teleop_turtle*. The structure of a message is defined in a static way. It composes different fields of primitive types, arrays of primitive types and/or other messages [QCG+09].

Besides nodes, ROS offers the ability to use **nodelets**. Nodelets behave similar to nodes, but have the benefit of zero copy cost between nodelets of the same nodelet handler. In other words, different algorithm run in different nodelets (within the same nodelet handler), without any copy or serialization cost between these nodelets.

An relevant package for the approach is the velodyne package [ros16]. It contains the three following sub-packages:

velodyne_driver: package containing driver for different Velodyne sensors

velodyne_msgs: defines Velodyne specific messages

velodyne_pointcloud: generates a point cloud from the sensor data

With this packages, it is possible to establish a connection to a Velodyne LiDAR sensor, receive the sensor data and create a point cloud out of it. Using this package in addition solves the problem with the proper motion of the vehicle described in section 2.1.1. The precondition is an existing odometry sensor on the vehicle. If it exists, the Velodyne package uses the last received odometry information to update the position of the sensor for each received package. Due to the high rate of odometry data, the errors based on the proper motion are reduced.

2.6 PCL

For different processing steps, the Point Cloud Library (PCL), which is fully integrated in ROS, is used. The introduction of cheap 3D sensors like the Microsoft Kinect is one of the main reasons for the demand of an open source library for point clouds and 3D geometry processing [RC11]. It supports a lot of different tasks like filtering, model fitting, segmentation, surface reconstruction, registration and feature estimation. Additionally it provides different tools for an increase in performance like OpenMP and Threading Building Blocks (TBB). Due to the integration in ROS, most algorithm can be executed as own nodelets.

Chapter 3

Implementation

To achieve the goal of an obstacle detection, the task is subdivided in multiple subtasks. Figure 3.1 shows the order of these subtasks. At the beginning a downsampling algorithm is used to reduce the big amount of received sensor data. Subsequently, a technique is needed to remove unnecessary data. In consequence, a random sample consensus algorithm (RANSAC) to eliminate the ground plane is used. Finally, the remaining elements of the origin point cloud are clustered to individual obstacles. Following the obstacle detection task, the obstacle tracking occurs, which uses the vehicle odometry as an additional source.

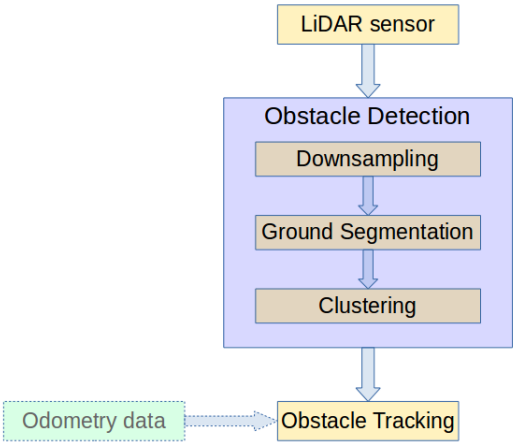

Figure 3.1: Subtasks for the obstacle detection. The tracking of the obstacles is a separated subsequent task. Additionally, it uses the vehicle odometry as further data source.

3.1 Downsampling

The received point cloud contains up to 1.3 million data points per second. Using all this data would be time-consuming and especially nearby measurements contain unnecessary details. For this reason, a voxel grid approach is used to downsample the number of data points. Within this filtering the input point cloud is segmented in smaller voxels (similar to smaller cube) with a fixed size. Now, for each voxel, the centroid of all contained data points will be determined and downsampled to the centroid. A two dimensional example is shown in figure 3.2. The chosen voxel size directly influences the amount and the position of the centroids.

With ongoing measurement distance, the distance between adjacent laser beams is growing linear. As a result, the voxel grid approach only works efficiently on nearby data points. An example is shown in figure 3.3. It illustrates a LiDAR scan with three parked cars on the right side and several trees on the centre strip. Comparing the detailed views of the tree in the back (a) and the tree nearby (b), is showing that the relative downsampling rate of tree (a) is worse than the one of tree (b). This behaviour results from the low details of the tree in the back previously.

In addition the downsample rate depends on the size of the voxel. As shown in figure 3.4, bigger voxels increase the downsampling rate considerably. Choosing a voxel size of 0.1 m reduces the amount of data points to approximately 50%. The raw input data were logged during a test drive at the university campus on a one-lane street with parking cars, trees and buildings nearby. This environment stands in opposition to a main street with more distance to the obstacles, where the downsampling rate would be less.
Depending on the application field, the results are more deviant. For example, at indoor obstacle detection the common distances are much smaller. In addition, the amount of data points is higher, due to the received data points of the walls. In contrast, part of the beams are not returned in outdoor areas, by virtue of the finite sensor range.

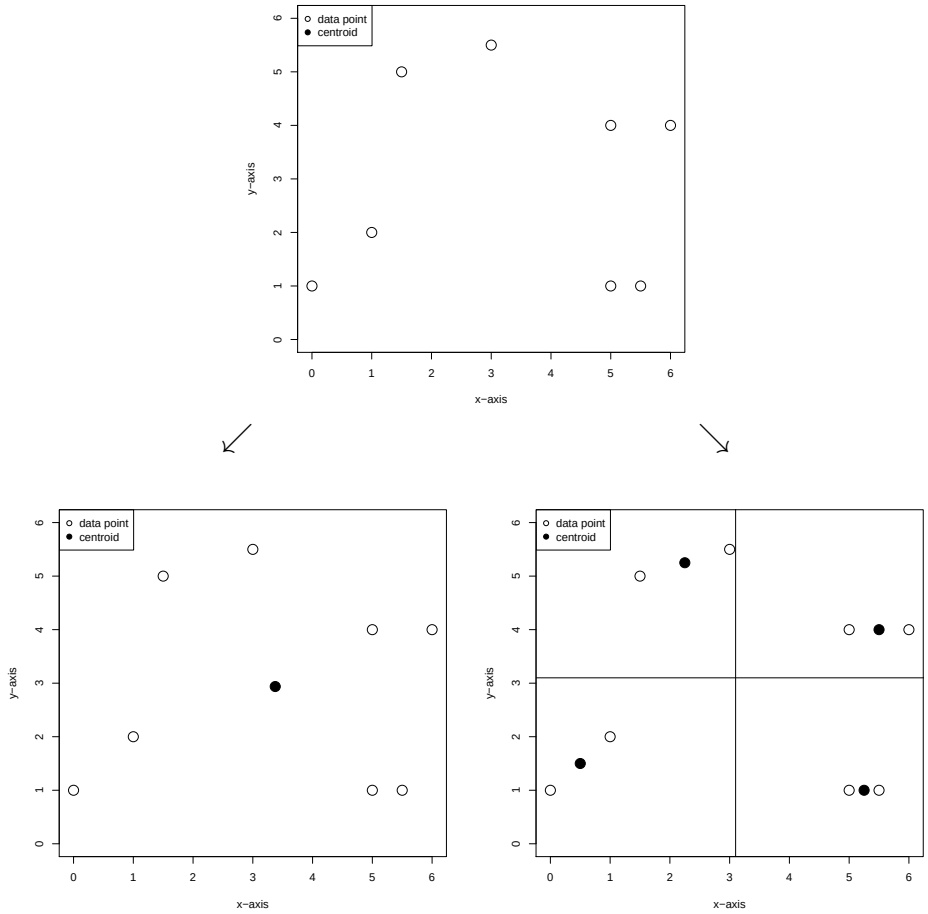

Figure 3.2: Example of a voxel filter. Black dots are the centroids of each voxel. Left example shows a big voxel size, where all data points are in the same voxel. Right example is a smaller voxel size, with four centroids.

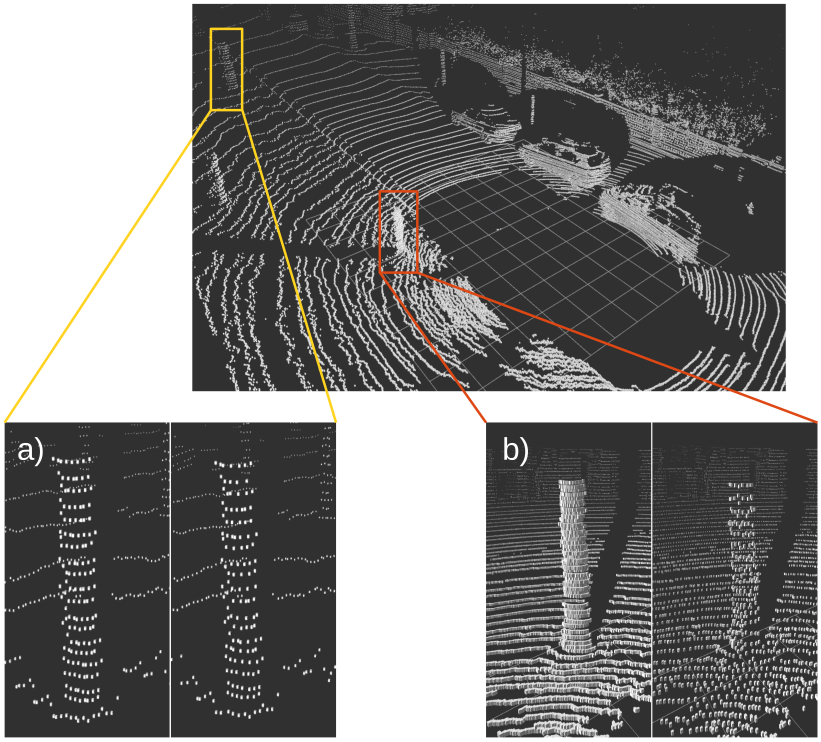

Figure 3.3: Example for the downsampling quality of a voxel grid filter on a) a tree further afar and b) on a tree nearby. The pictures on the left hand side show the input point cloud and the pictures on the right hand side illustrate the output of the voxel grid filter. The zoomed in pictures are turned approximately 90 °compared to the full picture.

3.2 Ground segmentation

After reducing the amount of input data, a function to remove unnecessary data is needed. That is why a separation between data points, which represent an obstacle, and data points, which are non-obstacle like the ground, is done. To simplify this task, the ground is assumed as an even plane. In other words, small elevations like curbside are ignored.

3.2.1 RANSAC

As theoretical approach a RANSAC algorithm is used to determine the data points, which are part of the ground. The idea of RANSAC was developed 1981 by Martin A. Fischler and Robert C. Bolles [FB81].

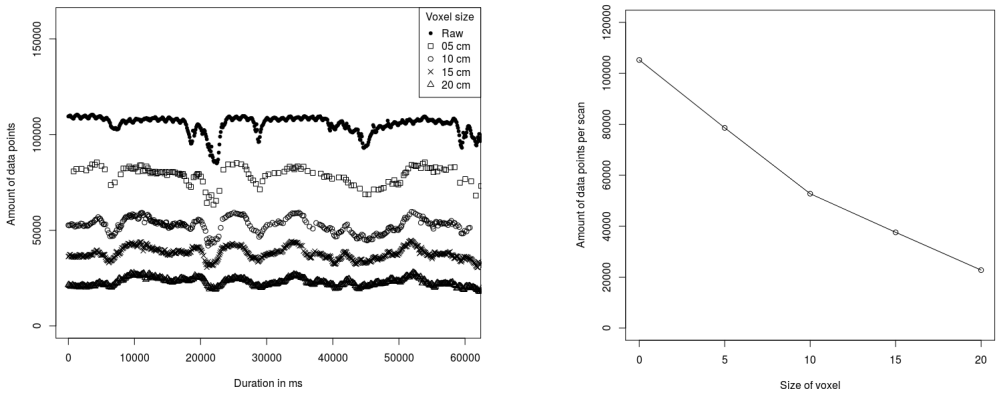

Figure 3.4: Downsampling rate of voxel filters. Left side shows the quantity of data points using different voxel sizes on the same scenario. Right side illustrates average amount of data points depending on the different size of voxels.

It is used to adjust a mathematical model to observed data, even if a lot of outliers or measuring errors exist. It contains following the steps:

1. Select randomly n unique data points from P, where P = Set of all data points, n = Minimum of data points to assign each free parameter and $n \leq \#(P)$
2. Create a mathematical model x with the selected n data points
3. $\forall p \in P$: Test if p fits in the model x ($\pm$ some threshold d)
4. Save the quantity of matching p as quality value for current model x
5. Repeat steps 1. - 4. to find a better model x

Following [Rus09] the number of iterations k to select n good data points depends on three factors:

- ϵ = probability to select a data point, which does not fit the model x
- n = number of selected data points
- q = desired probability of success

Knowing this probabilities, it is easy to calculate the opposite likelihoods. $(1 - \epsilon)$ is the probability to select at least one good selection. The chance to get n good selection is therefore $(1 - \epsilon)^n$. Repeating it k times, produces for

the likelihood of failure the formula $(1-(1-\epsilon)^n)^k$. Equated with $(1-q)$ for the desired probability of failure, produces:

$$(1-q) = (1-(1-\epsilon)^n)^k \tag{3.1}$$

Following this, receives for the number of iterations k:

$$k = \frac{\log(1-q)}{\log(1-(1-\epsilon)^n)} \tag{3.2}$$

3.2.2 Modifications of RANSAC

Apart from the normal RANSAC algorithm, several modified versions were published. One of them is the MLESAC algorithm [TZ00], which uses a different calculation of the quality value. In the classical RANSAC the quantity of matching data points is used as quality value. Precondition for a matching data point is a deviation smaller than the threshold. Under the presumption that a bad threshold is chosen and every data point is matching, each model would have the same quality value. By contrast, the MLESAC takes the deviation of matching data points from the model into consideration. Thus, data points with higher deviation influence the quality value proportionally.

Another enhancement is the local optimized RANSAC (LO-RANSAC) algorithm , which adds an optimization step after finding a group of matching data points. Within this step the matching data points are used to polish an optimized mathematical model by using a least square method. Afterwards, the data points are tested against this new model. As a result of this modification the number of iteration steps is decreased, while simultaneously the number of detected inliers increases. [CMO04]

A randomized RANSAC algorithm (RRANSAC) is proposed by Chum et al. [CM02]. Within their approach, they use a preverification step with a randomly chosen small number of all data points d, after the normal RANSAC algorithm created a mathematical model. Only if this preverification passes, the model quality with all data points is calculated. Through this modification, bad mathematical models are not compared to all data points, which leads to a large decrease of comparisons in total.

Once a good ground estimation x is received, all data points fitting this model are removed. The remaining data points are expected to be objects or

measurement noise. In practice, the initial simplification (ground is an even plane) causes some misinterpretation, especially on rise or slope of the road and next to the road. One solution is to increase the distance threshold, but this results in loosing of small obstacles. Another solution could be the separation of the point cloud into smaller cubes, while making an independent ground estimation for each of them. But choosing too small cubes causes wrong estimations, based on no contained ground data points in this particular cube. A good amount are four to eight cubes, depending on the scope.

Petrovskaya and Thrun presented a third solution to increase the accuracy [PT09]. Their idea is based on using another laser range sensor for the ground estimation. The IBEO Alasca, which is a preceding model of the IBEO Lux sensor, is mentioned as positive example. Through its four parallel horizontal scan lines, it is easy to mark, which of them are likely to hit the ground.

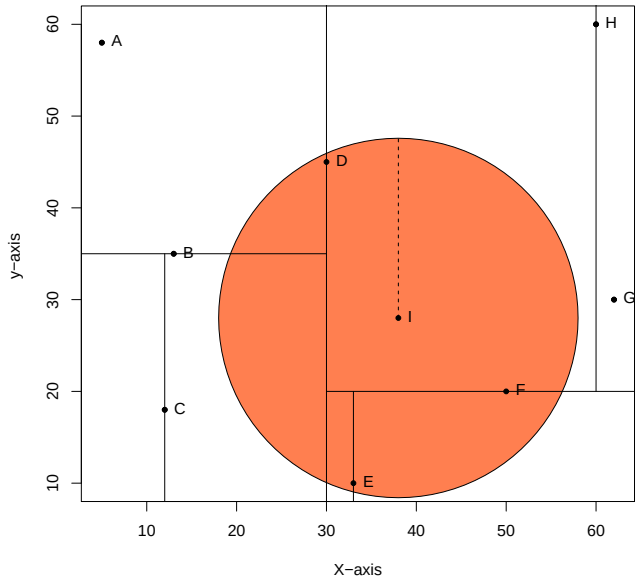

Figure 3.5: Example of a range search in a k-d tree. The search radius surrounding point I intersects with several areas, which are part of other subtrees.

3.3 Clustering

A range search algorithm based on the euclidean distance of the points is used, under the presumption that dense points represents the same object. As described in section 2.4 the remaining unsorted points are arranged in a k-d tree structure P. With this data, following steps are performed as described in [Rus09]:

1. Create an empty list of clusters C and a queue Q, which contains points, that are not already finished
2. $\forall\ p_i\ \in P$ do following sub-steps:
   (a) add p_i to queue Q
   (b) $\forall\ q_j\ \in Q$ do following steps:
      i. find all points q_k in a radius r around q_j and add them to a list R
      ii. $\forall\ r_l\ \in R$: add r_l to Q, if r_l is not already processed
   (c) after processing all points in Q, add Q the cluster list C
   (d) reset Q to an empty list

As a result of these steps, a list of clusters is received. Each cluster contains points representing one specific obstacle. Hereby, the most complex task is to find all points in a specific radius around q_j. The points in the k-d tree are sorted by their position. However using a range search or nearest neighbour approach on one specific node, often leads to an overlap with another subtree (see Figure 3.5). On that account, a distinction between three cases has been made. Firstly, the search area does not intersect with other subtrees. Secondly, a subtree is a complete part of the search area. Thirdly, the subtree and the search area intersect partly . The pseudo code in algorithm 2 considers this three cases, based on [DBVKOS00].

Defining a good, practicable range value for the clustering is no easy task. On the one hand, undersized values cause a breakdown of one large obstacle in multiple small clusters, on the other hand, oversized values cause a fusion of several closely related obstacles into one big obstacle (see figure 3.6).
Further, sensor noises causes sometimes small clusters containing only few data points. This problem is solved by defining a minimum number of points for each cluster.

Algorithm 2: Algorithm for range search in k-d tree

Input : root node q and range r
Output: list of all in-range nodes l

```
1  begin
2      if q is a leaf then
3          add q to l if it lies in range r;
4          return l;
5      else
6          if region of q.leftchild() completely in range r then
7              add all nodes in q.leftchild() to list l;
8              return l
9          else
10             if reqion of q.leftchild() and range r partly intersect then
11                 Do a recursive range search on q.leftchild();
12                 add result to list l
13             end
14         end
15         if region of q.rightchild() completely in range r then
16             add all nodes in q.rightchild() to list l;
17             return l
18         else
19             if reqion of q.rightchild() and range r partly intersect then
20                 Do a recursive range search on q.rightchild();
21                 add result to list l
22             end
23         end
24     end
25     return l
26 end
```

3.4 Tracking / Kalman Filter

After defining the steps to receive each obstacle of one measurement, recovery of the same obstacles in the next measurement is needed. To discover an obstacle and introduce it to the cars knowledge is already the first step, but for complex tasks like a dynamic path planning algorithm, more information like moving speed or direction of the obstacle is needed. This information is received by recovering obstacles and comparing the parameter modification. For the realization in ROS, a proprietary communication message is defined. To receive the information for this message a linear Kalman filter is used [Kal60]. Named after Rudolf E. Kálmán, the Kalman filter provides a good prediction of the system state, even with noised measurement input. Since its publication in the early 60s, the algorithm has been used in several areas, e.g. Apollo project of the National Aeronautics and Space Administration (NASA) [MS85].

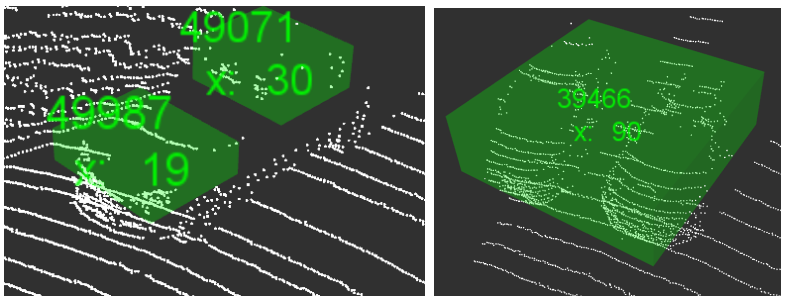

Figure 3.6: Example of different clustering problems. a) Undersized range causes separation of one obstacle in two different small clusters. b) Oversized range causes fusion of two closely related obstacles into one cluster.

The Kalman filter is based on two main steps. First, a prediction step, which gives an estimation of the current system state, based on the old system state, previous measurements and known noise. Second, an update step, which compares the predicted system state with the current measurement. The second step provides information about the prediction quality and the measurement quality. The following explanations are based on [WB06]

3.4.1 Prediction step

The prediction step contains the following two functions:

$$\hat{x}_k^- = A\hat{x}_{k-1} + Bu_{k-1} \tag{3.3}$$

$$P_k^- = AP_{k-1}A^T + Q \tag{3.4}$$

The function 3.3 gives an estimation of the state $\hat{x}_k^-$ based on a previous state $\hat{x}_{k-1}$, a state transition matrix A, an optional control vector u_{k-1} and a control-input matrix B. Due to the fact, no control can be performed on the tracked obstacles, the control vector u_{k-1} is not needed. Therefore, function 3.3 can be reduced to the following function:

$$\hat{x}_k^- = A\hat{x}_{k-1} \tag{3.5}$$

The matrix A defines the transition between the system state $\hat{x}_{k-1}$ and $\hat{x}_k$. Assuming that a system state describes the two dimensional position and velocity of an obstacle as

$$x_k = \begin{pmatrix} x \\ y \\ v_x \\ v_y \end{pmatrix} \tag{3.6}$$

the corresponding transition matrix A is

$$A = \begin{pmatrix} 1 & 0 & \Delta t & 0 \\ 0 & 1 & 0 & \Delta t \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{3.7}$$

With ongoing prediction steps, the Kalman filter gets more insecure with its estimations. This uncertainty is expressed in function 3.4. Q is the so called process noise covariance matrix, which is added to the error covariance matrix P^-. As a result, the error covariance changes, even if a zero initial error covariance matrix P_0 was chosen.

3.4.2 Update step

Once a new measurement z_k is received, the Kalman filter executes the update step to compare the estimation $\hat{x}_k^-$ with the current measurement z_k. This steps consist of the three following functions:

$$K_k = P_k^- H^T (H P_k^- H^T + R)^{-1} \tag{3.8}$$

$$\hat{x}_k = \hat{x}_k^- + K_k(z_k - H\hat{x}_k^-) \tag{3.9}$$

$$P_k = (I - K_k H) P_k^- \tag{3.10}$$

The function 3.8 is used to calculate the so called Kalman gain K_k. Among others, this calculation uses the measurement noise matrix R and the error covariance matrix P. Depending on these values the following update step relies mutually on the current measurement z_k. A derivation for the Kalman gain K_k can be found subsequently.
The matrix H is the observation matrix, which maps the estimation $\hat{x}_k^-$ in the measured space of measurement z_k (see formula 3.9). In this function occurs the update of the current system state. $z_k - H\hat{x}_k^-$ calculates the current deviation between the prediction $\hat{x}_k^-$ and the measurement z_k. Depending on the Kalman gain, this deviation affects the current estimation $\hat{x}_k^-$ to a greater or lesser extent. As a result, the Kalman filter gives a good estimation, even if the measured values are very noisy.
The last function 3.10 is the update step of the error covariance matrix P. Depending on the Kalman gain K_k the matrix values decrease.

3.4.3 Influence on the Kalman gain

The equation 3.11 depends on two uncertainties: On the one hand, the measurement noise matrix R and on the other hand, the error covariance matrix P_k^-.

$$K_k = P_k^- H^T (H P_k^- H^T + R)^{-1} \tag{3.11}$$

$$K_k = \frac{P_k^- H^T}{H P_k^- H^T + R} \tag{3.12}$$

With formula 3.12, the limits $R_k \to 0$ and $P_k \to 0$ are calculated.

$$\lim_{R_k \to 0} \frac{P_k^- H^T}{H P_k^- H^T + R} = H^{-1} \tag{3.13}$$

$$\lim_{P_k \to 0} \frac{P_k^- H^T}{H P_k^- H^T + R} = 0 \tag{3.14}$$

Keeping this in mind, while looking at the measurement update function 3.15, for $\lim_{P_k \to 0}$ the function is shortened to $\hat{x}_k = \hat{x}_k^-$. Confirming, that for a small error covariance P_k, the update function relies primary on the prediction value $\hat{x}_k^-$. On the opposite, a small measurement noise R causes more trust in the measurement value and as a consequent the update function gives more weight to the calculated deviation $z_k - H\hat{x}_k^-$.

$$\hat{x}_k = \hat{x}_k^- + K_k(z_k - H\hat{x}_k^-) \tag{3.15}$$

3.4.4 Match function

Centroid matching

So far the task to find the matching measurement was ignored. With each sensor message, almost 50-60 obstacles are extracted. A first solution uses the euclidean distance between the current track position and the new measurement. Even if this solution yield good results, there are a lot of situations, which are not covered.

As mentioned in section 2.2 there are situations, where only one side of an obstacle can be detected. Due to the proper movement, and sometimes also the movement of some obstacles, transitions appear, when suddenly another side of an obstacle (see figure 3.7) shows up. In the same moment an additional shift in the centroid position occurs. While the shift distance for cars

and pedestrian is smaller then one meter; for bigger obstacles like trucks this shift can be up to four or five meters. This position jump can cause also the loss of an active track.

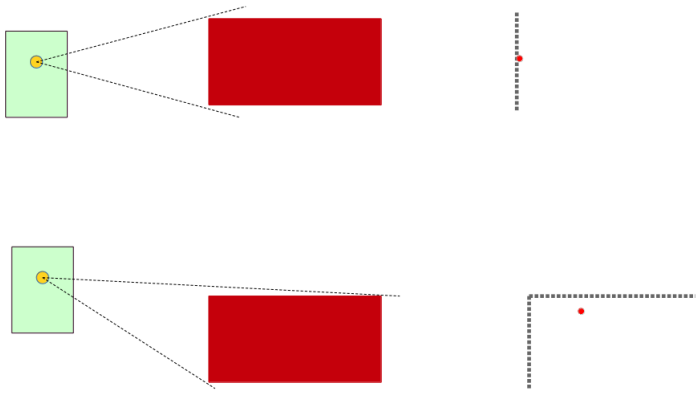

Figure 3.7: Left side: Position change of green car with sensor relatively to the red obstacle. Right side: Resulting centroid shift within the obstacle point cloud. Red dot represents the centroid.

Edge matching

A more complex match function approach compares the obstacles edges and detects occurred transitions additionally. For this reason, a rectangular bounding box is fitted in each obstacle and their edges are compared. For simplification, comparison is done in two dimensions. This step results in one edge with a minimal shift at least. In the example of figure 3.7, it is the nearest edge to the autonomous vehicle. But, using this approach leads to possible mismatches on several close by obstacles. The front edge of one obstacle could be matched to the back edge of the second obstacle, which is standing in front of the first. Therefore, the vertexes of the obstacles are enumerated and a distinction between different types of transitions is made, too.

The enumeration is done in a polar coordinate system. Each point in this coordinate system is defined by (r, Φ), where r is the radius from a reference point and Φ an angle from a reference direction. The reference point is the sensor position and the reference direction is the x-axis. Hence the x and y

coordinates of each vertex are converted by using the arctan2 function, which is a variation of the classical arctangent function. The converting formulas are shown in 3.16 and 3.17.

$$r = \sqrt[2]{x^2 + y^2} \tag{3.16}$$

$$\Phi = arctan2(x, y) = \begin{cases} \arctan(\frac{y}{x}) & \text{if } x > 0 \\ \arctan(\frac{y}{x}) + \pi & \text{if } x < 0 \text{ and } y \geq 0 \\ \arctan(\frac{y}{x}) - \pi & \text{if } x < 0 \text{ and } y < 0 \\ \frac{\pi}{2} & \text{if } x = 0 \text{ and } y > 0 \\ -\frac{\pi}{2} & \text{if } x = 0 \text{ and } y < 0 \\ \text{undefined} & \text{if } x = 0 \text{ and } y = 0 \end{cases} \tag{3.17}$$

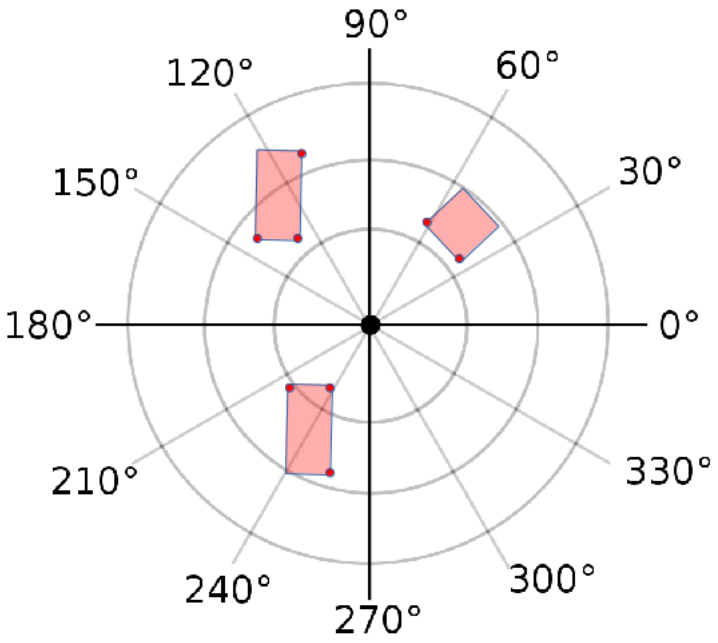

Figure 3.8: Obstacles in polar coordinate system. The reference point is the sensor position. The red dots illustrate the visible vertexes.

Figure 3.8 illustrates the position of the obstacles vertexes in a polar coordinate system. Obviously, the rectangular bounding box causes that the vertex with the largest distance r is always covered by nearer edges. The situation of two covert edges only occurs, if the angular values Φ of the two vertexes

in the back are between the angular values of the two vertexes in the front. In that case, both vertexes are covered by just one edge.
The enumeration order is based on the angle Φ and on the distinction of covert/ not covert vertexes. First, the visible vertexes are enumerated in increasing order to their angular. Subsequent, covert vertexes are enumerated in the same increasing order.

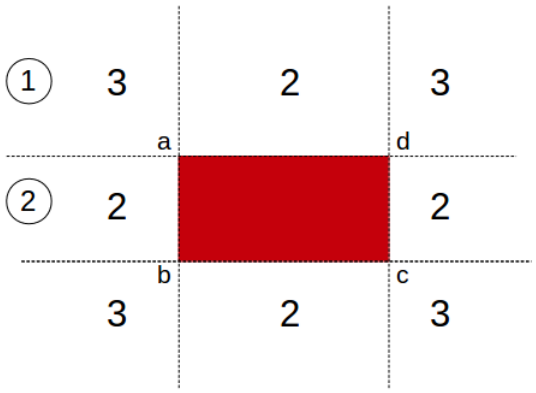

Figure 3.9: Amount of visible obstacle edges depending on own relative position. Circled numbers are starting points for the transition distinction. Letters are used to enumerate the vertex.

In figure 3.9, the amount of visible vertexes is shown. Depending on the position change of the sensor relatively to an obstacle, the amount varies between two or three visible vertexes. In consequence, the number of possible transitions is finite. Basically there are nine possible transitions. In figure 3.9 the starting points mentioned below are outlined. The possible transitions are:

1. Starting at 1. and detect two visible edges at the next measurement
   (a) Transition to the right and loosing vertex b
   (b) Transition down and loosing vertex d
2. Starting at 1. and detect three visible edges at the next measurement
   (a) No transition
   (b) Transition two fields down, loosing vertex d and detecting c

3. Starting at 2. and detect two visible edges at the next measurement
   (a) No transition
   (b) Transition diagonal down, loosing vertex a and detecting vertex c
   (c) Transition diagonal up, loosing vertex b and detecting vertex d
4. Starting at 2. and detect three visible edges at the next measurement
   (a) Transition up and detecting vertex d
   (b) Transition down and detecting vertex c

Regarding this list of possible transitions, the appearance or disappearance of an obstacle side can be taken into consideration. As a result, the tracking algorithm will be more stable when just taking the centroid position into account.

Chapter 4

Evaluation

In this chapter the received results are evaluated. While doing so, the results are distinguished between the accuracy of the obstacle detection and the obstacle tracking. As comparative value, the received results of the IBEO Lux sensor are used.

4.1 Setup

To receive comparable data, several test drives with the autonomous car MIG were done. As described in section 2.1, the car is equipped with the Velodyne HDL-64E and the IBEO Lux sensors, among others. The sensor's update rates were 10 Hz (HDL-64E) and 12.5 Hz (IBEO Lux). During the test drives, the raw sensor data on different traffic situations were recorded. Afterwards, the recorded raw data was replayed on an office PC and the algorithms for the obstacle detection and tracking were applied and evaluated. The possibility to replay special situations is one of the main advantages of the offline evaluation. So, the behaviour of the system with various modifications on identical situations can be tested.

For the evaluation, the following situations were under closer examination:

1. Driving on a main road with static obstacles and two-way traffic
2. Stopping at a traffic light and starting again
3. Driving through a small one-lane side road, with several static obstacles nearby

The following aspects are compared for each situation:

1. Amount of detected obstacles of different ranges
2. Average deviation of obstacle centroids of different ranges
3. Average age of tracks of different ranges

4.2 Results

The following results are sorted by the previous mentioned aspects, starting with the specific quantitative results. Subsequently the qualitative aspects are presented.

4.2.1 Amount of detected obstacles

The aim of the first experiment was to receive a quantitative overview of the two obstacle detection algorithms. It does not contain any information about accuracy. For each scan, the detected obstacles are counted depending on their distance. The charts in figure 4.1 are showing the average amount of obstacles for each scenario.

As expected, with ongoing distance, an increase of the obstacle amount is perceptible. Simultaneously, also the deviation is increasing between both sensors. Figure 4.2 illustrates the deviation between the IBEO Lux and the HDL-64E sensor for each scenario. For this increase, the main reason is the differing treatment of obstacles on the side walk.
Both algorithms are based on the idea of matching a rectangular into the data points of an obstacle. While this works well on obstacles like cars, it is not a good solution for shrubs and other plant obstacles, which have a more complex shape. Additionally, partial occultation causes unpredictable changes in the shape of obstacles. An approach to overcome this phenomenon is shown in the outlook of this thesis. Unlike the obstacle detection of the HDL-64E, the algorithm for the IBEO Lux fuses unstable small obstacles to one bigger and more stable obstacle. As a result one obstacle of the IBEO Lux is represented by three smaller obstacles of the HDL-64E, which leads to the deviation of the obstacle amount between both sensors. Even if the algorithm for the IBEO Lux sensor is more stable, the presented algorithm for the HDL-64E sensors contains more details for smaller obstacles.

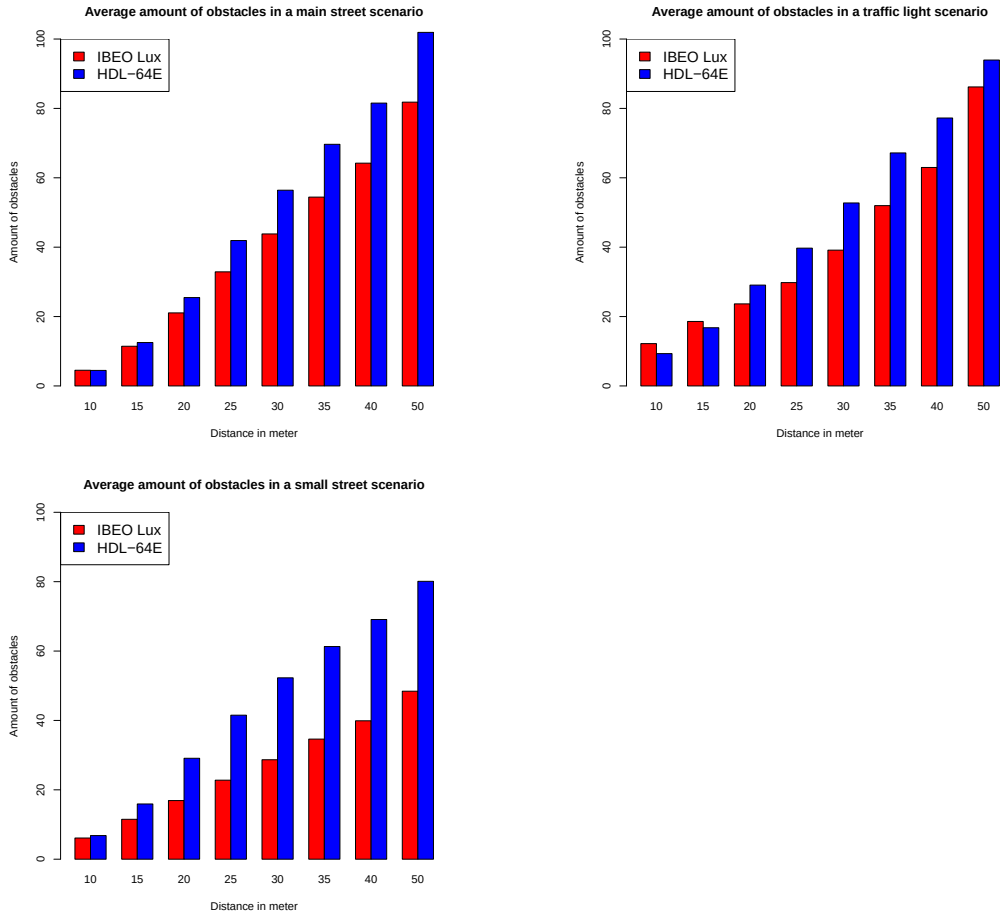

Figure 4.1: Amount of total detected obstacles for the three scenarios. For each scenario, the amount of obstacles is distinguished by its distance to the vehicle.

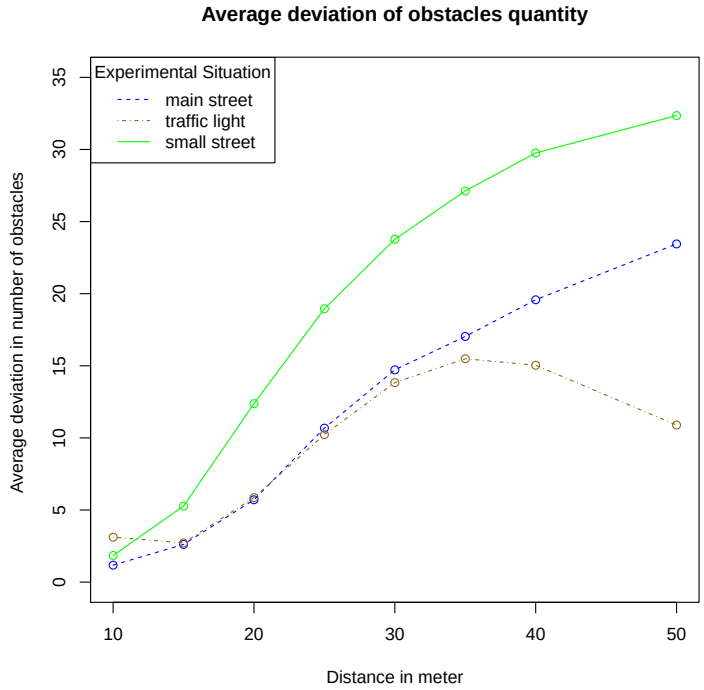

Figure 4.2: Average deviation in quantity of detected obstacles between the HDL-64E and the IBEO Lux

4.2.2 Average deviation of obstacle centroids

Within this experiment, the quality of the obstacle detection will be evaluated. Therefore, the centroid distance of an obstacle between both sensors are compared. As precondition, the identical obstacles in both sensor detections need to be found. The search area is limited up to two meters. Figure 4.3 shows the number of total matches for different distances. The first three graphs distinguish between positive and negative matches. Based on this, the fourth figure summarizes the relative amount of positive matches for each scenario. With ongoing distance, the relative amount of these positive matches decreases.

The deviation of the obstacle centroids between both sensors is determined for positive matches. Figure 4.4 illustrates the average centroid deviation for all scenarios. Two of these scenarios have an increase in the disparity for ongoing distance. The third situation (stop at a traffic light) has a decreasing diversion. This behaviour results from a different behaviour of the own vehicle compared to the other scenarios.

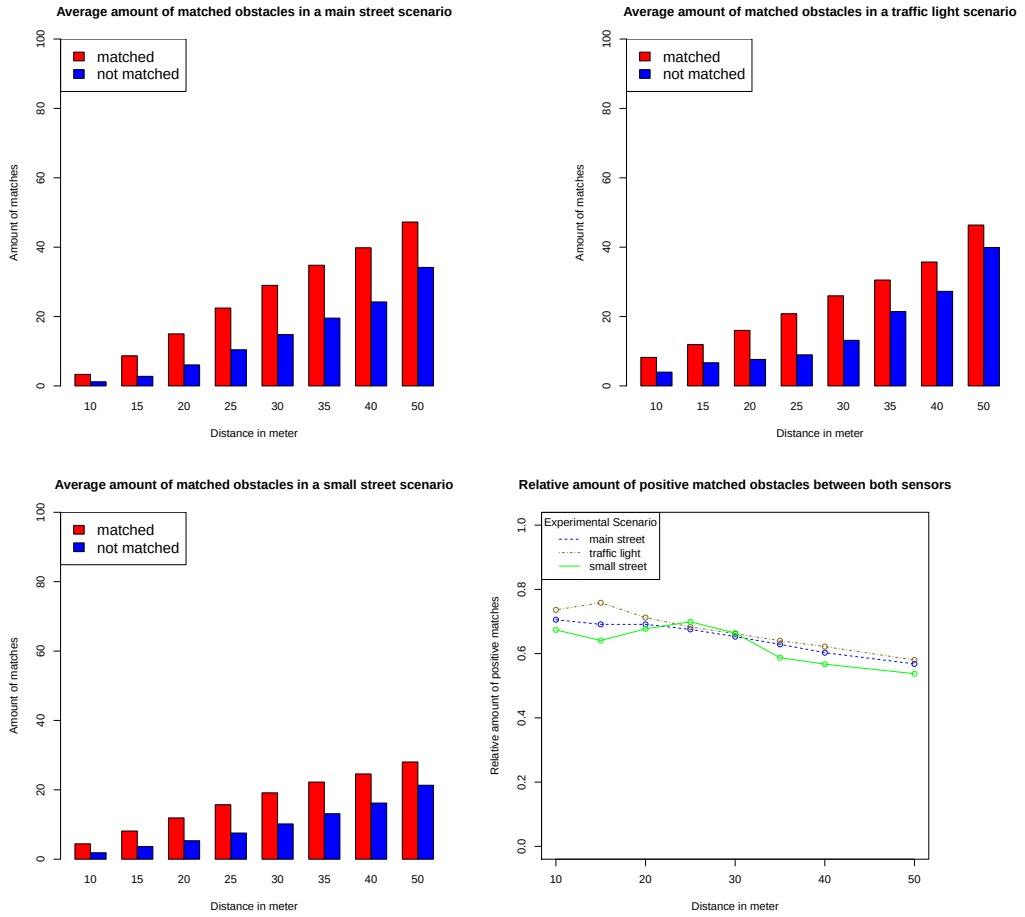

Figure 4.3: Amount of matched obstacles between both sensors. For a positive obstacle match, the centroid deviation of an obstacle has to be less then two meter.

Stopping at a traffic light involves a time period, when the vehicle stands still. This missing proper motion leads to an increased accuracy. However, the other scenarios (driving on a main street, driving on a small street) do not contain a stop of the vehicle.

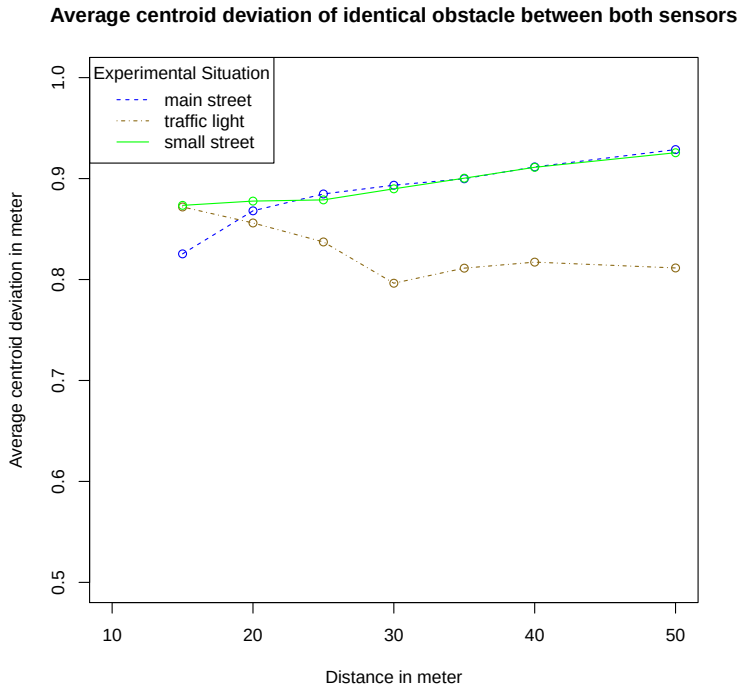

Figure 4.4: Average deviation of an identical obstacle centroid between the HDL-64E and the IBEO Lux.

4.2.3 Average age of obstacle tracks

While the previous experiments target the obstacle detection, the third experiment aims at the accuracy of the obstacle tracker. In that event, the age of the tracked obstacles is compared. An unreliable tracker often loses obstacles and re-detects them into new tracks. Within this re-detection the age of the track is reset. Hence, a higher average age of tracks implies a more reliable obstacle tracker.

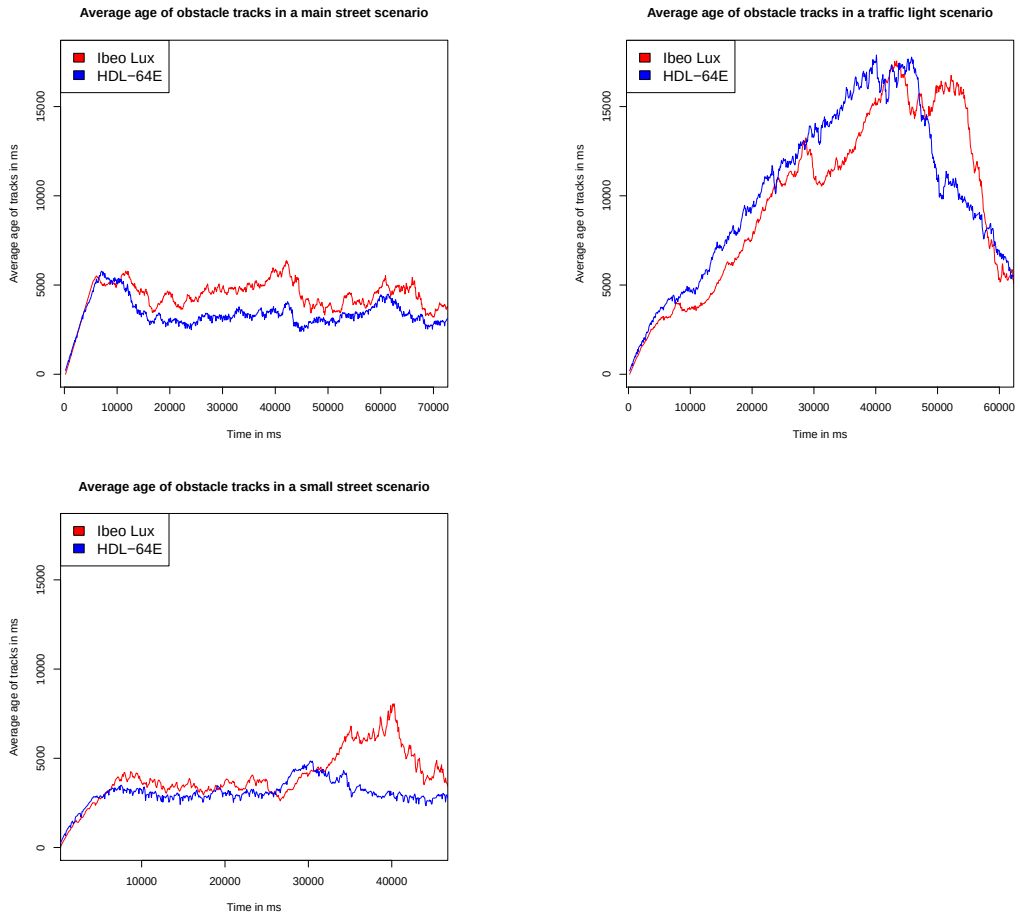

Figure 4.5: Average age of obstacle tracks between both sensors

Scenario	IBEO Lux	Velodyne
main street	4388,0 ms	3397,4 ms
traffic light	9364,0 ms	9839,5 ms
small street	3953,1 ms	3060,5 ms

Figure 4.6: Summarized average track age per each scenario

Figure 4.5 illustrates the average age of tracks for all three scenarios. Again, the difference between a static situation (stop at a traffic light) and a dynamic situation is clearly discernible. Comparing both dynamic scenarios, it is obvious that the algorithm for the IBEO Lux is more reliable than the approach of this thesis. As visualized in table 4.6, the average age for tracks in dynamic situations is approximately 25% smaller as track ages of the IBEO Lux algorithm.

The better results for the Velodyne HDL-64E on the traffic light scenario relies partly on the sensor position on top of the vehicle, whereas the Ibeo Lux sensors are mounted near the bumper. During this scenario several vehicles are standing nearby, whose block the field of view of the Ibeo Lux sensors. However, the field of view of the HDL-64E goes above this vehicles and let the sensor track obstacles behind the vehicles as well.

Chapter 5

Conclusion

In the course of the present thesis, the required steps to receive multi obstacle tracks based on the input data of a multi beam LiDAR sensor are examined. The first step is a downsampling task, which reduces the received data set to approximately 50 %. Afterwards, a plane segmentation based on a RANSAC algorithm is carried out. On the remaining data points a clustering algorithm is used to create several obstacles. These created obstacles are the base for the obstacle tracking algorithm, which additionally uses the vehicle odometry. The tracking algorithm is based on a linear Kalman filter.
As part of the evaluation, several aspects of the algorithm are examined. Comparing the quantity of the obstacle detection algorithm, shows a higher amount of obstacles on all test scenarios for the presented algorithm. Whereas, comparing the tracking age of the obstacle tracks, illustrates more stability for the tracking algorithm of the IBEO Lux sensor. In dynamic situations the age of tracks is approximately 25 % higher as the tracking age of the presented algorithm. Only in the traffic light scenario the average tracking age of the presented algorithm is higher. This is partly based on the different mounting positions of the sensor.

For future works, one task would be to solve the task of partial occultation, which would improve the accuracy of the presented algorithm remarkably. Tracks on the sidewalk often get lost and are detected as different obstacles afterwards. Main reason for this phenomenon, are partial occluded obstacles by smaller nearby obstacles (see figure 5.1). As a result, the clustering algorithm splits rear obstacles in several smaller obstacles, which can not be matched to any existing tracked obstacle.

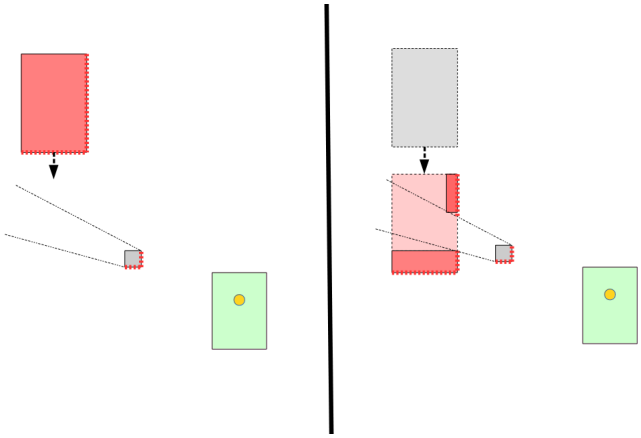

Figure 5.1: Example of partial occultation due to a blind spot, caused by a nearby obstacle.

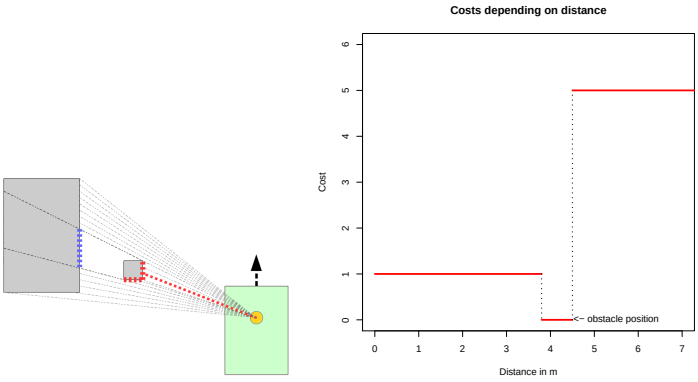

Figure 5.2: Individual beams hitting two obstacles. Blue dots represent expected distance values for several beams, red dots represent measured distance of the beams. Due to differential weighting the shorter beams influence the cost function less.

A solution to overcome this problem is shown in [PT09]. Their matching approach distinguishes from the presented approach fundamentally. The approach of this thesis uses the distance of obstacle centroids and bounding-box of two obstacles as a matching function and performs a measurement update on a valid match. But their matching approach is based more on the independent beam model [Mor88]. Thereby, each ray beam is treated as an independent measurement and tracked obstacles are compared directly to the ray beams. For performance reasons, adjacent beams with the same range are represented as cones, instead of individual lines. Based on this information the shape and the position of already known tracked obstacles are matched to corresponding beams. The difference between the actual beam distance and the expected beam distance is used as cost function. Here, the deviations are weighted differentially. If the actual beam distance is much shorter as expected, then the beam has hit a nearby obstacle. This deviation is weighted low. But, if the actual beam passes by the tracked obstacle and is longer as expected, the deviation weights higher (see figure 5.2). As a result, the influence of the partial occultation through nearby obstacles is considerable reduced.

The algorithm of this thesis appears as a good basis for future work in the context of autonomous driving. Each required step, from receiving a point cloud of sensor data, to create a stable obstacle tracking is presented. On these steps, future thesis and publications, which solve more specific tasks, can be developed.

Bibliography

[app] Specifications of Applanix POS LV 510, retrieved September 05 2016, from `http://www.applanix.com/pdf/specs/POSLV_Specifications_dec_2015.pdf`.

[Ben75] Jon Louis Bentley. Multidimensional Binary Search Trees Used for Associative Searching. *Communications of the ACM*, 18(9):509–517, 1975.

[BFK+08] Jonathan Bohren, Tully Foote, Jim Keller, Alex Kushleyev, Daniel Lee, Alex Stewart, Paul Vernaza, Jason Derenick, John Spletzer, and Brian Satterfield. Little Ben: the Ben Franklin Racing Team's Entry in the 2007 DARPA Urban Challenge. *Journal of Field Robotics*, 25(9):598–614, 2008.

[BKM+11] M Beetz, U Klank, A Maldonado, D Pangercic, and T Ruhr. Robotic Roomates Making Pancakes - Look Into Perception-Manipulation Loop. In *IEEE International Conference on Robotics and Automation*, pages 2587–2592, 2011.

[BOG+08] Patrick Beeson, Jack O'Quin, Bartley Gillan, Tarun Nimmagadda, Mickey Ristroph, David Li, and Peter Stone. Multiagent Interactions in Urban Driving. 2008.

[CM02] Ondrej Chum and Jirı Matas. Randomized RANSAC with Td,d test. In *Proc. British Machine Vision Conference*, volume 2, pages 448–457, 2002.

[CMO04] Ondrej Chum, Jirı Matas, and Stepan Obdrzalek. Enhancing RANSAC by Generalized Model Optimization. In *Proc. of the ACCV*, volume 2, pages 812–817, 2004.

[Com07] Wikimedia Commons. Schematic drawing of an octree, 2007.

[Com08] Wikimedia Commons. A basic LiDAR system with a single beam, 2008.

[dar07] DARPA Urban Challenge Rules, retrieved June 25 2016, from `http://archive.darpa.mil/grandchallenge/docs/Urban_Challenge_Rules_102707.pdf`, 2007.

[DBVKOS00] Mark De Berg, Marc Van Kreveld, Mark Overmars, and Otfried Cheong Schwarzkopf. Computational Geometry. In *Computational Geometry*. Springer, 2000.

[FB81] Martin A Fischler and Robert C Bolles. Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography. *Communications of the ACM*, 24(6):381–395, 1981.

[GLWR13] Daniel Göhring, David Latotzky, Miao Wang, and Raúl Rojas. Semi-Autonomous Car Control Using Brain Computer Interfaces. In *Intelligent Autonomous Systems 12*, pages 393–408. Springer, 2013.

[GWSG11] Daniel Göhring, Miao Wang, Michael Schnürmacher, and Tinosch Ganjineh. Radar/Lidar Sensor Fusion for Car-Following on Highways. In *Automation, Robotics and Applications (ICARA), 2011 5th International Conference on*, pages 407–412. IEEE, 2011.

[IB06] Karl Iagnemma and Martin Buehler. Editorial for Journal of Field Robotics - Special Issue on the DARPA Grand Challenge. *Journal of Field Robotics*, 23(9):655–656, 2006.

[Kal60] Rudolph Emil Kalman. A New Approach to Linear Filtering and Prediction Problems. *Journal of basic Engineering*, 82(1):35–45, 1960.

[LUG16] Stefan Lange, Fritz Ulbrich, and Daniel Goehring. Online Vehicle Detection using Deep Neural Networks and Lidar based Preselected Image Patches. In *Proceedings of the IEEE Intelligent Vehicles Symposium, Gothenburg, Sweden*, 2016.

[MBB+08] Michael Montemerlo, Jan Becker, Suhrid Bhat, Hendrik Dahlkamp, Dmitri Dolgov, Scott Ettinger, Dirk Haehnel, Tim Hilden, Gabe Hoffmann, Burkhard Huhnke, et al. Junior: The Stanford Entry in the Urban Challenge. *Journal of field Robotics*, 25(9):569–597, 2008.

[MCH+08] Isaac Miller, Mark Campbell, Dan Huttenlocher, Frank-Robert Kline, Aaron Nathan, Sergei Lupashin, Jason Catlin, Brian Schimpf, Pete Moran, Noah Zych, et al. Team Cornell's Skynet: Robust Perception and Planning in an Urban Environment. *Journal of Field Robotics*, 25(8):493–527, 2008.

[Mea82] Donald Meagher. Geometric Modeling Using Octree Encoding. *Computer graphics and image processing*, 19(2):129–147, 1982.

[MO14] Katharine M.Johnson and William B. Oiumet. Rediscovering the lost archaeological landscape of southern New England using airborne light detection and ranging (LiDAR). *Journal of Archaeological Science*, 43:9–20, 2014.

[Mor88] Hans P Moravec. Sensor Fusion in Certainty Grids for Mobile Robots. *AI magazine*, 9(2):61, 1988.

[MS85] Leonard A McGee and Stanley F Schmidt. Discovery of the Kalman Filter as a Practical Tool for Aerospace and Industry. 1985.

[MTH+12] Lorenz Meier, Petri Tanskanen, Lionel Heng, Gim Hee Lee, Friedrich Fraundorfer, and Marc Pollefeys. PIXHAWK: A Micro Aerial Vehicle Design for Autonomous Flight using Onboard Computer Vision. *Autonomous Robots*, 33(1-2):21–39, 2012.

[NLK+09] Hoa G Nguyen, Robin Laird, Greg Kogut, John Andrews, Barbara Fletcher, Todd Webber, Rich Arrieta, and HR Everett. Land, Sea, and Air Unmanned Systems Research and Development at SPAWAR Systems Center Pacific. In *SPIE Defense, Security, and Sensing*, pages 73321I–73321I. International Society for Optics and Photonics, 2009.

[PT09] Anna Petrovskaya and Sebastian Thrun. Model Based Vehicle Detection and Tracking for Autonomous Urban Driving. *Autonomous Robots*, 26(2-3):123–139, 2009.

[QBN+07] Morgan Quigley, Eric Berger, Andrew Y Ng, et al. STAIR: Hardware and Software Architecture. In *AAAI 2007 Robotics Workshop, Vancouver, BC*, pages 31–37, 2007.

[QCG+09] Morgan Quigley, Ken Conley, Brian Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, and Andrew Y Ng. ROS:

an open-source Robot Operating System. In *ICRA workshop on open source software*, volume 3, page 5. Kobe, Japan, 2009.

[RC11] Radu Bogdan Rusu and Steve Cousins. 3D is here: Point Cloud Library (PCL). In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 1–4. IEEE, 2011.

[ros16] Velodyne package for ROS, retrieved June 13 2016, from `http://wiki.ros.org/velodyne`, 2016.

[RP15] Satish K Reddy and Prabir K Pal. Segmentation of Point Cloud from a 3D LIDAR using Range Difference between Neighbouring Beams. In *Proceedings of the 2015 Conference on Advances In Robotics*, page 55. ACM, 2015.

[Rus09] Radu Bogdan Rusu. *Semantic 3D Object Maps for Everyday Manipulation in Human Living Environments*. PhD thesis, 2009.

[SFH+10] Siddhartha S Srinivasa, Dave Ferguson, Casey J Helfrich, Dmitry Berenson, Alvaro Collet, Rosen Diankov, Garratt Gallagher, Geoffrey Hollinger, James Kuffner, and Michael Vande Weghe. HERB: a home exploring robotic butler. *Autonomous Robots*, 28(1):5–20, 2010.

[SZS+99] David E Smith, Maria T Zuber, Sean C Solomon, Roger J Phillips, James W Head, James B Garvin, W Bruce Banerdt, Duane O Muhleman, Gordon H Pettengill, Gregory A Neumann, et al. The Global Topography of Mars and Implications for Surface Evolution. *Science*, 284(5419):1495–1503, 1999.

[TG15] Hamma Tadjine and Daniel Göhring. Acoustic/Lidar Sensor Fusion for Car Tracking in City Traffic Scenarios. In *3rd International Symposium on Future Active Safety Technology Towards zero traffic accidents*, 2015.

[TMD+06] Sebastian Thrun, Mike Montemerlo, Hendrik Dahlkamp, David Stavens, Andrei Aron, James Diebel, Philip Fong, John Gale, Morgan Halpenny, Gabriel Hoffmann, et al. Stanley: The Robot that Won the DARPA Grand Challenge. *Journal of field Robotics*, 23(9):661–692, 2006.

[TZ00] Philip HS Torr and Andrew Zisserman. Mlesac: A new robust estimator with application to estimating image geometry. *Computer Vision and Image Understanding*, 78(1):138–156, 2000.

[vela] Data sheet of Velodyne HDL-64ES2, retrieved September 05 2016, from `http://velodynelidar.com/docs/datasheet/63-9194_Rev-D_HDL-64E_Data%20Sheet_Web.pdf`.

[velb] Technical draw of Velodyne HDL-64ES2, retrieved September 08 2016, from `http://velodynelidar.com/docs/drawings/86-0105%20REV%20A%20OUTLINE%20DRAWING%20HDL-64E%20S2.pdf`.

[WB06] Greg Welch and Gary Bishop. An Introduction to the Kalman filter. Department of Computer Science, University of North Carolina, 2006.