Large-scale probabilistic feature mapping and tracking for autonomous driving

Probabilistisches Feature Mapping und Tracking für autonomes Fahren

Master-Thesis von Christian Merfels

Mai 2014

TECHNISCHE UNIVERSITÄT DARMSTADT

Department of Computer Science
Intelligent Autonomous Systems

Large-scale probabilistic feature mapping and tracking for autonomous driving
Probabilistisches Feature Mapping und Tracking für autonomes Fahren

Vorgelegte Master-Thesis von Christian Merfels

1. Gutachten: Prof. Dr. Jan Peters
2. Gutachten: Prasanth Jeevan, M.Sc.

Tag der Einreichung:

Erklärung zur Master-Thesis

Hiermit versichere ich, die vorliegende Master-Thesis ohne Hilfe Dritter nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die aus Quellen entnommen wurden, sind als solche kenntlich gemacht. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Darmstadt, den 22. Mai 2014

(C. Merfels)

Abstract

Autonomous driving requires a precise vehicle localization which can be achieved by using specific maps. This creates the challenge of constructing a system that generates these maps at a large scale by fusing sensor data. The issue of producing maps was in a previous project addressed by a cumbersome and error-prone manual process which only yielded feature-based maps. The approach of this thesis is to fuse sensor data and automatically create grid- and feature-based maps with a novel integrated technique. For this purpose, a software system called gridmap is developed and a supplementary analysis framework for the resulting maps is established. This software is being used in autonomous driving projects for grid- and feature-based localization approaches in the U.S. and in Germany.

Zusammenfassung

Autonomes Fahren erfordert eine präzise Lokalisierung des Fahrzeugs, die durch das Verwenden von speziellen Karten erreicht werden kann. Das begründet die Notwendigkeit für ein System, das solche Karten für große Flächen durch Sensordatenfusion erstellt. In einem früheren Projekt wurden Karten per Hand erstellt, was sich als schwerfälliger und fehleranfälliger Prozess herausstellte, der zudem nur featurebasierte Karten produzieren konnte. Der Ansatz dieser Arbeit ist das Fusionieren von Sensordaten und das automatische Erzeugen von grid- und featurebasierten Karten anhand eines neuartigen Verfahrens. Für diesen Zweck wird eine Software namens gridmap erarbeitet und ein zusätzliches Framework zur Analyse der erstellten Karten entwickelt. Diese Software wird in Projekten zu autonomen Fahren für grid- und featurebasierte Lokalisierungsansätze in den USA und in Deutschland verwendet.

Contents

1 Introduction **4**
1.1 Motivation . . . 4
1.2 Problem statement and objective . . . 5
1.3 Outline . . . 6

2 Literature review **7**
2.1 Mapping in probabilistic robotics . . . 7
2.2 Mapping in autonomous driving . . . 8

3 Gridmap **11**
3.1 Theoretical aspects of probabilistic mapping . . . 11
3.1.1 Problem definition of probabilistic mapping . . . 11
3.1.2 Binary Bayes filter with static state . . . 12
3.1.3 Occupancy grid mapping algorithm . . . 14
3.1.4 Relation to Gaussian filters . . . 15
3.2 Platform . . . 17
3.2.1 Hardware . . . 17
3.2.2 Software . . . 18
3.2.3 Feature detectors . . . 19
3.3 Implementation . . . 22
3.3.1 Grid architecture . . . 22
3.3.2 Extensions for large-scale mapping . . . 23
3.3.3 Tracking . . . 27

4 Evaluation **30**
4.1 Accuracy analysis . . . 30
4.1.1 Experimental design . . . 30
4.1.2 Results . . . 33
4.2 Performance analysis . . . 39
4.2.1 Experimental design . . . 39
4.2.2 Results . . . 40

5 Discussion **45**
5.1 Known issues . . . 45
5.2 Outlook . . . 45
5.3 Conclusion . . . 47

List of figures **48**

List of tables **49**

Bibliography **52**

1 Introduction

Changes in the automotive industry in the past decades have typically been evolutionary in nature. The concept of a driverless car introduces a change of revolutionary impact to the way we think of mobility, creating a true paradigm shift. Ever since the general availability of cars for personal usage over a hundred years ago [1], the consumer has come to expect the ability to go anywhere at any time. When self-driving cars hit the market, their enhanced capabilities will expand consumers' expectations even further, specifically with regards to *how* their transportation needs are satisfied.

Self-driving cars have for a long time been parts of fictional books, films and television shows. Soon, they will no longer be confined to the realm of science fiction: almost every major car manufacturer has publicly announced plans to launch semi- to fully autonomous driving functionality before the end of the decade [2] [3]. Cars with partial autonomous capabilities are already on the market and recent public showcases demonstrated that even cars with near-production technologies are capable of covering several hundred kilometers of urban streets and highways in self-driving mode [4] [5].

The benefits of a widespread adoption of automated vehicles will be significant and numerous: increased safety and fewer road accidents, savings on fuel and emissions due to a more dynamically managed traffic flow, a higher productivity as commuting time can be used more efficiently, increased convenience while driving and many more [6].

This thesis was written at the Volkswagen Electronics Research Laboratory (ERL) in Belmont, California. The ERL is part of the Volkswagen Group of America and collaborates in research and development projects with the brands Audi, Bugatti, Bentley, Lamborghini, Porsche, Ducatti and VW. The autonomous driving project, which this thesis is contributing to, is an ongoing international collaboration between Group Research of Volkswagen Germany and the ERL. An Audi A7 (see Figure 1.1) is used to develop and test autonomous driving functionalities in the project.

1.1 Motivation

Self-driving cars are autonomous vehicles that sense their environment and make decisions based on the measurements without human input. The process of driving is automated while usually still having a person sitting in the driver's seat. Five different levels of automation are defined by the National Highway Traffic Safety Administration for the U.S.[7], ranging from no automation (level zero) to full automation (level four). Intermediate levels are today already partially made available through the use of advanced driver assistance systems. Full automation requires a precise vehicle localization which can be achieved by comparing the vehicle's current perception to recorded data as stored in specific maps. These maps can contain landmarks, so called road features, which are commonly found on or next to the road, e.g. lane markings, stop lines, poles.

Having a precise map makes the problem of localization easier to solve and is in certain situations indispensable. Localizing without maps and relying exclusively on, for example, detected lanes poses problems when lanes are not available or perceivable (for example due to weather or partial occlusions).

Figure 1.1: *The Audi A7 prototype car which is used at the ERL in California to drive autonomously.*

Furthermore, maps for localization are extremely helpful in large unclear intersections in urban situations, as they can help the car to place itself in a lane model. Over all, using maps increases the reliability of localization, makes it generally available, and can potentially make it more accurate.

Therefore, one common challenge in autonomous driving is to generate a map in which a car can localize itself with the help of its sensors. The map has to take the information provided by these sensors into account.

In previous generations of the autonomous driving project at the ERL, these kinds of maps were generated manually. This is a long and tedious process and the resulting maps are valid for only small regions and often not accurate in the details. These circumstances resulted in the need for a software system that is capable of automatically creating maps for autonomous driving. Figure 1.2 gives an impression of a map of the previous generation.

1.2 Problem statement and objective

Localization for autonomous driving needs accurate and coherent maps of road features. Existing detectors are capable of generating estimates of the position of road features on a per frame basis. This thesis addresses the gap in between: how can these detections be fused in a way that a consistent map arises?

The intent of this thesis is therefore to present a system which fuses sensor data from multiple sources and generates coherent global and local maps. The maps are created in a probabilistic fashion as the detectors produce estimates of a probabilistic nature. In that way, they capture the measurement uncertainty as the sensor data is always affected by noise. The uncertainty is carried on into the maps.

A novel hybrid approach is demonstrated in order to generate grid-based and feature-based maps with an integrated method. The system is designed to automatically construct maps at a large scale and to also maintain local maps around the car. This gives localization subsystems locally coherent maps to compare against global maps.

The proposed solution is analyzed by implementing it for a prototype vehicle. A flexible and extensible analysis framework is developed for this purpose. The results of the analysis are to be evaluated with respect to the compliance of the requirements.

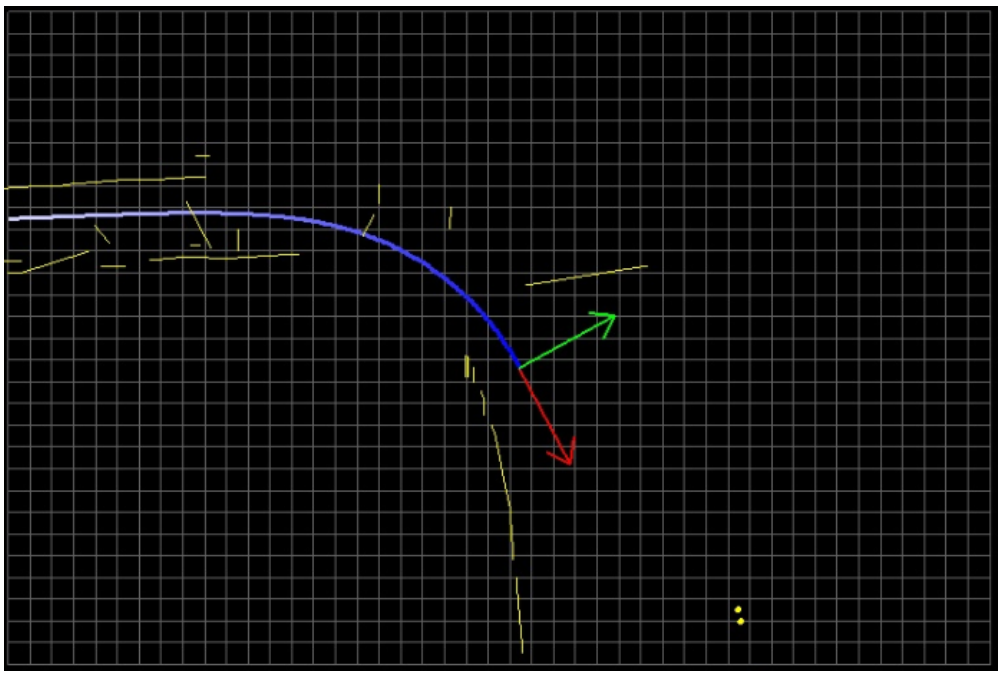


Figure 1.2: *Feature map as used prior to this thesis. The map was edited by hand so it was tedious to create. The blue line visualizes the trajectory of the car, the red and green arrows indicate the local reference frame and the yellow line segments mark landmarks.*

1.3 Outline

This chapter presents a general introduction to the topic of autonomous driving, describes the motivation behind mapping and points out the problems and objectives addressed within this thesis. Chapter two presents related work in the fields of autonomous driving and probabilistic robotics. The method of how to solve the described mapping problem is depicted in Chapter three. Here, a new hybrid approach, which combines grid-based and feature-based representations, is described to construct maps suitable for localization. Chapter four then analyzes the proposed system, while Chapter five concludes with a discussion and an outlook.

2 Literature review

Many robots rely on accurate models of the world surrounding them. Maps give them the ability to localize themselves, to perform motion and path planning and to modify their environment. Robotic mapping is the problem of generating these maps which are a spatial model of the robot's environment. Section 2.1 details the research in this area. Autonomous cars, which can be considered as robots, tackle this general problem in a specific fashion which is detailed in Section 2.2.

2.1 Mapping in probabilistic robotics

As a robot usually does not have a priori knowledge about its position, it has to perform localization and mapping in a loop. This problem is commonly referred to as SLAM[1]. Techniques to solve SLAM had various usages and adaptations in the field of robotics and arose as early as 1986 according to Durrant-Whyte and Bailey [8]. The SLAM problem is generally considered to be solved for many applications (see [8]) but remains an open problem for certain large-scale and real-time scenarios.

This thesis does not generate maps with a SLAM approach but focuses on exploiting known positional information and therefore performing **mapping with known pose**. General difficulties of this approach include errors due to noise in perception and possible perceptual ambiguity between landmarks. Large-scale mapping adds the problem of handling a potentially very large map compared to the car's perceptual range. The specific implementation in this thesis also deals with the problems of tracking features in a grid and performing under real-time constraints because part of the system is reused for localization.

Thrun et al. [9] divide the common mapping approaches into Kalman filter approaches, expectation maximization algorithms, hybrid approaches, object maps, mapping of dynamic environments and occupancy grid maps. The last one is being used in this work as it is very suitable for the problem of mapping with known poses and generally considered to be very robust.

Maps are commonly represented as either topological or metric maps. Topological maps represent the world as a graph in which nodes correspond to distinct landmarks and arcs correspond to the relative path between those. Metric maps treat the world as a two- or three-dimensional space in which every object is placed with absolute coordinates.

While a lot of effort has gone into research for grid-based and for topological methods, substantially less work has been done to combine these two paradigms. Thrun and Bücken [10] demonstrate a way of generating free space grid maps and converting those into a topological graph. The nodes of the graph represent coherent free space partitions that are connected by arcs which denote possible transitions for the robot. This approach lends itself well to a highly accurate and consistent metric description which is also topologically efficient. The system as described in the paper is in that configuration mostly useful for a robot that is only interested in calculating a rough obstacle-free path between two places.

A grid is a metric map and was first defined in [11] as a representation that

[1] SLAM = '*Simultaneous localization and mapping*'

> *"[...] employs a multidimensional (typically 2D or 3D) tessellation of space into cells, where each cell stores a probabilistic estimate of its state".*

Note that the concept of the grid is to be interpreted less as an occupancy grid for free-space representation and more as the presence or absence of certain features on or around the road.

Popular 3D space representations for metric maps include surfels, point clouds and Octrees. Surfels [12] can be used to represent only surfaces of objects and not free or unknown space. Point clouds cannot distinguish between free space and unknown space while Octrees can. Octrees [13] also offer the possibility to extract the grid at a coarser resolution by inspecting levels of the tree that are not at the leaf level and therefore generating multi-resolution maps.

Multi-level surface maps [14] are an alternative way of representing multiple surfaces at the same 2D position without using a full 3D approach. In that sense, they are ranging between 2.5D and 3D representations. Every cell is used to store a list of *surface patches* and measurements are being assigned to distinct surface patches which exist in intervals. A big advantage of this approach is that underpasses and bridges can be represented correctly, but on-ramps are still discretized into multiple height intervals.

The framework of choice is the open-source software OctoMap [15], developed by Hornung et al. at the University of Freiburg. The implementation portion of this thesis adapts OctoMap to fit the needs for large-scale mapping. The authors of OctoMap describe and evaluate in their paper [16] the OctoMap mapping framework. The biggest data set they use for evaluation (and also the biggest one publicly available) represents a mapped volume of 292 m × 167 m × 28 m. At a cell resolution of 10 × 10 cm this accounts for a memory usage of 990.66 MiB on a 32-bit architecture (with node pruning but without maximum-likelihood compression). Even if this mapped volume is impressively extensive for a small robot it appears rather confined when considering the use case of an autonomous car. The major problem with larger mapping areas is that the main memory is usually limited to a few gigabytes which in turn limits the maximum mapping area. This thesis proposes a mechanism to reduce this limitation to a large extent by storing and loading octree branches from disk on-the-fly (see Section 3.3.2).

Extensions to Octrees, which by themselves only represent three-dimensional space but have no semantic notion to the data contained in them, have been considered such as building up hierarchies of Octrees [17]. This paper focuses on speed improvements and memory reduction with respect to map generation. The authors achieve this by using predefined semantic labels to segment and classify point clouds into objects. Those objects are connected in a hierarchy so that changing parameters of nodes effects all of the child nodes. This allows moving an object (e.g. a table) which affects all of the child nodes (e.g. cups). It includes unfortunately a significant portion of manual interaction and is therefore not suited to automatically generate maps of road segments with semantic context.

2.2 Mapping in autonomous driving

Autonomous driving is generally understood as a specific application in the field of probabilistic robotics with the main challenges of mapping, localization and path planning. A survey of the state of the art shows that most mapping approaches for autonomous driving are based on some form of SLAM implementation. In fact, the autonomous car Stanley [18] won the first DARPA (Defense Advanced Research Projects Agency) Grand Challenge in 2005 by using a SLAM approach. The DARPA Grand Challenge of 2005 took place in the Mojave Desert. A 2D grid was build up from laser data and used to classify the surrounding environment into the three classes *drivable*, *occupied* and *unknown*. The car's position was estimated using an Unscented Kalman Filter with GPS and odometry data as input. Stanley was created by the Stanford Racing Team in cooperation with the VW ERL under the lead of Sebastian Thrun.

The Urban Challenge of 2007, which was also organized by DARPA, was set in an urban environment. Cars had to adhere to the common driving rules and interact with other road participants. The Stanford Racing Team was able to reach the second place. A clone of the car Junior, which has been used for that, is nowadays parked at the VW ERL. The winner of this competition was the vehicle Boss from CMU under the lead of William "Red" Whittaker [19]. Urmson et al. [20] describe that Boss employed an obstacle map in the form of a grid which was built up with laser data. Furthermore, Boss distinguishes between dynamic and static obstacles. Dynamic obstacles are tracked over time and predicted into the future. Multiple hypothesis about the state of tracked vehicles are maintained and are aligned using two different motion models.

DARPA is a United States agency of the Department of Defense and responsible for developing and researching new military technologies. The DARPA challenges demonstrated the interest of the US military for autonomous driving topics. The US military also experimented with semi-autonomous convoy driving as described in [21]. However, semi-autonomous driving poses a different problem because the lead vehicle is still operated manually; thus removing the challenge for the follower vehicles of building an accurate map of its surroundings and globally localizing itself. Instead, the follower vehicles try to follow the observed path from the lead vehicle and localize themselves only relative to the lead vehicle.

Agricultural interest has been another driving force behind autonomous vehicles. The problem of mapping in this field is much smaller as the space of operation is very confined, known and controllable. Routes are usually either being taught to the autonomous system or programmed in advance. Localization is often done with the help of DGPS receivers, camera-based visual tracking or external triangulation via markers on the agricultural machinery. Also multiple of these methods have been used in combination where a voting scheme was applied. Sophisticated approaches like an Extended Kalman Filter based on odometry and DGPS data as described in [22] have proven to fulfill the positional accuracy requirements as needed for autonomous farming even without maps.

Urban driving usually requires maps because a highly accurate localization is needed. Levinson, Montemerlo, and Thrun [23] construct maps of the ground's infrared reflectivity using lidar data. A 2D grid is employed for that with a spatial resolution of 5 cm. A GraphSLAM approach with an offline relaxation technique is utilized to align the grid at regions of self-overlap. By also integrating GPS and odometry data the system achieves a relative localization accuracy of about 10 cm to the map. Experiments empirically demonstrate that the file sizes are about 10 MiB per mile of road at this resolution.

Danescu, Oniga, and Nedevschi [24] construct a rather coarse occupancy grid at a resolution of 20 cm $\times$ 20 cm with the fixed dimensions of 50 m $\times$ 24 m. Particles are utilized to estimate occupancy and speed of the grid cells. They describe at the same time occupancy hypothesis for cells and physical objects that are being tracked by virtue of a motion model. As objects, they can migrate from cell to cell and therefore influence the occupancy probability distribution of cells. In this way, the approach combines a grid with a particle tracking filter.

Detection and tracking of moving objects (DATMO) is a challenge that in the automotive field is generally seen as complementary to SLAM. Vu, Aycard, and Appenrodt [25] tackle both problems in an integrated approach. The authors build up occupancy grid maps from laser sensor data, localize the vehicle with a new incremental scan matching technique and detect objects by comparing the current measurement to the previously constructed free space map. Inconsistencies in the detection of free and occupied space between the map and the measurement data help to decide whether a detected object is

likely static or dynamic. Filtering the dynamic objects from the map creates consistent free space maps, which are however only local and of small scale.

Wang, Thorpe, and Thrun [26] propose a different approach of combining SLAM with DATMO. The authors formulate the SLAM with DATMO problem in a Bayesian way. They represent the environment in a grid and maintain a moving and a stationary objects map. An Extended Kalman Filter is used to estimate the pose and objects are detected by motion with the help of a laser scanner.

Fully autonomous driving also poses challenges that are unrelated to driving on the road: autonomous parking is a use-case that has mostly been approached as an encapsulated problem. It is different from commercially already available parking assistants, which still require the driver to drive up to or next to a parking spot, engaging the assistant and letting the car back into the parking space. Audi introduced a system called *Parking Pilot* at the CES 2013 [27] which allows the removal of the driver from the situation. At CES 2014 a smartphone app was presented that allows remotely calling the car out of a parking garage. Kummerle et al. [28] presents an approach where the car is enabled to autonomously park itself in a multi-level parking structure. They rely heavily on pre-built sensor maps to localize the vehicle since neither a sufficiently accurate GPS signal is available due to the thick concrete walls nor common navigation maps as they are existing for roads. The multi-level parking structure is represented using multi-level surface maps. Those maps are generated by a GraphSLAM approach [29] and post-processed with a graph-based optimization procedure to ensure consistent maps.

Maps for driving autonomously can also be obtained by other measures than driving on a road, capturing the sensor data and processing it in some way. Pink [30] demonstrates a map building process based on the extraction of visual road features from geographically referenced aerial or satellite imagery. Those maps can be built in advance and the problem of SLAM is being avoided. The authors implement a map matching based on an iterative closest point algorithm to localize the vehicle. This approach requires high-precision imagery showing the non-occluded road features. Such imagery is at the moment attached with high costs and commercially not generally available in all areas. As a result, the mapping approach in this thesis is only using aerial imagery as a comparison metric for a limited test track.

3 Gridmap

The problem of generating maps for autonomous driving is tackled in this chapter. Section 3.1 presents the underlying theory and Section 3.2 presents the platform in terms of hardware and software at hand. Finally, Section 3.3 details the core of the implemented software system which is called *gridmap*.

3.1 Theoretical aspects of probabilistic mapping

This section defines the mathematical notation and the definition of the mapping problem from a probabilistic point of view. Most of the notation has been adapted from [31]. At first, the mapping problem is defined as a probabilistic problem. Then, the binary Bayes filter with static state is detailed as the approach to solve this problem. This results in the occupancy grid mapping algorithm which iteratively updates the solution. The relation to Gaussian filters is presented subsequently as lists of objects can be extracted from the grid-based map.

3.1.1 Problem definition of probabilistic mapping

Mapping as a probabilistic operation comes down to an estimation problem of the posterior probability of maps given a set of measurements in different robot poses. This posterior probability is $p(m|z_{1:t}, x_{1:t})$, where $z_{1:t}$ denotes the set of all measurements between time 1 to t, $x_{1:t}$ the set of all robot (or car) poses between time 1 to t and m the entire map. A grid represents a discrete approximation of the continuous space of the mapping area. A single grid cell with the index i is denoted $\boldsymbol{m}_i$ so that m forms the set over all $\boldsymbol{m}_i$ as given by

$$m = \{\boldsymbol{m}_i\}. \tag{3.1}$$

Each grid cell $\boldsymbol{m}_i$ is either occupied (represented by a "1") or free (represented by a "0"). The probability for a grid cell $\boldsymbol{m}_i$ to be occupied is written as $p(\boldsymbol{m}_i = 1)$, or short as $p(\boldsymbol{m}_i)$. For now it is assumed that there is only one type of object that can be present in grid cells. This will later be justified by arguing that every feature type is stored in its own map.

Let x be the state and x_t the state at time t. The state is supposed to be *complete*, which means that it incorporates all information about the environment that is necessary to know in order to make the best (stochastic) prediction about the future. Put differently, state completeness assumes that no further knowledge of past states or measurements influences the stochastic evolution of the future. The temporal sequence of states $x_{1:t}$ is therefore a *discrete-time Markov chain* (see [31, p. 17]) as formally expressed by the equality

$$p(x_t|x_{1:t-1}) = p(x_t|x_{t-1}, x_{t-2}, \ldots, x_1) = p(x_t|x_{t-1}). \tag{3.2}$$

Also, the measurement z at time t depends only on the current state x_t. It is conditionally independent of all other states $x_{1:t-1}$. As the state x_t is assumed to be complete, no measurement prior to the current measurement provides any additional information resulting in

$$p(z_t|x_{1:t}, z_{1:t-1}) = p(z_t|x_t, x_{t-1}, \ldots, x_1, z_{t-1}, z_{t-2}, \ldots, z_1) = p(z_t|x_t). \tag{3.3}$$

The number of possible maps (the hypothesis space) is high-dimensional and it is therefore infeasible to calculate the full posterior over all possible maps. This problem is dealt with by decomposing the map inference problem over the complete map into single-cell estimation problems. The difficulty then shifts to estimating

$$p(\boldsymbol{m}_i|z_{1:t}, x_{1:t}) \tag{3.4}$$

for all grid cells $\boldsymbol{m}_i$. The decomposition introduces the assumption of statistical independence between individual grid cells. Obviously this is for most sensor models not entirely true as measurements for nearby grid cells usually affect each other (e.g. sonar or lidar range finders). Therefore, some of the structure of the problem might be lost since dependencies between neighboring cells cannot be modelled. However, this assumption allows delivering satisfactory results in practice while keeping the computational cost small.
The product of the marginals approximates the posterior of a map as given by

$$p(m|z_{1:t}, x_{1:t}) = \prod_i p(\boldsymbol{m}_i|z_{1:t}, x_{1:t}). \tag{3.5}$$

The remaining task is to estimate the quantity $p(\boldsymbol{m}_i|z_{1:t}, x_{1:t})$ of Equation 3.4 for all grid cells $\boldsymbol{m}_i$. This can be considered as an estimation problem for a fixed binary state (either the grid cell is free or occupied, and this actual state does not change over time) given a series of measurements. The binary Bayes filter approaches this kind of problem.

3.1.2 Binary Bayes filter with static state

A binary Bayes filter is a non-parametric filter and does not assume a certain type of probability distribution for the posterior. In occupancy grid mapping it is used to estimate the probability of a cell being occupied given a certain set of measurements and robot poses, that is to say $p(\boldsymbol{m}_i|z_{1:t}, x_{1:t})$.

The corresponding posterior represents the robot's belief in the value of $\boldsymbol{m}_i$ at a given time t as given by

$$bel_{t,i}(\boldsymbol{m}_i) = p(\boldsymbol{m}_i|z_{1:t}, x_{1:t}). \tag{3.6}$$

In order to compute this term, we apply Bayes' rule and obtain

$$p(\boldsymbol{m}_i|z_{1:t}, x_{1:t}) = \frac{p(z_t|\boldsymbol{m}_i, z_{1:t-1}, x_{1:t})p(\boldsymbol{m}_i|z_{1:t-1}, x_{1:t-1})}{p(z_t|z_{1:t-1})}, \tag{3.7}$$

$$= \frac{p(z_t|\boldsymbol{m}_i, x_t)p(\boldsymbol{m}_i|z_{1:t-1}, x_{1:t-1})}{p(z_t|z_{1:t-1})}. \tag{3.8}$$

The step from Equation 3.7 to 3.8 is justified by the Markov assumption for the measurement data as stated in Equation 3.3. With Bayes' rule applied to the measurement model, which is represented by the term $p(z_t|\boldsymbol{m}_i, x_t)$ in the numerator, we obtain

$$p(z_t|\boldsymbol{m}_i, x_t) = \frac{p(\boldsymbol{m}_i|z_t, x_t)p(z_t)}{p(\boldsymbol{m}_i)}. \tag{3.9}$$

Substituting this result into Equation 3.8 we obtain

$$p(\boldsymbol{m}_i|z_{1:t}, x_{1:t}) = \frac{p(\boldsymbol{m}_i|z_t, x_t)p(z_t)p(\boldsymbol{m}_i|z_{1:t-1}, x_{1:t-1})}{p(\boldsymbol{m}_i)p(z_t|z_{1:t-1})}. \tag{3.10}$$

Equation 3.10 represents the belief of the robot for a given time t into the value $\boldsymbol{m}_i$ of cell i (cf. Equation 3.6). For numerical stability, this belief is usually expressed as *log odds ratio* because it maps the probability value from its original codomain of $[0,1]$ to the codomain $(-\infty,+\infty)$. The *odds* of $\boldsymbol{m}_i$ are defined as the function

$$\text{odds}(\boldsymbol{m}_i) := \frac{p(\boldsymbol{m}_i)}{p(\neg\boldsymbol{m}_i)} = \frac{p(\boldsymbol{m}_i)}{1-p(\boldsymbol{m}_i)}. \tag{3.11}$$

The *log odds* of $\boldsymbol{m}_i$ is denoted by $l_i(\boldsymbol{m}_i)$ and is defined accordingly as

$$l_i(\boldsymbol{m}_i) := \log(\text{odds}(\boldsymbol{m}_i)) = \log\frac{p(\boldsymbol{m}_i)}{1-p(\boldsymbol{m}_i)}. \tag{3.12}$$

The log odds ratio of a belief $bel_{t,i}(\boldsymbol{m}_i)$ is therefore expressed as $l_{t,i}(bel_{t,i}(\boldsymbol{m}_i)) = \log(\text{odds}(bel_{t,i}(\boldsymbol{m}_i)))$ and the belief can be recovered from the log odds ratio via the equation

$$bel_{t,i}(\boldsymbol{m}_i) = 1 - \frac{1}{1+\exp(l_{t,i})}. \tag{3.13}$$

As we want to compute $l_{t,i}(bel_{t,i}(\boldsymbol{m}_i))$ we need to derive an expression for the probability of the opposite event with $\boldsymbol{m}_i = 0$, which is obtained by the same reasoning which allowed us to derive Equation 3.10. This leads to

$$p(\neg\boldsymbol{m}_i|z_{1:t},x_{1:t}) = \frac{p(\neg\boldsymbol{m}_i|z_t,x_t)p(z_t)p(\neg\boldsymbol{m}_i|z_{1:t-1},x_{1:t-1})}{p(\neg\boldsymbol{m}_i)p(z_t|z_{1:t-1})}. \tag{3.14}$$

Ultimately, the log odds ratio of $bel_{t,i}(\boldsymbol{m}_i)$ can be calculated to

$$\begin{aligned}
l_{t,i}(bel_{t,i}(\boldsymbol{m}_i)) &= l_{t,i}(p(\boldsymbol{m}_i|z_{1:t},x_{1:t})),\\
&= \log\left(\frac{p(\boldsymbol{m}_i|z_{1:t},x_{1:t})}{p(\neg\boldsymbol{m}_i|z_{1:t},x_{1:t})}\right),\\
&= \log\left(\frac{p(\boldsymbol{m}_i|z_t,x_t)p(\boldsymbol{m}_i|z_{1:t-1},x_{1:t-1})p(\neg\boldsymbol{m}_i)p(z_t|z_{1:t-1})p(z_t)}{p(\neg\boldsymbol{m}_i|z_t,x_t)p(\neg\boldsymbol{m}_i|z_{1:t-1},x_{1:t-1})p(\boldsymbol{m}_i)p(z_t|z_{1:t-1})p(z_t)}\right),\\
&= \log\left(\frac{p(\boldsymbol{m}_i|z_t,x_t)p(\boldsymbol{m}_i|z_{1:t-1},x_{1:t-1})p(\neg\boldsymbol{m}_i)}{p(\neg\boldsymbol{m}_i|z_t,x_t)p(\neg\boldsymbol{m}_i|z_{1:t-1},x_{1:t-1})p(\boldsymbol{m}_i)}\right),\\
&= \log\left(\frac{p(\boldsymbol{m}_i|z_t,x_t)}{1-p(\boldsymbol{m}_i|z_t,x_t)}\,\frac{p(\boldsymbol{m}_i|z_{1:t-1},x_{1:t-1})}{1-p(\boldsymbol{m}_i|z_{1:t-1},x_{1:t-1})}\,\frac{1-p(\boldsymbol{m}_i)}{p(\boldsymbol{m}_i)}\right),\\
&= \log\left(\frac{p(\boldsymbol{m}_i|z_t,x_t)}{1-p(\boldsymbol{m}_i|z_t,x_t)}\right) + \underbrace{\log\left(\frac{p(\boldsymbol{m}_i|z_{1:t-1},x_{1:t-1})}{1-p(\boldsymbol{m}_i|z_{1:t-1},x_{1:t-1})}\right)}_{l_{t-1,i}} - \underbrace{\log\left(\frac{p(\boldsymbol{m}_i)}{1-p(\boldsymbol{m}_i)}\right)}_{l_{0,i}}
\end{aligned} \tag{3.15}$$

where $p(\boldsymbol{m}_i)$ represents the prior distribution over $\boldsymbol{m}_i$ and $l_{0,i}$ the log odds ratio of the prior belief before any measurement was sensed for the cell i. Equation 3.15 contains the core of the update mechanism in the occupancy grid mapping algorithm.

3.1.3 Occupancy grid mapping algorithm

The belief of the robot for a given time t that the grid cell i is occupied can be calculated recursively by virtue of Equation 3.15. The new belief value at time t is computed as a function of the old belief value at time $t-1$, the pose x_t and the measurement z_t. The algorithm is quoted from [31, p. 286]:

Algorithm 1: occupancy_grid_mapping($l_{t-1,i}$, x_t, z_t)

```
1 forall the cells m_i do
2     if m_i in perceptual field of z_t then
3         l_{t,i} = l_{t-1,i} + inverse_sensor_model (m_i, x_t, z_t) − l_{0,i}
4     else
5         l_{t,i} = l_{t-1,i}
6     end if
7 end forall
8 return l_{t,i}
```

The algorithm iterates over all grid cells $\boldsymbol{m}_i$ and computes the new value of $l_{t,i}$. If the current cell i is not in the perceivable field of view of the detector then no new information was gathered and the old belief $l_{t-1,i}$ is assigned to $l_{t,i}$ (line 5). Otherwise, if measurement data about cell i was obtained (line 3), then $l_{t,i}$ is updated according to Equation 3.15. If the prior $p(\boldsymbol{m}_i)$ is set to 0.5 then the term $l_{0,i}$ equals zero and can be omitted.

Extreme grid cell probability values of 0 and 1 pose a problem in practice. If a cell reaches a state where it is absolutely certain to contain a feature (= 1) or to not contain a feature (= 0), then this value can never be changed by the update mechanism again. This is easy to see as the log odds of 0 and 1 equal negative and positive infinity respectively. Minimal and maximal clamping thresholds are employed to prevent this saturation effect.

It might become necessary to assign discrete states to cells (labelling them *occupied*, *empty* or *unknown*) in order for non-probabilistic planners or localization modules to work. In this case, an optimal estimator such as the maximum a posteriori (MAP) decision rule can be applied. Elfes employs this rule in [11], which translates in the notation at hand to:

> "A cell i is *occupied* if $p(\boldsymbol{m}_i) > p(\neg\boldsymbol{m}_i)$, *empty* if $p(\boldsymbol{m}_i) < p(\neg\boldsymbol{m}_i)$, and *unknown* if $p(\boldsymbol{m}_i) = p(\neg\boldsymbol{m}_i)$."

Applying the MAP decision rule holds the advantage of generating more compressed maps as less information per cell needs to be stored. However, it is almost always preferable to keep the probabilistic representation of the map (if possible) because this allows for a better use of probabilistic localization, path planning and other application modules as any deterministic decision is pushed as far back in the decision making chain as possible.

The MAP decision rule minimizes the misclassification rate. However, if the *cost* of a misclassification is not equal for the different classes, then the decision rule can be adapted to minimize the overall *risk* instead. Risk is the expected cost for a decision. The decision rule can be adapted to favor the labelling which invokes the least amount of risk given a cost function. In this case, the decision rule changes to a decision based on risk:

> "Label a cell i as *occupied* if $R(\alpha_2|\boldsymbol{m}_i) > R(\alpha_1|\boldsymbol{m}_i)$, *empty* if $R(\alpha_2|\boldsymbol{m}_i) < R(\alpha_1|\boldsymbol{m}_i)$, and *unknown* if $R(\alpha_2|\boldsymbol{m}_i) = R(\alpha_1|\boldsymbol{m}_i)$."

Here, α_1 denotes the decision to label the cell as occupied and α_2 to label the cell as empty. The function R computes the risk associated with taking a decision given a certain information (here, the occupancy probability of a cell). R is defined as

$$R(\alpha_1|\boldsymbol{m}_i) = \lambda(\alpha_1|C_1)p(C_1|\boldsymbol{m}_i) + \lambda(\alpha_1|C_2)p(C_2|\boldsymbol{m}_i), \tag{3.16}$$
$$R(\alpha_2|\boldsymbol{m}_i) = \lambda(\alpha_2|C_1)p(C_1|\boldsymbol{m}_i) + \lambda(\alpha_2|C_2)p(C_2|\boldsymbol{m}_i) \tag{3.17}$$

where $\lambda(\alpha_i|C_j)$ is the loss function for taking the decision α_i while the true class is C_j. If we assume that the loss function λ penalizes a wrong decision more than a correct decision, we can easily derive that α_1 is the risk-minimizing decision if

$$\frac{p(\boldsymbol{m}_i|C_1)}{p(\boldsymbol{m}_i|C_2)} > \frac{\lambda(\alpha_1|C_2) - \lambda(\alpha_2|C_2)}{\lambda(\alpha_2|C_1) - \lambda(\alpha_1|C_1)} \times \frac{p(C_2)}{p(C_1)}. \tag{3.18}$$

It is obvious to see that the rule, which minimizes the misclassification rate, is a special case of the rule, which minimizes the risk, if we assume that the loss is a binary function which rewards a correct decision with a loss of 0 and penalizes a wrong decision with a loss of 1.

Currently, there is no explicit definition for the loss function λ and the rule which minimizes the misclassification rate is used in practice. However, this can easily be changed for different use cases once for example localization finds a meaningful interpretation for its application.

3.1.4 Relation to Gaussian filters

Gaussian filters are a family of widely used filters[1] to recursively estimate an unknown state.

Noisy measurement data is integrated over time into a single estimate which is usually more precise than just the latest measurement. Gaussian filters approximate the unknown quantity by a (multivariate) normal distribution. While this unimodal model doesn't allow for multiple distinct hypotheses to be represented, it has the advantage of being entirely characterized by only its first two moments; the mean $\boldsymbol{\mu}$ and the covariance $\boldsymbol{\Sigma}$.

The most important practical benefit of the reduction of the representation with two parameters is that less storage space is required. Generated maps will consume less memory and disk space which is important for large-scale mapping. A potential online transfer of maps from a server to the car will be faster as less data has to be exchanged. Localization approaches could be faster (depending on the implementation) because they have to handle less information as compared to a grid-based representation. On the other hand, grids allow representing arbitrary spatial belief distributions. This yields a higher storage space requirement but generally approximates reality better.

The global focus of this work is to produce a consistent probabilistic map around the car which estimates the position of certain features. The two major approaches to achieve this goal are either using an occupancy grid or temporally tracking the features (which can for example be done with Gaussian filters). To the best knowledge of the author of this thesis, no method exists in the literature to directly combine those two approaches in the following way. This work implements an **occupancy grid** to handle the positional uncertainty of features and it allows **extracting the mean and the covariance** of their position. The produced maps can be regarded as **hybrid maps** which exist **as a grid and as a list of features** at the same time. They can therefore be used by either a **grid-based or a feature-based localization approach**, as the grid mimics the outcome of a Gaussian filter. Additionally, **features are fused** in the grid to combine the information from multiple sensors and to improve the detectors' accuracy.

[1] Popular filters among the Gaussian filters are the *Kalman filter* and its variations. Kalman filters are designed to work with linear system dynamics whereas several of its extensions, such as the *extended Kalman filter* or the *unscented Kalman filter*, work on nonlinear system dynamics. They all have in common that they produce a stochastic estimate by performing a *prediction step* (based on the system dynamics) which is then compared to the measurement data in the *update step*.

Two essential techniques are developed for this: first, keeping track of which cells belong to which feature and identifying the same feature over multiple runs or at different sampling points, and second, extracting the mean and covariance of a discretized feature dispersion. Any feature-based localization algorithm, that can cope with the feature tracking result of a Gaussian filter can consequently use $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$ as the output of the grid to take into account the positional uncertainty of features in the map matching process.

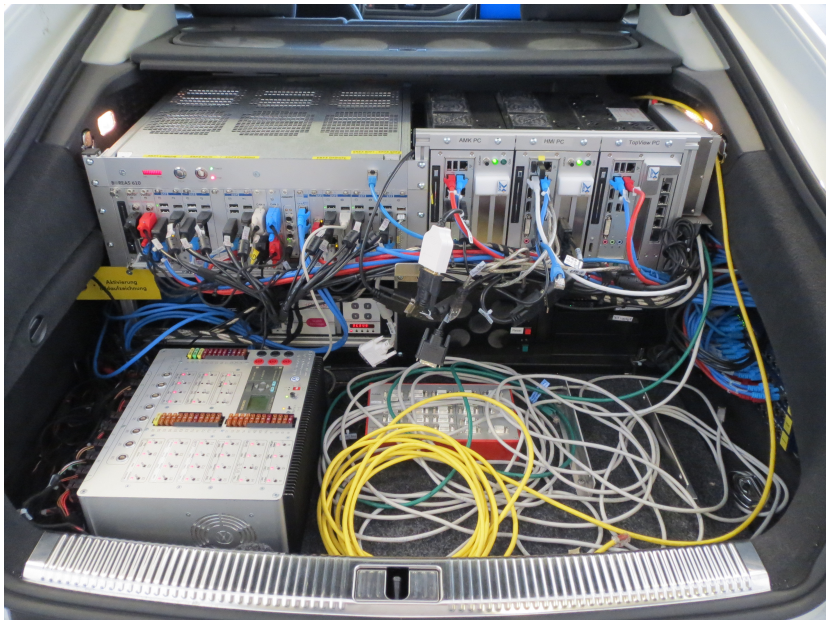


Figure 3.1: *Computer hardware in the trunk of a prototype A7 used for autonomous driving. A big portion of the PCs will become redundant once the zFAS is available which will push the autonomous driving technology closer to series production status.*

3.2 Platform

A certain set of hardware and software components forms the environment for the mapping software. The hardware is being detailed in Section 3.2.1 and the software in Section 3.2.2. A couple of feature detectors are used as input for the mapping module and treated as part of the existing software environment as they are not developed as part of this thesis. Section 3.2.3 explains the mode of operation of feature detectors in detail.

3.2.1 Hardware

The mapping software presented here is deployed on an Audi A7 prototype which has been specifically prepared for autonomous driving tasks. It has a certain set of human-machine interface features that indicate the autonomous or manual driving mode to the driver. Furthermore, several PCs and dedicated hardware components in the trunk provide the necessary computing power. Sensors constitute a crucial piece of hardware because they provide the car with information about its environment. Different kind of sensors are installed in the vehicle, e.g. radar, lidar, ultrasonic, mono / stereo cameras etc. The major sensors for mapping and localization are four topview cameras (capturing image data nearby the vehicle and providing a 360 degree bird's-eye view), a front-facing lidar (a laser-based, active rangefinder) and two front-facing monocameras. The goal for the sensor setup is to only require a minimal set of sensors in order to achieve a reliable perception of the external vehicle environment. Keeping the number of sensors low means less hardware cost for the final production vehicles, less effort to bring the sensors to series-production status and fewer sensors to integrate into the car.

At the CES in January 2014, Audi announced the zFAS hardware [32]. zFAS is a specialized book-sized hardware component powered by two Nvidia chips and designed to receive sensor input, execute piloted driving software and transmit commands to the car. The intended design goal of the zFAS is to replace almost all of the current PCs in the trunk, and therefore offers a way of bringing piloted driving functionality in series-production cars. Using topview and front-facing cameras as well as a lidar will also be possible in series production.

There is however one sensor, that will only be available only in prototype vehicles but not in production cars: the D-GPS and IMU unit called POS LV from the company Applanix [33]. It provides highly accurate positional data by fusing differential GPS data (D-GPS) from multiple antennas with orientation data from a inertial measurement unit (IMU). While traditional GPS data is only accurate up to 5 meters, a D-GPS unit can already achieve up to 10 cm accuracy in best-case scenarios. As the POS LV also incorporates post-processed data in addition to that, it is accurate up to 2 cm RMS. As the information for post-processing the positional data is usually only available a day or two after the initial measurement, our prototype mapping car records sensor information combined with the positional data in order to generate a map several days after the actual recording. The expensive POS LV sensor is therefore only used in prototype mapping vehicles for *producing* maps and even that can potentially be avoided with additional work in the future. For *using* maps and performing localization when driving autonomously, this sensor isn't necessary. The position is computed by comparing the local map around the car to the prior created global map. The local map is created by using stock or near-stock sensors such as the topview and front-facing cameras plus the lidar.

3.2.2 Software

A certain software framework exists in which the work of this thesis has to fit. The environment on the software side of things is detailed in the subsection *Framework*, whereas the subsection *Requirements* describes the functional and non-functional requirements for the software development process. The last subsection *Software libraries* mentions the used libraries for the sake of completeness.

Framework

As there are currently up to a dozen different PCs running in the trunk of the prototype A7, there are different operating systems and software environments in use. The software framework used for mapping is Windows 7 in 64-bit in combination with the automotive software middleware called ADTF ("Automotive Data and Time-Triggered Framework" [34]). This middleware has been developed by Audi in 2001 and is widely adapted and used all over in the VW group for advanced driver assistance engineering. It essentially provides mechanism to easily control the program and data flow between different software modules. These modules are called filters in ADTF and represent dynamic-linked libraries (.dll). Filters are the pivotal concept in the ADTF development. They are developed in C++, compiled to a shared library and then connected in an ADTF configuration. These configurations can be run offline as a simulation on a laptop or online with real measurement data in the car. Most of the core implementation work of this thesis consists in the C++ development of filters.

One advantage of ADTF is the possibility to record sensor data of all kind in a live run in the vehicle under real-world conditions, which can later be analyzed in the lab. This allows reproducing situations of interest and to adapt algorithms to deal with them. In fact, most of the development work has been done in this way: data has been recorded on the road and only a certain time later it is used for development. This proved to be of fundamental importance as the positional data from the POS LV D-GPS unit can only be post-processed a couple of days later.

Another advantage of ADTF is the reusability of filters. As some code is employed in the process of creating a map as well as in the localization later on, it is of great value to utilize the same filters again. Examples for such filters are the perception filters which detect features (either for creating a map in the first place or for finding those features again in a previously generated map in order to localize the vehicle) or filters that take care of converting coordinate systems. The grid, which is described in depth in Chapter 3.3, is another example. It is going to store all information in mapping mode. In localization mode however, it still has the same set of methods and operations but it is discarding obsolete information in a window around the vehicle. The corresponding filter for the grid needs to be only slightly adapted to achieve this.

Requirements

The software implementation of this thesis is part of a collaborative group project. The primary functional requirement is to implement a system which is capable of creating and maintaining local and global maps in a probabilistic fashion. The system is intended to produce grid-based maps and to extract feature-based maps from them. Input data from multiple sensors has to be fused into coherent representations.

The non-functional requirements include fulfilling the deliverable constraints and meeting the intermediate deadlines and deliverables so that iterative system integration can be conducted. The existing software environment as listed above has to be used for the implementation portion. Also, the existing hardware environment has to be respected. This means strict resource requirements so that the software can run in the car. All ADTF filters that are used in localization mode have to run in real-time, while the mapping mode as an off-line process does not have a timing constraint. The design of the system has to be in a way that maps at a large scale (several hundred kilometers) are computationally manageable.

Software libraries

Different third party software libraries were used in the development of this thesis, all of them open source. This includes OpenCV[2] for image processing, Geographiclib[3] for performing conversion between local Cartesian coordinate systems and UTM, Eigen[4] for matrix manipulations in C++, zlib[5] for using gzip compression, MatIO[6] for storing C++ data structures as Matlab compatible files and OctoMap[7]. Furthermore, gtest[8] was used as unit test framework for C++ classes.

3.2.3 Feature detectors

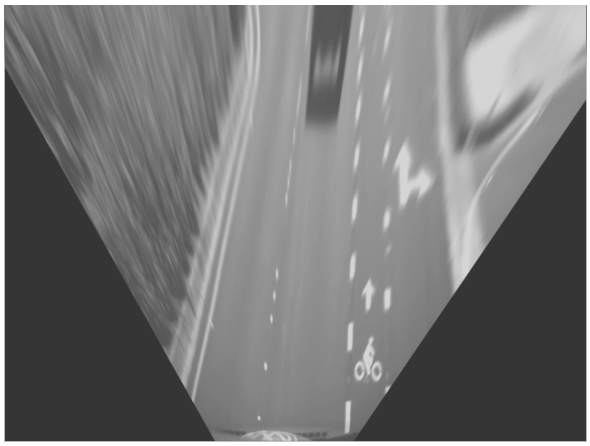
A multitude of features can be extracted from the sensor data obtained by the car. The features have to be easily re-observable and should not occur to often (the information content is too low) or too rarely (not generally usable). In practice, several landmarks such as lane markings, poles, stop lines and arrows are detected by the car and used as input for the mapping and localization modules.

The grid uses a modular approach to allow any kind of feature detector to fuse data into a grid layer. Feature detectors have to implement the common input interface in order to input data into the layer (see Section 3.3.1). Several detectors for the same feature type can fuse information into the same layer without having to perform sensor fusion in beforehand. One example for this kind of sensor fusion is given in Section 3.2.3 by detailing the lane detectors, which operate on images from different topview cameras but fuse data into the same layer. This detector has not been developed as part of this thesis but is one of the detectors that have been used as input source throughout the development of the grid.

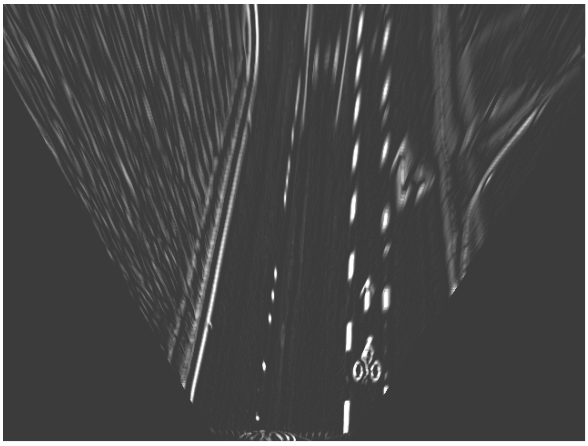
The grid itself is for the most part agnostic to the feature type: it can range from general free space, drivable road surface or a road lane model over to any kind of feature that is present on or next to roads. The shape of the feature type is important though when it comes to extracting a list of features from the grid. Also, the sensor source doesn't matter as long as it produces consistent detections in the environment.

One example feature detector is briefly illustrated to better understand the input of the grid, which is important for the definition of the input interface.

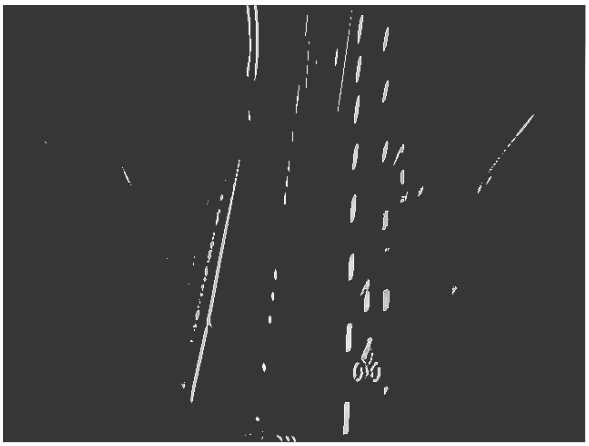
[2] http://opencv.org/
[3] http://geographiclib.sourceforge.net/
[4] http://eigen.tuxfamily.org/
[5] http://www.zlib.net/
[6] http://sourceforge.net/projects/matio/
[7] http://octomap.github.io/
[8] https://code.google.com/p/googletest/



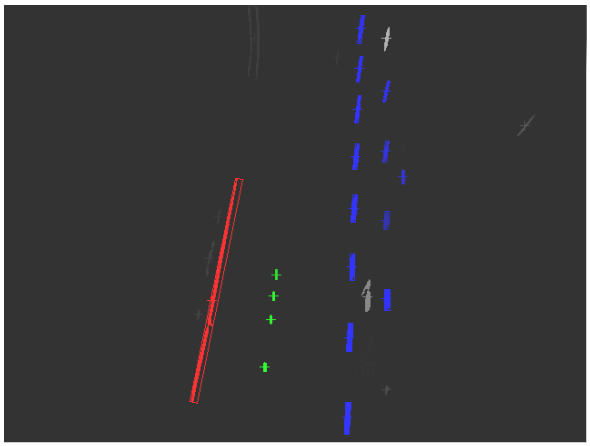
(a) *Raw image from the topview camera after undistortion and virtual topview transformation.*



(b) *Background subtraction to obtain the brighter image areas.*



(c) *Contours in the foreground plane after binarization.*



(d) *Contours classified into feature types (red: continuous lane line, blue: dashed lane line, green: Botts' dots, black: unknown).*

Figure 3.2: *Work flow of the lane line detector. First, the image background is removed and the contours of all objects in the foreground plane are found. These contours are then classified into different road feature types. Only the maximum likelihood class is visualized here, and the brightness of the color in d) describes the certainty of being a certain feature type.*

Dashed and continuous lane line detector

The lane line detector is a module which takes as input images from topview cameras and outputs the position of all dashed and continuous lane lines as well as Botts' dots[9] in the sensing area. The output of Bott's dots from this detector is currently not used in the project.

Figure 3.2a shows the raw image of the topview camera in front of the vehicle after the image has been undistorted and transformed to a virtual bird's eye perspective. In a first step, the background is subtracted from the image and Figure 3.2b is obtained. The foreground is then binarized with a dynamic threshold which results in Figure 3.2c. All remaining objects are being considered as possible feature candidates. They are being processed as the input by a machine learning model which has been trained with a supervised learning approach. The model has four output classes which are *dashed lane line*, *continuous lane line*, *Botts' dots* and *unknown*. It assigns to every candidate input the probabilities of it

[9] In many parts of the U.S. lanes are not only separated by dashed and continuous lane lines but also by small round markers called Botts' dots. They are raised in order to provide tactile feedback to the driver.

being element of each class. Figure 3.2d shows a visualization of this, where only the class with the maximum probability for each candidate is being shown. The more intense the colors are, the higher the probability value.

The detector finally outputs a list of contours of all features associated with the computed probability value. As the extrinsic parameters of the virtual topview camera are known the detector can also determine the world coordinates of all features. Furthermore, the detector assigns IDs to all features which are unique for that image. The feature detection is also being performed for the left, right and rear topview cameras. As their field of views either overlap at the same point of time or potentially in a later or earlier point of time, it is common that one feature is eventually being detected by multiple cameras over multiple frames. The information of each camera is then being used as update information for the grid.

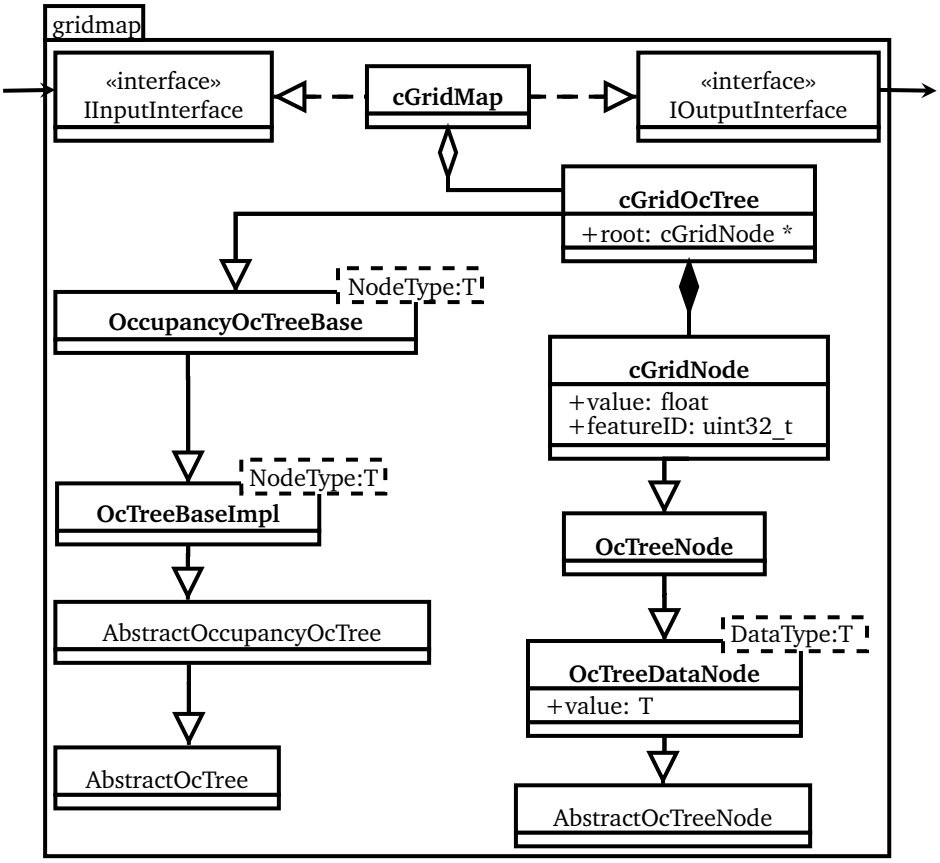


Figure 3.3: *UML specification of the gridmap. It is encapsulated by the use of interfaces. Internally, it builds up on the library OctoMap and its classes. The classes cGridMap, cGridOctree and cGridNode are integrated into the class hierarchy to enable the storage of feature IDs.*

3.3 Implementation

The architecture of the gridmap software is presented in Section 3.3.1. It uses an existing library, for which the most notable extensions are summarized in Section 3.3.2. The method of how objects are extracted from the grid-based map is detailed in Section 3.3.3.

3.3.1 Grid architecture

The grid architecture was designed from scratch. The software specification was refined in an iterative process to meet the requirements of all involved parties who are interested in using the grid. One of the main design goals was therefore to provide very clean and complete input and output interfaces in an early stage. The internals of the *gridmap*, as the software is called, are being kept as a black box to outside modules. This allows the exchange of internal modules such as the underlying grid implementation, which is currently the open-source mapping framework OctoMap [15]. The gridmap is developed as a self-contained static C++ library. It is easily integrated into different ADTF filters but can also be used in pure C++ applications. This advantage has been exploited to create standalone executables which can read in the gridmap data and convert it into different formats. The UML representation of the C++ implementation is depicted in Figure 3.3.

The class **cGridMap** is the core of the implementation. It is externally addressable over the **IInputInterface** and the **IOutputInterface**. Both declare only a single function: the first one declares

InsertObjects(const GridContainer *) and the latter one GetGrid(cGridContainer *). They work with a data container of type **cGridContainer** which stores all necessary information such as the probability values of the local grid, the dimensions, the offset of the lower left corner and the resolution. The **IOutputInterface** allows other applications to request arbitrary regions of the accumulated grid. In localization mode this is typically used to request either a fixed region around the current car's position or to request the region which has been updated on the last input. Note that both input and output interface only work with local grids. The transformation into object lists happens in an independent step.

Figure 3.3 shows the relation between **cGridMap** and OctoMap. There exist already a few available implementations of the abstract class **OccupancyOcTreeBase<NodeType>** in OctoMap, but none of them fulfils all necessary requirements. As every node needs to store not only its occupancy belief value but also the associated feature ID, the tree class **cGridOcTree** and the node class **cGridNode** have been developed. With the rest of the OctoMap package, this set of classes is compiled to the static C++ library called **octomap-gridoctree**. It is this library that stores the actual grid data. The class **cGridmap** takes care of managing the memory, storing and reading tree branches from disk and deciding the associations of feature IDs. Note that the cells always have a quadratic shape and the resolution is therefore always symmetrical in x and y direction.

For mapping, the gridmap stores feature detections with absolute world coordinates. In order to comply with existing coordinate systems, the *Universal Transverse Mercator* (UTM) [35] coordinate frame is chosen. It is a global coordinate frame which projects every world position to a two-dimensional grid. This makes it necessary to convert every location in the vehicle reference frame to UTM but simplifies plausibility checks and the comparison with reference aerial imagery.

3.3.2 Extensions for large-scale mapping

The gridmap uses Octrees as underlying data structure in the form of the mapping package OctoMap (see Section 3.3.1). Octrees are designed to represent three-dimensional space in a very efficient manner. Even though the space in the gridmap is currently represented in a two-dimensional way, the possibility to switch to a full three-dimensional representation will become important once the localization and path planner modules will be able to handle three-dimensional information.

Several adoptions are necessary to use the OctoMap framework efficiently in a large-scale mapping project and have been implemented as part of this thesis.

Port OctoMap to Windows 64-bit

The authors of OctoMap originally developed the library as a 32-bit application. As the source code is available under BSD license it is possible to compile it as 64-bit library. However, this requires some changes in the source code as various variable overflows occur which at times is hard to track down. This includes assumptions about the number of bytes being used for pointers which differs from 4 byte under 32-bit to 8 byte under 64-bit. Also, the range of values changes for certain data types like *size_t*. Furthermore, the library was at first under Windows not able to store acquired map files on disk and load them again due to a compiler bug with Visual Studio. All of these issues have been resolved.

Swapping of tree branches on disk

The OctoMap framework provides mechanisms to load complete tree structures from disk. The tree is originally always kept in its entirety in the main memory after the initial loading. As the mapping of large-scale areas requires trees that cannot fit entirely into the main memory, a technique to swap branches of the tree on disk is implemented. A simple heuristic determines whether a tree branch is

likely to be used in the next cycle and loads the corresponding branch from disk, or stores it on disk otherwise.

Larger tree structures

The original OctoMap implementation always uses a tree size of up to 16 levels in the tree. This allows the map at a resolution of 0.04 m × 0.04 m to cover a space of $2^{16-1} \times 0.04$ m = 1310.72 m in each dimension. In the context of large-scale mapping with potentially several hundred square kilometers of map area this means however a limitation that has to be tackled. In their paper [16] the authors of OctoMap state that:

> *"[...] while this [number of 16 depth levels] is sufficient for most indoor applications, the implementation can directly be extended to 32 depth levels, allowing to cover* $2^{32} \times 0.01$ *m = 42 949 672.96 m [sic] at a resolution of 1 cm."*

While it is true that the tree depth can be passed as a parameter to the OctoMap library, a lot of minor adjustments and bug fixes were necessary to actually get this rarely customized feature to work. Most of the library assumes 16 depth levels or less. Also the visualization called *octoviz*, which comes with OctoMap, is only capable to visualize trees up to 16 depth levels.

A reference frame has to be fixed when creating large-scale maps. The coordinate system UTM has been chosen in order to stay conform with existing global reference frames. UTM cells differ in size with the largest cell having a maximum edge length of 885 km. At a resolution of 4 cm × 4 cm a tree with 26 depth levels[10] is required to represent this area. Combining the possibility to use more than 16 depth levels and to dynamically swap branches between disk and main memory allows the use of one tree per UTM cell.

Merging of multiple maps

A procedure has been implemented to merge multiple maps into a single map file. This allows to produce single map files with different test runs and to combine them afterwards. The area in the map can be extended by this approach or the already mapped area can be refined. For the latter, both grid maps are considered as single measurements and combined using the Bayes' update mechanism. Combining maps for the same route and extending the mapped area can also happen at the same time as the update mechanism handles all cells in a generic way.

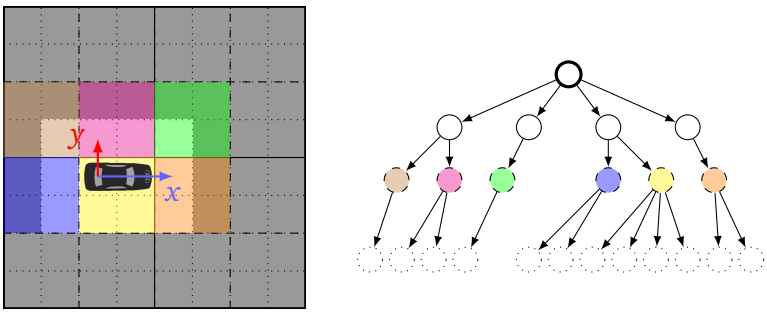
Endless driving mode

While mapping requires a large tree to represent an entire UTM cell, localization can be implemented with a smaller tree. This is because old information far away from the car's position can be discarded. In practice, a tree with 16 depth levels is used which gives a lower bound of some hundred meters of distance before information gets removed. This way of maintaining the car's absolute position is not a problem when outputting features because they are being outputted in coordinates relative to the vehicle. Figure 3.4 demonstrates the changes to the tree structure when grid cells are being freed. The implementation of the endless driving mode permits to drive an arbitrary distance while the memory consumption can at the same time be guaranteed to be limited to a maximum size.

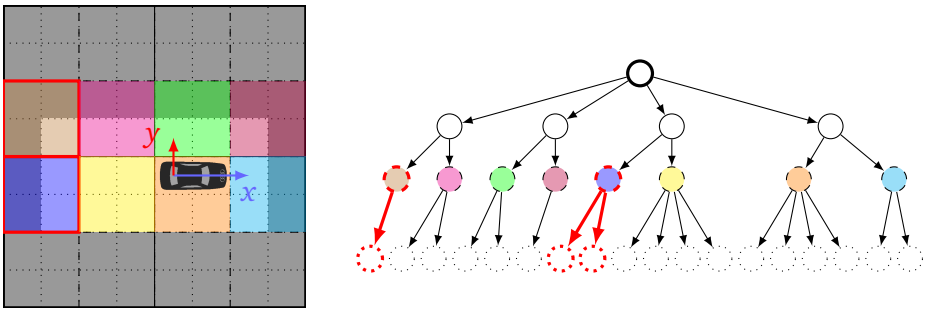
Node pruning in 2D

OctoMap offers the possibility to prune tree nodes if the information stored in all children nodes is identical. The space reduction achieved by pruning is virtually lost when octrees are used to exclusively store data which has been projected into the two-dimensional ground plane. Four out of eight leaf nodes store information about the ground plane (those four nodes that correspond to the quadtree that describes the same region) but the other four nodes will always be unknown (because the volumetric information has already been projected into the ground plane). In this configuration pruning can only

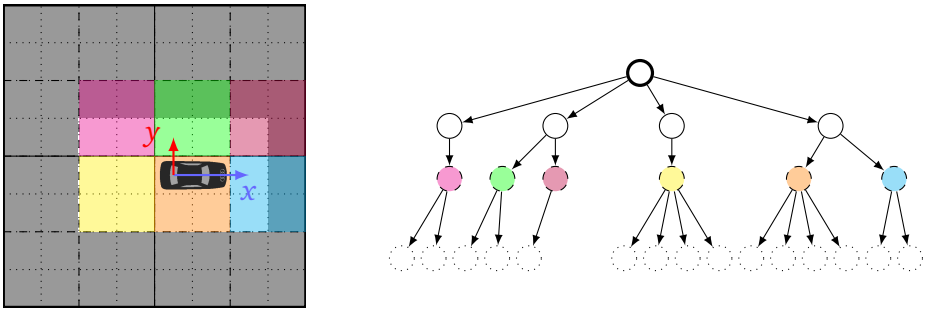
[10] A tree with 26 depth levels covers up to $2^{26-1} \times 0.04$ m = 1342177.28 m per dimension



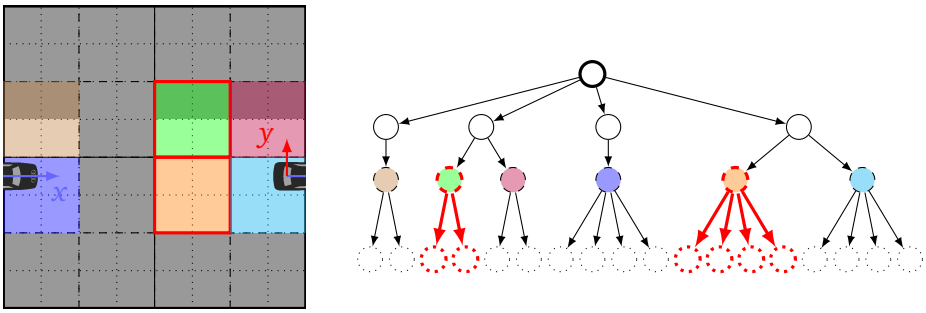
(a) *Vehicle starting to drive on a finite grid with the corresponding tree structure. The bright area around the car marks the currently known and mapped area; the rest of the grid is unknown. The colors visualize the correspondences between grid cells and tree nodes shown on the right. Unknown cells result in non-existing nodes in the tree.*



(b) *The known area grows as the car moves along the x axis which results in a more developed tree. Grid areas, that are considered to be far away from the car, are marked for deletion and visualized with a red border.*



(c) *The area which has been marked for deletion in the previous step has been freed. The corresponding nodes in the tree have been discarded and the area is considered as unknown again.*



(d) *The car reaches the boundary of the grid and starts reusing old areas again. Meanwhile, new areas get marked for deletion. The transition between dropping of the right boundary and entering the left boundary is seamless.*

Figure 3.4: *Endless driving mode for localization: when the car leaves the right border of the grid, it enters on the left side again in a seamless fashion. Old information about areas far away from the car is at some point discarded so that the area can be used again.*

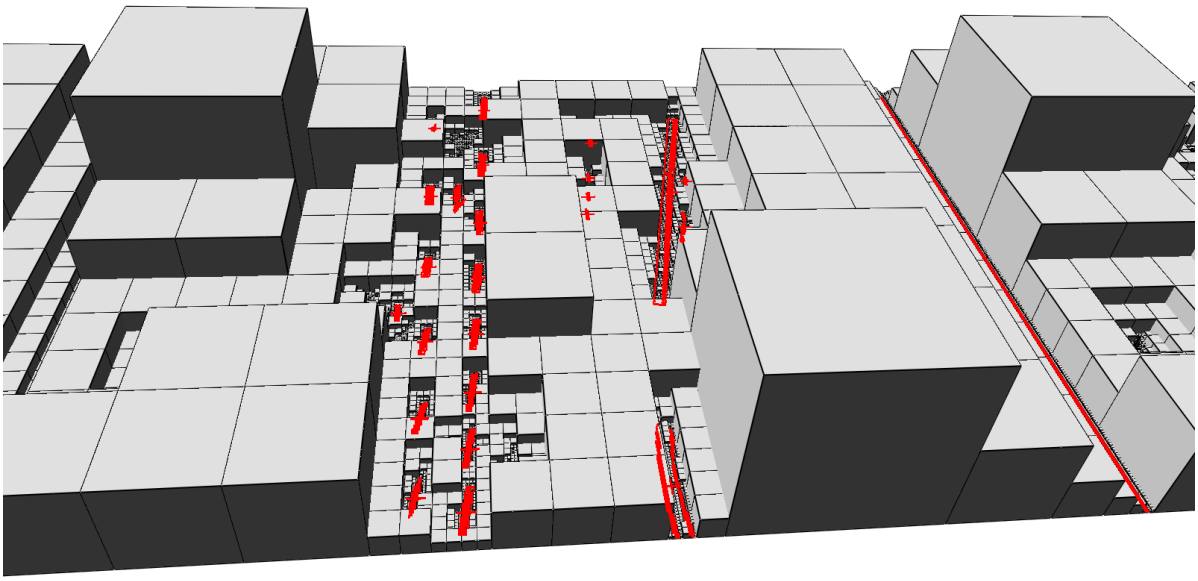


Figure 3.5: *Effect of 2D pruning on the tree structure of a volume representing a stretch of a road. The red lines represent detected dashed and solid lane lines. The cells have been binarized for visualization purposes. The cubes show the 3D structure of the Octree: the tree structure is fully developed down to the smallest cell if that cell is occupied. Large grey boxes indicate that the tree is significantly less developed and that it is pruned at a higher level. The bigger the boxes are, the more levels of nodes are pruned and the more space, memory and processing time is saved.*

take place if all eight nodes are unknown. While this saves an enormous amount of space in all regions that have never been mapped, it is not as efficient as it could be. Pruning in 2D is implemented to regain the benefit of space compression for areas that store identical information in the ground plane. This technique collapses eight child nodes into one parent node given that the four child nodes which describe the ground plane are identical. Special care has to be taken if a measurement update changes information so that a formerly pruned node has to be extended: while all child nodes can be populated with the parent's node information in the case of 3D pruning, only four of them have to be populated in 2D pruning and four have to be marked as *unknown*[11]. Figure 3.5 visualizes the effect of successful 2D pruning on a stretch of the map.

Selective updating

Code profiling shows that the biggest amount of processing time of the grid in localization mode is spent when information is inserted into or recovered from the grid. A selective cell update rule is introduced to decrease the number of cells that have to be updated in each step. This mechanism makes use of the fact that a sensor's field of view overlaps quite heavily in two consecutive time steps t and $t+1$. Also, the information in the grid for the area of the sensor's field of view at t can be cached. If at time $t+1$ a new update has to be integrated into the local grid, some cells do not benefit from being updated because they are already at their minimum (or maximum) value and a further negative (or positive) reinforcement would not change this.

[11] Nodes that represent cells that lie within an unknown region are actually not labelled as unknown but correspond to a non-existing node. They can therefore be considered as *implicitly labelled*.

Multithreading

Every layer in the map stores another feature type. The layers are functionally decoupled in a way that they do not interfere with each other. This design decision allows performing multithreaded in- and outputting of data. Multithreading is also implemented in the ADTF filters which have limited multithreading support off the shelf.

These extensions have been developed for different purposes. Some of them are vital for the system to work at all, like porting OctoMap to Windows 64-bit. Maps could not be generated at a large-scale without the larger tree structures. In practice, the map merging is also crucial as it allows to split mapping runs into multiple parts or to refine existing maps with new data. Swapping tree branches to disk enables the system to produce maps that cannot be fully contained in the main memory anymore. The endless driving mode is important for localization so that the car isn't limited to a confined area. Running the system in real-time for localization requires some performance improvements. These are implemented in terms of 2D node pruning, selective updating and multithreading. Numerical analysis in Section 4.2 shows their influence on the performance.

3.3.3 Tracking

Maintaining information in a common occupancy grid means that features are not associated to previous measurements of the same feature. As the grid implementation in this thesis seeks to create an output similar to that of Gaussian filters, mechanisms have to be defined to reidentify features and to extract the mean and the covariance for a given feature. Reidentification of features is the problem of keeping track of which cells belong to which feature and maintaining a unique ID per feature. Extraction of the mean and the covariance is the problem of describing the spatial uncertainty per feature in a feature-based way by approximation using the uncertainty information inherently stored in the grid.

Re-identification of features

Features are a set of connected cells. The probability of a cell of being occupied by the same feature type is higher than a given threshold for all cells in the describing set. It is easy to identify all cells that belong to one feature when this feature has been spotted for the first time because the detector output contains this information. However, this task becomes more difficult for subsequent detections of the feature for two reasons: first, because the same feature can be detected at a slightly offset location and second, because cells, which were formerly thought of as containing the feature, have been updated to probability values below the threshold. The form and the location of the feature are therefore evolving over time. This is known as the association problem.

Three strategies have been designed to solve this problem in a grid-based representation:

1. **Connected cells.**

   All cells that are connected and have a non-zero feature ID will be maintained in a way that they all have the same ID at all times. This is done by translating the local feature ID that has been given to a feature by its detector, into a globally unique ID which is then recursively spread out over neighboring cells. This strategy basically assumes that the same feature always has some spatial overlap in its detections and that distinct features are not too close to each other, so that incorrect feature merges are unlikely. These prove to be valid assumptions for short range sensors such as the topview cameras. Figure 3.6 illustrates this strategy.

4				5	
			5	5	
	3		5	5	
	3				
3	3	3	6		
	3	3			

(a) *Initial grid with some distinct features as shown by different IDs. A new feature with the ID 6 just got inserted.*

4				5	
			5	5	
	3		5	5	
	3				
3	3	3	**3**		
	3	3			

(b) *The feature with the ID 6 gets converted to ID 3.*

4				5	
			5	5	
	3		5	5	
	3		6		
3	3	3	3		
	3	3			

(c) *Another cell update triggered by the feature with the ID of 6.*

4				**3**	
			3	**3**	
	3		**3**	**3**	
	3		**3**		
3	3	3	3		
	3	3			

(d) *All neighbor cells get recursively converted to the common minimal feature ID of 3.*

Figure 3.6: *The fusion strategy "connected cells". The grids show only the feature IDs while the probability values are omitted (the feature type is identical for all features). Numbers in blue denote freshly inserted feature IDs and bold numbers mark cells that changed their feature ID. A feature with the ID 6 gets iteratively inserted into the grid. This influences its ID as well as the feature with the ID 5. Cells always recursively synchronize themselves to have the lowest ID among all neighbor cells so that higher IDs can be reused.*

2. **Overlap of area.**

   A minimal spatial overlap can be defined which is needed to identify a detection as an existing feature in the grid. If marking all connected cells as the same feature would cause multiple adjacent jittering features to be blurred into one big feature, employing this strategy could counteract that unwanted behavior. In practice though, this behavior has not been observed.

3. **Maximum distance.**

   If the same object is often detected far enough from the current believed position in the grid so that there is no overlap at all, a more flexible strategy is advisable. For a single feature, its nearest neighbor could be searched for and the distance to it could be used to decide whether they are associated. If a set of features is inserted into the grid at once then ICP[12] in one of its variants [36] can be used to find the best match between the detectors output and the existing map. This has not been necessary with the detectors at hand.

It turns out that current feature detections are always very close to the last detection with a significant aerial overlap. This is mostly due to the used topview cameras which by design only cover a field of view very close to the vehicle. The image for this close region is usually quite stable and detections are therefore quite accurate. Good results have for this reason been achieved with the first strategy. It is also the fastest because it can operate on a cell basis instead of having to consider the entire detection. The practical implementation is done in form of the design pattern called **Strategy Pattern** (see [37]) which makes it possible to conveniently exchange the strategy if necessary.

Extraction of the mean and the covariance

The grid map explains which cells belong to which feature. Several steps are involved in order to track single features over time:

1. The first step is to extract a list of features out of the grid-based representation. To accomplish this, the contours of all feature ID regions are found and kept in a list. A feature ID region is

[12] the *Iterative Closest Point* algorithm

an area in the map where all the cells have the same feature ID. With the previously described re-identification strategy called *connected cells*, all of the cells in this area are connected and form a closed and completely filled contour.

2. For every contour, a minimal bounding box is determined. Depending on the size, this bounding box is subdivided into multiple sub boxes so that every sub box can safely be assumed to contain a linear part of the feature shape.

3. For every sub box, the centroid[13] is computed. If the sub box is at the borders of the entire shape, then the centroid is moved on the border of the smaller side so that start and end points are located on the border. These start and end points are defined to be the means of the features.

4. Start and end points of features are tracked over time. For this, the sample covariance matrices of the point set describing the start and end points of the same feature as it changes over time are calculated. Using an iterative formula to calculate the sample covariance, only the last mean and covariance matrices have to be stored along with a count.

This procedure allows extracting the start and end points of features. Also, the sample covariance for features is computed which signifies the temporal variation of these points.

[13] The 2D centroid is defined as the first moment of an area over its zeroth moment. OpenCV is used to compute the shape descriptors, see http://docs.opencv.org/modules/imgproc/doc/structural_analysis_and_shape_descriptors.html.

4 Evaluation

The proposed mapping system is analyzed with respect to the accuracy of the output and the runtime performance. Sections 4.1 and 4.2 both first define the experimental design used to obtain measurements and present subsequently the results.

4.1 Accuracy analysis

The accuracy is measured in a grid-based and in a feature-based way, motivated by the fact that maps are produced in these two representations. Section 4.1.1 explains the approach of how to compare the maps to the ground truth data in order to compute the accuracy. Section 4.1.2 shows the application of the analysis to generated test maps and presents the resulting values.

4.1.1 Experimental design

Maps are supposed to describe parts of the world. Determining the accuracy of a map therefore means measuring the relative differences between the map and the world. The maps at hand are at the same time grid- and feature-based, which will be reflected by performing two comparisons. The association strategy of feature IDs to cells is not evaluated. As detailed in Section 3.3.3, this is because the observed associations almost always connect the correct feature cells and a simple strategy is sufficient to achieve this. The accuracy evaluation will be carried out on the dashed lane line detector as a lot of test data is available for it. It is to note, that the maps as produced by the grid depend heavily on the accuracy of the detectors. The numerical results of the described analysis are therefore only valid for the dashed lane line detector but the same analysis can be reused for other kinds of detectors.

For ground truth, this thesis relies on geographically referenced aerial imagery data with hand-annotated labels. This imagery is of very high accuracy with a maximum distance of up to 10 cm to the real world. It is only available for certain routes and spans about 2.3 km of road. These routes have been mapped with a prototype vehicle and the resulting maps are compared to the ground truth data. In a first step, the ground truth data has been preprocessed. This includes trimming, filtering and multiple file format conversions to go from so-called Shapefiles [38] to Matlab compatible MAT files[1]. A grid-based ground truth representation has been created out of the available feature-based representation. The visualization in Figure 4.1 shows the features in the ground truth data as shown by the open source tool OpenJUMP [39], which underlays the ground truth Shapefiles with geographically referenced aerial imagery.

Grid-based comparison

Grid maps assign probability values to cells that denote for each cell the likelihood of it belonging to a certain class. They are thus compared cell by cell to the ground truth. A cell in the ground truth map is containing a value of 1 if a feature is present in that cell. A threshold τ is used to classify cells into positive and negative predictions. Positive predictions (map cells with probability values greater or equal than τ) count as a *true positive (TP)* if the ground truth cell contains a 1, and as *false positive (FP)* otherwise. The *true negatives (TN)* and *false negatives (FN)* are computed correspondingly.

[1] Numerical analysis has been performed in Matlab.

Figure 4.1: *Ground truth features in Shapefile format in OpenJUMP. Green line segments mark hand-annotated dashed lane lines.*

The ground truth map contains many more feature-less cells than those with a feature (roughly 96% of the cells in the ground truth map contain zeros). This results in the fact that a map, that exclusively predicts zeros for all cells, produces a largely correct but of course unusable set of predictions. The correctness measures called *precision* (which is equal to the *positive predictive value (PPV)* for binary classification) and *recall* (which is equal to the true positive rate) are therefore favored as comparison metrics as given by

$$\text{precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}, \tag{4.1}$$

$$\text{recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}. \tag{4.2}$$

Also, a more probabilistic correctness measure is used in the form of the likelihood of the ground truth (GT) given the predicted map. To derive this, let $\boldsymbol{GT}_i$ be the probability value in the ground truth map for the cell i and let GT be the entire ground truth map so that

$$GT = \{\boldsymbol{GT}_i\}. \tag{4.3}$$

This definition yields for the likelihood $\mathscr{L}$

$$\mathscr{L}(GT|m) = \prod_i p(\boldsymbol{GT}_i|\boldsymbol{m}_i). \tag{4.4}$$

All the $\boldsymbol{GT}_i$ are either 1 or 0. With this information in mind, the likelihood for a single cell is defined as

$$p(\boldsymbol{GT}_i|\boldsymbol{m}_i) = \begin{cases} \boldsymbol{m}_i & \text{if } \boldsymbol{GT}_i \text{ is } 1, \\ 1 - \boldsymbol{m}_i & \text{if } \boldsymbol{GT}_i \text{ is } 0. \end{cases} \tag{4.5}$$

There is a catch with this definition: the minimum and maximum clamping thresholds prevent the likelihood per cell to ever reach a value of 1 or 0. Adjusting the ground truth values to be at the boundaries of the clamping thresholds would however bias the result in the opposite way: every probability value close to one of the clamping thresholds would be treated as a near-perfect prediction. The given definition is therefore more conservative. The best and worst cases for the average likelihood value are computed

to get a better understanding of this effect. An ideal map is constructed that contains the minimum clamping threshold value in every cell where the ground truth map contains a zero, and the maximum clamping threshold value in every cell where the ground truth map contains a one. For obtaining the worst case likelihood, an analogous map is constructed that contains the worst possible prediction.

In numerical analysis, the logarithm of the likelihood is usually preferable as it is numerically more stable to take the sum over the logarithmic values than to calculate the product of values between zero and one. However, as the map sizes differ significantly, comparing the log likelihood is not possible in a meaningful way as it linearly increases with the amount of cells. Normalizing the log likelihood comes back to calculating the average log likelihood value because a sum gets divided by the amount of its summands. At this point it makes sense to investigate the average non-log likelihood instead as it is more intuitive. It is thus one of the computed values. For a random guess given the prior distribution over positive and negative cells an average likelihood value of 0.5 would be expected, and better predictions tend towards a value of one[2]. Section 4.1.2 presents the result of the application of this methodology to the test data.

Feature-based comparison

Feature-based maps are stored as a list of features with global positions. Dashed lane lines in particular are stored as start and end points for both the test and the ground truth data. Extracting a feature map from a grid map requires calculating a single belief value from an arbitrary discretized probability dispersion. This is done by taking the average over all cells that contain the same feature ID.

The accuracy measurement involves the following steps:

1. For every feature in the map, try to associate it with its globally nearest neighbor (GNN). Identify outliers by applying a certain distance threshold. Associated features are counted as true positives and outliers as false positives.

2. Calculate the Euclidean distances between the start and the end points for all associated features.

3. Count the number of false negatives. For that, take the D-GPS positional information for the test data set and reprocess the trace of the car's position with a sliding window approach. The size of the sliding window equals the sensor's field of view. If a ground truth feature exists in that window that does not exist in the predicted map then classify this instance as a false negative. The effect of this is that every ground truth feature, that has not been detected but could have been detected, is counted as a false negative.

In the end, the output contains the number of true positives, false positives and false negatives. As feature-based maps are not dense maps, there is no reasonable definition for true negatives and the focus is therefore on the positive predictions. The probabilistic measures *recall* and *precision* are used for the feature-based comparison in the same way as they are defined for grid-based comparison (see Definitions 4.2 and 4.1).

Since the distances between predicted and actual features are also of high interest, the distances from start to start and from end to end feature points are calculated. The results will present the average distances along with the standard deviation. As a preliminary visual inspection suggests that there currently is a systematic offset between predicted and ground truth features, the translational vectors between the start and between the end points are derived from the test sets. Assuming that there is no systematic offset one would expect these vectors to fluctuate closely around zero (with a given standard deviation).

[2] Neither a likelihood of 0 or 1 can actually be reached because of the minimal and maximum clamping thresholds, see Section 3.1.3.

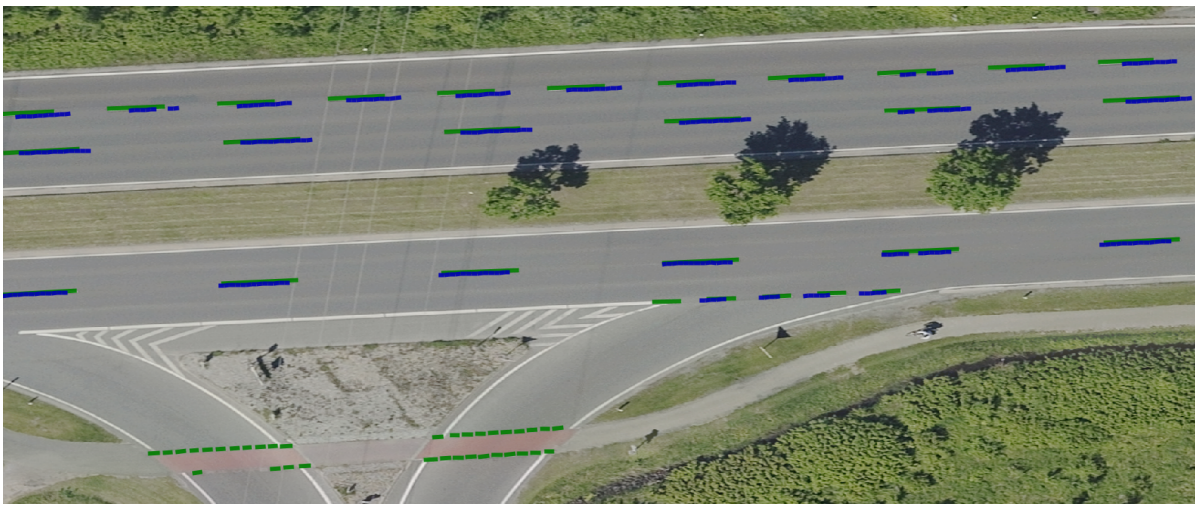


Figure 4.2: *Predicted map for dashed lane lines with underlaid aerial imagery. Green: ground truth markings, blue: detected features. The small dashed lane lines at the bike path are not detected due to their different shape and their location at the far outside end of the sensing area.*

4.1.2 Results

The accuracy analysis is defined for the grid-based and the feature-based maps. Therefore, results are also being divided into these two categories. Evaluation has been carried out for six different test runs on the same route. A seventh map has been created as a result of the combination of the six test runs.

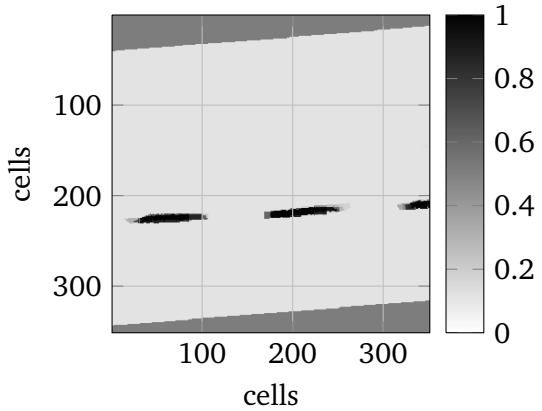
A first inspection is done on a purely visual basis. For this, the produced maps are converted into an OpenJUMP compatible format[3] which describes geographical features which is then used to underlay the resulting maps with aerial imagery. Figure 4.2 displays a part of the map.

A visual qualitative examination shows consistent maps for large parts of the test runs. Apparently the feature detectors have trouble detecting features correctly and entirely when the car is turning. This is likely due to the feature vector that is used to train the classification as it mostly contains geometrical properties that are not rotationally invariant. For a quantitative analysis the grid-based and feature-based numerical evaluation is performed.

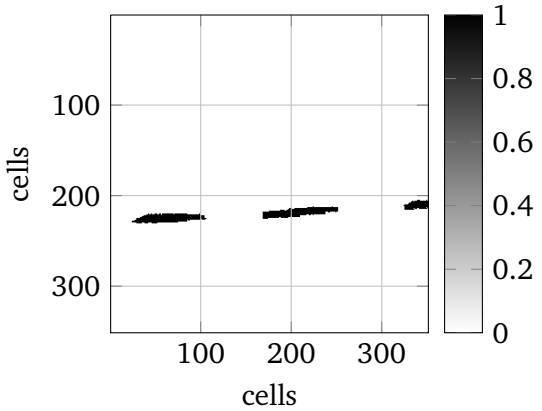
Results of grid-based comparison

Grid maps are compared cell by cell to the ground truth maps. In a first step, true positives / negatives and false positives / negatives are determined. Figure 4.3 illustrates the methodology for a small sample grid. The grid resolution is set to 4 cm and the belief values are color-coded. It is to note that the high grid resolution is very sensitive to any kind of noise: small lateral or longitudinal offsets result in a lot of cells that are only incorrect by a small translation. This particular map extract is from a map that has been produced from test data that has been recorded driving from East to West. The features suffer from a translational inaccuracy in longitudinal driving direction due to timing latency issues of the cameras (see Section 5.1). This effect can be reproduced in the opposite direction when investigating the accuracy of maps that are generated by driving from West to East.

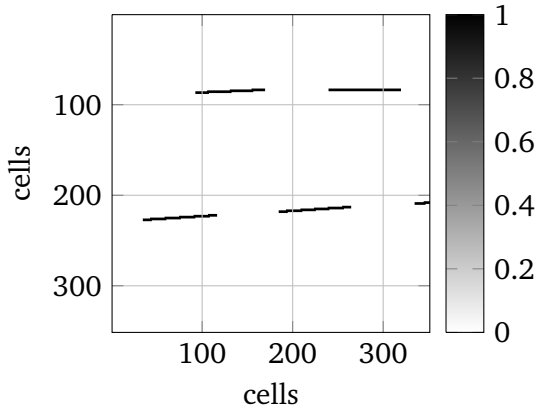
[3] The format is called *Geography Markup Language* (GML) which is an XML grammar. The grammar definition can be found at `http://www.opengeospatial.org/standards/gml`.



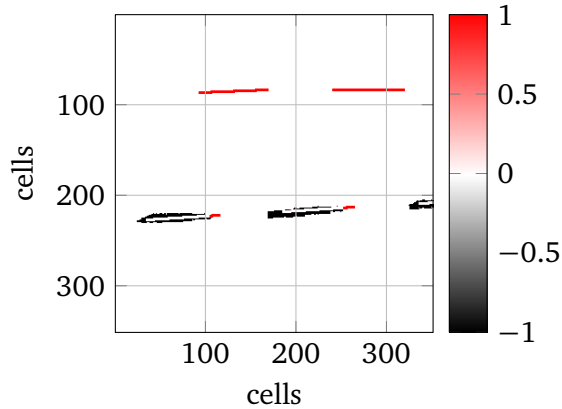
(a) *Extract of the grid map. The outer grey regions with a value of 0.5 indicate the unknown area. The light grey signifies that no features have been detected within the field of view of the sensors. The brightness of the colors indicate the probability value.*



(b) *The same region of the map after binarization with a threshold value of $\tau = 0.5$.*



(c) *The corresponding part of the ground truth.*



(d) *Comparison between the ground truth map in c) and the binarized map in d). Red cells indicate false negatives. True positives and negatives are represented by white cells while the black areas denote false positives.*

Figure 4.3: *Grid-based comparison between a map file and the corresponding ground truth for dashed lane lines. The grid is thresholded and the overlap between cells in the predicted map and cells in the ground truth map determines the true positives.*

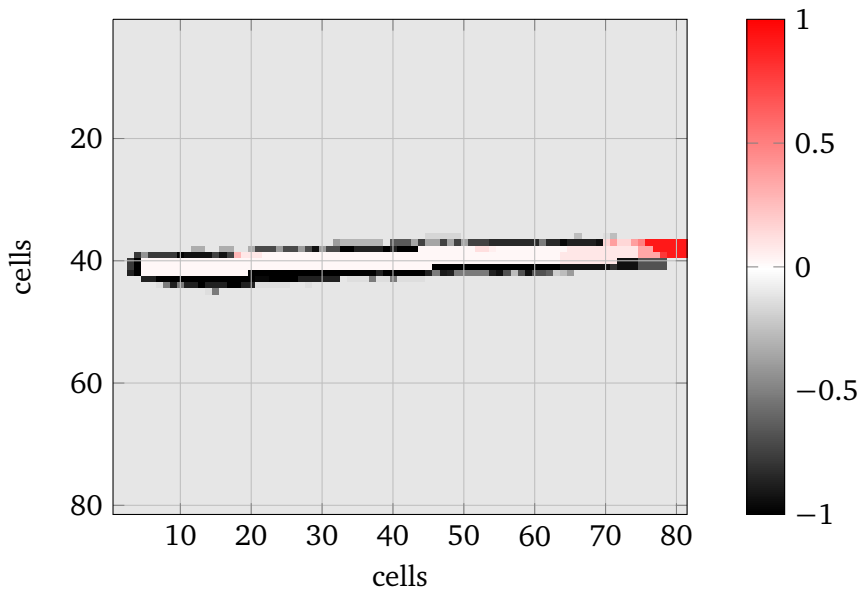


Figure 4.4: *Zoom on a single dashed lane line feature in comparison with the ground truth map. The detected feature is subtracted from the corresponding ground truth feature. White regions signify high likelihood values whereas black and red areas denote low likelihood values: red marks cells that are present in the ground truth but not in the prediction and black marks cells that are labeled as very likely in the predicted map but not present in the ground truth. The grey background results from the subtraction of the minimum clamping threshold from cells containing zeros in the ground truth map.*

Figure 4.4 shows a detection of a single feature for a grid resolution of 4 cm. It appears a bit blurred out and shows signs of imperfection (as is to be expected from real test data). Overall, its shape is clearly recognizable though which indicates that the data has been successfully fused.

The procedure as depicted in Figure 4.3 is applied to the entire area of all seven grid maps, which are grouped into runs from West to East and East to West. The likelihood per cell is also determined in the way that Figure 4.4 suggests. Table 4.1 shows the statistics over the correctly and wrongly predicted number of cells as well as the average likelihood value per cell.

These numbers are obtained for a grid resolution of 4 cm. As it can be seen, the false positive and true negative rates are of low interest due to the very high number of easy to predict negatives. The

	East to West	West to East	combined map
recall	0.159	0.296	0.334
precision	0.130	0.265	0.242
average likelihood per cell	0.559	0.627	0.629
false negative rate	0.841	0.704	0.666
false positive rate	0.007	0.005	0.006
true negative rate	0.993	0.995	0.994

Table 4.1: *Results of the accuracy analysis on the grid-based maps for different driving directions. The combined map merges all runs into a single map. The average likelihood value can only vary between 0.1246 and 0.8807 due to the use of clamping thresholds.*

	front camera	left camera	rear camera	right camera	fused map
recall	0.156	0.121	0.168	0.270	0.171
precision	0.179	0.150	0.149	0.157	0.209
average likelihood per cell	0.557	0.537	0.550	0.588	0.579
false negative rate	0.844	0.879	0.832	0.730	0.829
false positive rate	0.005	0.008	0.007	0.006	0.005
true negative rate	0.995	0.992	0.993	0.994	0.995

Table 4.2: *Results of the accuracy analysis for local maps in localization mode before and after the gridmap. A threshold of $\tau = 0.5$ is applied to determine the true and false predictions. The precision for the fused local map is higher than the average and maximum precision for the single camera detections.*

overall accuracy seems to have a substantial amount of space for improvement. However, the very fine grid resolution implies that a small translational error in detections causes a lot of false positives and false negatives. Creating feature detectors is not part of this thesis and is active work in progress. It is important to note though, that the fusion of the gridmap increases the accuracy: the recall and average likelihood values of the combined map are higher than the single mapping runs while the false negative decreases.

The average likelihood value is not bound between zero and one because of the usage of minimum and maximum clamping thresholds. An ideal map results in an average likelihood of 0.8807 while the worst possible prediction of a map yields an average likelihood of 0.1246.[4]

The same analysis is applied for investigating the effect of using the gridmap software in localization mode. For that, the local detection grids as generated by the detectors for each frame in localization mode are compared to the fused detection grids after the gridmap. As the gridmap is responsible for fusing the detections from the four topview cameras around in the car into local map, the detections before the gridmap are separately evaluated for each camera. The output after the gridmap is set to the same region of interest that has been used on the input and one local map is produced for every input. Table 4.2 shows the results as averaged over six test runs. Only dashed lane lanes as detected by the topview cameras are used for the comparison. First of all, a notable difference between the detections on the images of the different cameras is observed which is due to different training data, camera parameters and lightning conditions (this additionally illustrates the difficulty of constructing a coherent local map based on non-coherent sensor data). The recall for the fused local map (0.171) is very close to the average recall of the cameras (0.179): the gridmap cannot make the detections themselves any better than the output of the detectors. However, the significant advantage of using the gridmap lies in the reduction of weakly detected false positives, which explains why the precision for the fused map is with a value of 0.209 higher than the precision for any of the detectors running on a single camera. The average likelihood per cell is also slightly better with 0.579 than the average likelihood of 0.558 for the single camera detections (keep in mind that the likelihood has an upper and lower bound different than zero and one, see above).

[4] The maximum clamping threshold is set to 0.971 and the minimum clamping threshold to 0.1198 as suggested by the authors of OctoMap. The feature type is dashed lane lines and the grid resolution is set to 4 cm. All of these parameters are identical for the experiments from which the values in the rest of this chapter have been obtained.

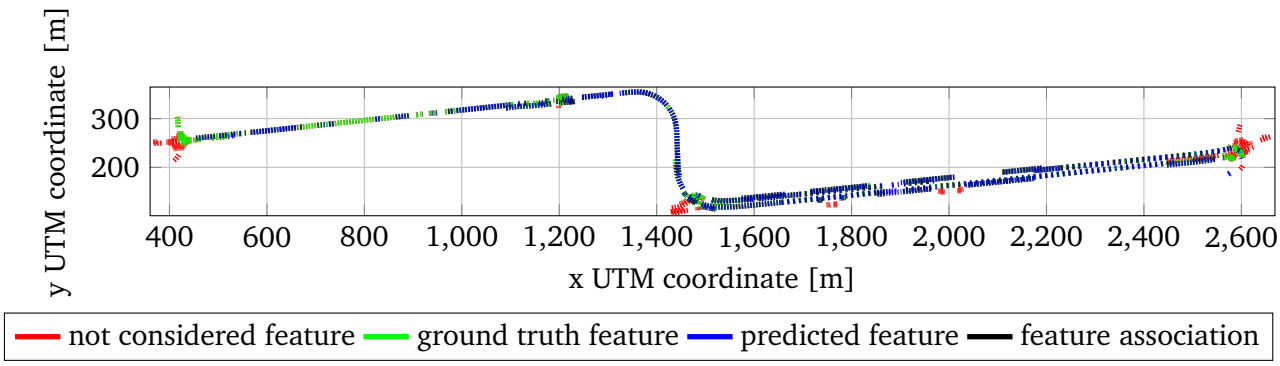


Figure 4.5: *Overview of the analysis of the feature map. Red lines denote features that are not included for the analysis since they have never been in the field of view of any sensor. Green: ground truth features, blue: predicted features, black: association between detected and ground truth features.*

A manual inspection of the false positive and false negative rates shows that the detection results get significantly worse while the car is turning. This effect is reproducible for both the single camera detections as well as the fused local map. The negative impact on the detection accuracy originates from the fact that the detectors are not trained to find features on rotated images and are not invariant to rotation either.

With the results in Table 4.1 and Table 4.2, the fusion of the gridmap is demonstrated to enhance the quality of global and local maps. The gridmap is of course by itself not able to detect any feature that has not been detected by any of the single camera detectors, but it allows to remove a lot of false positives.

Results of feature-based comparison

Feature map comparison implies the matching of map features to ground truth features. Hence, the entire set of dashed lane line features from the map and the ground truth features are compared for a chosen belief threshold. Higher thresholds imply that only features with a higher confidence are used for the comparison. This leads to a higher precision and a lower false positive rate but also means that less ground truth features are recognized (higher false negative rate and lower recall). One sample map is picked to demonstrate the analysis (the threshold is at $\tau = 0.5$, as suggested in Section 3.1.3). The global map is pictured in Figure 4.5, whereas Figure 4.6 presents a closer view for this sample map.

This kind of analysis is run on six feature maps resulting directly from available data sets. A seventh virtual feature map is created by merging the six test runs into one. The data sets are divided into test runs going from East to West, West to East and the combined run. All runs have been performed on the same route though and the only features with a belief value higher than $\tau = 0.5$ are considered for the comparison. Table 4.3 sums up the results.

The reason that these tables are divided into mapping runs from different driving directions is to highlight a known systematic latency issue (see Section 5.1). It can easily be seen from the numerical results that the average translational vector for both start and end points varies depending on the driving direction. The features are on average always a bit closer to the car because of a fluctuating negative timing latency from the topview cameras. The combined map doesn't show this behavior as the latency

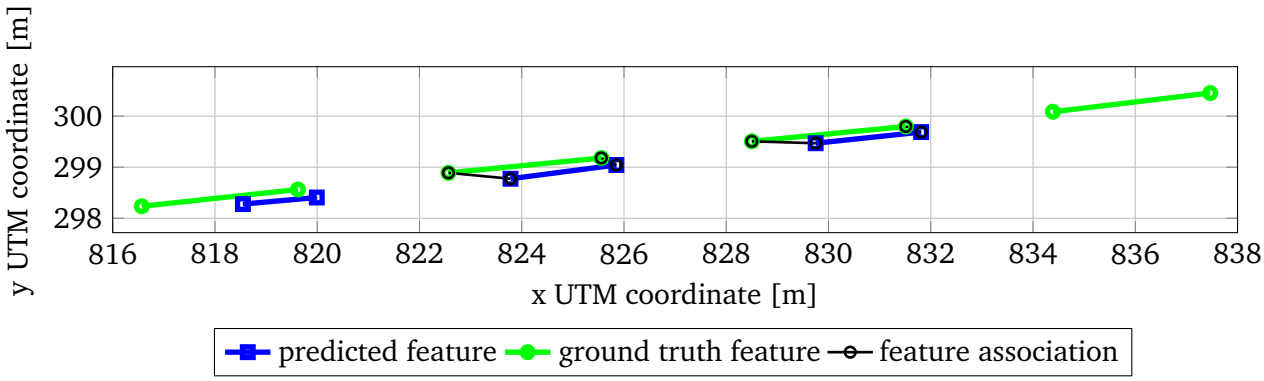


Figure 4.6: *Comparison of the feature map to the ground truth features, zoomed in. Black lines mark the successful associations between features in the map (blue) and ground truth features (green). The green feature in the lower left bottom is not associated to the blue feature because the distance between these two features is bigger than the distance threshold of 1.6 m, which leads to a false negative and a false positive prediction. The two features in the middle are successfully associated (both represent a true positive). The green feature in the upper right corner has no counterpart in the feature map: it is counted as a false negative.*

	East to West	West to East	combined map
number of true positives	264	271	477
number of false positives	78	84	146
number of false negatives	307	290	457
precision	0.773	0.762	0.766
recall	0.462	0.484	0.511
average distance of start and end points [m]	0.69 / 0.86	0.57 / 0.49	0.63 / 0.63
std. dev. distance of start and end points [m]	0.33 / 0.25	0.28 / 0.37	0.27 / 0.39
average translation of start points [m]	(0.47, -0.22)	(-0.46, -0.15)	(0.03, -0.13)
std. dev. translation of start points [m]	(0.52, 0.19)	(0.36, 0.18)	(0.64, 0.20)
average translation of end points [m]	(0.79, -0.15)	(-0.12, -0.05)	(0.30, -0.10)
std. dev. translation of end points [m]	(0.36, 0.19)	(0.52, 0.30)	(0.64, 0.21)
average length ratio	0.87	0.83	0.90
standard deviation length ratio	0.20	0.19	0.21

Table 4.3: *Results of the accuracy analysis on the feature-based maps for different driving directions. The combined map merges all runs into a single map. The abbreviation std. dev. denotes the standard deviation.*

influences are averaged out. The average translational vector for start points for the combined map is down to (0.03 m, -0.13 m) but for the cost of an increased standard deviation.

The recall for the combined map is higher than the average over the single runs. This is due to the distribution of the features on the road: the detector has troubles identifying features on adjacent lanes on the far outside of the field of view which are then classified as a false negative for a run from one direction to the other. On the run in the opposite direction, these features are nearby to the car and detected with high confidence and result in one less false negative.

The grid-based and feature-based analysis is developed as a general framework for accuracy-related investigations and serves multiple purposes. Obviously, it allows determining whether a given map achieves a desired accuracy and adheres to the requirements. This permits to reject bad test runs and to quantify the need for a better mapping of certain regions.

Moreover, it allows spatially highlighting regions of low accuracy within the map by applying the analysis in a sliding window approach. This can potentially lead to discover systematic problem areas for detectors such as turns or ramps.

It can also be run on maps produced from single sensors at a time to measure their impact on the overall quality[5].

Another use for the evaluation framework is to test different extensions or hypothesis for any module in the toolchain: for instance, the timestamps of the raw images were manually adjusted to correct for the latency issues with the cameras. The resulting maps were compared to the maps without timestamp correction and the quite drastic negative impact on the accuracy was demonstrated. The output of detectors before and after fusion into a local map around the car can also be analyzed in the same fashion as entire maps. Problems due to calibration offsets between cameras or due to roll and pitch of the car can be determined in this way.

Furthermore, detectors can find possibly optimal models by creating a receiver operating characteristic (ROC curve) which plots the true positive rate versus the false positive rate.

4.2 Performance analysis

The performance analysis focuses on the file size of maps, the runtime and the memory usage. Section 4.2.1 defines the analysis method while Section 4.2.2 presents the results.

4.2.1 Experimental design

Runtime is being measured by using the tools offered by the framework ADTF. Memory usage and file size is being measured with system tools. All measurements are averaged over the available test data. The requirements are quite different when producing maps or when using parts of the toolchain in localization mode.

Mapping

The runtime is of minor importance when generating maps because it is an off-line process. Sensor data is recorded while driving and post-processed in the lab. However, the file size of the map files does matter. For this, map files generated over six test runs for the same route are being used. To increase

[5] This technique has helped for example to identify the rear view camera as very susceptible to changes in illumination and made some test runs therefore unusable. It is especially unfortunate if this happens to the rear view camera as it updates the local map last in localization mode and has the potential to decrease the belief in true positive detections before they leave the field of view.

feature type	size per meter of route in $\frac{\text{KiB}}{m}$	size per square meter of mapped area in $\frac{\text{KiB}}{m^2}$	average file size in KiB
dashed lane lines	0.346	0.030	858
solid lane lines	0.366	0.031	907
arrows	0.238	0.020	590
stop lines	0.215	0.018	532
poles	0.285	0.010	707

Table 4.4: *File size of the grid-based maps for different feature types. The test track has a total length of 2.48 km.*

the sample size, every top view camera is generating its own map. The resulting file size is measured as a function of the mapped area and length of the route. Additionally, the influence of the grid resolution and two of the gridmap extensions are measured in the same way.

Localization

The grid is used in online localization mode to fuse features from multiple sensors in a local window around the car. Therefore, the runtime and the memory usage is important. The runtime for multiple datasets is measured over the entire chain of involved grid software modules for one feature detector. The memory usage is periodically checked and the constant overhead stemming from ADTF is subtracted. Both runtime and memory consumption are measured for different configurations of the system in order to quantify the optimizations as described in Section 3.3.2.

4.2.2 Results

The results of the performance measurements for mapping and localization mode are detailed in the following two sections. For mapping, the focus lies on investigating the file size per map while it shifts to the memory consumption and runtime for localization.

Mapping

Multiple datasets are used to compute the average map file size per feature. Comparing the area size per map is more meaningful than comparing the route length (because the sensors' fields of views differ) but from a practical point of view the route length is still interesting. Table 4.4 presents the different file sizes per feature type.

Dashed and solid lane lines appear much more common than arrows, stop lines and poles. The effect is that more space can be pruned for the rare features so that the resulting tree is more compact and the file smaller. Poles are detected with long-range sensors and the mapped area is therefore bigger for the same route as compared to all other features, which are detected with short or mid-range cameras. This explains the larger file sizes for poles in terms of $\frac{\text{KiB}}{m}$ but smaller file size when comparing the size in $\frac{\text{KiB}}{m^2}$.

The influence of some parameters to the resulting file size is also measured. For that, maps for dashed lane lines are generated with coarser resolutions and with disabled extensions. Table 4.5 presents the

configuration	average file size in KiB for maps containing dashed lane lines
4 cm resolution, all features enabled	858
8 cm resolution, all features enabled	376
16 cm resolution, all features enabled	171
4 cm resolution, without 2D pruning	4318
4 cm resolution, without gzip	21058

Table 4.5: *File size of the grid-based maps for different configurations. The test track is the same as for Table 4.4 and has a total length of 2.48 km.*

results of the measurements. Maps with a resolution of 4 cm are considered as baseline to which the other measurements are compared. The measurements with resolutions of 4 cm, 8 cm and 16 cm are taken with all extensions enabled to not distort the effects resulting from changing other parameters. The map sizes for the two extensions 2D pruning and gzip are measured with a resolution of 4 cm.

The file size for maps with a resolution of 8 cm is about 2.28 times smaller than for maps with a resolution of 4 cm and roughly 2.20 times bigger than for maps with a resolution of 16 cm. Theoretically, a four times smaller map could at most be expected when lowering the resolution by factor two in both dimensions. However, there is a certain fixed overhead to the maps as the tree structures, which contain the information in the map, are of a fixed size and are mostly empty anyway. The implementation of 2D pruning saves about factor 5.03 of storage space. Compressing the files with gzip results in 24.54 times smaller map files. As a consequence, the gridmap software automatically saves and loads maps that are compressed with gzip.

These values apply for the size of grid maps. Feature maps can be extracted out of grid maps and are currently stored as .csv files. These are on average about factor 2.58 smaller - which is not a lot, given that the information stored is significantly less. This is mostly due to the grid map being stored in an extremely compact and optimized binary format whereas the feature maps still have a huge optimization potential regarding file size. At the moment, the human-readable format of CSV files is for maintenance and plausibility checks preferred over any optimized binary format as file size reduction is not a pressing necessity.

Overall, the file sizes are reduced to a usable dimension so that even mapping at large scale becomes feasible.

Localization

The runtime and the memory consumption of the grid and all of its subcomponents are measured with timing spots for all six test runs. The runtimes are averaged and presented in Figure 4.7 as a cumulative plot which shows the median and average value as well as the 75, 85 and 95 quantiles.

These numbers include all of the extensions and optimizations that are implemented for the gridmap. As it can be seen from this chart, the gridmap runs in real-time up to a frequency of 1000 ms / 26.92 ms = 31.15 Hz in 95% of the inputs. These absolute values depend of course heavily on the environment in which the software is run: they change with a different amount of sensors, a different sensing area or different hardware.

Some gridmap extensions have been disabled one by one in order to measure their impact on the runtime. This refers to the extensions *selective updating* and *2D node pruning* as detailed in Section

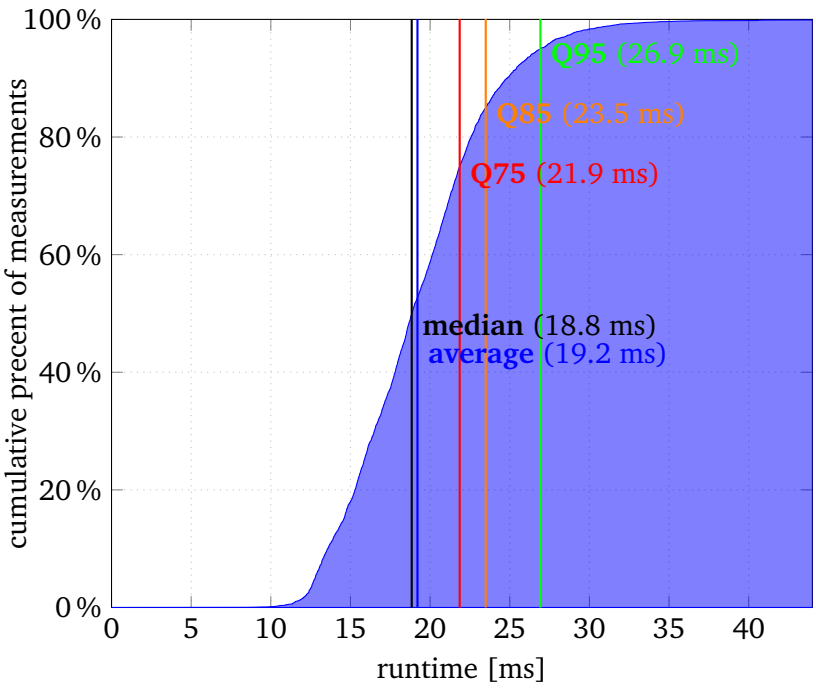


Figure 4.7: *Cumulative plot of the runtime of the gridmap. The grid resolution is at 4 cm and the detected feature type is dashed lane lines.*

3.3.2. The grid resolution is also made coarser (while all of the extensions are enabled) to gain a better understanding of its influence on the runtime. Figure 4.8 demonstrates the results.

The selective updating strategy already saves about 10% of runtime which is due to the reductions of costly inputs into the gridmap. The implementation of the 2D pruning mechanism saves about another 15% of runtime as the tree doesn't have to be traversed to the bottom if cells are empty. These two extensions make for the bulk of runtime savings and enable the gridmap to operate in real-time in conjunction with detectors with an input frequency of 30 Hz (e.g. camera-based detectors).

The most important runtime savings though are clearly achieved with a reduction of the grid resolution. It is important to note that the runtime and accuracy tuning is always focused on grids with a very fine[6] resolution of 4 cm. That is, if the performance requirements can be satisfied for this very fine resolution, then larger cell sizes will not only also be fast enough but theoretically even result in a quadratic speedup.

The memory usage is also measured. The software environment ADTF needs about a constant amount of main memory which is subtracted from the measurements to provide the raw memory demand of the grid in localization mode. The results are depicted in Figure 4.9.

Utilizing coarser grid resolutions effects the memory consumption significantly as the memory usage drops by about 19% when switching from a resolution of 4 cm to a resolution of 16 cm. There are multiple influences of changing the grid resolution on the memory consumption: first of all, less probability values need to be stored. The tree structure however stays the same and the maximum area which can be mapped gets larger. Pruning can potentially be applied less often as fewer cells contain values at the maximum or minimum clamping threshold (which is the most often occurring use case for pruning).

2D pruning saves on average 3.79 times memory compared to not using this extension. For understanding why it has such a big impact on the memory usage, we need to elaborate on the amount of

[6] Chris Urmson, lead director of Google's self-driving car project, disclosed at his speech at the RoboBusiness 2013 in San Francisco that their mainly lidar-based maps recently switched from a grid resolution of 15 cm to 10 cm.

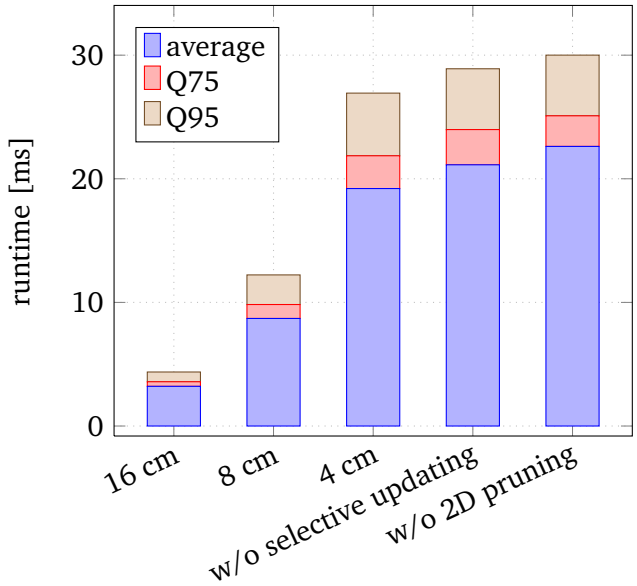


Figure 4.8: *Runtime for different gridmap extensions in localization mode. Using coarser grid resolutions leads to lower runtime requirements. Implementing selective updating saves about 10% and 2D pruning about 15% of necessary runtime as compared to the implementation at 4 cm resolution without extensions.*

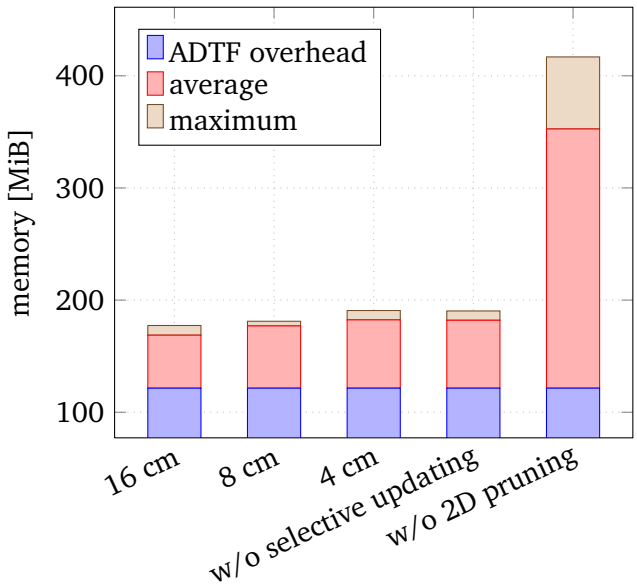


Figure 4.9: *Memory usage for different gridmap extensions. Most of the memory can be saved by implementing the 2D pruning which results in pruning significantly more nodes and creating a sparser tree structure.*

memory needed per node on a 64-bit system. Leaf nodes in the tree are 24 byte big: 4 byte for storing the float probability value, 4 byte for the uint32_t feature ID, 8 byte for the pointer to the array of children and 8 byte for the uint64_t pruning history. This last entry is needed to be able to recursively expand the node structure if it has been pruned in 2D. As a size of 24 byte is a multiple of 8 byte, no padding is added by the compiler. An inner node in the tree is 88 byte big: 24 byte for the same information as stored for a leaf node, and 8×8 byte for the pointers to the children (this array only gets allocated for inner nodes). This results in saving the memory needed for 7 leaf nodes and 1 inner node when pruning one level in the tree and corresponds to 256 byte or more than ten times the space needed for a leaf node.

Selective updating requires slightly more memory than not using it. This is due to the fact that the latest local map is stored in memory to accelerate the memory access in the next step. For the data used to generate the plot in Figure 4.9 a local window at a resolution of 4 cm with an area of 8 m $\times$ 8 m around the car is kept in memory. This local map stores information for $\frac{8\text{m}\times 8\text{m}}{0.04\text{m}\times 0.04\text{m}} = 40000$ cells. One cell is 8 byte big with 4 byte for the float probability value and 4 byte for the uint32_t feature ID. This leads analytically to an additional memory consumption of 40000×8 byte ≈ 0.3 MiB (ignoring a small constant overhead) which is verified by the empiric measurements (average additional memory usage of about 0.34 MiB). This small price in terms of memory consumptions yields a significant improvement when comparing the runtime (see Figure 4.8).

Memory savings alone are of less interest than runtime improvements. However, as most of the runtime in the current setup is a result of copying memory, the memory savings of the 2D pruning mechanism are reflected in the runtime. The absolute memory usage with 60.88 MiB on average for a grid resolution of 4 cm is rather low and negligible given the available hardware in the car.

The performance analysis shows that the proposed system is fulfilling all performance requirements for mapping and localization mode. It is however not independent of its technical environment: running on specialized hardware in the car would mean changes to the performance requirements and results. Also, using detectors with an extremely high field of view, such as some radar scanners, might result in higher data streams on the input and output of the grid. This could reduce the output frequency of the grid in localization mode. As of now though, the system has been measured on the actual hardware in the car with the available feature detectors and is capable of performing in real-time in localization mode and producing compact maps in mapping mode.

5 Discussion

The previous chapters presented the problem of mapping, described how this problem is being tackled and analyzed the results. In this chapter, the known issues are discussed, followed by an outlook of the system usage and a conclusion.

5.1 Known issues

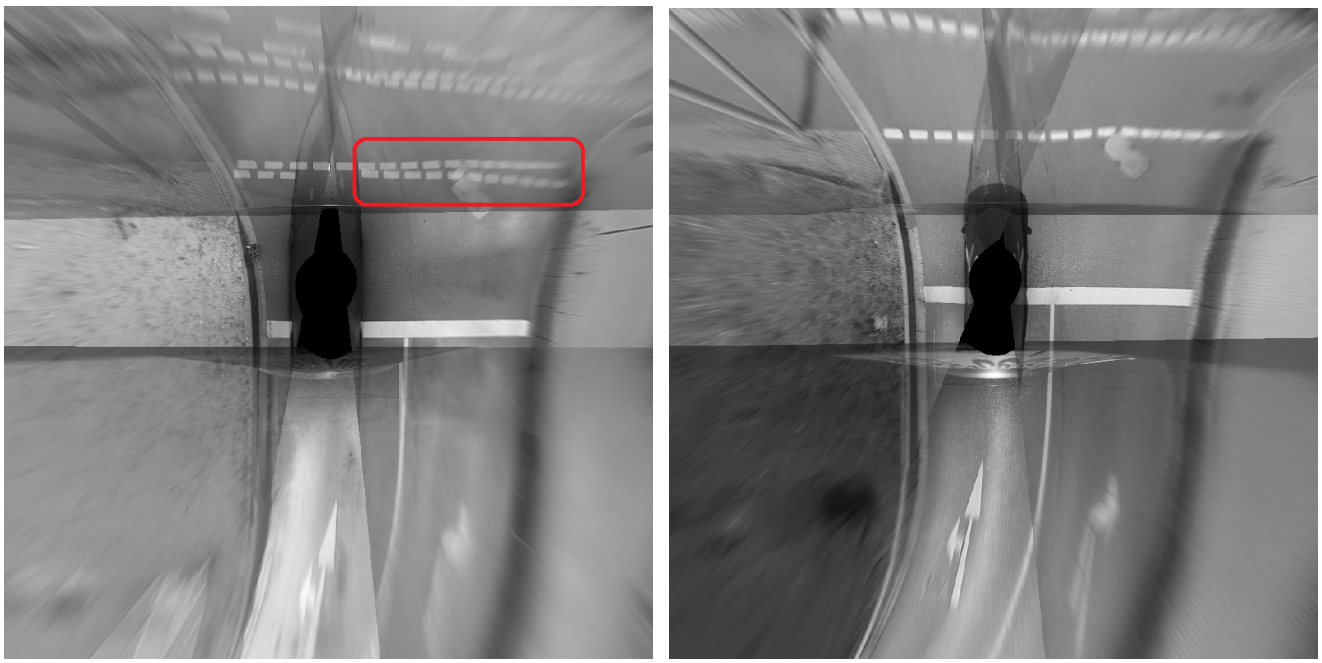
During most of the development phase the biggest accuracy issues stemmed from sensor latencies and camera calibration concerns. A mixture of absolute (time difference as compared to GPS time) and relative sensor latencies (timing differences between sensors that are thought to be the same, e.g. topview cameras) introduced a systematic error in the conversion from location in camera space to world position. The detectors were correctly detecting features in image space, but the image from the video stream was erratically timestamped. This made it impossible to associate the exact pose of the car when the image was taken. As a result, the features' positions were slightly offset to their actual positions even though the car's pose was known with high precision at all times and the detectors were functioning as expected.
Camera calibration issues arose because the intrinsic and especially the extrinsic parameters of the topview cameras were not highly accurately determined in the beginning. The ultra-wide-angle fisheye lenses of the cameras are nontrivial to calibrate because of the significant image distortion. An imprecise calibration however results in a wrong mapping of image coordinates to world coordinates and therefore decreases positional accuracy. Figure 5.1a illustrates the camera calibration issues and Figure 5.1b shows the state of the cameras after calibration. The calibration issues have been mostly dealt with over the course of time and were improved to a state where the resulting accuracy is acceptable.

A shortcoming of the current mapping software is that height information is not integrated. This is primarily based on the fact that neither the feature detectors produce nor the localization modules handle height information. Also, timing constraints in localization mode are much harder to meet if feature association and cell updates in the grid have to be performed in 2.5 or 3D instead of pure 2D. In the long run, this issue has to be addressed anyway as bridges or overpasses can't be represented in 2D only. Multi-level surface maps [14] or 3D approaches can be evaluated in close cooperation with the feature detector and localization modules.

5.2 Outlook

The mapping system as described in this thesis is going to be utilized in the foreseeable future for several projects at the Volkswagen AG in Germany and in the USA. As it is an evolving system, it has been designed in a modular and extensible way. It offers a unified approach for fusing sensor information from multiple sources and maintaining grid-based and feature-based maps at the same time without storing redundant information. Furthermore, an extensive analysis framework allows to investigate the detector and the resulting map quality. Some extensions for it are already known to be needed in the medium term while others have more of an optional character.

Currently, high precision map-based localization is being implemented in a feature-based and in a grid-based fashion. Both use Monte Carlo localization approaches and mostly differ in the map-matching technique. Comparing a feature-based and a grid-based localization approach makes therefore direct



(a) *Overlaid topview images with insufficient camera calibration. The region marked in red shows an example of a region of deviation between the front and the right topview camera.*

(b) *Overlaid topview images with better camera calibration. A precise investigation shows that the calibration still is not perfect but is improved substantially over the prior state.*

Figure 5.1: *Example of the camera calibration issues. The four topview camera images are overlaid by showing the average grayscale value for every pixel.*

use of the hybrid maps that are generated by this mapping system. Even the implementation of a hybrid localization technique has been contemplated.

At the moment the mean and the covariance of the position of a feature are extracted from the grid. This assumes that those two moments are sufficient for describing the underlying generating probability distribution. This is only true for Gaussians, therefore this extraction is limited to represent the result of a Gaussian filter as other distributions are described by other parameters. While the Gaussian filters are amongst the most popular temporal tracking filters, the parameter extraction could be adapted in the future to match the result of other filters (e.g. multimodal filters such as a particle filter).

Several feature detectors will be adapted in the near future to work with the grid. More features mean more complete maps but also the need for more ground truth data in order to evaluate the detectors' accuracies. Features could also be extracted from geographically referenced aerial imagery if this becomes widely available in a good quality.

For now, the generated maps are solely being used for autonomous driving with prototypes. Closer integration with third party maps might be an upcoming topic once steps towards a larger fleet usage will be considered. Necessary adjustments for that will include aligning the hybrid grid maps with lane models of the third party maps (which are usually stored as topological maps). A possible approach could also be the integration of features into separate map layers of stock GPS devices in the car. Going forward from a pure prototyping system to actually mapping large areas means the need for more mapping resources. As the currently used mapping vehicles don't scale because of their very expensive Applanix D-GPS system, other means have to be employed. Major routes could be mapped by prototype vehicles and stock cars could create local imprecise maps and align those to the major routes or known lane models.

5.3 Conclusion

The overall goal of this thesis is to provide a software system which is capable of generating grid- and feature-based maps for autonomous driving. It has been shown that the gridmap software is capable of accomplishing this: it achieves this goal by using an occupancy grid to fuse the input of feature detectors and to extract a list of features afterwards. The gridmap is also suitable for constructing local feature maps around the vehicle for autonomous localization. An extensive numerical analysis framework has been developed which allows to visualize and analyze the map content. The gridmap is running on actual prototype vehicles and provides a subsystem for autonomous driving functionality. It has therefore been tested and evaluated on real vehicle data and proven its viability.

List of Figures

1.1 Audi A7 prototype car 5
1.2 Feature map as used prior to this thesis 6

3.1 Computer hardware in the trunk of a prototype A7 17
3.2 Work flow of the lane line detector 20
3.3 UML specification of the gridmap 22
3.4 Endless driving mode for localization 25
3.5 The effect of 2D pruning on the tree structure 26
3.6 The fusion strategy connected cells 28

4.1 Ground truth features on aerial imagery 31
4.2 Ground truth features and detections on aerial imagery 33
4.3 Grid-based comparison for accuracy analysis 34
4.4 Zoom on a single feature in grid-based comparison 35
4.5 Overview of the entire map for feature-based accuracy analysis 37
4.6 Zoom on features compared to the ground truth for feature-based accuracy analysis 38
4.7 Cumulative plot of the runtime of the gridmap 42
4.8 Runtime for different gridmap extensions 43
4.9 Memory usage for different gridmap 43

5.1 Calibration issues with the topview cameras 46

List of Tables

4.1 Results of the accuracy analysis on the grid-based maps 35
4.2 Results of the accuracy analysis comparing the results before and after the gridmap 36
4.3 Results of the accuracy analysis on the feature-based maps 38
4.4 File size of the grid-based maps for different feature types 40
4.5 File size of the grid-based map for different configurations 41

Bibliography

[1] The Henry Ford Museum. *Henry Ford: the Innovator and Ford Motor Company*. 2013. URL: http://www.hfmgv.org/exhibits/hf/The_Innovator_and_Ford_Motor_Company.asp (visited on 05/22/2014).

[2] Morgan Stanley. "Autonomous Cars: Self-Driving the New Auto Industry Paradigm". In: *Morgan Stanley Research Blue Paper* (2013).

[3] Financial Times. *Tesla moves ahead of Google in race to build self-driving car*. 2013. URL: http://www.ft.com/cms/s/0/70d26288-1faf-11e3-8861-00144feab7de.html\#axzz2r0CrhkKq (visited on 05/22/2014).

[4] Telegraph. *Self-driving Mercedes S-class – beyond the Frankfurt show*. 2013. URL: http://www.telegraph.co.uk/motoring/car-manufacturers/mercedes-benz/10300094/Self-driving-Mercedes-S-class-beyond-the-Frankfurt-show.html (visited on 05/22/2014).

[5] Chris Urmson. *The latest chapter for the self-driving car: mastering city street driving*. 2014. URL: http://googleblog.blogspot.com/2014/04/the-latest-chapter-for-self-driving-car.html (visited on 05/22/2014).

[6] Gary Silberg et al. "Self-driving cars: The next revolution". In: *KPMG and Center for Automotive Research* (2012), pp. 10–15.

[7] National Highway Traffic Safety Administration. *U.S. Department of Transportation Releases Policy on Automated Vehicle Development*. 2013. URL: http://www.nhtsa.gov/About+NHTSA/Press+Releases/U.S.+Department+of+Transportation+Releases+Policy+on+Automated+Vehicle+Development (visited on 05/22/2014).

[8] Hugh Durrant-Whyte and Tim Bailey. "Simultaneous localization and mapping: Part 1". In: *Robotics & Automation Magazine* (2006), pp. 99–110.

[9] Sebastian Thrun et al. "Robotic mapping: A survey". In: *Exploring artificial intelligence in the new millennium* (2002), pp. 1–35.

[10] Sebastian Thrun and Arno Bücken. "Integrating grid-based and topological maps for mobile robot navigation". In: *Proceedings of the National Conference on Artificial Intelligence*. 1996, pp. 944–951.

[11] Alberto Elfes. "Using occupancy grids for mobile robot perception and navigation". In: *Computer* (1989), pp. 46–57.

[12] Martin Habbecke and Leif Kobbelt. "A surface-growing approach to multi-view stereo reconstruction". In: *Computer Vision and Pattern Recognition*. IEEE. 2007, pp. 1–8.

[13] Donald JR Meagher. *Octree encoding: A new technique for the representation, manipulation and display of arbitrary 3-d objects by computer*. Electrical and Systems Engineering Department Rensseiaer Polytechnic Institute Image Processing Laboratory, 1980.

[14] Rudolph Triebel, Patrick Pfaff, and Wolfram Burgard. "Multi-level surface maps for outdoor terrain mapping and loop closing". In: *Intelligent Robots and Systems*. IEEE. 2006, pp. 2276–2282.

[15] Armin Hornung et al. *OctoMap, An Efficient Probabilistic 3D Mapping Framework Based on Octrees*. 2014. URL: http://octomap.github.io/ (visited on 05/22/2014).

[16] Armin Hornung et al. "OctoMap: An Efficient Probabilistic 3D Mapping Framework Based on Octrees". In: *Autonomous Robots* (2013).

[17] Kai Wurm et al. “Hierarchies of octrees for efficient 3d mapping”. In: *Intelligent Robots and Systems*. IEEE. 2011, pp. 4249–4255.

[18] Sebastian Thrun et al. “Stanley: The robot that won the DARPA Grand Challenge”. In: *Journal of field Robotics* (2006), pp. 661–692.

[19] Carnegie Mellon University. *Carnegie Mellon’s Tartan Racing Wins $2M DARPA Urban Challenge*. 2007. URL: http://www.cmu.edu/homepage/practical/2007/fall/victory.shtml (visited on 05/22/2014).

[20] Chris Urmson et al. “Autonomous driving in urban environments: Boss and the urban challenge”. In: *Journal of Field Robotics* (2008), pp. 425–466.

[21] Ingo Schwartz. “PRIMUS: autonomous driving robot for military applications”. In: *AeroSense*. International Society for Optics and Photonics. 2000, pp. 313–323.

[22] Anthony Stentz et al. “A system for semi-autonomous tractor operations”. In: *Autonomous Robots* (2002), pp. 87–104.

[23] Jesse Levinson, Michael Montemerlo, and Sebastian Thrun. “Map-Based Precision Vehicle Localization in Urban Environments”. In: *Robotics: Science and Systems*. Citeseer. 2007.

[24] Radu Danescu, Florin Oniga, and Sergiu Nedevschi. “Modeling and Tracking the Driving Environment With a Particle-Based Occupancy Grid”. In: *IEEE Transactions on Intelligent Transportation Systems* (2011), pp. 1331–1342.

[25] Trung-Dung Vu, Olivier Aycard, and Nils Appenrodt. “Online Localization and Mapping with Moving Object Tracking in Dynamic Outdoor Environments”. In: *IEEE Intelligent Vehicles Symposium* (2007), pp. 190–195.

[26] Chieh-Chih Wang, Charles Thorpe, and Sebastian Thrun. “Online simultaneous localization and mapping with detection and tracking of moving objects: Theory and results from a ground vehicle in crowded urban areas”. In: *Robotics and Automation*. IEEE. 2003, pp. 842–849.

[27] Audi AG. *Piloted Parking Made Audi a Showstopper at 2013 CES*. 2013. URL: http://www.audisite.com/2712/piloted-parking-made-audi-a-showstopper-at-2013-ces/?doing_wp_cron=1397694031.9284610748291015625000 (visited on 05/22/2014).

[28] Rainer Kummerle et al. “Autonomous driving in a multi-level parking structure”. In: *Robotics and Automation*. IEEE. 2009, pp. 3395–3400.

[29] Sebastian Thrun and Michael Montemerlo. “The graph SLAM algorithm with applications to large-scale mapping of urban structures”. In: *The International Journal of Robotics Research* (2006), pp. 403–429.

[30] Oliver Pink. “Visual map matching and localization using a global feature map”. In: *Computer Vision and Pattern Recognition Workshops*. IEEE. 2008, pp. 1–7.

[31] Sebastian Thrun, Wolfram Burgard, and Dieter Fox. *Probabilistic robotics*. MIT press, 2005.

[32] MIT Technology Review. *Self-Driving Cars Get Smaller Brains*. 2014. URL: http://www.technologyreview.com/news/523351/ces-2014-audi-shows-off-a-compact-brain-for-self-driving-cars/ (visited on 05/22/2014).

[33] Applanix. *POS LV*. 2014. URL: http://www.applanix.com/products/land/pos-lv.html (visited on 05/22/2014).

[34] Audi Electronics Venture GmbH. *ADTF Software environment for applications and tools*. 2014. URL: http://www.audi-electronics-venture.de/aev/brand/en/services/development_tools/adtf.html (visited on 05/22/2014).

[35] Steven Dutch. *The Universal Transverse Mercator System*. 2014. URL: https://www.uwgb.edu/dutchs/FieldMethods/UTMSystem.htm (visited on 05/22/2014).

[36] Szymon Rusinkiewicz and Marc Levoy. “Efficient variants of the ICP algorithm”. In: *3-D Digital Imaging and Modeling*. IEEE. 2001, pp. 145–152.

[37] Erich Gamma et al. *Design patterns: elements of reusable object-oriented software*. Pearson Education, 1994.

[38] ESRI. “Shapefile technical description”. In: *An ESRI White Paper* (1998).

[39] openjump.org. *OpenJUMP*. 2014. URL: http://openjump.org (visited on 05/22/2014).