Sensor Fusion in Localization, Mapping and Tracking

Applications in Urban Autonomous Driving

Constantin Wellhausen

March 2024

Dissertation zur Erlangung des Grades eines
Doktors der Ingenieurwissenschaften
– Dr.-Ing. –

Universität Bremen – Fachbereich 3 Mathematik/Informatik

Title Figure: Representation of the environment of the autonomous vehicle using an occupancy grid map. The environment is discretized into cells, with each cell containing a Bernoulli distributed random variable that represents the occupancy of the cell. The vehicle is localized in its environment, shown as a blue vehicle in the center of the image.

Sensor Fusion in Localization, Mapping and Tracking

Applications in Urban Autonomous Driving

Constantin Wellhausen

Date of Submission
March 5, 2024

Date of Defense
June 6, 2024

Reviewers
Prof. Dr. Kerstin Schill, University of Bremen, Germany
Prof. Dr. Kevin Hipp, University of Applied Sciences of Zwickau, Germany

Abstract

Making autonomous driving possible requires extensive information about the surroundings as well as the state of the vehicle. While specific information can be obtained through singular sensors, a full estimation requires a multi sensory approach, including redundant sources of information to increase robustness. This thesis gives an overview of tasks that arise in sensor fusion in autonomous driving, and presents solutions at a high level of detail, including derivations and parameters where required to enable re-implementation. The thesis includes theoretical considerations of the approaches as well as practical evaluations. Evaluations are also included for approaches that did not prove to solve their tasks robustly. This follows the belief that both results further the state of the art by giving researchers ideas about suitable and unsuitable approaches, where otherwise the unsuitable approaches may be re-implemented multiple times with similar results. The thesis focuses on model-based methods, also referred to in the following as classical methods, with a special focus on probabilistic and evidential theories. Methods based on deep learning are explicitly not covered to maintain explainability and robustness which would otherwise strongly rely on the available training data. The main focus of the work lies in three main fields of autonomous driving: localization, which estimates the state of the ego-vehicle, mapping or obstacle detection, where drivable areas are identified, and object detection and tracking, which estimates the state of all surrounding traffic participants. All algorithms are designed with the requirements of autonomous driving in mind, with a focus on robustness, real-time capability and usability of the approaches in all potential scenarios that may arise in urban driving.

In localization the state of the vehicle is determined. While traditionally global positioning systems such as a Global Navigation Satellite System (GNSS) are often used for this task, they are prone to errors and may produce jumps in the position estimate which may cause unexpected and dangerous behavior. The focus of research in this thesis is the development of a localization system which produces a smooth state estimate without any jumps. For this two localization approaches are developed and executed in parallel. One localization is performed without global information to avoid jumps. This however only provides odometry, which drifts over time and does not give global positioning. To provide this information the second localization includes GNSS information, thus providing a global estimate which is free of global drift. Additionally the use of LiDAR odometry for improving the localization accuracy is evaluated.

For mapping the focus of this thesis is on providing a computationally efficient mapping system which is capable of being used in arbitrarily large areas with no predefined size. This is achieved by mapping only the direct environment of the vehicle, with older information in the map being discarded. This is motivated by the observation that the environment in autonomous driving is highly dynamic and must be mapped anew every time the vehicles sensors observe an area. The provided map gives subsequent algorithms information about areas where the vehicle can or cannot drive. For this an occupancy grid map is used, which discretizes the map into cells of a fixed size, with each cell estimating whether its corresponding space in the world is occupied. However the grid map is not created for the entire area which could potentially be visited, as this may be very large and potentially impossible to represent in the working memory. Instead the map is created only for a window around the vehicle, with the vehicle roughly in the center. A hierarchical map organization is used to allow efficient moving of the window as the vehicle moves through an area. For the hierarchical map different data structures are evaluated for their time and space complexity in order to find the most suitable implementation for the presented mapping approach.

Finally for tracking a late-fusion approach to the multi-sensor fusion task of estimating states of all other traffic participants is presented. Object detections are obtained from LiDAR, camera and Radar sensors, with an additional source of information being obtained from vehicle-to-everything communication which is also fused in the late fusion. The late fusion is developed for easy extendability and with arbitrary object detection algorithms in mind. For the first evaluation it relies on black box object detections provided by the sensors. In the second part of the research in object tracking multiple algorithms for object detection on LiDAR data are evaluated for the use in the object tracking framework to ease the reliance on black box implementations. A focus is set on detecting objects from motion, where three different approaches are evaluated for motion estimation in LiDAR data: LiDAR optical flow, evidential dynamic mapping and normal distribution transforms.

The thesis contains both theoretical contributions and practical implementation considerations for the presented approaches with a high degree of detail including all necessary derivations. All results are implemented and evaluated on an autonomous vehicle and real-world data. With the developed algorithms autonomous driving is realized for urban areas.

Kurzfassung

Um autonomes Fahren zu ermöglichen, sind umfangreiche Informationen über die Umgebung und den Zustand des Fahrzeugs erforderlich. Während bestimmte Informationen durch einzelne Sensoren gewonnen werden können, erfordert eine vollständige Schätzung einen multisensorischen Ansatz, der redundante Informationsquellen einschließt, um die Robustheit zu erhöhen. Diese Arbeit gibt einen Überblick über die Aufgaben, die beim autonomen Fahren in der Sensorfusion zu lösen sind, und stellt Lösungen mit einem hohen Detaillierungsgrad vor, einschließlich Ableitungen und Parametern, die für eine Implementierung erforderlich sind. Die Arbeit enthält sowohl theoretische Überlegungen zu den Ansätzen als auch praktische Bewertungen. Enthalten sind dabei auch Ansätze, die sich als nicht robust genug erwiesen haben, um im autonomen Fahren zum Einsatz zu kommen. Dies folgt der Überzeugung, dass beide Arten von Ergebnissen den Stand der Technik vorantreiben, indem sie den Forschern Hinweise auf geeignete und ungeeignete Ansätze geben, wo die ungeeigneten Ansätze ansonsten mehrfach mit ähnlichen Ergebnissen neu implementiert würden. Der Schwerpunkt der Arbeit liegt auf modellbasierten Methoden, die im Folgenden auch als klassische Methoden bezeichnet werden, mit einem besonderen Fokus auf probabilistischen und evidenzbasierten Theorien.

Methoden, die auf Deep Learning basieren, werden explizit nicht behandelt, um die Erklärbarkeit und Robustheit zu erhalten, die ansonsten stark von den verfügbaren Trainingsdaten abhängen würden. Der Schwerpunkt der Arbeit liegt in drei Hauptbereichen des autonomen Fahrens: Lokalisierung, die den Zustand des Ego-Fahrzeugs schätzt, Kartierung bzw. Hinderniserkennung, bei der befahrbare Bereiche identifiziert werden, und Objekt-erkennung und -verfolgung, die den Zustand aller umliegenden Verkehrsteilnehmer schätzt. Alle Algorithmen werden mit Blick auf die Anforderungen des autonomen Fahrens entwickelt, wobei der Schwerpunkt auf Robustheit, Echtzeitfähigkeit und Nutzbarkeit der Ansätze in allen möglichen Szenarien liegt, die im Stadtverkehr auftreten können.

In der Lokalisierung wird der Zustand des Fahrzeugs bestimmt. Während traditionell oft globale Positionierungssysteme wie ein globales Navigationssatellitensystem (GNSS) für diese Aufgabe verwendet werden, sind diese anfällig für Fehler und können Sprünge in der Positionsschätzung erzeugen, die zu unerwartetem und gefährlichem Verhalten führen können. Der Schwerpunkt der Forschung in dieser Arbeit liegt auf der Entwicklung eines Lokalisierungssystems, das eine glatte Zustandsschätzung ohne Sprünge erzeugt. Hierfür werden zwei Lokalisierungsansätze entwickelt und parallel ausgeführt. Eine Lokalisierung wird ohne globale Informationen durchgeführt, um Sprünge zu vermeiden. Diese liefert jedoch nur Odometrie, die über die Zeit driftet und keine globale Positionierung liefert. Um diese Informationen zu erhalten, wird die zweite Lokalisierung mit GNSS-Informationen durchgeführt und liefert so eine globale Schätzung, die frei von globaler Drift ist. Zusätzlich wird die Verwendung von LiDAR-Odometrie zur Verbesserung der Lokalisierungsgenauigkeit bewertet.

In der Kartierung liegt der Schwerpunkt dieser Arbeit auf der Bereitstellung eines effizienten Kartierungssystems, das in beliebig großen Gebieten ohne vordefinierte Größe eingesetzt werden kann. Dies wird erreicht, indem nur die unmittelbare Umgebung des Fahrzeugs kartiert wird, wobei ältere Informationen in der Karte verworfen werden. Dies wird durch die Beobachtung motiviert, dass die Umgebung im autonomen Fahren sehr dynamisch ist und jedes Mal neu kartiert werden muss, wenn die Sensoren des Fahrzeugs ein Gebiet beobachten. Die bereitgestellte Karte gibt den nachfolgenden Algorithmen Informationen über Bereiche, in denen das Fahrzeug fahren kann oder nicht. Dazu wird eine Grid-Map verwendet, die die Karte in Zellen einer festen Größe unterteilt, wobei jede Zelle schätzt, ob der ihr entsprechende Bereich in der Welt belegt ist. Die Grid-Map wird jedoch nicht für den gesamten Bereich erstellt, der potenziell besucht werden könnte, da dieser sehr groß sein kann und möglicherweise nicht im Arbeitsspeicher dargestellt werden kann. Stattdessen wird die Karte nur für ein Fenster um das Fahrzeug herum erstellt, wobei sich das Fahrzeug ungefähr in der Mitte befindet. Eine hierarchische Kartenorganisation wird verwendet, um ein effizientes Verschieben des Fensters zu ermöglichen, wenn sich das Fahrzeug durch ein Gebiet bewegt. Für die hierarchische Karte werden verschiedene Datenstrukturen hinsichtlich ihrer Rechenzeit- und Speicherkomplexität bewertet, um die am besten geeignete Implementierung für den vorgestellten Mapping-Ansatz zu finden.

Schließlich wird für das Tracking ein Late-Fusion-Ansatz für die Multi-Sensor-Fusion der Schätzung der Zustände aller anderen Verkehrsteilnehmer vorgestellt. Die Objekterkennung erfolgt durch LiDAR-, Kamera- und Radarsensoren, wobei eine zusätzliche Informationsquelle aus der Vehicle-to-Everything-Kommunikation gewonnen wird, die ebenfalls in der Late-Fusion zusammengeführt wird. Die Late-Fusion wurde für eine einfache Erweiterbarkeit und mit Blick auf beliebige Objekterkennungsalgorithmen entwickelt. Für eine erste Bewertung stützt sie sich auf die von den Sensoren gelieferten Black-Box-Objekterkennungen. Im zweiten Teil der Forschung zum Objekt-Tracking werden mehrere Algorithmen zur Objekterkennung auf LiDAR-Daten für die Verwendung im Rahmen des Objekt-Trackings bewertet, um die Abhängigkeit von Black-Box-Implementierungen zu verringern. Ein Schwerpunkt liegt auf der Erkennung von Objekten anhand ihrer Bewegung, wobei drei verschiedene Ansätze zur Bewegungsschätzung in LiDAR-Daten bewertet werden: LiDAR-Optical-Flow, Evidential-Dynamic-Mapping und Normal-Distribution-Transforms.

Die Arbeit enthält sowohl theoretische Beiträge als auch praktische Überlegungen zur Implementierung zu den vorgestellten Ansätzen mit einem hohen Detaillierungsgrad einschließlich aller notwendigen Ableitungen. Alle Ergebnisse werden auf einem autonomen Fahrzeug und realen Daten implementiert und evaluiert. Mit den entwickelten Algorithmen wird autonomes Fahren im urbanen Raum realisiert.

Acknowledgements

I am thankful to everyone at the Cognitive Neuroinformatics working group for the great working atmosphere and fruitful discussions, making my time at the university thoroughly enjoyable. In particular I would like to thank Joachim Clemens for his guidance in my research as well as the uncountable hours of discussions that led to many of the results of this thesis. I would also like to thank Matthias Rick and Andreas Folkers for the many patient hours spent together in the research vehicle.

I thank my supervisor Kerstin Schill for her support, the opportunity to conduct research under her supervision and for all the encouraging words that helped me in finishing this thesis. In addition, I would like to give special thanks to Kevin Hipp for agreeing to review this thesis. I would also like to thank Joachim Clemens for proof-reading the thesis.

This work was partially supported by the German Aerospace Center (DLR) with financial means of the German Federal Ministry for Economic Affairs and Climate Action (BMWK), projects OPA3L (grant No. 50 NA 1909) and MUTIG-VORAN (grant No. 50 NA 2202A).

Contents

List of Figures

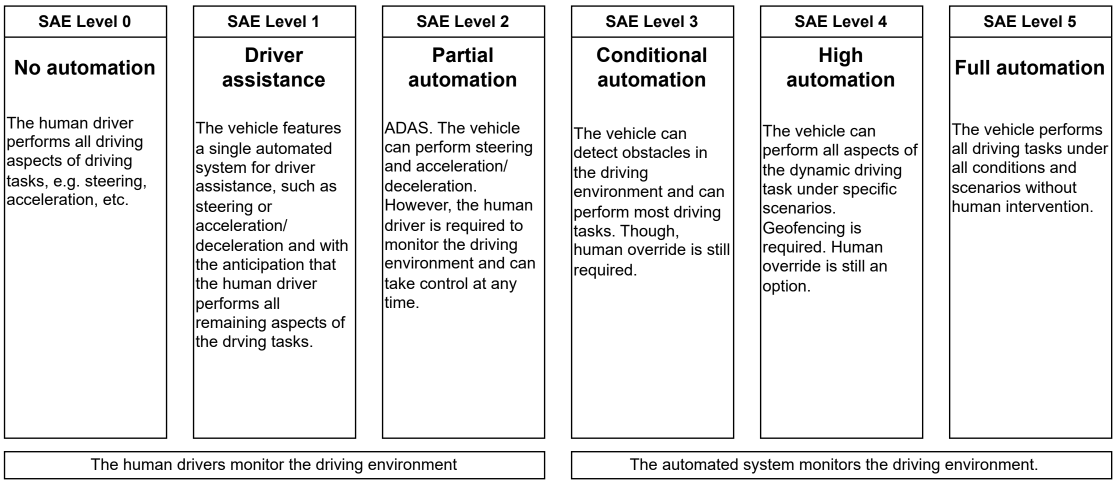
List of Tables

1 Introduction

Enabling an autonomous vehicle to navigate safely through complex urban areas requires information about its surroundings as well as the position of the vehicle and its current movement. Depending on the desired level of automation the requirements on this knowledge differs. Many current mass-produced vehicles integrate a number of advanced driver assistance systems (ADAS). While these systems rely on sensory surveillance of the scene, the required robustness and level of scene understanding is much lower compared to full self driving and it is often obtained by a single sensor. Infrequent failures of the systems are undesired but acceptable as the responsibility to drive safely lies with the driver in these systems.

This however changes when higher levels of autonomy are desired, as shown in Fig. 1.1. Here, the driver remains fully responsible for monitoring the environment until level 2 automation, after which the responsibilities are slowly shifted away from the driver towards the autonomous system until full autonomous driving is reached. In this work the focus lies on the development of systems that do not rely on driver intervention, with the ultimate goal of aiding in the development of a fully autonomous vehicle.

To reach this goal a much more thorough understanding of the surroundings as well as the state of the vehicle is required. Obtaining this information is however a difficult task. Sensors observe different parts of the surroundings as well as the vehicle state, however individually this only results in partial information. Instead many sensors built into the vehicle are combined to obtain it, each measuring different characteristics. Every sensor measures parts of the environment or parts of the state of the vehicle, however all sources of information contain inaccuracies, may even be contradictory and in itself do not result in a complete understanding of the current scene. Only by combining the data and taking into account all the different characteristics of the sensors and the information they produce is it possible to obtain this understanding. This process is called multi-sensor fusion, although in this work it will be referred to as sensor fusion to improve readability. For sensor fusion to obtain a good estimate it is imperative to have prior knowledge of the uncertainties of each measurement that contributes to the estimate. By taking into account how accurate a measurement is when combining it with other sources a significantly higher accuracy can be achieved compared to an estimation on a single source or a fusion that does not utilize uncertainty. It is useful in many different applications, however in autonomous driving there are three main tasks to be solved: localization, mapping and tracking. Localization is the task of estimating the state of the ego vehicle, which contains its pose and dynamics including the velocity, acceleration and



SAE Level 0	SAE Level 1	SAE Level 2	SAE Level 3	SAE Level 4	SAE Level 5
No automation	**Driver assistance**	**Partial automation**	**Conditional automation**	**High automation**	**Full automation**
The human driver performs all driving aspects of driving tasks, e.g. steering, acceleration, etc.	The vehicle features a single automated system for driver assistance, such as steering or acceleration/ deceleration and with the anticipation that the human driver performs all remaining aspects of the drving tasks.	ADAS. The vehicle can perform steering and acceleration/ deceleration. However, the human driver is required to monitor the driving environment and can take control at any time.	The vehicle can detect obstacles in the driving environment and can perform most driving tasks. Though, human override is still required.	The vehicle can perform all aspects of the dynamic driving task under specific scenarios. Geofencing is required. Human override is still an option.	The vehicle performs all driving tasks under all conditions and scenarios without human intervention.
The human drivers monitor the driving environment			The automated system monitors the driving environment.		

Figure 1.1: Levels of driving automation as defined by the Society of Automotive Engineers(SAE). (Figure partially adopted from [Yeong et al., 2021].)

turn rate of the vehicle. Mapping refers to the estimation of the vehicles' environment. Finally tracking describes the estimation of the pose, dynamics and size of dynamic objects in the vicinity of the vehicle. Solving these tasks lays the groundwork for all subsequent tasks that arise in autonomous driving that finally lead to a safe control of the autonomous vehicle. While the localization is a prerequisite for most autonomous driving functionality, the map containing obstacles and the state of dynamic objects are especially used for vehicle control and path planning in order to avoid collisions and plan efficient driving paths. The integration of the tasks in a full autonomous driving stack is shown in Fig. 1.2. These three tasks are introduced in detail in the following.

1.1 Sensor Fusion Tasks in Autonomous Driving

The first task is the localization of the vehicle, where the position and dynamic properties of the vehicle such as its velocity, acceleration and turn rate are estimated. The resulting vehicle state estimate is used by virtually every algorithm on the autonomous vehicle either directly or indirectly. This leads to a strong requirement for robustness and accuracy. Especially the algorithms responsible for vehicle control rely on an accurate and smooth estimation of the recent movement of the vehicle. Inaccuracies or even jumps may lead to over-corrections that feel unpleasant and unnatural for potential passengers and may even produce dangerous situations. The localization relies on many different sensors that measure its relative movement, such as inertial measurement units (IMU), or its absolute position in the world, such as global navigation satellite systems (GNSS). However additional sensors such as wheel speed sensors, and sensors measuring the steering wheel angle aid in the estimation as well.

The second task is the mapping of the vehicles surroundings, where the state of the vehicles' environment is estimated. The specific state that is estimated depends on what

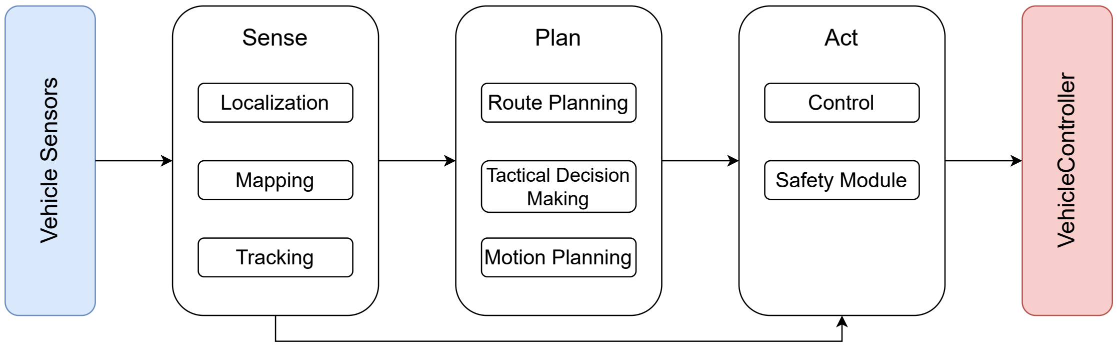


Figure 1.2: Full autonomous driving stack in which the sensor fusion is integrated. (Figure partially adopted from [Folkers et al., 2022].)

the map is used for, however the main objective of the mapping is to give information about where the vehicle can or cannot drive. Most mapping approaches limit the environment estimation to static obstacles such as trees, houses or traffic signs, although there exist approaches that extend this representation to also take into account moving obstacles such as other vehicles. For the mapping sensors are utilized that measure the surroundings of the vehicle. Light Detection and Ranging (LiDAR) are most commonly used, however Radar and Camera sensors may supply additional information that can be fused to obtain a more complete understanding of the surroundings including semantic information.

Finally the third task is the detection and tracking of other traffic participants. This task involves identifying other traffic participants in the surroundings of the vehicle and to estimate their classification, pose, size and dynamic properties such as velocity and acceleration. The state that is estimated is therefore very similar to the one estimated in the localization, however no intrinsic measurements are available. Instead the state must be estimated from external sensors such as LiDAR, Camera and Radar. Obtaining robust estimates of all surrounding traffic participants is therefore a challenging task and remains an active field of research. In recent years there has been a surge in research specifically in the area of deep neural networks. However these approaches lack explainability and rely on a large amount of training data to ensure operability under all scenarios. Situations not present in the training data may result in unexpected results. To avoid these shortcomings the focus of this work is on developing and exploring model-based approaches without the use of deep learning.

1.2 Thesis Contribution

Contributions are made in all of the fields of sensor fusion mentioned above. The work in these fields focuses on approaches for real-world applications. The algorithms are developed for a demonstrator with the goal to offer a shuttle service for urban areas [Folkers et al., 2022]. Thus the algorithms take into account the challenges that arise in these areas. Compared to highway scenarios where systems like the Drive Pilot by

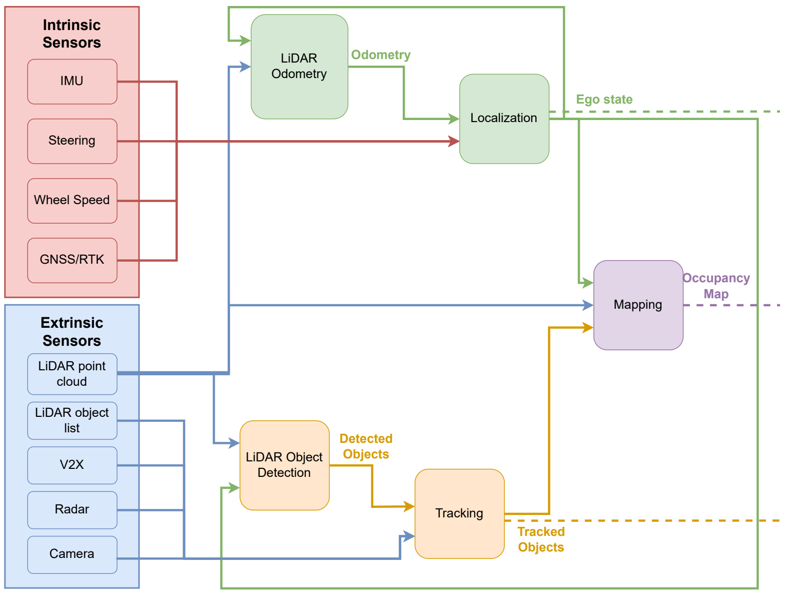


Figure 1.3: Architecture of the sensor fusion stack for autonomous driving. Intrinsic sensory input is shown in red, extrinsic input in blue. The modules corresponding to localization functionality are shown in green, mapping is shown in purple while object detection and tracking is shown in orange.

Mercedes Benz [Mercedes-Benz, 2023] and the Driver Assistance by BMW [BMW, 2023] have made significant progress in offering full self driving, urban areas still offer many unsolved areas of research. While highways are highly structured with strict rules to be followed, urban areas are often cluttered scenes, with less structure in the motion of other traffic participants. In addition urban areas contain many underpasses, bridges and overgrown areas, making sensors that rely on satellite reception (i.e. GNSS) unreliable in some situations. An overview of the entire sensor fusion stack is shown in Fig. 1.3. The contributions made in these fields are listed in the following.

1.2.1 Localization

For the localization a system is proposed specifically for the requirements that exist in autonomous driving. It provides a vehicle state estimate that is highly accurate over short periods of time while also being smooth without any jumps. This is done to accommodate subsequent algorithms that rely on a continuous and accurate estimate of the vehicles' recent movement. To obtain a smooth localization however, GNSS information can not be included in the estimate as this inherently produces jumps. Therefore the resulting localization has no external reference and only estimates relative movement or odometry. Thus it drifts over time and does not give global position information. There are however algorithms such as a routing algorithm that relies on a global position estimate to choose the best route to a desired destination. To provide this global information a second localization is performed in parallel which includes GNSS information in its estimate. This localization gives its state in a global reference system and can thus be used for routing between a passenger pickup and its destination or to take into account static information about the environment. As this localization takes into account GNSS data it will not drift over time, however it may produce jumps especially in areas with weak GNSS signal. By offering both types of localization each module may choose which one to use depending on whether a jump-free and smooth localization is beneficial or whether global positioning without drift is required. Both approaches are evaluated for the use on the demonstrator vehicle in multiple real-world scenarios. Additionally the use of LiDARs to aid in the reduction of drift over time is evaluated by utilizing LiDAR odometry. As the vehicle moves through an area it will measure large parts of the environment in multiple subsequent scans, enabling the use of scan matching approaches to align these measurements. This alignment can be used as additional information for the localization, and as each alignment takes into account previous information, the inherent drift caused by only using relative measurements is reduced.

1.2.2 Mapping

Creating a map of the vehicles' surroundings in autonomous driving poses some task-specific difficulties as well. Areas that are mapped are often assumed to have a fixed size, limited to the area for which the mapping algorithm was developed. Especially in autonomous driving this is however not always possible. The area that should be mapped depends on the desired route of the passengers and this can potentially cover a very large area between the origin and the destination. Depending on the distance it is often infeasible to map the entire area. However due to the highly dynamic nature of the mapped areas in autonomous driving it is not necessary to map the entire area and keep all information in memory. As an example, parked vehicles may block parts of a street when first visiting an area, and then be gone at a later time. Therefore to make decisions based on the environment it is important to always have updated information available, meaning that areas that were previously mapped need to be mapped again upon revisiting. Instead of keeping the entire history in the map this thesis thus uses a moving-window approach where the mapped area is reduced to the direct vicinity of the vehicle. For this a special hierarchical map structure is used where the map is split into multiple submaps. The main contribution of this work is the comparison between different data structures used in the hierarchical map. Autonomous driving requires fast, real-time capable algorithms and any processing time saved in each algorithm can be utilized for more complex estimations in others. Thus an evaluation is performed to compare processing time and memory demands to find the optimal implementation for a hierarchical moving-window implementation of the mapping algorithm. This evaluation is again performed on real-world data to optimize for the use on the demonstrator vehicle.

1.2.3 Object Detection and Tracking

Detecting other traffic participants in the vehicles vicinity and estimating their state, also referred to as object tracking, are among the most crucial and challenging tasks in autonomous driving. Due to their difficulty a number of approaches are evaluated in this thesis. The thesis contains both positive and negative results of these evaluations, to give an overview of approaches and their strengths and weaknesses in different scenarios. In the current research climate negative results are however often not publishable. This lead to the conscious decision to write this thesis as a monograph to share both positive and negative results in the search for an object detection and tracking approach.

Contributions are made in two parts. Firstly a late fusion framework is given for fusing information from several commonly used sensors. The term late fusion refers to the fact that information is first processed for each sensor individually and fused at the latest possible stage, as opposed to early or mid fusion, where information is combined at earlier stages. A late fusion is however independent of the specific object detection algorithm, making it easier to extend it to multiple sensors and evaluate the performance of different object detection algorithms, which makes it the ideal approach for this thesis. The main sensors used in this work are multiple LiDARs built into the sides, front and back of the vehicle, however the late fusion allows fusion of objects detected by a camera and Radar as well. Additionally it enables the fusion of information communicated by other vehicles over vehicle-to-everything (V2X) communication. By combining all available sensors as well as utilizing communicated information obtained by other vehicles a robust tracking system is proposed, that is still able to operate when single sensors malfunction.

Secondly multiple object detection algorithms on LiDAR data are implemented and evaluated for the use on the available data of the demonstrator vehicle. A focus is put on algorithms that do not rely on knowledge about the specific objects classes that may occur but rather on a general approach to determine moving objects. The difficulty in this thesis in particular lies in the available sensor data, as the vehicle is equipped with LiDARs with a very limited vertical field of view. Thus only 2D object detection algorithms are evaluated. The applicability of these algorithms to certain scenarios as well as measures to improve their performance by additional sensors or additional computational power are discussed.

1.3 Publications Overview

While the thesis is written as a monography parts are based on previously published work, which is cited in the corresponding chapters and extended and textually adapted for better readability. An overview over the autonomous driving stack was published in [Folkers et al., 2022]. The results in the field of jump-free localization were published in [Clemens et al., 2020]. Work regarding a moving-window grid mapping approach was published in [Wellhausen et al., 2021]. In addition in [Höffmann et al., 2022] a global localization system is utilized which is based on the localization system presented in this thesis. While no authorship was claimed due to the focus of the paper being on the presented autonomous lawnmower, the described global localization system was partially developed as a part of this thesis.

1.4 Thesis Outline

The remainder of the thesis is structured as follows. In Chap. 2 different methods commonly used in sensor fusion are introduced. This especially includes the Bayes filter and the Kalman filter as a specific implementation. Vehicle state representations are discussed and the $\boxplus$-method is introduced for handling manifolds as part of the state. In addition grid mapping is introduced. In Chap. 3 an overview over the research vehicle as well as the used sensors is given.

Chap. 4 begins the description of contributions made in this thesis and covers the implementation of the two localization approaches used in this work. In addition the use of a LiDAR to decrease localization drift when estimating odometry or when no GNSS is available is evaluated. In Chap. 5 the moving-window mapping implementation is presented. A theoretical analysis is given for the time and space complexity when different data structures are used as storage. This is then evaluated on real-world data and suggestions are given for suitable data structures. Chap. 6 first introduces the late fusion approach for fusing information from different sources. Second it introduces multiple object detection approaches on LiDAR data. Finally Chap. 7 concludes the thesis by summarizing the work and giving an outlook on possible extensions and future work.

2 Methods in Sensor Fusion

While many ways exist to fuse sensor data, some standard approaches have been established that find use in many fields. In the following an overview over these methods is given to serve as a basis for the developed methods in this work. For an overview over methods in sensor fusion with a special focus on handling uncertain information the reader is also referred to [Clemens, 2018]. The following overview takes inspiration from this work with an additional focus on methods used in autonomous driving, extending it where necessary. First an overview is given over Bayes filters and the Kalman filter as a specialization. Second suitable representations for vehicle states are presented. Third grid mapping is introduced. Finally the handling of timestamped data in sensor fusion algorithms is discussed.

2.1 Bayes Filter

A Bayes filter is a probabilistic technique for state estimation from uncertain measurements and control processes [Russell and Norvig, 2010]. The goal is to estimate a state $x_t \in \mathcal{S}$ given the measurements $z_{0:t} = z_0, ..., z_t$, with $z_i \in \mathcal{Z}$, $i = 0, ..., t$ and the control sequence $u_{1:t} = u_1, ..., u_t$, with $u_i \in \mathcal{U}, i = 1, ..., t$. While $\mathcal{S}$ is the state space, $\mathcal{Z}$ and $\mathcal{U}$ are the measurement and control space respectively, and t denotes the current point in time. Note that in autonomous driving, the control is often a measured process as well, where inertial measurements are used to more precisely predict the movement of the vehicle. The state estimation is expressed as a probabilistic problem to model sensor and process noise. Obtaining a state estimate at time t given all measurements and controls can be expressed as a conditional probability $p(x_t|z_{0:t}, u_{1:t})$. Two assumptions are required to enable efficient computation: (i) the measurement z_t depends only on the current state x_t and (ii) the current state depends only on the previous state x_{t-1} and the current control u_t, which is also referred to as the Markov assumption [Thrun et al., 2005, Sect. 2.4.4]. Using these two assumptions, the probability for a measurement, also referred to as measurement model, can be expressed as $p(z_t|x_t)$, while the probability for a state transition with the current control is given by $p(x_t|u_t, x_{t-1})$, also known as the motion model.

To now derive a recursive method for state estimation, first the Bayes rule is utilized:

$$P(X|Y) \propto P(Y|X)P(X) \tag{2.1}$$

Using this rule as well as (i), we may formulate the state estimation as follows:

$$\begin{aligned} p(x_t|z_{0:t}, u_{1:t}) &\propto p(z_t|x_t, z_{0:t-1}, u_{1:t})p(x_t|z_{0:t-1}, u_{1:t}) \\ &= p(z_t|x_t)p(x_t|z_{0:t-1}, u_{1:t}) \end{aligned} \tag{2.2}$$

Using (ii) and the law of total probability, we obtain the prior $p(x_t|z_{0:t-1}, u_{1:t})$:

$$p(x_t|z_{0:t-1}, u_{1:t}) = \int_S p(x_t|u_t, x_{t-1})p(x_{t-1}|z_{0:t-1}, u_{1:t-1})dx_{t-1} \tag{2.3}$$

Since the term $p(x_{t-1}|z_{0:t-1}, u_{1:t-1})$ is the posterior from the previous step, this results in a recursive algorithm where the full measurement and control sequence is not required. Together, (2.2) and (2.3) form the recursive Bayes filter. Equation (2.3) is often referred to as the prediction step, which uses the current control input and the previous state to propagate the state forward in time. On the other hand, equation (2.2) can be seen as the correction step, which uses the current measurement to correct errors accumulated in the prediction step. While the Bayes filter gives a framework for probabilistic state estimation, there are multiple implementations of it, with one of the most frequently used one being the Kalman filter. This approach and two of its specializations are presented in the following.

2.1.1 Kalman Filter (KF)

The Kalman filter [Kalman, 1960; Thrun et al., 2005, Sect. 3.2] is an implementation of the Bayes filter. It consists of two steps, a state prediction step, which in autonomous driving relates to the prediction of the vehicles movement, and a measurement (or correction) step, where the state is corrected based on current measurements. In autonomous driving the prediction is usually performed based on the type of vehicle that is estimated, taking into account its physical properties and limitations. Whenever possible this is often modeled as a measured process instead of relying on a static control input, where inertial measurements precisely measure the current movement of the vehicle. Corrections on the other hand are solely performed using sensor data. The Kalman filter predicts what a sensor would measure if the current state was correct, and when this does not fit the received sensor measurement the state is corrected to reduce this discrepancy.

The Kalman filter makes multiple assumptions to enable efficient calculation. First, the filter assumes linear motion, and the noise has to be Gaussian distributed with zero mean:

$$x_t = A_t x_{t-1} + B_t u_t + \epsilon_t, \quad \epsilon_t \sim \mathcal{N}(0, R_t) \tag{2.4}$$

Here, the Matrix A_t models the influence of the old state on the new state, while matrix B_t does the same for the control. Additionally, the Gaussian process noise ϵ_t with covariance R_t is added to the state estimate.

Second, the filter assumes a linear measurement model, again with Gaussian noise and zero mean:

$$z_t = C_t x_t + \delta_t, \quad \delta_t \sim \mathcal{N}(0, Q_t) \tag{2.5}$$

Here, the Matrix C_t transforms the state into the measurement space, while δ_t is the Gaussian process noise and the Matrix Q_t is its measurement noise.

The Kalman filter estimates the Normal Distribution $x_t | z_{0:t}, u_{1:t} \sim \mathcal{N}(\mu_t, \Sigma_t)$. More specifically, the mean μ_t and the covariance Σ_t need to be estimated. For this the prediction and correction step are executed periodically. In the prediction, the previous mean μ_{t-1} and covariance Σ_{t-1} are predicted to the current time t using the control u_t:

$$\bar{\mu}_t = A_t \mu_{t-1} + B_t u_t, \tag{2.6}$$

$$\bar{\Sigma}_t = A_t \Sigma_{t-1} A_t^T + R_t \tag{2.7}$$

Using these predicted values, the Kalman Gain is calculated, which weighs the influence of new measurements against the existing estimate:

$$K_t = \Sigma_{t-1} C_t^T (C_t \bar{\Sigma}_t C_t^T + Q_t)^{-1} \tag{2.8}$$

Using this factor, the prediction is corrected in the correction step by

$$\mu_t = \bar{\mu}_t + K_t (z_t - C_t \bar{\mu}_t), \tag{2.9}$$

$$\Sigma_t = (I - K_t C_t) \bar{\Sigma}_t, \tag{2.10}$$

where I is an identity Matrix.

While the result is always the optimal solution as long as the assumptions 2.4 and 2.5 of linear models with Gaussian noise hold, there are only very few real-world problems where a linear motion and measurement model can be assumed. To overcome this restriction, several extensions to the Kalman filter have been proposed, most notably the extended Kalman filter and the unscented Kalman filter. Both of these extensions find application in this thesis and are introduced in the following.

2.1.2 Extended Kalman Filter (EKF)

The extended Kalman filter [Jazwinski, 2007; Thrun et al., 2005, Sect. 3.3] removes the requirement for a linear motion and measurement model by linearizing them using first order Taylor series expansion. While this linearization requires both models to be reasonably linear locally around the linearization point, this assumption is significantly less restrictive and holds for many real-word applications. To allow non-linear models, a number of changes are required. First, the motion model 2.4 becomes

$$x_t = g(u_t, x_{t-1}) + \epsilon_t, \quad \epsilon_t \sim \mathcal{N}(0, R_t), \tag{2.11}$$

where the function $g : \mathcal{U} \times \mathcal{S} \to \mathcal{S}$ is the non-linear state transition function using the previous state and the control input. Similarly, 2.5 becomes

$$z_t = h(x_t) + \delta_t, \quad \delta_t \sim \mathcal{N}(0, Q_T), \tag{2.12}$$

with $h : \mathcal{S} \to \mathcal{Z}$ being the non-linear measurement function mapping the current state to the measurement space.

Using these two non-linear functions, the extended Kalman filter updates the state as follows:

$$\bar{\mu}_t = g(u_t, \mu_{t-1}) \tag{2.13}$$

$$\bar{\Sigma}_t = G_t \Sigma_{t-1} G_t^T + R_t \tag{2.14}$$

Here, the linearization is used in the Jacobian $G_t = \frac{\partial g}{\partial \mu_{t-1}}$ of the state transition function. It is calculated using first order Taylor series expansion from the linearization point μ_{t-1}. Next, the Kalman gain is calculated using

$$K_t = \bar{\Sigma}_t H_t^T (H_t \bar{\Sigma}_t H_t^T + Q_t)^{-1}. \tag{2.15}$$

Here, the Jacobian $H_t = \frac{\partial h}{\partial \bar{\mu}_t}$ is again calculated using first order Taylor series expansion.

Using the Kalman gain, the prediction is corrected with

$$\mu_t = \bar{\mu}_t + K_t(z_t - h(\bar{\mu}_t)), \tag{2.16}$$

$$\Sigma_t = (I - K_t H_t)\bar{\Sigma}_t. \tag{2.17}$$

As can be seen the extended Kalman filter is very similar to the standard Kalman filter, with the major change being the substitution of the linear mapping between state space and measurement space or state space and control space with non-linear mappings. While in theory the extended Kalman filter is simple to implement, the difficulty often lies in obtaining the Jacobians, which may be difficult to obtain depending on the corresponding function. Additionally while the assumption of locally linear models holds well enough for many applications, for very non-linear models this approximation fails. In these cases, an unscented Kalman filter may be used instead.

2.1.3 Unscented Kalman Filter (UKF)

In the unscented Kalman filter [Julier and Uhlmann, 1997; Thrun et al., 2005, Sect. 3.4], a deterministic sampling method is used for linearization instead of calculating the Jacobian by Taylor series expansion. Samples are drawn around the linearization point according to the current covariance, and propagated through the measurement and motion model to estimate the resulting distribution.

For prediction first sigma points are generated around the previous estimate μ_{t-1} by

$$\mathcal{X}_{t-1} = \begin{pmatrix} \mu_{t-1} & \mu_{t-1} + \gamma\sqrt{\Sigma_{t-1}} & \mu_{t-1} - \gamma\sqrt{\Sigma_{t-1}} \end{pmatrix}, \tag{2.18}$$

with $\gamma = a^2(n+k) - n$. While n is the dimensionality of the state space, a and k are scaling parameters which determine the spread of the sigma points from the previous mean. This results in $2n+1$ sigma points, with one being the mean μ_{t-1} and one being generated for every dimension of the state space in both directions around the mean. Each sigma point is propagated through the control function g by

$$\bar{\mathcal{X}}_t^* = g(u_t, \mathcal{X}_{t-1}). \tag{2.19}$$

From these sigma points the predicted mean and covariance is calculated by

$$\bar{\mu}_t = \sum_{i=0}^{2n} w_m^{[i]} \bar{\mathcal{X}}_t^{*[i]}, \tag{2.20}$$

$$\bar{\Sigma}_t = \sum_{i=0}^{2n} w_c^{[i]} \left(\bar{\mathcal{X}}_t^{*[i]} - \bar{\mu}_t\right) \left(\bar{\mathcal{X}}_t^{*[i]} - \bar{\mu}_t\right)^T + R_t. \tag{2.21}$$

Given the predicted mean and covariance the correction step is prepared by sampling around these predicted values. This is done by

$$\bar{\mathcal{X}}_t = \begin{pmatrix} \bar{\mu}_t & \bar{\mu}_t + \gamma\sqrt{\bar{\Sigma}_t} & \bar{\mu}_t - \gamma\sqrt{\bar{\Sigma}_t} \end{pmatrix}. \tag{2.22}$$

From these sigma points an estimated measurement is generated by propagating the sigma points through the measurement function and calculating the mean $\hat{z}_t$ and its uncertainty S_t by

$$\bar{\mathcal{Z}}_t = h(\bar{\mathcal{X}}_t), \tag{2.23}$$

$$\hat{z}_t = \sum_{i=0}^{2n} w_m^{[i]} \bar{\mathcal{Z}}_t^{[i]}, \tag{2.24}$$

$$S_t = \sum_{i=0}^{2n} w_c^{[i]} \left(\bar{\mathcal{Z}}_t^{[i]} - \bar{\mu}_t\right) \left(\bar{\mathcal{Z}}_t^{[i]} - \bar{\mu}_t\right)^T + Q_t. \tag{2.25}$$

The Kalman Gain K_t is then calculated by

$$\bar{\Sigma}_t^{x,z} = \sum_{i=0}^{2n} w_c^{[i]} \left(\bar{\mathcal{X}}_t^{[i]} - \bar{\mu}_t\right) \left(\bar{\mathcal{Z}}_t^{[i]} - \bar{\mu}_t\right)^T, \tag{2.26}$$

$$K_t = \Sigma_t^{\bar{x},z} S_t^{-1}. \tag{2.27}$$

With the Kalman Gain the corrected new mean μ_t and covariance Σ_t are calculated by

$$\mu_t = \bar{\mu}_t + K_t(z_t - \hat{z}_t), \tag{2.28}$$

$$\Sigma_t = \bar{\Sigma}_t - K_t S_t K_t^T. \tag{2.29}$$

While the UKF generally outperforms the EKF where parts of the models are highly non-linear, this improvement comes at a significant increase in computational complexity. With every sample being propagated through the measurement and motion model, computation time becomes a limiting factor especially where the state has a high dimensionality.

2.2 Vehicle State Representation

One of the most prominent tasks in autonomous driving is the estimation of a vehicle state. This may be the state of the ego vehicle or that of other traffic participants. In both cases, very similar states need to be estimated, with minor differences that will be discussed in Chap. 4 and Chap. 6 for the specific scenarios. A minimal vehicle state consists of the pose, which contains the position and orientation of the object as well as its velocity. With this information, a simple trajectory prediction is possible. While the position and velocity are estimated in euclidean space, and are therefore an element of $\mathbb{R}^3$, for the rotation a suitable representation requires some additional considerations. Rotations in 2D or 3D are part of the special orthogonal group $SO(2)$ and $SO(3)$ respectively, which are manifolds with a complex topology unlike a vector space. Since the Kalman filter is designed for states in a euclidean space, when estimating rotations there are some measures that need to be taken in order to ensure correct results. In the following, different representations of rotations will be considered. Subsequently, a method for encapsulating rotations for filtering using the so-called $\boxplus$-method is introduced.

2.2.1 Rotations in 2D

While the vehicle moves in three dimensional space, in some scenarios it can be beneficial to reduce the estimation of object states to two dimensions, causing the orientation to be part of the special orthogonal group $SO(2)$ instead. The simplest representation for a 2D rotation is through an angle $\psi \in \mathbb{R}, -\pi \leq \psi < \pi$. While this representation gives a clear understanding of the corresponding rotation in 2D space, there are some difficulties that need to be considered when using this representation. Firstly, there is no inherent way to apply this rotation to a vector. Secondly there are discontinuities between $-\pi$ and π where a small change in rotation results in a large difference in the representation. Lastly concatenating rotations requires normalization to keep the resulting angle between $-\pi$ and π.

Instead an over-parameterized representation can be chosen. The most commonly used representation is the 2D rotation matrix, defined as

$$R = \begin{bmatrix} \cos\psi & -\sin\psi \\ \sin\psi & \cos\psi \end{bmatrix}. \tag{2.30}$$

Using this representation applying a rotation $R^{A\to B}$ to a vector $p^A \in \mathbb{R}^2$ can be done by matrix-vector multiplication as

$$p^B = R^{A \to B} \cdot p^A. \tag{2.31}$$

A rotation matrix can be inverted by transposing the matrix

$$R^{(B \to A)} = \left(R^{(A \to B)}\right)^{-1} = \left(R^{(A \to B)}\right)^T, \tag{2.32}$$

with $R^{(A \to B)} R^{(B \to A)} = I$. Finally, rotations may be concatenated by matrix multiplication

$$R^{(A \to C)} = R^{(B \to C)} R^{(A \to B)}. \tag{2.33}$$

Using a rotation matrix for state representation the optimization or estimation algorithm would need to be aware of the structure of the matrix introduced in (2.30), which complicates the estimation problem. Additionally the dimensionality of the state would grow. Therefore for 2D state representation in this thesis angles are used directly, while rotation matrices are used in order to rotate vectors. Handling the resulting discontinuities in state estimation is explained in Sect. 2.2.3.

2.2.2 Rotations in 3D

Similar to rotations in 2D multiple possible representations exist for rotations in three dimensional space, each with different drawbacks and benefits. A major difference between representations is the amount of parameters used to represent a rotation in 3D. A minimal representation has three parameters, one for each degree of freedom in $SO(3)$. This however always comes at the cost of the representation containing singularities [Stuelpnagel, 1964], where multiple values for a parameter on one axis lead to similar resulting rotations. Thus, changes in the parameter space will not necessarily result in a proportional change in the state space. In the most extreme case a gimbal lock may occur [Lepetit et al., 2005, Sect. 2.2], where the three parameters only allow rotations around two axes. To avoid this over-parameterized representations can be used that use more than three parameters in order to represent rotations in $SO(3)$ without containing singularities. An overview over commonly used representations and a discussion on their usability in a Kalman filter is given in the following.

One of the most prominent and intuitive representations are Euler angles. Euler angles represent the rotation using three angles that determine the rotation around the x-,y- and z-axis, also called roll, pitch and yaw as a vector $[\phi \; \theta \; \psi]^T \in \mathbb{R}^3$. Using this representation, it is possible to directly understand from the parameters what kind of rotation is performed. This makes it comparably easy to use, however, the representation comes with a number of downsides. First, there exist singularities at the poles, where a large change in the parameter space only has a minor influence on the state space. Second, using Euler angles for optimization or state estimation using Kalman filters comes with the additional difficulty of discontinuities around $\pm\pi$. Here, a jump in the representation exists, which only leads to minimal changes in the state space. To use Euler angles for these applications, special handling of these jumps is required.

An alternative representation is the so-called scaled-axis representation. Any rotation in $SO(3)$ can be expressed as a vector $v \in \mathbb{R}^3, \|v\| = 1$ which defines the axis around which the rotation is performed and an angle $\alpha \in \mathbb{R}$ which defines the angle of rotation. In the scaled-axis representation, this angle is however encoded directly in the vector, by scaling the vector v using

$$u = \alpha v. \tag{2.34}$$

This representation is minimal in the amount of parameters used, however it is again not singularity-free and to use it for multiple subsequent rotations is difficult as there is no simple way to calculate the resulting combined rotation.

Instead, a rotation matrix may be used to define a rotation. To represent a three-dimensional rotation from frame A to frame B, a matrix $R^{(A \to B)} \in \mathbb{R}^{3 \times 3}$ is required. This matrix has to be an orthogonal matrix with determinant 1. Using this representation, a number of operations can be easily defined. First, to rotate a vector $p^A = [x\ y\ z]^T$ expressed in frame A to frame B, matrix-vector multiplication is performed by

$$p^B = R^{(A \to B)} p^A. \tag{2.35}$$

Second, a rotation can be inverted by transposing the matrix

$$R^{(B \to A)} = \left(R^{(A \to B)}\right)^{-1} = \left(R^{(A \to B)}\right)^T, \tag{2.36}$$

with $R^{(A \to B)} R^{(B \to A)} = I$. Third, rotations may be concatenated by matrix multiplication

$$R^{(A \to C)} = R^{(B \to C)} R^{(A \to B)}. \tag{2.37}$$

The rotation matrix can be extended to represent full transformations, where a translation $t^{(A \to B)} \in \mathbb{R}^3$ from frame A to B is additionally applied. For this, a transformation matrix is constructed by

$$T^{(A \to B)} = \begin{bmatrix} R^{(A \to B)} & t^{(A \to B)} \\ 0_{1 \times 3} & 1 \end{bmatrix} \tag{2.38}$$

Using this, a transformation is again applied using matrix-vector multiplication, however the vector p^A is embedded using homogeneous coordinates. As such the transformation is given by

$$p^B = T^{(A \to B)} \begin{bmatrix} p^A \\ 1 \end{bmatrix}, \tag{2.39}$$

where p^B is also in homogeneous coordinates. Using rotation matrices, no discontinuities or singularities exist, however the rotation is now significantly over-parameterized. While a rotation matrix uses nine parameters, there exist representations that do not contain singularities with only four parameters. Additionally, requiring the matrix to be orthogonal with determinant 1 is difficult to model in optimization tasks.

One representation with only four parameters is a quaternion. It is based on the idea, that every rotation can be described by an axis $v = [v_x \; v_y \; v_z]^T \in \mathbb{R}^3$ similar to the scaled axis representation. However in this case the axis vector is normalized to $\|v\| = 1$ and an angle parameter $\alpha \in \mathbb{R}$ is added. The rotation quaternion [Titterton and Weston, 2004, Sect. 3.6.4] is defined as

$$q = \cos\frac{\alpha}{2} + (v_x i + v_y j + v_z k)\sin\frac{\alpha}{2}, \tag{2.40}$$

where i, j, k are complex units for which holds $i^2 = j^2 = k^2 = ijk = -1$. To describe a valid rotation in $SO(3)$, the quaternion has to be a unit quaternion. As such, the set of valid rotation quaternions $\hat{\mathbb{H}}$ is defined as

$$\hat{\mathbb{H}} = \{q \in \mathbb{H}, \|q\| = 1\}, \tag{2.41}$$

where $\mathbb{H}$ is the set of all quaternions. To rotate a vector $p^A = [x \; y \; z]^T$ expressed in frame A to frame B using a quaternion $q^{(A\to B)}$, a pure quaternion $\bar{p}^A$ is constructed from p^A by setting the vector part v to p^A and the real part α to zero. The rotation is then performed by

$$\bar{p}^B = q^{(A\to B)}\bar{p}^A {q^{(A\to B)}}^{-1}, \tag{2.42}$$

where $\bar{p}^B$ again is a pure quaternion where the vector part is the rotated vector p^B. For a quaternion $q^{(A\to B)}$, the inverse $q^{(B\to A)}$ can be calculated using

$$q^{(B\to A)} = {q^{(A\to B)}}^{-1} = {q^{(A\to B)}}^{*}, \tag{2.43}$$

where ${q^{(A\to B)}}^{*}$ is the conjugate of $q^{(A\to B)}$. To obtain the rotation defined by combining two quaternions $q^{(A\to B)}$ and $q^{(B\to C)}$, the concatenation is calculated as

$$q^{(A\to C)} = q^{(B\to C)} q^{(A\to B)}. \tag{2.44}$$

While the quaternion is still over-parameterized with four parameters, it uses significantly fewer than the rotation matrix without having singularities. Just like rotation matrices, quaternions require special handling to ensure that they remain valid rotation quaternions with $\|q\| = 1$ when used in optimization tasks.

2.2.3 Encapsulating Rotations using the $\boxplus$-Method

To use rotations in $SO(2)$ or $SO(3)$ in a Kalman filter difficulties arise due to discontinuities and additional difficulties are encountered in 3D due to singularities. In order to handle these manifolds in the state, the $\boxplus$-method is used to encapsulate the manifold state space [Hertzberg et al., 2013]. In this method the state representation is separated from the representation of updates of the state. A key assumption behind this method is that updates to the state are performed in short intervals, which leads to the update being small. Thus discontinuities and singularities do not occur in the representation of the rotation describing the update, regardless of which representation is used, as these occur only for large updates. Updates to the state using the $\boxplus$-method depend on the specific representations used, which is shown in detail in the following first for rotations in 3D and then for 2D rotations.

While many representations exist for the $SO(3)$ state space, there is no single ideal representation to choose. When using a minimal representation like Euler Angles, singularities and discontinuities exist. These can be avoided by using quaternions or rotation matrices, however then the state space is over-parameterized. Algorithms that calculate updates to the rotation are not aware of the constraints that result from this over-parameterization (e.g. $\|q\| = 1$ for quaternions). As such, the updates may result in invalid parameters, and a normalization to restore the constraints will lead to sub-optimal updates.

To avoid these issues, the $\boxplus$-method is used to encapsulate the manifold state space [Hertzberg et al., 2013]. The rotation is globally represented using an over-parameterized representation such as quaternions, while updates are applied using a minimal representation. Updates are done in small intervals and are therefore numerically small. The manifold space appears like a vector space locally, which enables algorithms such as the Kalman filter, where a vector space is expected, to calculate the updates on this local vector space. With these updates being small, they are far away from problematic singularities and discontinuities. The global representation in the manifold space on the other hand accumulates these updates and may contain arbitrary rotations, as the over-parameterization ensures that no singularities occur. Two operators are defined to map from manifold state $\mathcal{S}$ to the vector space $\mathbb{R}^n$ and vice versa, where n determines the degrees of freedom in $\mathcal{S}$. These are defined as

$$\boxplus : \mathcal{S} \times \mathbb{R}^n \to \mathcal{S}, \tag{2.45}$$

$$\boxminus : \mathcal{S} \times \mathcal{S} \to \mathbb{R}^n. \tag{2.46}$$

The $\boxplus$-operator updates the state $\mathcal{S}$ using an element from the vector space. Intuitively, this operation applies an update to the state, much like an addition. The $\boxminus$-operator calculates a difference between two states on the manifold space, which is expressed as an element of the vector space. This can be seen as a subtraction. There are a number of constraints these operators have to satisfy, with $x \in \mathcal{S}$ and given the deviations are sufficiently small:

$$x \boxplus 0 = x \tag{2.47}$$

$$x \boxplus (y \boxminus x) = y, \forall y \in \mathcal{S}, \tag{2.48}$$

$$(x \boxplus \delta) \boxminus x = \delta, \delta \in \mathbb{R}^n, \tag{2.49}$$

$$\|(x \boxplus \delta_1) \boxminus (x \boxplus \delta_2)\| \leq \|\delta_1 - \delta_2\|, \delta_1, \delta_2 \in \mathbb{R}^n. \tag{2.50}$$

The most common state space in this work is $SO(3)$, which occurs in any state where a 3D-pose is estimated. In this work, a quaternion is used for global representation of the manifold space, while a scaled-axis representation is used for representing deviations in the vector space. This combination has multiple advantages. The global representation remains small compared to using rotation matrices, however it is still possible to perform basic operations directly on the representation as shown in (2.42) - (2.44). For the vector space the choice is largely arbitrary as long as it is minimal, as this deviation is not used for further processing besides calculating the update on the manifold space. It does however influence how the covariance of the rotation in the state is interpreted, as this is expressed with respect to the chosen representation.

Both $\boxplus$ and $\boxminus$ can be defined for any combination, however in the following they are given only for the combination used in this work. The equations for combining rotation matrices with scaled axis can be found in [Hertzberg et al., 2013], while the combination of quaternions and Euler angles is shown in [Clemens and Schill, 2016]. A more thorough discussion on the use of different combinations including the combination of quaternions and Euler angles is additionally given in [Clemens, 2018].

For quaternions and scaled axes, the operators are defined as [Hertzberg et al., 2013]

$$x \boxplus d = x \exp \frac{d}{2} \tag{2.51}$$

$$y \boxminus x = 2 \,\overline{\log}\, (x^{-1} y), \tag{2.52}$$

with $x \in \hat{\mathbb{H}}$ and $y \in \hat{\mathbb{H}}$ being quaternions, and $d = [d_x \; d_y \; d_z]^T \in \mathbb{R}^3$ being a scaled axis. The functions $\exp : \mathbb{R}^3 \rightarrow \hat{\mathbb{H}}$ and $\overline{\log} : \hat{\mathbb{H}} \rightarrow \mathbb{R}^3$ are defined as

$$\exp d = \cos\|d\| + (d_x i + d_y j + d_z k) \frac{\sin\|d\|}{\|d\|}, \tag{2.53}$$

$$\overline{\log}\, q = \begin{cases} 0, & q_v = 0 \\ \frac{\tan^{-1}(\|q_v\|/q_0)}{\|q_v\|} q_v, & q_v \neq 0, q_0 \neq 0 \\ \pm \frac{\pi/2}{\|q_v\|} q_v, & q_v \neq 0, q_0 = 0, \end{cases} \tag{2.54}$$

with $q \in \hat{\mathbb{H}}$. The symbols $q_0 \in \mathbb{R}$ and $q_v \in \mathbb{R}^3$ represent the real part and the vector part of the quaternion respectively. The functions exp and $\overline{\log}$ map between the different spaces. The function exp converts an element in the vector space to the manifold space, while the $\overline{\log}$ functions performs the opposite conversion.

Similar to rotations in 3D the $\boxplus$-method can also be applied to 2D rotations in order to avoid issues with discontinuities around π and $-\pi$ [Hertzberg et al., 2013]. For this the global representation of the orientation in the state is chosen to be an angle $\psi \in \mathbb{R}$. To avoid issues with discontinuities, periodic equivalents $\psi + 2\pi k, k \in \mathbb{Z}$ are treated as the same value, and as such the internal representation can exceed the bounds of π and $-\pi$. Thus adding a rotation to the global representation is a simple addition, while the $\boxminus$ operator must take into account that the internal representation is not normalized to the correct bounds. The $\boxplus$-operator is thus defined as

$$x \boxplus \delta = x + \delta. \tag{2.55}$$

The $\boxminus$-operator on the other hand is defined as

$$y \boxminus x = v_\pi(y - x), \tag{2.56}$$

where $v_\pi(\delta) := \delta - 2\pi\lfloor\frac{\delta+\pi}{2\pi}\rfloor$. This definition gives the smallest angular difference between x and y, which is an intuitive result.

With these functions defined, it is now possible to update a manifold state using small deviations both in 2D and 3D. It should however be noted, that most frequently the state is a composite of many different spaces, some of which may be manifold while others are simple vector spaces. In the latter case the $\boxplus$ and $\boxminus$ operators reduce to a simple addition and subtraction for the vector parts of the state, defined as

$$x \boxplus d = x + d, \tag{2.57}$$

$$y \boxminus x = y - x. \tag{2.58}$$

In some cases, the state may also be composed of multiple manifolds S_1 and S_2. In those cases, the operations are performed component-wise on the corresponding parts of the state. This is defined as

$$\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \boxplus \begin{bmatrix} d_1 \\ d_2 \end{bmatrix} = \begin{bmatrix} x_1 \boxplus_{S_1} d_1 \\ x_2 \boxplus_{S_2} d_2 \end{bmatrix}, \tag{2.59}$$

$$\begin{bmatrix} y_1 \\ y_2 \end{bmatrix} \boxminus \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} y_1 \boxminus_{S_1} x_1 \\ y_2 \boxminus_{S_2} x_2 \end{bmatrix}. \tag{2.60}$$

In most sensor fusion algorithms, simply representing the state of an object is not sufficient. Instead, a lot of information is contained in the uncertainty of the estimate. Most multivariate probability distributions are however defined on the vector space. To represent uncertainty of the manifold elements, normal distributions can be defined on $\boxplus$-manifolds [Hertzberg et al., 2013]. For this, an element $\mu \in \mathcal{S}$ of the manifold is used as mean, while the multivariate normal distribution $\mathcal{N}$ is defined on the vector space $\mathbb{R}^n$ and then mapped to the manifold around μ using the $\boxplus$ operator. This is defined as

$$\mathcal{N}_{\mathcal{S}}(\mu, \Sigma) = \mu \boxplus \mathcal{N}(0, \Sigma), \tag{2.61}$$

where $\Sigma \in \mathbb{R}^{n \times n}$ is the covariance matrix.

Many sensor fusion algorithms require derivations of functions, for example for optimization or state estimation using the EKF. For this, the derivation in the vector space around the element of the manifold is required, even when the original space is a manifold. In the example of an EKF this is the case since updates of the state are calculated on the vector space, and therefore the derivatives are required on the same space. The derivatives of functions on a manifold space in the vector space can be obtained by

[Hertzberg, 2015, Appx. B.1].

$$\frac{d}{dx} f(x) = \frac{d}{d\delta} \left(f(x \boxplus \delta) \boxminus f(x) \right), \tag{2.62}$$

calculated at $\delta = 0$, with $\delta \in \mathbb{R}^n$. Here, the function $f : \mathcal{S} \to \mathcal{X}$ is a function on the manifold space $\mathcal{S}$ which maps to a manifold space $\mathcal{X}$.

While the thesis contains derivatives for all functions where they are required, two of the most common operations and their derivatives when working with a $SO(3)$ state space are given here. When rotating a vector with a quaternion the two relevant derivations are given by

$$f(q, v) = qvq^{-1}, \tag{2.63}$$

$$\frac{\partial f}{\partial q} = -R[v]_\times, \tag{2.64}$$

$$\frac{\partial f}{\partial v} = R, \tag{2.65}$$

where $[v]_\times$ is the skew-symmetric matrix of v and R is the rotation matrix corresponding to the rotation defined in q. Additionally often two rotations are concatenated. The derivative is defined as

$$f(q_1, q_2) = q_1 q_2, \tag{2.66}$$

$$\frac{\partial f}{\partial q_1} = R_{q_2}^T, \tag{2.67}$$

$$\frac{\partial f}{\partial q_2} = I. \tag{2.68}$$

For a thorough overview of commonly used operations and their derivations on $\boxplus$-Manifolds the reader is referred to [Sola et al., 2018].

2.3 Grid Mapping

Grid maps are one of the most commonly used methods for representing information about the environment. It is an approximate representation, where information about the environment is aggregated in cells to handle large sets of data. The environment is discretized in square cells in 2D or cubes in 3D. Their size can be either fixed or variable, with each cell representing the state of the entire area it covers. This abstraction leads to a loss of information depending on the resolution of the grid map, however this is often required when using high-frequent high-resolution measurements like LiDAR data to enable real-time capabilities. One of the most frequently used types of grid maps is the so called occupancy grid map [Elfes, 1989; Thrun et al., 2005, Chap. 9]. Here, a grid map $y = y_{1:M} \in \mathcal{Y}^M$ is used to represent the state of world, where y represents the grid map containing all cells, M determines the amount of cells and $\mathcal{Y}$ is the set of possible states. Each cell $y_i \in \mathcal{Y} = \{o, f\}, 1 \leq i \leq M$ can represent occupied space denoted o

or free space denoted f. Any sensor that measures larger parts of the environment can be used to build such a map, however the most common implementations use LiDAR, Radar, ultrasonic or a combination of these sensors. While it is possible to build grid maps that assign binary labels of either free or occupied, it is difficult to build and reason on those grid maps, as cells may be measured multiple times, sometimes with conflicting results. To overcome this shortcoming, occupancy grid maps make use of a probabilistic framework, with each grid cell being a Bernoulli-distributed random variable, where the parameter of this distribution is estimated. This is defined as

$$y_i|x_{0:t}, z_{0:t} \sim Bernoulli(\pi_i),\ 1 \leq i \leq M, \tag{2.69}$$

where $\pi_i \in [0,1]$ represents the probability of a cell to be occupied, depending on the states $x_{0:t}$ and all previous measurements $z_{0:t}$. As the states $x_{0:t}$ are fixed, this type of mapping is referred to as mapping with known pose [Thrun et al., 2005, Sect. 9.2]. To estimate the posterior $P(y_i|x_{0:t}, z_{0:t})$, a recursive approach [Thrun et al., 2005, Sect. 9.2], maximum likelihood estimation (MLE) or maximum a posteriori estimation (MAP) [Thrun et al., 2005, Sect. 9.4] can be used. In the recursive approach, the posterior $P(y_i|x_{0:t}, z_{0:t})$ is estimated by fusing the previous estimate $P(y_i|x_{0:t-1}, z_{0:t-1})$ with the result of the inverse sensor model $P(y_i|x_t, z_t)$ probabilistically. The inverse sensor model attempts to determine the state of the map while being conditioned on measurements that were produced by measuring this world. This model can be difficult to formulate depending on the sensor. In contrast, MLE and MAP rely on the forward sensor model $p(z_t|x_t, y)$. This model instead formulates the probability of a measurement given the current map. To perform MLE or MAP, they solve

$$y^* = \underset{y \in \mathcal{Y}^M}{\arg\max} \prod_{j=0}^{t} p(z_j|x_j, y)P(y). \tag{2.70}$$

The prior $P(y)$ is assumed to be uniformly distributed in MAP, while it is ignored for the estimation in MLE. This way of estimating the map is called forward model mapping, since it only uses the forward sensor model. While this method can be easier to use, due to the forward model being more intuitive to formulate, the map estimation is performed as an optimization problem in each step instead of being an iterative process. Therefore, this method of map estimation can be computationally demanding.

Instead of estimating the occupancy of a cell, another approach is to estimate the fullness $\theta_i \in [0,1]$ of a cell. This approach can be used whenever highly precise measurements are taken, such as LiDAR measurements. With the noise of the measurement being significantly smaller than the error introduced by the discretization process, the measurement noise can be neglected. This reduces the map generation to $\theta_i = k_i/(k_i+l_i)$, with k_i indicating how often an endpoint of a scan hit the cell and l_i representing how often the cell was traversed but not hit, which would indicate that it is free [Clemens, 2018]. This particular simplification enables the use of an highly efficient algorithm which simply increments the counters according to the current measurement z_t.

One particular problem these approaches have in common is that no distinction can be made between cells for which no information is available and cells that are measured with conflicting results. In both cases the result would be a uniform distribution $P(f) = P(o) = 0.5$, making the cases indistinguishable. To overcome this shortcoming, belief function theory can be utilized [Dempster, 1968; Dempster, 1967]. Instead of estimating the state of a cell by determining the probability of a binary variable $y_i \in \{o, f\}$ each cell is represented by an evidential variable. In this variable, belief mass can not only be assigned to each individual element of the set of possibilities $\Theta_Y := \{o, f\}$. Instead the full frame of discernment contains all subsets, including the empty set $\emptyset$ and the superset Θ_Y. Depending on the subset different properties can be expressed. While belief mass on the sets $\{o\}$ and $\{f\}$ represents whether the cell is occupied or free, belief mass on $\emptyset$ indicates a conflict in information where a cell was measured as free as well as occupied. On the other hand, belief mass on the superset Θ_Y indicates that no information is available for this cell. With this additional distinction, more informed decisions can be made about the environment [Reineking and Clemens, 2014].

This representation can be extended to gain additional information about the environment by extending the frame of discernment to include dynamics [Steyer et al., 2018]. The modified set $\Theta = \{f, s, d\}$ allows hypotheses for free as well as statically and dynamically occupied areas and any combination of these hypotheses. In addition, within each cell a velocity is estimated, giving more detailed information about the dynamics within the environment. This is in contrast to the previously mentioned methods where dynamics are not considered and need to be excluded before building the grid map. The capabilities and applicability of this extension are further explored in Chap. 6.

2.4 Coordinate Systems

In autonomous driving, there exist a number of relevant coordinate systems in which data is expressed. In the following, an overview is given on all relevant coordinate systems that are used in this work.

2.4.1 Vehicle Frame

The vehicle frame, denoted with V is positioned on the center of the vehicles rear axle, with the origin in the ground plane. It is a cartesian coordinate system where the x-axis is pointed in driving direction, the y-axis towards the left of the vehicle, and the z-axis points upwards. The IMU is positioned right above the origin of the vehicle frame to avoid frequent transformations that would be required to compensate an offset. The positioning on the rear axle is done to simplify computations and reduce undesired measurements of latitudinal acceleration experienced during turns. During a turn the rear axle is always positioned orthogonally to the circular path the vehicle is following as long as the wheels do not drift, which is assumed in this thesis. This is not the case for any other point on the vehicle, which would cause the acceleration to be measured on different axes. This would need to be compensated, making the vehicle models more complicated. Additionally, since the front axle is used for steering it shifts during a turn which would causes undesired changes in the measured acceleration if the IMU (and thus the vehicle frame) was positioned on this axle, although this effect would be minimal.

2.4.2 Navigation Frame

The navigation frame, also referred to in this thesis as the odometry frame and denoted with N is one of the most commonly used coordinate system in this work. It is a cartesian frame and its origin and orientation are initialized as zero, which reflects the fact that the vehicle has not moved in the beginning. In this frame the odometric vehicle state is estimated, which is a jump-free estimate of the vehicle pose and dynamics, as explained in Sect. 4.2. Since the vehicle state is expressed in this frame, most computations in this work are performed in the odometry frame.

2.4.3 Global Frame

In contrast to the odometry frame, which is always centered wherever the vehicle was started by initializing the position as zero, the global frame, denoted as G has a fixed reference. The global frame is a polar coordinate system which is centered in the center of the WGS84 reference ellipsoid [Wendel, 2007, Sect. 3.1]. The orientation is however not expressed in the coordinate system centered in the center of earth, as this would lead to unintuitive axes of rotation since the z-axis points towards the top of the ellipsoid. Instead the orientation is expressed in the east-north-up frame centered in the object for which the orientation is described. This leads to a consistent description of the orientation, with the z-axis pointing towards the sky orthogonally from the point on the ellipsoid at the position of the object. This effectively makes the rotation of the z-axis the rotation around the ”up” axis, which is consistent with the odometry frame. The x-axis points east, while the y-axis points north.

2.4.4 Sensor Frames

Many different sensor frames exist on the vehicle, with each camera, LiDAR and radar defining its own coordinate system, denoted C, L and R respectively. The LiDAR and radar coordinate systems are simple cartesian coordinate systems and the data can easily be transformed to the vehicle or odometry frame for further processing. The camera on the other hand measures on its 2D image plane. For the camera frame it is therefore more complicated as this sensor measures in 2D and no direct transformation to the vehicle frame is possible. A coordinate system is still fixed in the origin of the camera, with the z-axis pointing in the direction the camera is measuring in, while x points towards the right and y points downwards. Measurements in the camera are however expressed in pixel coordinates on the image plane. Converting these measurements to the 3D coordinate system is only possible under certain conditions using prior knowledge, additional sensors (e.g. stereo camera) or strong assumptions. The handling of these measurements is described in detail in Sect. 6.2.4.

2.5 Measurement Scheduling

Within a complex real-time system such as an autonomous vehicle, it is important to consider each sensor measurement according to the time it was measured. This is however not necessarily the time at which the data is available to the algorithm. Many factors play a role such as delays within the sensor between measuring and transmitting the data, delays within the network of the vehicle or delays due to multi-threading within the software stack of the vehicle. As a result, measurements from different sensors will often arrive out of order with a moderate delay. Especially when using Kalman filtering it is however difficult and tied to additional computational costs to retroactively fuse measurements into the state estimate. The current state is assumed to only depend on the previous state as well as the current measurement. The history is encoded in the relationships between parts of the state within the cross-covariance matrix. To include an old measurement would require the state to be re-estimated using all previous measurements in the correct order.

To overcome this issue the measurements are sorted before being handed over to the individual algorithms. It is however not possible to sort measurements in real-time, as it is unknown if even older messages will be received after obtaining a measurement. Therefore a delay $d \in \mathbb{R}^+$ needs to be introduced which is long enough to compensate for out of order measurements but still short enough to allow for a reactive autonomous driving system. To not loose any data during this delay a number of queues $q_i, i \in S$ are created, where S is the set of all available sensors on the vehicle. As new measurements arrive they are placed into their respective queue. All queues are checked periodically to evaluate whether any queue contains measurements older than d. When a measurement exceeds this delay it is automatically processed as it is assumed that no measurements arrive with a larger delay.

This implementation results in an autonomous system with a delay of d, where in this thesis $d = 20\text{ms}$ is chosen. However the true delay can often be reduced significantly. Whenever all queues contain an entry the most recent measurement can be processed until one queue is empty. This is due to the fact that the measurements for each sensor are expected to arrive in order. Thus when each queue is filled the ordering of all available measurements can be determined until one queue is empty again. Using this method the true delay of the system is variable and can go up to d, however often it will remain significantly lower.

3 Research Vehicle

Testing the capabilities and limitations of sensor fusion approaches is a difficult task, as the results often depend on many variables and slight changes in the input data may alter it significantly. Therefore, testing on simulated data is only done for select cases where all relevant error sources can be modeled sufficiently well. Often, approaches can be tested on publicly available datasets such as the KITTI Vision Benchmark Suite [Geiger et al., 2012] or Waymo Open Dataset [Sun et al., 2020], which already contain real-world data. However, to fully test an approach for autonomous driving, the algorithm needs to be executed on an autonomous vehicle where the sensor fusion result is used for decision making and vehicle control. Therefore, all algorithms in this work are developed to be used in real-time on an autonomous vehicle. The vehicle and its sensors are presented in the following.

The vehicle itself is a customized VW Passat GTE. It is equipped with a number of sensors which provide information about the vehicle as well as its surroundings. The sensors built into the vehicle are shown in Fig. 3.1. The resulting coverage of the surroundings with the available sensors is shown in Fig. 3.2. Note that while ultrasound sensors and area view cameras are also built into the vehicle these are mainly useful in parking scenarios due to the low range of ultrasound sensors and the placement and strong distortion of the area view cameras. Thus these sensors are not used in this work. The sensor information is processed on a custom PC, which is built into the trunk of the vehicle. The specific sensor models used can be found in Tab. 3.1 including their approximate original prices. In addition to the sensors noted here the vehicle comes equipped with ultrasound, radar, area view camera, wheel speed and steering wheel angle sensors, which are however included in the series production vehicle with no additional cost at-

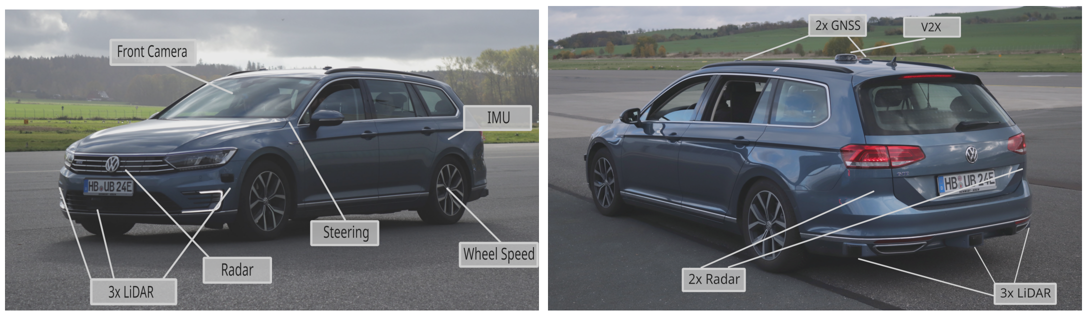


Figure 3.1: The VW-Passat and its built in sensors.

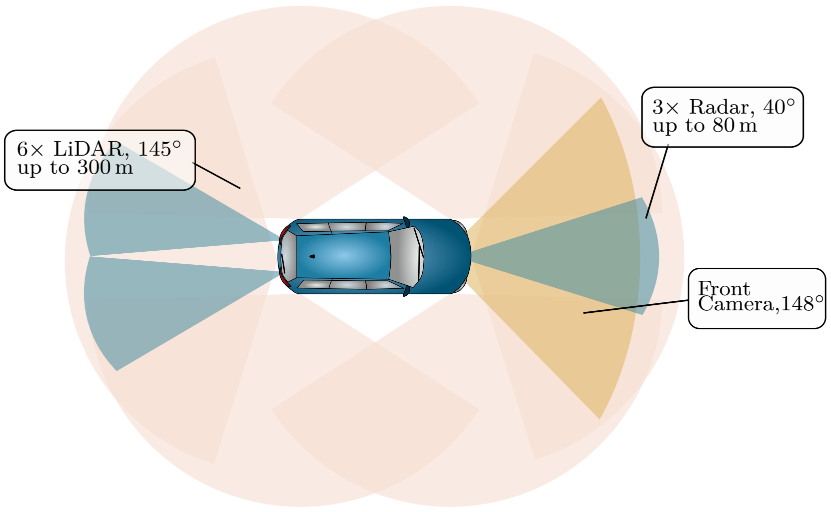


Figure 3.2: Sensor coverage of the sensors used in this thesis (not to scale).

Sensor type	Amount	Sensor model	Price per sensor	Year
IMU	1	ADIS 16488	1000€	2021
LiDAR	6	SCALA Gen 1	10000€	2018
Front Camera	1	LI-AR0233-GW5200-GMSL2	600€	2022
GNSS Receiver	3	u-blox (2xF9P, 1xM8T)	400€, 75€	2019
V2X	1	Nordsys WaveBee	8000€	2021

Table 3.1: Custom sensors in the vehicle, their approximate original prices and the time at which they were integrated in the research vehicle.

tached. However accessing these sensors is possible by connecting to the modified CAN Bus. Note that especially the LiDAR was originally very expensive. Since six LiDARs are required for a full coverage this accounted for a significant portion of the sensor cost. However with recent developments in sensor technology similar LiDARs with more layers are available around 2000€. This indicates that this sensor has the potential to be cost efficient enough to be utilized in autonomous vehicles on a larger scale in the future which would not be the case with the previous cost. The individual types of sensors are introduced in the following, to give an overview about the available information for the sensor fusion system.

3.1 Inertial Measurement Unit (IMU)

An inertial measurement unit (IMU) enables the precise measurement of an object's motion characteristics, including its acceleration and turn rate. The IMU achieves this through the integration of various sensors, such as accelerometers, gyroscopes, and often magnetometers, each responsible for measuring specific aspects of the object's movement. In this work, microelectromechanical systems (MEMS) IMUs are considered due to their low cost and size while providing high accuracy [Otegui et al., 2020]. The accelerometer measures accelerations along specific axes. It operates on the principle of inertia, measuring the displacement of a mass within the sensor which is proportional to the acceleration of the sensor. In a 3-axis IMU, three accelerometers are oriented orthogonally. By measuring the accelerations along these axes simultaneously, the IMU can precisely determine the object's linear motion in three-dimensional space.

Gyroscopes are responsible for measuring angular rate, which refers to the object's rate of rotation around its axes. Gyroscopes utilize the principles of angular momentum to detect rotational movement. While traditionally gyroscopes used to measure angular momentum using a spinning mass mounted to a gimbal, modern MEMS IMUs use a tuning fork configuration, where two masses are attached using a spring, with the rotational axis in between. Any rotation will assert force in different directions for the two masses, while a linear movement would result in a similar force for both. From the difference in force, the angular momentum is calculated.

In addition to the data from accelerometers and gyroscopes, some IMUs also integrate magnetometers. Magnetometers measure the Earth's magnetic field and provide information about the object's orientation relative to the magnetic north. However, as the IMU is mounted within a vehicle which has an interfering magnetic field itself, this information is not used in this work.

The IMU is a crucial sensor in accurately estimating a vehicles movement over short periods of time. With the sensor measuring acceleration and turn rate, to determine a velocity or even position of the vehicle the sensor data must be accumulated over time. This accumulates errors over time, however over short periods it is highly accurate and gives information at a very high frequency, often significantly above 100Hz, with the IMU used in this thesis measuring at 800Hz.

3.2 Vehicle Speed and Steering Angle

In addition to the IMU the vehicle comes equipped with several sensors which give additional information about the vehicle odometry. These additional sensors are built into the vehicle by the manufacturer and the specific hardware is unknown, however the information still proves useful for sensor fusion. The sensor data is obtained using the vehicle Controller Area Network (CAN) bus. Firstly, the vehicle calculates a velocity both for the vehicle as a whole and for each tire individually. This is done by wheel encoders that measure the rate at which the wheels turn. In combination with some wheel slip compensation and knowledge about the wheel circumference this is used to determine the velocity. Note that a velocity can also be determined from accumulating IMU measurements, however in contrast to the IMU the wheel odometer does not suffer from sensor drift which occurs when noisy, relative measurements are accumulated into a higher order estimate. The frequency at which the odometer provides data is however

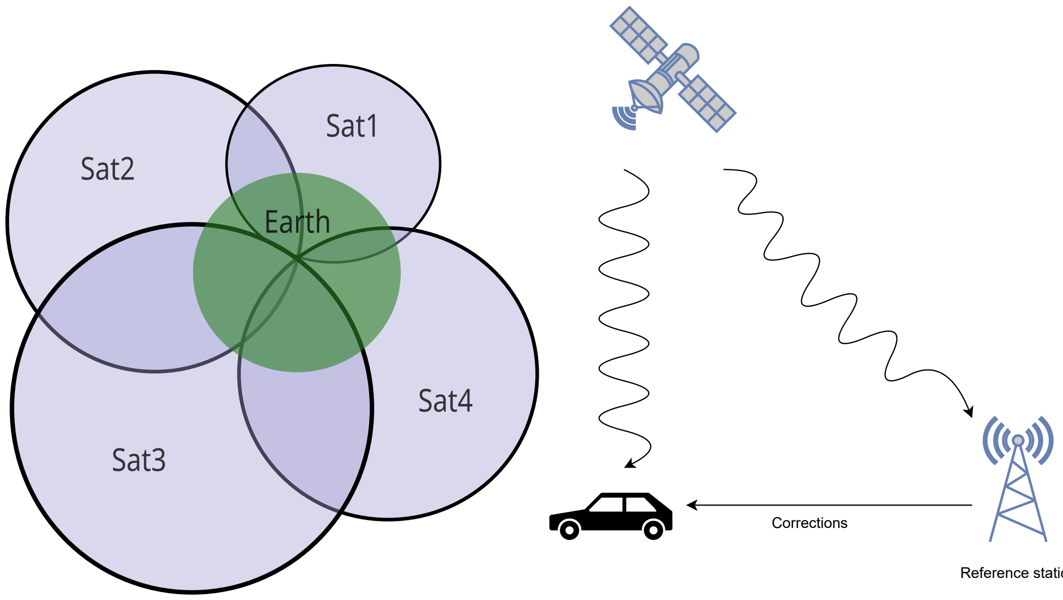


Figure 3.3: Visualization of the global positioning using GNSS and RTK. On the left: Distance to four Satellites shown as a circle and the resulting unambiguous intersection. On the right: Correction of GNSS position using RTK.

significantly lower at 50Hz. Thus this combination of sensors is an ideal scenario for sensor fusion approaches to obtain a highly accurate estimate at a high frequency without sensor drift.

Secondly, the vehicle provides information about the steering wheel angle. To make this angle usable, a calibration is performed to convert it to the actual steering angle of the wheels. The information provided by the steering angle is partially redundant to the turn rate provided by the gyroscope, however again the measurement rates differ and combining the two information sources leads to a more robust estimate of the movement of the vehicle.

3.3 Global Navigation Satellite System (GNSS)

A Global Navigation Satellite System (GNSS) is a network of satellites that enables precise positioning, navigation, and timing services on a global scale. It is available anywhere with a clear view of the sky, making it an almost indispensable tool for autonomous navigation on earth.

The GNSS system uses constellations of satellites orbiting the Earth. The most well-known and widely used GNSS systems include the United States' Global Positioning System (GPS) and Russia's Global Navigation Satellite System (GLONASS) and Galileo from the European Union, however there are multiple more like, BeiDou (China) and NavIC (India). To aid in obtaining a strong GNSS signal modern receivers often use a combination of multiple different systems, which is also the case in this thesis.

To calculate a position, a GNSS receiver is built into the vehicle, which is equipped with an antenna that receives signals from multiple satellites. This signal contains information about the position of the satellite as well as the time of sending. Using this information as well as the time of flight of the signal, which travels at the speed of light, a distance to each satellite can be calculated. When three or more satellites are received, this information can be used to triangulate the position of the receiver as shown in Fig. 3.3. In modern receivers more information like the phase and wavelength is used to calculate a more accurate position. Current openly available GNSS systems reach accuracies of around 1-2 meters in good conditions. This is however not sufficient for autonomous driving, where at least a lane-accurate localization is required. To boost GNSS performance, real-time kinematic positioning (RTK) is used. A reference station is placed within the range of the vehicle, which first measures its own position very precisely over an extended period of time. After this initialization it can be used to determine errors in the GNSS signal that are created from atmospheric disturbances. A correction signal is then sent to the vehicle, which is utilized to correct the errors in the signals the vehicle receives. Using this method, an accuracy of a few centimeters is reached.

Unlike IMU or odometer measurements, the measurement obtained by the GNSS receiver contains an absolute position directly without the need to accumulate measurement (and with it errors) over time. In addition a velocity can be determined from the Doppler effect observable over time. Especially the absolute measurement of the position is immensely useful as otherwise obtaining a globally correct position without drift would require much more sophisticated algorithms. For this thesis a two-receiver setup is used, which allows for direct heading estimation using GNSS measurements. The heading is calculated from the relative position measured between the two receivers, which is explained in detail in Sect. 4.3. The rate at which GNSS measurements arrive is comparably limited, often between 1-10Hz, although modern receivers can reach up to 100Hz. The receivers used in this thesis provide data at 10Hz. Thus using GNSS as the only source of information for positioning results in large inaccuracies during times where no current data is available. Additionally GNSS coverage can not be guaranteed in all situations making it a difficult sensor to work with in autonomous driving as the system can not rely on global positioning information to be available at all times. Especially when no clear view to the sky is available the satellite data can not reach the GNSS receiver. However other issues occur as well in autonomous driving especially in urban areas. Driving next to large buildings causes reflections of the satellite signal, leading to incorrect assumptions about the distance to the satellite. This is called a multipath error and can be reduced by modern hardware, for example by checking the direction from which a signal was received, however it can not currently be fully eliminated. Dealing with these shortcoming is discussed in detail in Chap. 4.

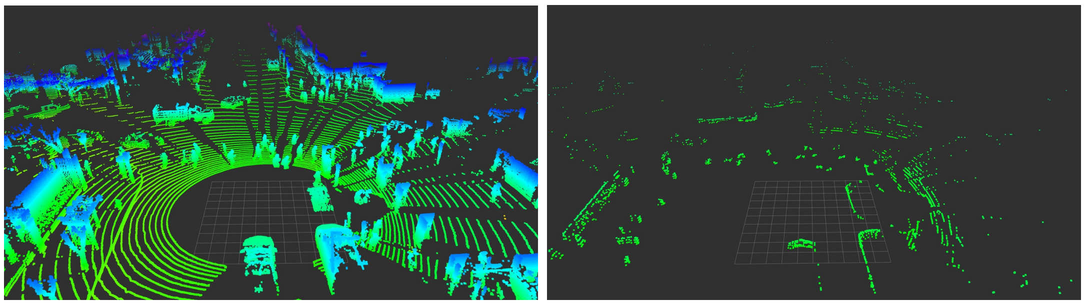


Figure 3.4: Comparison of data obtained by a 128-layer LiDAR with data from a 4-layer LiDAR. On the left: A point cloud from the Waymo dataset which includes a 128-layer LiDAR mounted to the top of the vehicle [Sun et al., 2020]. On the right: A point cloud of the same scene from a 4-layer LiDAR.

3.4 Light Detection and Ranging (LiDAR)

Light Detection and Ranging (LiDAR) is a remote sensing technology that measures distances by emitting laser pulses and analyzing their reflections. The system calculates the distance by measuring the time it takes for the laser pulse to travel to an object and return, utilizing the principle of time-of-flight. The majority of currently available commercial LiDARs are so called scanning LiDARs, which utilize a spinning sensor to scan the field of view (FOV) sequentially. This is done in horizontal layers, with the amount of layers ranging from only a single layer (2D LiDAR) to more than 100 layers. Depending on the amount of layers, the resulting point cloud contains vastly different amounts of information, as can be seen in Fig. 3.4. Most research vehicles for autonomous driving use a 360 degree LiDAR mounted to the top of the vehicle, which measures on at least 32 layers. This is made evident by most publicly available autonomous driving datasets containing a top-mounted 3D LiDAR [Sun et al., 2020; Geiger et al., 2012; Caesar et al., 2020]. To keep the vehicle close to a series production, this is not the case in this thesis. Instead, six Scala LiDARs are almost seamlessly built into the exterior of the car as seen in Fig. 3.1. Each LiDAR has a horizontal FOV of 145 degrees and a vertical FOV of 3.2 degrees. They measure along four layers spread evenly over this field of view, although each scan only contains three layers, alternating between the top three and the bottom three.

The coverage reached with this setup can be seen in Fig. 3.2. With every part of the surroundings being covered by at least one LiDAR, this sensor is ideal for obstacle avoidance and object detection on the vehicle. However, with only a 3.2 degree opening angle and only three layers per scan, the information that can be extracted from it is limited when compared to top-mounted 360 degree scanners. This limitation is a driving factor in selecting suitable approaches for both obstacle avoidance and object detection in this work. While the LiDAR is useful in determining the location of objects or obstacles in the environment, it is difficult to determine semantic information from it. Especially determining object classes such as cars, pedestrians or bikes is crucial for safe and predictive driving. However without any texture information to use for classification determining object classes is often not possible. Thus using only LiDAR sensors would result in limited understanding of the environment.

3.5 Camera

To provide the missing texture information about the environment a front-facing camera is built into the research vehicle. This is used to gain information about upcoming traffic signs, available lanes and to aid in object detection and classification. The data provided by the camera is complementary to the point clouds generated by the LiDARs. In contrast to LiDARs the Camera data measures its environment on a 2D image plane. Thus every point in the world is projected to a flat surface, losing the information about the location of the point in the world. On the other hand color information is available, making texture information usable for tasks like object classification. Cameras are a highly potent sensor in autonomous driving, as they provide information similar to what a human uses when driving. Since humans are very good at determining other objects in the environment and where the vehicle can drive, it could be argued that most information required for autonomous driving can be gathered from camera data. However the goal in autonomous driving is not to match human performance but to achieve the highest possible safety. As a result, to make the system as robust as possible the addition of other sensors is desirable and it is unlikely that autonomous vehicles will ever solely rely on camera data.

A difficulty in using cameras is, that the most accurate models rely on a large amount of training data, which is only available to a small number of very large companies leading the developments in autonomous driving. This data is used to train deep neural networks, which are also not a focus of this thesis due to their lack of explainability, and thus the Camera is only used to support the LiDAR detections and add additional information to it. It is used for object detection in front of the vehicle, however this object detection is seen as a black box in this work and object detection or tracking algorithms do not depend on the existence of camera data. This is especially important as no full coverage of the environment by cameras exists on the vehicle, with only one front-facing camera being used.

Four additional Area View Cameras are built into the car by the manufacturer, however they have very limited resolution and their positions make them unusable for most tasks, so their data is not used in this work.

3.6 Vehicle-to-Everything (V2X)

An emerging technology in autonomous driving is the Vehicle-to-Everything (V2X) communication. Instead of placing the burden of detecting every aspect of the environment on every single vehicle, the idea here is that every relevant piece of infrastructure and as many traffic participant as possible estimates their own state and broadcasts this for others to use. While this is still far from reality, first concepts are proposed in this work to use this information to aid object tracking. For communication between vehicles, dedicated V2X hardware is used. A choice can be made between using a cellular network like 4G or when available 5G, or using Dedicated Short-Range Communication (DSRC) with a range of roughly 500 meters [Bae et al., 2020]. Since 4G may not be available everywhere, the vehicle is equipped with the latter, enabling it to communicate with any vehicles in its direct vicinity.

Communication between intelligent vehicles enables sharing of a large variety of useful information. While the simplest case is sharing the ego state (e.g. position, orientation, velocity) with other vehicles, other information that is more difficult to obtain can be broadcast as well. Other detected objects, surrounding traffic signs or even raw sensor data like LiDAR measurements are examples of data that could potentially be shared among vehicles to increase the robustness of each individual vehicle. In this thesis the focus is set on broadcasting the ego state as a first proof of concept that this can be used to increase robustness and eliminate misdetections. Determining the ego state is a much simpler task than detecting other traffic participants and estimating their state from those measurements. Thus by broadcasting this information an additional source of highly certain information becomes available to the fusion algorithm of all other traffic participants, where they would otherwise rely on less certain information. By fusing the received information with available sensor data a more robust estimate of the traffic participant can be estimated. An additional benefit is the possibility of receiving broadcasted states of vehicles that are not currently observable due to occlusion, for example at sharp corners with low visibility, which could otherwise go unnoticed, thus causing collisions.

4 Localization

To navigate a vehicle through highly complex and often narrow streets, a very precise understanding of the current location and dynamics of the vehicle are required. The task of determining these is solved by the localization. Obtaining an accurate localization is one of the most crucial tasks in autonomous driving, as nearly all subsequent algorithms require information about the positioning of the vehicle, or about its current or recent movement. For many of these tasks, accuracy of the localization is a driving factor in simplifying or even enabling finding a solution. Many algorithms use the recent movement of the vehicle as input to transform the results from the last iteration to the current step, for example to use it as an initialization point. For some, the accuracy of the localization even directly influences the capabilities of the algorithm. Route planning for example requires lane-accurate global positioning, and algorithms such as LiDAR optical flow which is presented in Sect. 6.3.1 depend on an accurate estimate of the vehicle movement to be able to detect changes in the environment.

However, accuracy is not the only requirement. Especially in autonomous driving a continuous, jump-free estimate is necessary to ensure safe driving. A sudden change in the position estimate may lead to a drastic response from the vehicle control as it tries to realign itself with a previously estimated lane or avoid obstacles which now shifted according to the jump. This requirement of a jump-free estimation strongly restricts the possible solutions applicable for localization in autonomous driving, as many approaches inherently produce jumps in the estimated position. An overview of these approaches can be found in Sect. 4.1.

A common denominator in all approaches that produce jumps is, that they view the localization task as a problem where a global solution is the wanted result. As measurements arrive, the current estimate is corrected towards the globally correct solution. Whenever the previous estimate contained inaccuracies, e.g. due to sensor drift, the solution will jump upon correction. As a result, such an approach is not suitable for many tasks in autonomous driving.

However, many tasks in autonomous driving do not require a global solution. A large number of algorithms only make use of the relative movement of the vehicle, also called odometry. Odometry is not corrected using any global reference system and as such it does not jump. Over time it will however accumulate an error, which means it is not globally correct. Over short periods of time this error remains small and as such the odometry can be used to obtain a relative movement of the vehicle.

As can be seen from these observations, there is no single solution which satisfies all

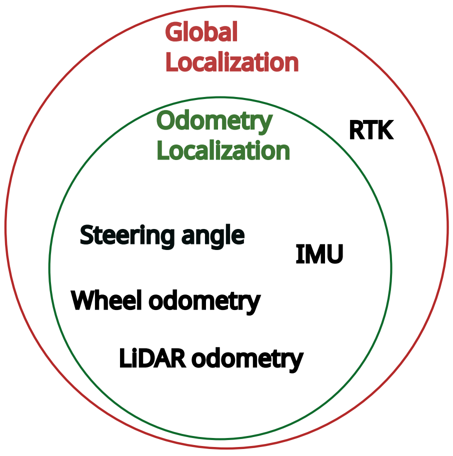


Figure 4.1: Architecture of the localization system.

requirements for autonomous driving. Instead, a localization architecture is used, which calculates both a global solution as well as an odometry. This architecture is shown in Fig. 4.1. The global filter and the odometry filter largely use the same input and the same models, with the exception of global filter being updated using RTK data. As such, the global filter estimates the vehicle state in the global WGS84 frame, while the odometry filter creates a result in a local frame which is centered in the starting position. This local frame is referred to as the odometry frame. To make global information usable for the vehicle, it is transformed to the odometry frame, which is explained in Sect. 4.4.

In the following, the localization system is explained in detail. First, an overview over the current state of the art is given in Sect. 4.1. Then the odometry and the global filter are explained in Sect. 4.2 and Sect. 4.3. Finally, the use of LiDAR odometry is evaluated to aid localization and reduce drift for the odometry filter as well as the global filter in GNSS-denied areas.

4.1 State-of-the-art

The task of localization has been extensively studied in a wide variety of fields, ranging from aviation, over robotics and ground vehicles to underwater localization. Many different approaches exist, and the applicability depends on the available sensors as well as the requirements set by the particular application. In the following an overview over these methods and a discussion on their suitability for this thesis is given. The focus is set on approaches for autonomous driving, where the sensor setup is comparable to the one on the research vehicle presented in Chap. 3.

Traditionally, much research has been done in the direction of inertial navigation systems in combination with global navigation systems (INS/GNSS) [Groves, 2013; Breßler et al., 2016]. In this approach, an inertial measurement unit (IMU) is used together with a global navigation satellite system (GNSS) to calculate a global vehicle state. While the IMU gives a precise estimate of the vehicles movements over a short period of time, this will drift over time, which is then corrected using the GNSS information. The GNSS positioning is locally less precise and arrives at a much lower frequency, however it does not drift over time. By combining the two sensors a global estimate without drift can be calculated, which is still locally precise due to the inclusion of the IMU measurements. While this is a comparably mature field of research, some advancements were still made in recent years. In [Dai et al., 2020] a recurrent neural network (RNN) is utilized to estimate the error made by the INS system when GNSS is not available in order to minimize the prediction error. In [Jin et al., 2021] a INS/GNSS system aided by a camera is presented for fast initialization of the localization system. A rough orientation is estimated from subsequent GNSS measurements while the vehicle is moving, giving an initial guess for the parameters of the visual inertial odometry. By subsequently comparing the estimated trajectory of the INS/GNSS system with the one estimated by the visual-inertial system these parameters are refined until convergence. In [Chiang et al., 2020] INS/GNSS is integrated with a simultaneous localization and mapping (SLAM) algorithm, which is another possible approach for solving the localization problem which is introduced in the following. The combination of INS/GNSS ensures lane-level accuracy even in areas with little to no GNSS reception, utilizing a LiDAR for calculating the SLAM.

One possibility to localize the vehicle is by utilizing a pose-graph representation of the trajectory. It consists of nodes that represent the vehicle poses at certain points in time, that are connected by measurements of the movement between the nodes. The types of measurements used vary between approaches, with the simplest implementation using only odometry measurements directly provided by the vehicles sensors, such as inertial measurements and the velocity. However, when using only relative measurements the estimate will drift over time, with no way to correct the error. To circumvent this issue, SLAM can be used [Grisetti et al., 2010]. In this approach, the task of generating a map of the environment and finding the correct location within this map are viewed as dependent on each other and are solved simultaneously. This is done by additionally representing measurements of the environment in the pose-graph. The type of measurements used vary, however the most common approaches are visual SLAM (vSLAM) and LiDAR SLAM, where measurements are obtained by camera and LiDAR respectively. Whenever an area is visited twice, similar measurements are associated and loop closure is performed in order to obtain a globally consistent map.

One of the state of the art approaches for vSLAM is the ORB-SLAM2 algorithm [Mur-Artal and Tardós, 2017]. Features are extracted from an RGB-D image at regular intervals and matched to existing features using bundle adjustment (BA). This is performed on a local scale to reduce drift and on a global scale to perform loop closure. The approach is extended in ORB-SLAM3 [Campos et al., 2021] to improve initialization speed, place recognition and overall localization accuracy. While ORB-SLAM relies on feature extraction for determining movement, there are also approaches that work directly on the image. The Direct Sparse Odometry with Loop Closure (LDSO) [Gao et al., 2018] calculates a sparse representation of the environment directly from the camera image. It is an extension of the DSO [Gao et al., 2018] where the photometric error is used to determine odometry from changes in subsequent images. In LDSO a pose-graph representation is used for solving the SLAM problem, while loop-closure is performed using a feature-based bag-of-words approach. A similar approach is presented in [Engel et al., 2014] where the full image without any feature-based abstraction is used directly in order to build a depth map of the environment against which new images are matched. These depth-maps are stored in a pose-graph of keyframes to solve the SLAM problem.

Instead of using a camera, a LiDAR is often utilized for measuring the environment. In [Hess et al., 2016] the 2D-SLAM framework Cartographer is presented, which uses scan matching between subsequent scans to obtain an odometry, while using global scan matching on submaps to calculate loop closures. Another approach is LOAM [Zhang and Singh, 2014], where a 3D-LiDAR is used to aid localization. In this work the SLAM problem is split into calculating an odometry and creating a map. As such, no loop closure is performed and the result is an odometry with reduced drift. This work is extended in LIO-SAM [Shan et al., 2020] to use a factor-graph which enables loop-closure and inclusion of additional sensors. Finally, a combination of both LiDAR and Camera can be considered, which is implemented in LIMO [Graeter et al., 2018]. Here, the LiDAR is used for extracting depth information of the camera features. For odometry estimation, bundle adjustment is used between key frames. In this approach, only odometry is estimated without loop closure.

Instead of estimating motion directly from observations of the environment, filter-based approaches can be used to fuse information from different sensors in a more loosely-coupled way. In this case, the mapping and localization task are not seen as a joint problem, and the localization is solved separately. Using Kalman filtering as introduced in Sect. 2.1.1, uncertain measurements can be probabilistically fused into a state estimate without the need for complicated vehicle model assumptions. A benefit of the Kalman filter is that it allows incorporating arbitrary information sources, making it applicable for both odometry estimation as well as global positioning.

To now make a selection from existing approaches the requirements for autonomous driving need to be taken into account. While all approaches besides the simple pose-graph implementation can localize the vehicle with sufficient accuracy, in most approaches it is possible to produce jumps in the state estimate. An example of such a jump when using GNSS for localization is shown in Fig. 4.2. In the classical GNSS/INS fusion this happens due to errors in the GNSS information. Especially in urban driving this may happen near large structures or under bridges, where non-line-of-sight (NLOS) and multipath errors lead to inaccurate GNSS solutions. In addition, whenever the GNSS signal is lost for a while, for example in GNSS-denied areas, the localization will start to drift without the global correction. Upon receiving global information again, this drift will be corrected, leading to potentially large jumps. Even when using no global correction for local sensor drift, there are still possible sources for jumps. When calculating a SLAM, this happens during loop-closure. As an area is seen twice and the old and new measurements are associated, this allows for a correction of the previously estimated trajectory in the pose graph. This may lead to the node representing the current position in the graph being moved in order to fit the associated measurement. When using a Kalman filter which only incorporates odometry measurements without

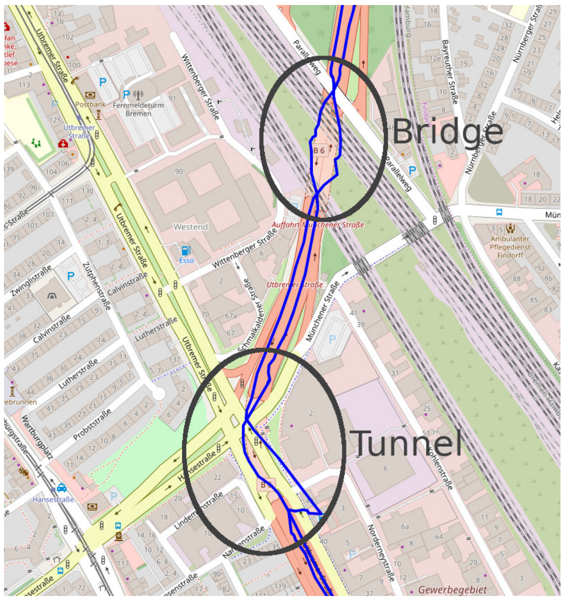


Figure 4.2: Example of the position as estimated by the GNSS receiver in difficult situations. (Maps generated using UMap based on OpenStreetMaps data. Figure adopted from [Clemens et al., 2020].)

global corrections, jumps may still occur in the state estimate [Sun et al., 2018]. This is due to the growing uncertainty over the actual position in the world. As the position is not observable only from relative measurements, this uncertainty grows unbounded and eventually leads to instabilities in the filter.

As such, there are two observations to be made.

- No existing localization system which includes global correction data is jump-free, as the estimate will always jump whenever global correction data was unavailable for an extended period of time or when the correction data is erroneous.

- Even when only odometry is estimated, there still occur jumps, either due to loop closure, or in the case of Kalman filters due to growing uncertainty over the position.

To address the first observation, a localization system is proposed which splits the two tasks of obtaining a jump-free odometry and a global localization. For this two Kalman filters are executed in parallel, one estimating odometry without any global correction and one global filter which performs INS/GNSS. However, as per the second observation, jumps still occur in state-of-the-art odometry estimation. Therefore, a variant of the Kalman filter is introduced, which avoids unbounded covariance growth, and therefore does not suffer from jumps caused by filter instability. In the following the odometry filter is introduced in Sect. 4.2, followed by the global filter in Sect. 4.3.

4.2 Odometry Filter

The Odometry filter fuses all measurements of the relative vehicle movement into a state estimate. The resulting estimate contains precise information about the movement of the vehicle over a short period of time. It will however drift over longer periods of time, limiting its use to algorithms that only rely on short-term movement. The drift is however minimal, making it usable even over stretches of more than 100 meters, which is sufficient for most algorithms that do not rely on a global positioning. The work surrounding the odometry filter was first published in [Clemens et al., 2020] and textually adapted and partially extended for this thesis for better readability. Additionally this thesis is extended to contain the Jacobians of the transition function as well as the measurement functions. As shown in Chap. 3, the vehicle is equipped with a number of sensors for measuring vehicle movement. While the IMU measures acceleration and turn rate, additional sensors in the wheels and steering wheel provide a velocity and steering angle. These measurements are fused in a $\boxplus$-Kalman filter, which uses the $\boxplus$-method as described in Sect. 2.2.3. This is used here as the orientation of the vehicle is part of the special orthogonal group $SO(3)$, which is a manifold. This Kalman filter estimates the state containing a pose in 3D, which consists of a 3D position and orientation. Additionally, the velocity of the vehicle is estimated. Lastly, as IMU data contains significant biases which change over time, these are estimated in the state to correct the measurements accordingly. Thus, the state space is defined as

$$\mathcal{S} = \mathbb{R}^3 \times SO(3) \times \mathbb{R}^3 \times \mathbb{R}^3 \times \mathbb{R}^3. \tag{4.1}$$

The state x_t at time t is defined as

$$x_t = \begin{bmatrix} r_t \\ q_t \\ v_t \\ \bar{a}_t \\ \bar{\omega}_t \end{bmatrix}. \tag{4.2}$$

Here, the position is given by $r_t \in \mathbb{R}^3$ and the orientation $q_t \in SO(3)$ is represented by a quaternion. In $v_t \in \mathbb{R}^3$ the velocity is represented in body-fixed coordinates. In $\bar{a}_t \in \mathbb{R}^3$ and $\bar{\omega}_t \in \mathbb{R}^3$ the biases of the accelerometer and the gyroscope are estimated.

In the following, first the motion and measurement models are given. In the second part, the problem of growing uncertainties in the position as well as the heading of the vehicle is discussed and a solution proposed.

4.2.1 $\boxplus$-Kalman Filter for Odometry Estimation

The motion model, often referred to as the prediction step of the Kalman filter, predicts the state x_{t-1} to the current time t given the control u_t. The control input is often assumed to be some fixed process like driving forward a certain distance, with a certain process noise to model inaccuracies in the execution. In this thesis the process is driven by the acceleration $a_t \in \mathbb{R}^3$ and turn rate $\omega_t \in \mathbb{R}^3$ measured by the IMU, denoted $u_t = \begin{bmatrix} a_t^T & \omega_t^T \end{bmatrix}^T \in \mathbb{R}^6$. As a result, the motion model is defined as

$$f(x_t, u_t) = \begin{bmatrix} r_t + q_t v_t q_t^{-1} \cdot \Delta t \\ q_t \exp((\omega_t - \bar{\omega}_t) \cdot \frac{\Delta t}{2}) \\ v_t + \Delta v_t \\ \bar{a}_t \\ \bar{\omega}_t \end{bmatrix}, \tag{4.3}$$

with $\Delta v_t = (a_t - \bar{a}_t + q_t^{-1} g q_t - [\omega_t - \bar{\omega}_t]_\times v_t) \cdot \Delta t$. Here, $g \in \mathbb{R}^3$ is the gravity vector, Δt refers to the time between t and $t-1$, and exp is defined as shown in (2.53). It should be noted, that the measured acceleration and turn rate are influenced by the rotation of the earth. The compensation of those influences are explored in [Groves, 2013, Sect. 5.3]. However, this influence is negligible as the vehicle is equipped with a MEMS IMU, where the noise is significantly larger than the produced error [Schmid et al., 2012].

To perform an update of the Kalman filter, the covariance $R_t \in \mathbb{R}^{15\times15}$ of the state transition as well as the Jacobian $J_t \in \mathbb{R}^{15\times15}$ around the current estimate is required. The covariance R_t of the state update is defined as a matrix where its diagonal $\bar{R}_t$ contains the following vector

$$R_t = \begin{bmatrix} 0_{3\times1} \\ 0_{3\times1} \\ 0_{3\times1} \\ \Delta t \cdot \hat{\bar{a}} \\ \Delta t \cdot \hat{\bar{\omega}} \end{bmatrix}, \tag{4.4}$$

with each entry being a 3-dimensional vector, and $\hat{\bar{a}} \in \mathbb{R}^3$ and $\hat{\bar{\omega}} \in \mathbb{R}^3$ being the random walk for the acceleration bias and turn rate bias respectively. For the state transition Jacobian, the function $f(x_t)$ is derived with respect to all elements of the state. The resulting Jacobian is defined as

$$J_t = \frac{\partial f}{\partial x} = \begin{bmatrix} I_{3\times3} & \frac{\partial f_r}{\partial q} & \frac{\partial f_r}{\partial v} & 0_{3\times3} & 0_{3\times3} \\ 0_{3\times3} & \frac{\partial f_q}{\partial q} & 0_{3\times3} & 0_{3\times3} & -I_{3\times3}\Delta t \\ 0_{3\times3} & \frac{\partial f_v}{\partial q} & \frac{\partial f_v}{\partial v} & -I_{3\times3}\Delta t & \frac{\partial f_v}{\partial \bar{\omega}} \\ 0_{6\times9} & & & I_{6\times6} & \end{bmatrix}, \tag{4.5}$$

where the non-zero and non-identity parts of the Jacobian are defined as

$$\frac{\partial f_r}{\partial q} = -R[v]_\times \Delta t, \tag{4.6}$$

$$\frac{\partial f_r}{\partial v} = R\Delta t, \tag{4.7}$$

$$\frac{\partial f_q}{\partial q} = \exp((\omega - \bar{\omega})\Delta t), \tag{4.8}$$

$$\frac{\partial f_v}{\partial q} = [Rg]_\times \Delta t, \tag{4.9}$$

$$\frac{\partial f_v}{\partial v} = I - [\omega - \bar{\omega}]_\times \Delta t. \tag{4.10}$$

$$\frac{\partial f_v}{\partial \bar{\omega}} = [v]_\times^T \Delta t. \tag{4.11}$$

With the state transition function, transition covariance and Jacobian defined, the state transition can be implemented both for an EKF, where the Jacobian is required and for an UKF where the linearization is done using sigma point propagation. The resulting prediction is corrected using multiple sensor measurements. The measurement functions including their Jacobians are presented in the following.

4.2.2 Steering Angle

The vehicle measures the current steering wheel angle, which can be converted to the steering angle $z_t^{sa} \in \mathbb{R}$. Using a single track model [Rajamani, 2011], this steering angle can be related to the speed v_t^x in x-direction and turn rate ω_t^z around the z-axis, as well as its bias $\bar{\omega}_t^z$. The relation is defined as

$$\omega_t^z - \bar{\omega}_t^z = \frac{v_t^x}{b} \cdot \tan z_t^{sa}, \tag{4.12}$$

where b denotes the wheel base. Solving this for z_t^{sa} gives the measurement function

$$h^{sa}(x_t) = \tan^{-1} \frac{\omega_t^z - \bar{\omega}_t^z b}{v_t^x}. \tag{4.13}$$

This measurement is assumed to be affected by additive, normally distributed noise with covariance Q_t^{sa} with zero mean. The Jacobian of the measurement function is given by

$$J^{sa} = \frac{\partial h^{sa}}{\partial x} = \begin{bmatrix} 0_{1\times 3} & 0_{1\times 3} & \frac{\partial h^{sa}}{\partial v} & 0_{1\times 3} & \frac{\partial h^{sa}}{\partial \bar{\omega}} \end{bmatrix}, \tag{4.14}$$

with

$$\frac{\partial h^{sa}}{\partial v} = \begin{bmatrix} v_t^x \cdot \frac{b}{(v_t^x)^2 + (\omega_t^z)^2 \cdot b^2} & 0 & 0 \end{bmatrix}, \tag{4.15}$$

$$\frac{\partial h^{sa}}{\partial \bar{\omega}} = \begin{bmatrix} 0 & 0 & (\omega_t - \bar{\omega}_t) \cdot \frac{b}{(v_t^x)^2 + (\omega_t^z)^2 \cdot b^2} \end{bmatrix}. \tag{4.16}$$

4.2.3 Vehicle Speed

The vehicle speed is provided by the vehicle, which measures its own longitudinal velocity $s_t \in \mathbb{R}$ using odometers in the wheels. In addition, two pseudo-measurements are introduced along the y- and z-axis where zero velocity is expected. This models the constraints of a land vehicle, which cannot drive laterally [Groves, 2013, Sect. 15.4.1]. Lateral movement can still happen when the vehicle drifts, however this is not an intended movement and when minimal drifting occurs it is modeled in the measurement noise. The measurement is defined as

$$z_t^v = \begin{bmatrix} s_t & 0 & 0 \end{bmatrix}^T, \tag{4.17}$$

while the measurement function $h^v : \mathcal{X} \to \mathbb{R}^3$ is defined as

$$h^v(x_t) = v_t. \tag{4.18}$$

The measurement is again assumed to be affected by additive, normally distributed noise with covariance Q_t^v and zero mean. The Jacobian of the measurement function is given by

$$J = \begin{bmatrix} 0_{3\times3} & 0_{3\times3} & I_{3\times3} & 0_{3\times3} & 0_{3\times3} \end{bmatrix}. \tag{4.19}$$

4.2.4 Zero Update

When the vehicle stops, additional information about the vehicle state can be inferred from the accelerometer and gyroscope measurements. The accelerometer always measures the gravity, as that force is continually experienced. In the case where the vehicle is not moving, gravity should be measured according to the currently estimated attitude, pointing downward in the navigation frame. However the bias of the accelerometer must be considered. The gyroscope on the other hand should measure only the turn rate bias when no movement is performed. Accordingly, the measurement is

$$z_t^{zu} = \begin{bmatrix} a_t^T & \omega_t^T \end{bmatrix}^T, \tag{4.20}$$

while the measurement function is defined as

$$h^{zu}(x_t) = \begin{bmatrix} \bar{a}_t - q_t^{-1} g q_t \\ \bar{\omega}_t \end{bmatrix}. \tag{4.21}$$

The Jacobian for the zero update is given by

$$J = \frac{\partial h^{zu}}{\partial x} = \begin{bmatrix} 0_{3\times3} & [R^{-1} - g]_\times & 0_{3\times3} & I_{3\times3} & 0_{3\times3} \\ 0_{3\times3} & 0_{3\times3} & 0_{3\times3} & 0_{3\times3} & I_{3\times3} \end{bmatrix}. \tag{4.22}$$

This update is only applicable when the time at which the vehicle stops and starts driving can be reliably detected. In this work the detection is performed using accelerometer measurements as well as velocity measurements. When both indicate a stop, the zero update is performed. In addition, it is only applicable when the global z-axis is known. This is not the case in general when only calculating odometry, as no direct relation exists to the global coordinate system. For the use of this localization algorithm it is therefore required to perform an initialization where the vehicle start on a flat, horizontal surface. When no such surface is available, the zero update is disabled.

4.2.5 Moving Reference Kalman Filter

With these models a Kalman filter can be implemented that estimates odometry from a wide range of sensors. There is however a practical limitation when implementing a Kalman filter for pure odometry. Parts of the state are unobservable and therefore accumulate uncertainty over time. This is the case for both the position estimate r_t as well as the rotation around the z-axis q_t^z. These parts of the state are estimated based on derived measurements that are accumulated to a higher order estimate. No absolute information on these parts is available from measurements, either directly or indirectly. Note that while the rotation around the z-axis is unobservable, the rotation around the x- and y-axis are observable from measuring the gravity vector which is roughly orthogonal to the x-y plane. While growing uncertainty over the unobservable parts is the correct and expected behavior, there are a number of issues that arise, which need to be addressed for localization of an autonomous vehicle.

Firstly, the estimated uncertainty quickly looses any meaningful information for any subsequent algorithm. As the uncertainty grows larger and the vehicle moves further from the origin, the covariance reflects the possible area where the vehicle may have moved and its resulting orientation, considering that all measurements are noisy. This space grows large quickly and using it to make decisions in subsequent algorithms is therefore difficult. Secondly, the growing covariance leads to numerical issues in the filter. As the covariance grows, the cross-covariance grows as well. When the vehicle comes to a stop, parts of the state become easier to observe, such as the IMU biases. In combination with the large uncertainty of the position estimate, a large correction of the position estimate results, which reflects the better IMU bias estimate. This correction in the position is the numerically correct result, however when the position uncertainty is large enough, it produces jumps in the state estimate as shown in Fig. 4.3. The initial motivation to calculate odometry instead of a INS/GNSS was to overcome jumps in the state estimate. As such, even while being the correct solution, measures need to be taken to avoid producing jumps in the state estimate.

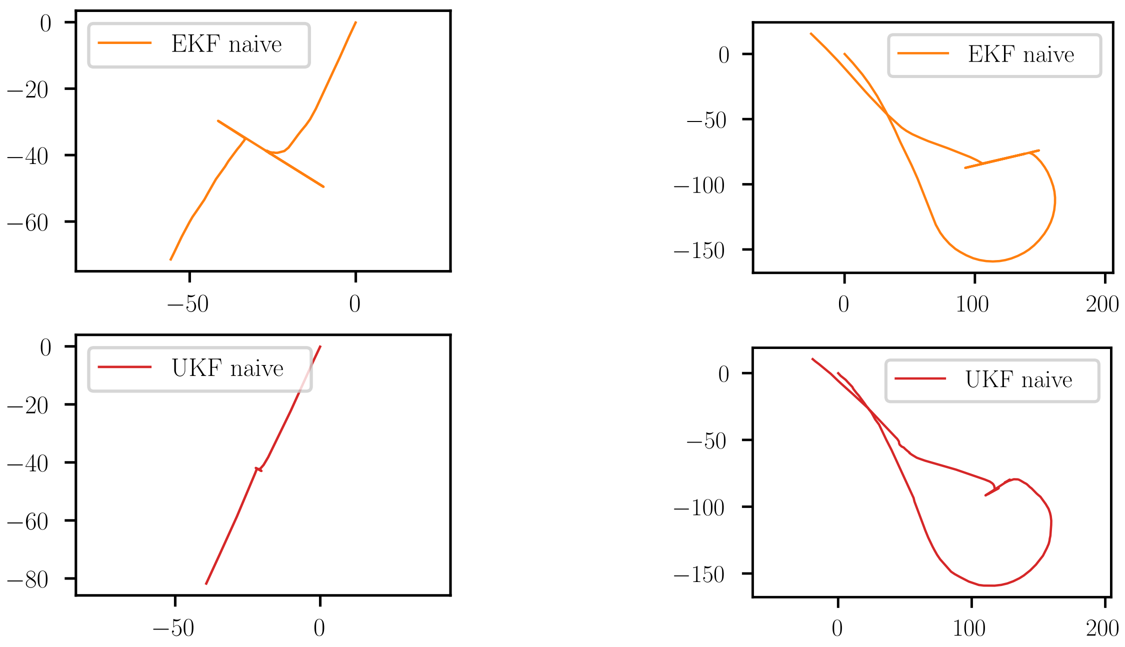


Figure 4.3: Position estimate of the vehicle for a stop after a distance of 746 and 1803 m. (Axes scales are meters, Figure adopted from [Clemens et al., 2020].)

To overcome both the aforementioned issues, the odometry localization can be modified to always estimate the covariance towards a recent reference state. This is similar to a sliding window, where only the data within a certain time frame is used to calculate the odometry. Using a Kalman filter there is however no inherent way to only include recent measurements besides calculating the current state based on all measurements in the sliding window in every time step. Excluding a single measurement from the state estimation is not possible due to the Markov assumption used in both the EKF and the UKF as shown in Sect. 2.1.1. The influence a single measurement had on the estimate is lost as no history is kept besides the accumulated covariance. Calculating the entire estimate again with the measurement excluded is an option, however this is computationally expensive and may not be feasible depending on the dimensionality of the state, the measurement frequency and the sliding window size.

Instead, two Kalman filters can be used in parallel to achieve similar functionality as a sliding window in a Kalman filter. The filters are reset periodically so the state is always calculated only based on a limited amount of measurements. The underlying idea is that the covariance is always calculated with respect to a certain previous state. In the standard case, this reference state is located in the starting pose of the vehicle. Whenever a filter is reset by setting its covariance to zero, the reference state is moved to the current pose of the vehicle. Note that this new reference state is only used for the covariance, while the vehicle pose is still estimated with respect to the odometry frame. Resetting a filter while the vehicle is in motion is however not a safe operation, as the first few measurements may lead to unexpected behavior when the filter is not

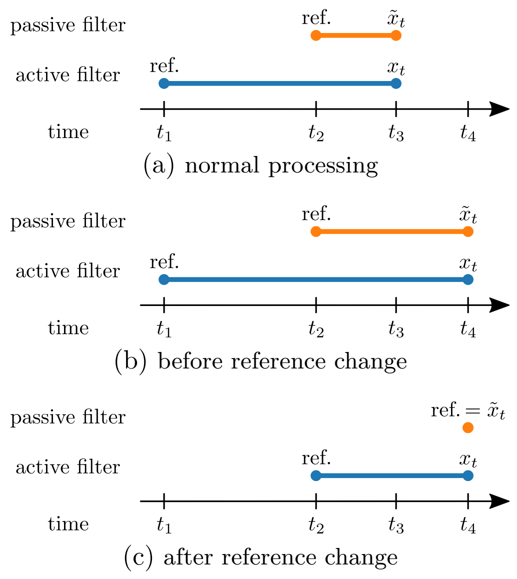


Figure 4.4: Temporal relation between the estimates of both filters and their reference states. The relations between the time points are $t_2 = t_1 + \tau$, $t_4 = t_2 + \tau = t_1 + 2\tau$, and $t_2 < t_3 < t_4$. (Figure adopted from [Clemens et al., 2020].)

stabilized around an estimate yet. A safe reset can however be achieved by using two filters in parallel. One filter, referred to as the active filter, is used as a stable filter that is well initialized and always gives a jump-free estimate. The second filter is used for moving the reference state forward in time and is referred to as the passive filter. This filter resets its covariance periodically in order to move the reference state forward. It is however not used as active output and therefore the instabilities that may occur during reset do not affect the localization performance. To now use the new reference state in the active filter, the covariance that was prepared in the passive filter is periodically transferred to the active filter. This behavior is visualized in Fig. 4.4. As shown, the two filters always have a shifted reference state. As the active filter moves far from its own reference, the reference of the passive filter is transferred to the active filter, while the passive filter is reset, making the current state its new reference state.

In combination, the two filters produce a localization where the reference state is periodically shifted forward in time, while avoiding instabilities around the reset points. By keeping the reference state towards which the covariance is calculated in a limited range, the covariance no longer grows unbounded, fixing the numerical issues mentioned before. In addition, the covariance of the position and rotation around the z-axis now contain meaningful information. They give the accumulated uncertainty since the last reset which can now be used by subsequent probabilistic algorithms. In the following, the process of resetting the passive filter and transferring its uncertainty to the active filter is explained in detail in the context of a $\boxplus$-KF.

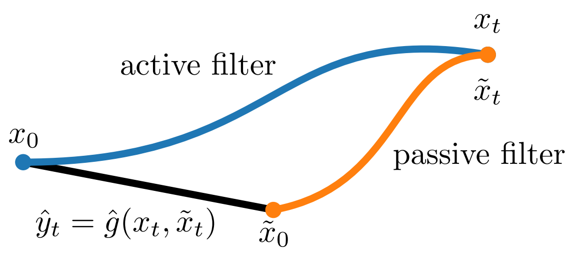


Figure 4.5: Difference in coordinate systems during the covariance transfer. (Figure adopted from [Clemens et al., 2020].)

4.2.6 Uncertainty Transfer

When transferring the uncertainty from the passive to the active filter, special care needs to be taken that the mean of the active filter does not change in the process, as this would again lead to jumps in the state estimate. In addition, since the mean of the active filter and the passive filter may diverge over time, the covariances are expressed in different coordinate systems. Simply copying the covariance of the passive filter to the active filter is therefore not an option, as the uncertainty would potentially affect different axes. Instead, the covariance of the passive filter needs to be transformed to the coordinate system of the active filter. This process is visualized in Fig. 4.5. To project the covariance of the passive filter $\tilde{x}_t$ to the mean x_t of the active filter, $\tilde{x}_t$ is passed through the function

$$g(\tilde{x}_t, \hat{y}_t) = \begin{bmatrix} \hat{r}_t + \hat{q}_t \tilde{r}_t \hat{q}_t^{-1} \\ \hat{q}_t \tilde{q}_t \\ \hat{v}_t + \tilde{v}_t \\ \hat{\bar{a}}_t + \tilde{\bar{a}}_t \\ \hat{\bar{\omega}}_t + \tilde{\bar{\omega}}_t \end{bmatrix}, \tag{4.23}$$

where $\tilde{\cdot}$ denotes elements of $\tilde{x}_t$ and $\hat{\cdot}$ refers to the elements of $\hat{y}_t$. The parameter $\hat{y}_t$ represents the difference between origins of x_t and $\tilde{x}_t$, which is obtained by

$$\hat{y}_t = \hat{g}(x_t, \tilde{x}_t) = \begin{bmatrix} r_t - q_t \tilde{q}_t^{-1} \tilde{r}_t (q_t \tilde{q}_t^{-1})^{-1} \\ q_t \tilde{q}_t^{-1} \\ v_t - \tilde{v}_t \\ \bar{a}_t - \tilde{\bar{a}}_t \\ \bar{\omega}_t - \tilde{\bar{\omega}}_t \end{bmatrix}. \tag{4.24}$$

It is important to note that the parameter $\hat{y}_t$ is fixed in (4.23) and does not change with the parameter $\tilde{x}_t$. By passing $\tilde{x}_t$ through $g(\tilde{x}_t, \hat{y}_t)$, the resulting mean will be equal to that of x_t, while the covariance of $\tilde{x}_t$ is projected to the frame of x_t.

In the UKF this is done using sigma point propagation. The algorithm is shown in

Change Reference UKF

```
Input: $\mu_t$, $\Sigma_t$, $\tilde{\mu}_t$, $\tilde{\Sigma}_t$
    // Transfer uncertainty of passive filter to active filter
 1: $\hat{y}_t \leftarrow \hat{g}(\mu_t, \tilde{\mu}_t)$
 2: $\tilde{\mathcal{S}}_t \leftarrow (\tilde{\mu}_t\ \tilde{\mu}_t \boxplus \sqrt{\tilde{\Sigma}_t}\ \tilde{\mu}_t \boxplus -\sqrt{\tilde{\Sigma}_t})$
 3: $\mathcal{S}^*_t \leftarrow g(\tilde{\mathcal{S}}_t, \hat{y}_t)$
 4: $\mu_t \leftarrow$ MeanOfSigmaPoints($\mathcal{S}^*_t$)
 5: $\Sigma_t \leftarrow \sum_{i=0}^{2M} w^{[i]} (\mathcal{S}^{*[i]}_t \boxminus \mu_t)(\mathcal{S}^{*[i]}_t \boxminus \mu_t)^\top$
    // Reset uncertainty of passive filter
 6: $\hat{r}_t \leftarrow r_t$, $\hat{q}^z_t \leftarrow q^z_t$
 7: $\mathcal{S}_t \leftarrow (\mu_t\ \mu_t \boxplus \sqrt{\Sigma_t}\ \mu_t \boxplus -\sqrt{\Sigma_t})$
 8: $\tilde{\mathcal{S}}^*_t \leftarrow \tilde{g}(\mathcal{S}_t, \hat{r}_t, \hat{q}^z_t)$
 9: $\tilde{\mu}_t \leftarrow$ MeanOfSigmaPoints($\tilde{\mathcal{S}}^*_t$)
10: $\tilde{\Sigma}_t \leftarrow \sum_{i=0}^{2M} w^{[i]} (\tilde{\mathcal{S}}^{*[i]}_t \boxminus \tilde{\mu}_t)(\tilde{\mathcal{S}}^{*[i]}_t \boxminus \tilde{\mu}_t)^\top$
11: return $\mu_t$, $\Sigma_t$, $\tilde{\mu}_t$, $\tilde{\Sigma}_t$
```

Figure 4.6: Algorithm for changing the reference in a UKF. $\mathcal{S}$ denotes the set of sigma points, w is the set of corresponding weights, M denotes the number of degrees of freedom of the state space (here $M = 15$), and MeanOfSigmaPoints is implemented according to [Hertzberg et al., 2013, Tab. 3]. (Figure adopted from [Clemens et al., 2020])

Change Reference EKF

```
Input: $\mu_t$, $\Sigma_t$, $\tilde{\mu}_t$, $\tilde{\Sigma}_t$
    // Transfer uncertainty of passive filter to active filter
 1: $\hat{y}_t \leftarrow \hat{g}(\mu_t, \tilde{\mu}_t)$
 2: $\mu_t \leftarrow g(\tilde{\mu}_t, \hat{y}_t)$
 3: $\Sigma_t \leftarrow G_t \tilde{\Sigma}_t G_t^\top$
    // Reset uncertainty of passive filter
 4: $\hat{r}_t \leftarrow r_t$, $\hat{q}^z_t \leftarrow q^z_t$
 5: $\tilde{\mu}_t \leftarrow \tilde{g}(\mu_t, \hat{r}_t, \hat{q}^z_t)$
 6: $\tilde{\Sigma}_t \leftarrow \tilde{G}_t \Sigma_t \tilde{G}_t^\top$
 7: return $\mu_t$, $\Sigma_t$, $\tilde{\mu}_t$, $\tilde{\Sigma}_t$
```

Figure 4.7: Algorithm for changing the reference in an EKF. (Figure adopted from [Clemens et al., 2020])

line 1-5 of Fig. 4.6. For the EKF the transfer is done using linearization of the function $g(\tilde{x}_t, \hat{y}_t)$ using its Jacobian $G_t = \frac{\partial g}{\partial \tilde{x}_t}$. This is shown in Line 1-3 of Fig. 4.7. The Jacobian is given by

$$G_t = \frac{\partial g}{\partial \tilde{x}_t} = \begin{bmatrix} \hat{R} & 0_{3\times3} & 0_{6\times9} \\ 0_{3\times3} & \hat{R} & \\ 0_{9\times6} & & I_{9\times9} \end{bmatrix}, \tag{4.25}$$

where $\hat{R}$ is the rotation matrix constructed from the pose of $\hat{y}_t$.

4.2.7 Passive Filter Reset

After transferring the covariance from the passive filter to the active filter, the passive filter needs to be reset to move the reference state forward. However, not the entire covariance needs to be reset, since large parts of the state are observable and therefore do not suffer from unbounded growth. Instead, only the unobservable parts of the covariance need to be reset, namely the position and the rotation around the z-axis. By leaving the observable parts of the covariance untouched, the filter keeps a better understanding of relationships between parts of the state, leading to more accurate state updates. The reset is performed by passing x_t through the function

$$\tilde{g}(x_t, \hat{r}_t, \hat{q}_t^z) = \begin{bmatrix} \hat{r}_t \\ q_t^x q_t^y \hat{q}_t^z \\ v_t \\ \bar{a}_t \\ \bar{\omega}_t \end{bmatrix}. \tag{4.26}$$

Note that $\hat{r}_t, \hat{q}_t^z$ are given as input and therefore remain fixed, resulting in zero covariance in those parts of the state. This effectively resets the covariance, moving the reference state to the current state. In the UKF this is again done by sigma point propagation. The corresponding algorithm is defined in Fig. 4.6, line 6-10. In the EKF the covariance is updated by linearization using the Jacobian $\tilde{G}_t = \frac{\partial \tilde{g}}{\partial x_t}$. The Jacobian is defined as

$$\tilde{G}_t = \frac{\partial \tilde{g}}{\partial x_t} = \begin{bmatrix} 0_{3\times15} & & \\ 0_{3\times3} & \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} & 0_{3\times9} \\ 0_{6\times9} & & I_{9\times9} \end{bmatrix}. \tag{4.27}$$

The corresponding algorithm is given in line 4-6 in Fig. 4.7.

Table 4.1: Datasets used for the evaluation. (Table partially adopted from [Clemens et al., 2020].)

Name	**Duration**	**Distance**	**Med. Speed**	**Max Speed**	**Trajectory**
Parking Lot	5:17	1.57 km	17.9 km/h	37.5 km/h	Fig. 4.8a
Braunschweig	27:21	15.02 km	39.2 km/h	66.3 km/h	Fig. 4.8b
Bremen	36:39	20.36 km	36.2 km/h	79.9 km/h	Fig. 4.8c
BS to HB	1:29:19	126.53 km	84.4 km/h	172.7 km/h	Fig. 4.8d

4.2.8 Evaluation

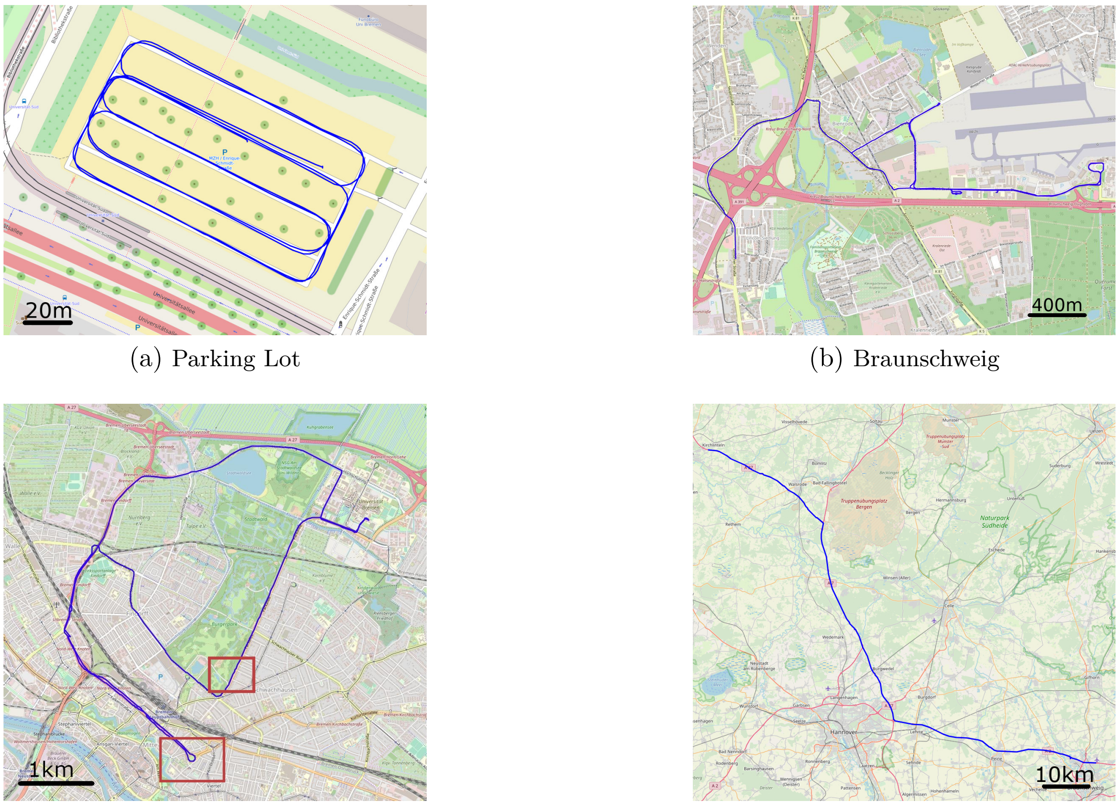
The evaluation of the odometry localization was performed on multiple datasets which were recorded using the VW Passat GTE described in Chap. 3. For ground truth the positioning obtained by the GNSS receiver is used, in this case not corrected using RTK as no correction data is available in the datasets. While the GNSS may contain drift and errors itself, the influence is minimal while the GNSS reception is good and thus these errors are tolerable for an evaluation of the odometry. The datasets were additionally checked for errors in the GNSS estimate, and areas with noticeable errors in the GNSS positioning were excluded from the evaluation. The datasets are listed in Tab. 4.1, while the ground truth trajectories are shown in Fig. 4.8. The datasets were selected to reflect different driving situations. The Parking Lot features low driving speeds with many sharp turns. The Braunschweig dataset contains areas that resemble driving in the countryside, where medium speeds are reached. The Bremen dataset represents an inner city scenario again with medium speeds. Finally, the BS to HB dataset contains a highway drive from Braunschweig to Bremen at high speeds. The evaluation compares five approaches for odometry estimation. As baseline a single-track model is integrated using

$$t_{t+1} = r_t + q_t \begin{bmatrix} s_t & 0 & 0 \end{bmatrix}^T q_t^{-1} \cdot \Delta t, \tag{4.28}$$

$$q_{t+1} = q_t \exp\left(\begin{bmatrix} 0 & 0 & \frac{s_t}{b} \cdot \tan z_t^{sa} \end{bmatrix}^T \cdot \frac{\Delta t}{2} \right). \tag{4.29}$$

To evaluate the difference a jump-free localization makes in the presented scenarios, both an EKF and a UKF are evaluated using a naive implementation (denoted EKF/UKF naive), as well as the modified implementation with a moving reference state (denoted EKF/UKF mov ref). Note, that besides the filter switching, both versions of the EKF and the UKF use the same measurement and state transition models. For the moving reference implementation, the time τ between filter switches is set to 10s to allow the filter to be properly initialized. In this time, the covariance will be sufficiently built up to allow corrections to be applied, however the covariance will not grow large enough that jumps are produced.

In order to evaluate odometry, special metrics are required to take into account that only the localization performance over shorter periods of time are relevant. Looking



(a) Parking Lot

(b) Braunschweig

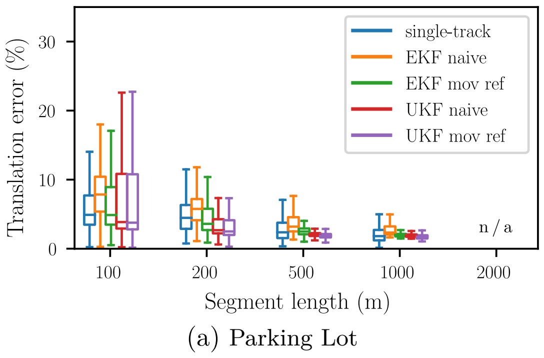
(c) Bremen

(d) BS to HB

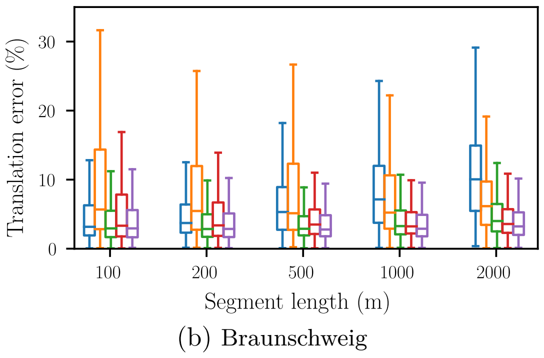
Figure 4.8: Ground-truth trajectories of the evaluation datasets. The red squares in (c) indicate the areas of focus in the evaluation. (Maps generated using UMap based on OpenStreetMaps data. Figure adopted from [Clemens et al., 2020].)

at the absolute error in the estimated pose is meaningless, as a small error in heading in the start of the estimation would lead to very large errors later on. Even if the relative localization performance after the initial error would be superior, this would not be reflected in the evaluation result. Instead, the performance is evaluated over short trajectories as proposed in [Geiger et al., 2012], with the evaluation script taken from [Zhang and Scaramuzza, 2018]. The segment lengths were chosen between 100m and 2000m to evaluate performance both over short periods of time and after significant distances. The trajectories are aligned with the ground truth using their poses at the start of the trajectory and the error in the last pose of the segment is summed up over all trajectories to obtain an error value. This is shown in Fig. 4.9.

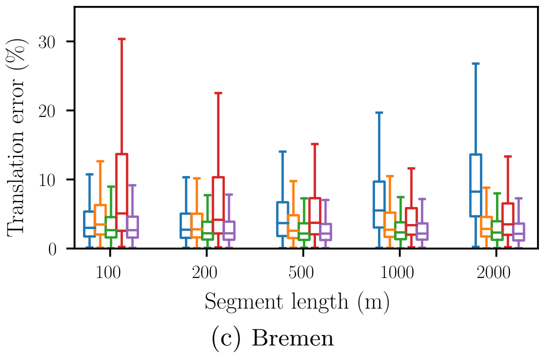
In the first dataset containing a drive on a parking lot (shown in Fig. 4.9a), the moving reference implementations consistently achieve similar or better localization results when compared to their naive counterparts. However, the differences in the results are relatively minor, and even the single-track model integration performs well on this dataset, sometimes even outperforming other approaches. This is likely due to a strong



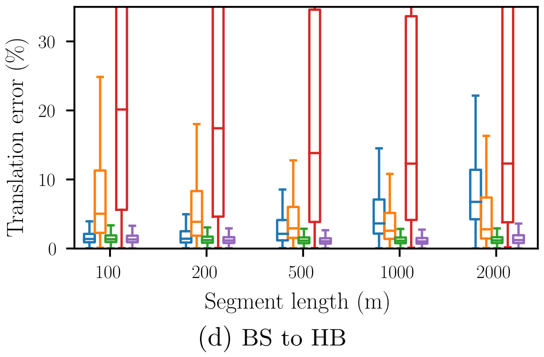
(a) Parking Lot



(b) Braunschweig



(c) Bremen



(d) BS to HB

Figure 4.9: Average translation error of a pose relative to the previous pose at a specific distance. (See [Geiger et al., 2012; Zhang and Scaramuzza, 2018] for further details on the error calculation. Figure adopted from [Clemens et al., 2020].)

non-linearity in this datasets with many sharp turns, in addition to the fact that the total length is only 1.57km. Since the velocity and steering obtained by the vehicle are very precise over shorter periods with only little drift, the single track model works well here.

On the second dataset containing a drive through Braunschweig (see Fig. 4.9b), the differences between the approaches become more apparent. The moving reference implementations noticeably outperform their naive counterparts, and especially for longer segments the performance of the single track model deteriorates. There are especially some very large error values for the naive EKF, indicating that the filter starts to become unstable. This is due to the length of the drive, where the covariance grows large enough over time to allow large corrections.

A similar picture can be seen in the Bremen dataset (see Fig. 4.9c). Again the moving reference implementations outperform the naive versions, with the naive implementations showing signs of instabilities as well. This dataset is even longer than the Braunschweig dataset and contains many stops, which generally lead to more instabilities. Therefore a worse performance of the naive implementations is to be expected.

Finally, in the highway dataset from Braunschweig to Bremen, the differences between

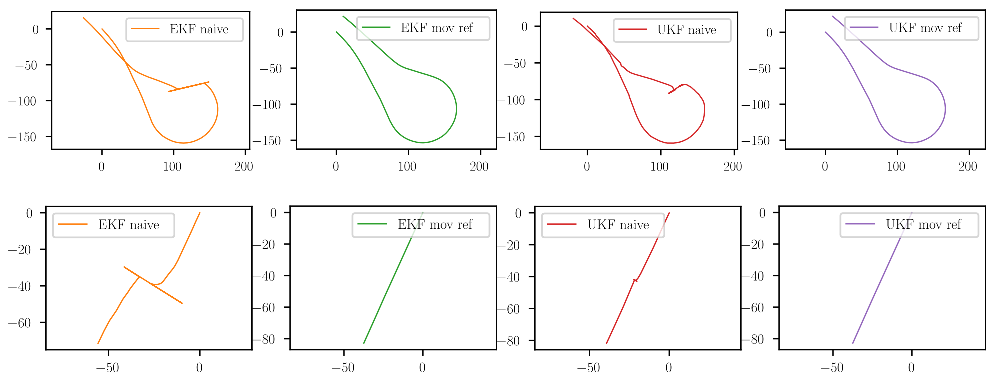


Figure 4.10: Example for a stop of the vehicle after a distance of 746 m since the last stop (top) and after 1803 m (bottom). (Axes scales are meters. Figure adopted from [Clemens et al., 2020].)

the moving reference implementations and the naive ones becomes drastic, as the naive filters become fully unstable over such a long drive of 126km. Here it becomes very apparent why a special handling of the covariance is needed when estimating odometry over such long periods of time.

This evaluation clearly shows that the proposed moving reference methods outperform their naive counterparts, however the issue that it was meant to solve was the jumping state estimate that may occur in the naive case. A closer look at the jumps that occur showed that they are caused by the vehicle stopping. When the vehicle is stopped, parts of the state such as the IMU biases can be observed better, allowing the filter to correct them. Since they are connected in the filter via the cross covariance, this leads to corrections in the position estimate, that become large as the uncertainty over the position grows. To evaluate the performance of the moving reference implementation in these stops we take a closer look at some stops that occur in the Bremen dataset, marked as a red square in Fig. 4.8c. The corresponding trajectories are shown for the evaluated algorithms in Fig. 4.10. In this figure, two stops are evaluated, one after driving 746 m since the last stop, which occurred after driving a total of 10,927 m, and one after 1803 m since the last stop, with a total distance of 5,082 m. As can be seen, the moving reference implementation no longer produces jumps in these scenarios, and careful evaluation of the entire trajectory shows that the localization is now fully jump-free even for the long highway dataset.

In addition to the relative localization, an evaluation on the absolute trajectory error is performed for completeness. In order to obtain this error, the trajectories are aligned by an $SE(2)$ transformation using the method proposed in [Zhang and Scaramuzza, 2018]. The results are shown in Tab. 4.2. The absolute errors show a similar result as the relative evaluation, with the single-track model drifting heavily when the trajectory is longer. The moving reference implementation shows improvements in the absolute error for every dataset besides the Bremen evaluation, where the EKF naive implementation performs slightly better. Overall the UKF moving reference implementation produces the lowest errors for all but the highway dataset, indicating that the better approximation

Table 4.2: Absolute root mean square (RMS) translation error. Table adopted from [Clemens et al., 2020].

Algorithm	Park. Lot	Braunschw.	Bremen	BS to HB
single-track	12.2 m	316.8 m	1,997.5 m	21,810.3 m
EKF naive	15.8 m	277.1 m	188.8 m	2,827.9 m
EKF mov ref	9.5 m	165.2 m	195.0 m	2,691.4 m
UKF naive	8.2 m	160.3 m	280.5 m	7,645.9 m
UKF mov ref	7.3 m	106.9 m	174.3 m	3,973.1 m

Table 4.3: Runtime of the algorithms. Table adopted from [Clemens et al., 2020].

Algorithm	Park. Lot	Braunschw.	Bremen	BS to HB
single-track	1.24s	5.45s	6.59s	18.16s
EKF naive	3.12s	16.24s	20.15s	52.65s
EKF mov ref	4.83s	25.61s	31.86s	85.26s
UKF naive	7.49s	42.53s	50.37s	135.09s
UKF mov ref	13.09s	74.07s	90.36s	207.31s

of non-linearity has some benefits in urban driving.

Lastly, a runtime evaluation is performed. All algorithms are run on an Intel Xeon E5-2699 CPU (2.20GHz, single-threaded). The results can be seen in Tab. 4.3. As expected, the single track model performs best in terms of computation time. The moving reference implementations perform all computations for two filters, so a factor of 2 is expected for the runtime. There is however some static overhead for both the naive and the moving reference implementations, reducing this factor slightly. The UKF overall requires more computations than the EKF due to the sigma point propagation, resulting in a significantly longer runtime. Overall all algorithms are still real-time capable, with the UKF moving reference needing 207s for a dataset that is 1h30m long.

4.2.9 Conclusion

In this section a localization system for estimating odometry was presented. It uses low to mid cost sensors as shown in Chap. 3 in order to perform jump-free localization. For this, a $\boxplus$-Kalman filter was used, where the equations were given for both an EKF and a UKF. To avoid issues caused by unbounded growth of the covariance of the position as well as the rotation around the z-axis, a moving reference Kalman filter was proposed. In this approach, two filters run in parallel, with the active filter generating the localization output, while the passive filter is periodically reset. This is used to move the reference state forward, effectively bounding the covariance growth. The covariance of the passive filter is then periodically transferred to the active filter to move its reference state forward. An evaluation was performed on four different datasets reflecting multiple common scenarios in autonomous driving. It was shown, that the moving reference implementations not only performs jump-free odometry estimation but also perform better in terms of relative and absolute localization performance. Finally, all algorithms were shown to be real-time capable. They were implemented on the research vehicle shown in Chap. 3 and the resulting localization is actively used as part of the autonomous driving stack.

4.3 Global Filter

While the Odometry localization achieves high levels of accuracy on a local scale and is thus used for all subsequent algorithms that require precise localization over a short time frame, some algorithms need global information to function. This is especially the case for routing algorithms, where a valid route depends on the current location in the world. For global localization, in this work an INS/GNSS fusion is performed using a $\boxplus$-Kalman filter, as presented in [Höffmann et al., 2022]. This allows for probabilistic fusion of the inertial measurements used in Sect. 4.2 with the global GNSS data. The GNSS is aided by RTK, providing highly precise measurements with centimeter accuracy. This localization runs fully decoupled from the odometry localization, however it does receive the same data as input and uses many of the same models. There are however some key differences, which will be discussed in the following.

First, the state space needs to be modified. Instead of estimating a position in Cartesian coordinates, a global position $p_t = \begin{bmatrix} \varphi_t & \lambda_t & h_t \end{bmatrix}^T$ is estimated, with φ_t and λ_t being the latitude and longitude, while h_t represents the height above the WGS84 ellipsoid which approximates the shape of the earth with respect to the sea level, excluding variables such as mountains. However, with most computations on the state being performed in vector space, using the polar coordinates as a representation is not ideal as it requires frequent conversions. Instead, changes to the position are accumulated in the vector space and periodically converted to polar coordinates. This is done by estimating the position $r_t = \begin{bmatrix} r_t^e & r_t^n & r_t^u \end{bmatrix}$ in the east-north-up (ENU) frame, with the coordinate system centered at a past position p_{t-k}. This reference position is periodically moved to the current position in order to avoid conversion inaccuracies that arise when the reference position is far away. With the position now expressed in vector space, a conversion is required to obtain the global position. This is however performed less frequently and as such does not add much overhead. The conversion is defined as [Wendel, 2007, Sect. 8.2.3]

$$p_t = p_{t-k} + \begin{bmatrix} \frac{r_t^n}{R_{t-k}^n + h_{t-k}} \\ \frac{r_t^e}{(R_{t-k}^e + h_{t-k} \cdot \cos\varphi_{t-k})} \\ r_t^u \end{bmatrix}, \tag{4.30}$$

where R^n_{t-k} and R^e_{t-k} denote the earth's north-south and east-west curvature radius respectively, each at the reference position in time $t-k$.

The rotation of the vehicle is again represented using a quaternion q_t, however this no longer represents the rotation in the local odometry frame. Instead, the orientation is estimated in the ENU frame, centered in the position of the vehicle. The state space is therefore defined as

$$\mathcal{S} = \mathbb{R}^3 \times SO(3) \times \mathbb{R}^3 \times \mathbb{R}^3 \times \mathbb{R}^3. \tag{4.31}$$

The state x_t at time t is defined accordingly as

$$x_t = \begin{bmatrix} r_t \\ q_t \\ v_t \\ \bar{a}_t \\ \bar{\omega}_t \end{bmatrix}, \tag{4.32}$$

Note that this state is identical to the odometry state, with the only difference being the coordinate systems in which the position and rotation are expressed. Therefore the motion model from the odometry filter as well as the measurement models remain the same.

To obtain a global estimate, the measurements of the GNSS receiver need to be fused into the state. The measured values are the latitude φ^g_t, longitude λ^g_t and height h^g_t of the antenna. To update the state this measurement needs to be converted to the ENU frame located in p_{t-k}. This is done using [Wendel, 2007, Chap. 8.2.2]

$$z^g_t = \begin{bmatrix} (\lambda^g_t - \lambda_{t-k}) \cdot (R^e_{t-k} + h_{t-k}) \cdot \cos\varphi_{t-k} \\ (\varphi^g_t - \varphi_{t-k}) \cdot (R^n_{t-k} + h_{t-k}) \\ (h^g_t - h_{t-k}) \end{bmatrix}. \tag{4.33}$$

To account for the offset of the antenna from the vehicle frame origin, the body-fixed position r^{a1} of the antenna is transformed using the estimated orientation of the vehicle in the ENU frame. This gives the measurement function

$$h^g(x_t) = r_t + q_t r^{a1} q_t^{-1}. \tag{4.34}$$

The measurement is assumed to be affected by additive, normally distributed noise with covariance Q^g_t and zero mean. The Jacobian of the measurement function is defined as

$$J^g = \frac{\partial h^g}{\partial x} = \begin{bmatrix} I_{3\times3} & -\left(R_t \left[r^{a1}\right]_\times\right) & 0_{3\times9} \end{bmatrix}. \tag{4.35}$$

In addition to the global position, information is contained in the GNSS measurements about the orientation of the vehicle in the world. While this is not possible with standard GNSS unless it is inferred from multiple measurements with the assumption of moving forward, this vehicle is equipped with multiple GNSS antennas. By measuring the position z_t^r of the second antenna in the ENU frame centered in the position of the first antenna, the orientation in the world can be estimated. For this, the measurement is given by

$$\hat{z}_t^r = \frac{z_t^r}{\|z_t^r\|}, \tag{4.36}$$

where a unit vector is used since the actual distance is not relevant. The direction of the vector gives the orientation of the vehicle. The corresponding measurement function is defined as

$$h^r(x_t) = q_t \frac{r^{a1} - r^{a2}}{\|r^{a1} - r^{a2}\|} q_t^{-1}, \tag{4.37}$$

with r^{a2} describing the position of the second antenna in the body-fixed frame of the vehicle. The measurement is assumed to be affected by additive, normally distributed noise with covariance Q_t^r and zero mean. The corresponding Jacobian is given by

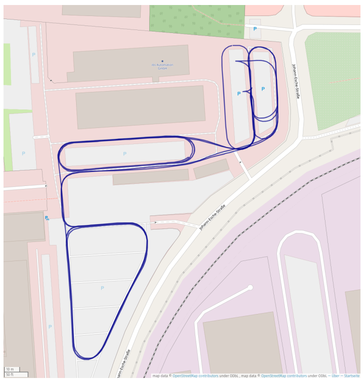
$$J^r = \frac{\partial h^r}{\partial x} = \begin{bmatrix} 0_{3\times3} & -\left(R_t \left[\frac{r^{a1}-r^{a2}}{\|r^{a1}-r^{a2}\|}\right]_\times\right) & 0_{3\times9} \end{bmatrix}. \tag{4.38}$$

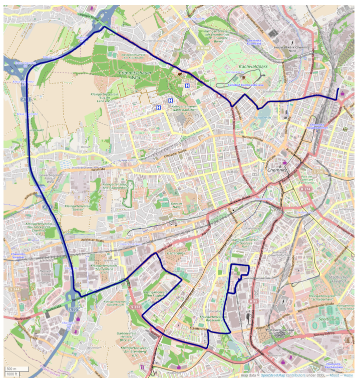
In addition to being able to update the global orientation directly, the dual receiver setup enables an accelerated initialization process. In the standard implementation with one GNSS receiver, initially the orientation is unknown. As the vehicle starts to move, and with the assumption that it moves forward or by obtaining the driving direction from the wheel sensors, the orientation can be inferred. Consequently the initialization process always requires a short drive to initialize the heading, which can be cumbersome. With the dual receiver setup, the orientation is measured directly on system startup and the filter is initialized accordingly. This leads to a significant improvement in usability of the system.

4.3.1 Evaluation

Evaluating the performance of the global filter is a difficult task, as the result has to be compared to a ground truth that is at least an order of magnitude more accurate than the evaluated algorithm. As the RTK system provides centimeter accuracy, the requirement for the ground truth system is very high. In fact, no ground truth system exists that does not require additional costly hardware, that can provide ground truth over stretches of multiple kilometers. In the work of [Höffmann et al., 2022] the global localization system is evaluated on a small scale using such a reference system, however this evaluation was performed on an industrial lawn mower at low speeds with a limited range. In this work, a qualitative evaluation is performed on multiple datasets containing various difficult scenarios where the GNSS quality is very low or where no GNSS is received at all. This is meant to complement the previously performed quantitative evaluation on the mower.

The evaluation data is similar to the setup presented in Sect. 4.2, with the main difference being the inclusion of the two receiver RTK system, which provides centimeter


(a) Parking Lot


(b) Highway

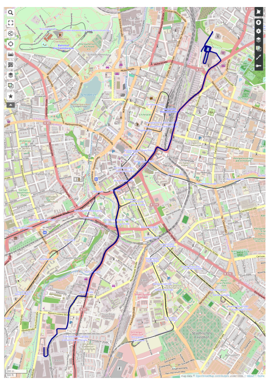

(c) Garage

Figure 4.11: Trajectories estimated by the global localization. (Maps generated using UMap based on OpenStreetMaps data.)

Table 4.4: Overview of the Datasets.

Name	Duration	Distance	Max. Speed	Trajectory
Parking Lot	11:11	3.04 km	49.2 km/h	Fig. 4.11a
Highway	42:28	25.8 km	101.4 km/h	Fig. 4.11b
Garage	21:27	8.0 km	60.9 km/h	Fig. 4.11c

accurate positioning as well as heading for the vehicle. The RTK is calculated on the receivers, and fused into the localization as described above. In the evaluation GNSS is only used in the filter when RTK is available. When no RTK solution is found, the measurements are discarded to avoid errors introduced by faulty measurements and the filter relies on dead-reckoning without global correction. In this evaluation, the u-blox F9P receivers were used.

The evaluation was performed on three datasets, which are listed in Tab. 4.4. The first dataset is a drive on a parking lot with many sharp turns. The speed is comparably low and the GNSS signal strong. The second dataset contains a drive through Chemnitz on various urban streets as well as a long stretch of highway. This dataset contains higher speeds as well as multiple sections with poor GNSS signal. In addition, it contains drives under bridges, where no GNSS data is available. The last dataset was recorded in large parts in a multi-level parking garage with very poor GNSS reception. The car was thereafter driven over urban roads for multiple kilometers at medium speeds.

In the following, multiple difficult scenarios are observed more closely to evaluate the performance of the global localization in such cases. Firstly, looking at the parking lot dataset in Fig. 4.11a, no significant jumps can be detected in the estimate, which shows a smooth trajectory. In the case of the global filter this is not guaranteed, however especially with good GNSS signal this is the expected result. As comparison, the GNSS

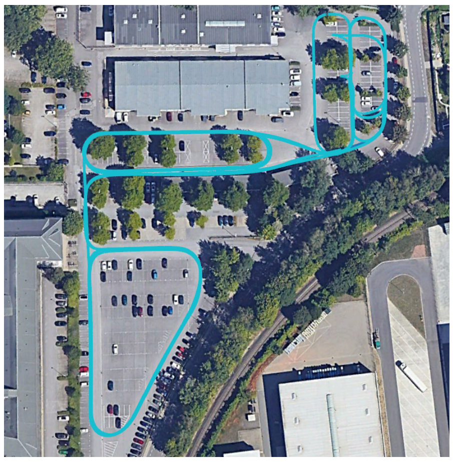
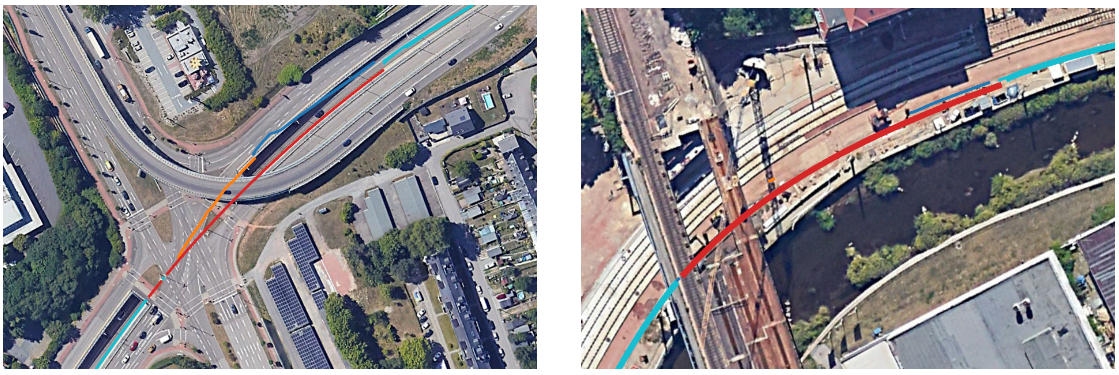


Figure 4.12: GNSS signal strengths of the Parking Lot scenario. Light blue trajectories were corrected using RTK, while all other colors have impaired accuracy. (Maps generated using Google Earth Pro.)



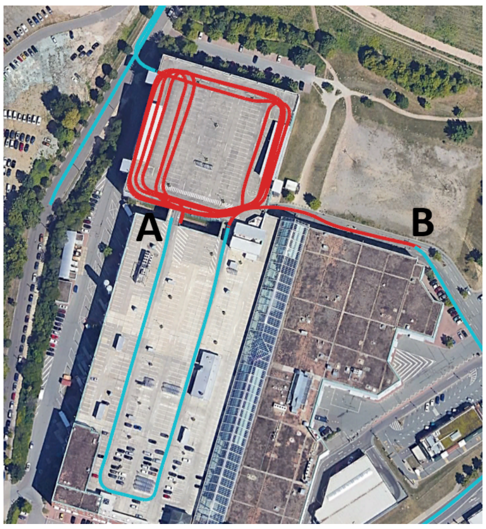
(a) Bridge 1

(b) Bridge 2

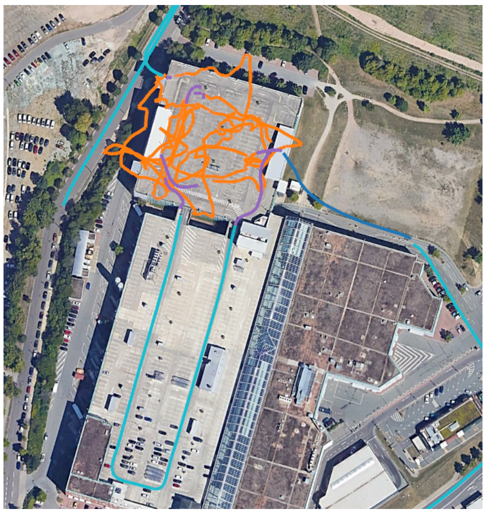
Figure 4.13: Showcase of two problematic areas where no GNSS signal is available. The trajectory in red shows the global state estimate where no RTK was available. Light blue trajectories are aided by RTK, while orange and dark blue show the incoming GNSS position estimate with reduced GNSS quality, which are discarded. (Maps generated using Google Earth Pro.)

signal strength is plotted in Fig. 4.12. As can be seen, the whole scenario is performed with RTK, with the exception of one stretch of about four meters. Over the duration of 11 minutes, no visible drift or deviations from the repeating path are visible.

The highway dataset is considerably longer and therefore contains multiple parts where poor or no GNSS signal is present. Two of those scenarios are shown in Fig. 4.13. As can be seen here, while the GNSS gives erroneous measurements and shows strong drifts and jumps, the global filter detects this and no longer uses these measurements for state updates. Instead, the filter performs dead reckoning. This is expected to contain drift, which is visible in both scenarios, where at the end of the bridge the



(a) State Estimate



(b) GNSS position estimate

Figure 4.14: Showcase of two problematic areas where no GNSS signal is available. (Maps generated using Google Earth Pro.)

estimate jumps as soon as RTK is received again. This is however expected behavior and does not influence the safety of the autonomous system as it is not used for vehicle control. The last dataset contains a long drive through a parking lot that stretches over multiple floors. The resulting state estimate for this part of the drive can be seen in Fig. 4.14a. The corresponding GNSS position estimates given by the GNSS receiver are shown in Fig. 4.14b. This evaluation clearly demonstrates the difficulty of localizing in this scenario, with the GNSS receiver giving strongly erroneous measurement. Again, these measurements are excluded in the global filter and dead reckoning is performed. However, even with no GNSS correction for an extended period of time, the estimate only drifts by a few meters, which can be seen upon exiting the garage onto a parking deck in point A and exiting the garage onto the street in point B. The rest of the drive from point B was performed to show the stability of the filter even after a prolonged time without any GNSS corrections. As shown in Fig. 4.11 the drive was completed successfully and closer inspection of the trajectory showed no unexpected behavior.

In addition to this evaluation a test drive was performed to test the behavior of the initialization when the vehicle is already moving. For this the vehicle was accelerated to 20 km/h before the localization system was started. The result can be seen in Fig. 4.15. As can be seen the trajectory is smooth immediately after system startup and the filter is stable even without any initialization procedure.

4.3.2 Conclusion

For obtaining a global state estimate the $\boxplus$-Kalman filter presented in Sect. 4.2 was modified to include measurements from two RTK-corrected GNSS receivers. The result is

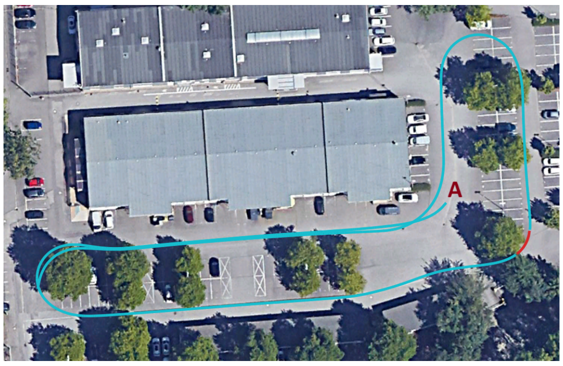


Figure 4.15: Evaluation of the global filter initialization. Starting position marked with a red A.

a robust global localization system with a very simple initialization procedure, which even allows on the fly initialization while moving. The performance of the localization was evaluated qualitatively on three datasets which contained a number of difficult sections that were investigated more closely. In all scenarios the localization remained robust, however there was some expected drift when no GNSS updates were performed due to poor signal strength or when no signal was received. As with the odometry localization, the resulting global localization system was implemented on the research vehicle and is used in real time as part of the autonomous driving stack.

4.4 Transforming Global Information to the Odometry Frame

Most algorithms in autonomous driving rely on the odometry estimate for localization as this is expressed in cartesian coordinates and does not contain jumps. However there are tasks where global information is required, which then needs to be transformed to the odometry frame to make it usable for the algorithm. An example of this are lane boundaries defined in a previously measured high definition map. These boundaries are used for vehicle control as additional information where the vehicle can drive, however the vehicle control is performed in the odometry frame to make use of the jump-free localization. Thus the boundaries expressed in the global frame must be transformed to the odometry frame. This transformation is however not fixed, as the odometry drifts over times while the global estimate does not. Thus the transformation must be calculated with respect to the most recent localization estimates from both filters. Additionally the global information is expressed in the WGS84 frame and must be converted to a cartesian frame to make use of it.

To transform a point $\bar{p}^G = \begin{bmatrix} \bar{\varphi} & \bar{\lambda} & \bar{h} \end{bmatrix}^T$ from the global frame G to the odometry frame N the assumption is used that the required global information is close to the vehicle and can thus be expressed on the vector space around the position of the vehicle without considering the curvature of the earth. The transformation is performed by

$$T_t^{V \to N} R_t^{G \to V} - \left(p_t^G \boxminus \bar{p}^G \right), \tag{4.39}$$

where $T_t^{V \to N}$ is given by the odometry localization, $R_t^{G \to V}$ is given by the global localization and p_t^G is the current global position estimate, also given by the global localization. The $\boxminus$-operator is defined as

$$a \boxminus b = \begin{bmatrix} (a_\lambda - b_\lambda) \cdot (c1 + b_h) \cdot \cos(b_\varphi) \\ (a_\varphi - b_\varphi) \cdot (c2 + b_h) \\ a_h - b_h \end{bmatrix}, \tag{4.40}$$

where $c1$ is the east-west curvature of the WGS84 ellipsoid in the position of a and $c2$ is the corresponding north-south curvature [Wendel, 2007, Chap. 8]. They are calculated by

$$c1 = \frac{1}{a} \cdot \sqrt{1 - e^2 \cdot \sin(a_\varphi)^2}, \tag{4.41}$$

$$c2 = \frac{a \cdot (1 - e^2)}{(1 - e^2 \cdot \sin(a_\varphi)^2)^{\frac{3}{2}}}, \tag{4.42}$$

with $a = 6378137$ and $e = 0.0818191908426$ [Wendel, 2007, Chap. 3.1].

4.5 Visual/LiDAR Inertial Odometry

To aid the localization visual odometry is often utilized in current research. The perceived movement of the environment in the sensor data is used to determine how the vehicle moved between the two measurements. By creating this connection between past and current measurements the odometry localization no longer relies solely on relative measurements and instead directly measures its movement over a short period of time. Similarly this approach can aid in global localization in GNSS-denied areas to reduce sensor drift and avoid larger jumps in the state estimate. In previous research performed using a Camera, IMU, speed and steering measurements, [Serov et al., 2022] found, that no notable improvements over the odometry localization presented in Sect. 4.2 could be gained when using a medium-cost IMU. However an increase in accuracy could be achieved when less precise IMU measurements were used.

4.5.1 Truncated Signed Distance Functions (TSDF)

In this work a similar evaluation is performed for LiDAR odometry. For this evaluation truncated signed distance functions (TSDF) [Curless and Levoy, 1996] are utilized. Similar to the work presented in [Daun et al., 2019] the TSDF is utilized to build a grid map of the environment. In this grid each cell contains the distance to the closest measured point, with cells in front of the measured obstacle having a positive distance and cells behind the measurement having a negative distance. The distance is truncated with all cells outside of the truncated area containing the maximum defined distance. This is visualized in Fig. 4.16. The TSDF map is built up over time and for each new

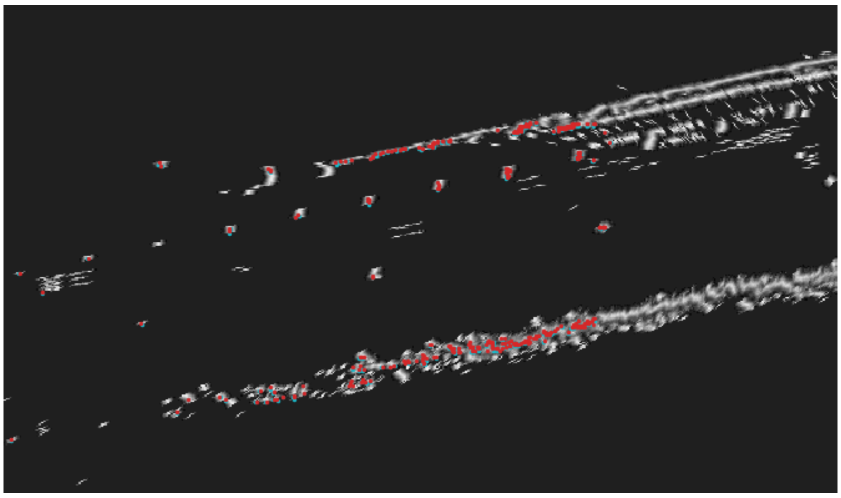


Figure 4.16: Example of a TSDF Map. In greyscale the (absolute) truncated distance to previous measurements is shown, while the current measurement is shown in blue and the optimized scan is shown in red. Note that the red points overlap with the blue measurement, as the optimization resulted only in a minimal change in this example.

scan a matching step is performed, where the current scan is aligned with the TSDF map to minimize the total distance from previous measurements. The resulting pose is then fused using the Kalman filter to obtain a new state estimate, and the TSDF map is updated using the LiDAR scan. This approach is explained in detail in the following and subsequently evaluated for the use on the research vehicle.

To build a TSDF map of the environment the mapping system is utilized, which is explained in detail in Chap. 5. This is generally used to build an occupancy map of the environment to determine drivable areas, however it is implemented as a generalized framework and cell types and measurement updates can be exchanged. First the scan $S^L = \{m_0, ..., m_N\}, m_i \in \mathbb{R}^2, N \in \mathbb{N}$ is used to determine an upper and lower bound for each scan point m_i, between which the TSDF is to be estimated. For this a truncation distance $\phi \in \mathbb{R}$ is defined, which bounds how far away cells are updated. The distances are calculated along the scan line on which the LiDAR measures. As such the upper bound u_i and lower bound l_i in which cells are updated are calculated by

$$u_i^L = m_i + \phi \cdot \frac{m_i}{|m_i|} \tag{4.43}$$

$$l_i^L = m_i - \phi \cdot \frac{m_i}{|m_i|}. \tag{4.44}$$

These bounds are transformed from the LiDAR frame to the navigation frame using

$$u_i^N = T^{V \to N} \cdot T^{L \to V} \cdot u_i^L, \tag{4.45}$$

$$l_i^N = T^{V \to N} \cdot T^{L \to V} \cdot l_i^L, \tag{4.46}$$

where $T^{V \to N}$ is the vehicle pose in the navigation frame obtained by the odometry localization and the pose of the LiDAR in the vehicle $T^{L \to V}$ is a fixed transformation given by the LiDAR configuration. These bounds are calculated for each point m_i in the scan to determine the first and last point to be updated in the TSDF map. These points are transformed from coordinates in the navigation frame to map coordinates by discretizing them to integer indices in a grid. This step is described in detail in Chap. 5. The resulting points in the map are used to draw a pixel line using the Bresenham algorithm [Bresenham, 1998]. This pixel line contains indices of cells that are on the measurement line leading from the LiDAR sensor to the hit point m_i, however only when their distance to the hit is smaller than ϕ. For each cell the distance d_i to m_i is calculated and the cell is updated according to the update function presented in [Daun et al., 2019] in equations (4)-(6).

To perform the matching step of the current scan given the previously estimated map a non-linear optimization problem is solved, which is defined by

$$\underset{\bar{T}^{V \to N}}{\arg\min} \sum_{i=1}^{N} \left(\Phi \left(\bar{T}^{V \to N} \cdot T^{L \to V} m_i \right) \right), \tag{4.47}$$

where Φ obtains the distance in the TSDF map for a point expressed in the navigation frame. For this all cells around the point are retrieved from the map and bi-linear interpolation is performed. The optimization problem can be solved by any non-linear optimizer such as the Ceres Solver [Agarwal et al., 2023]. In this work however a self-implemented version of a gradient-based optimization is used which iteratively modifies the pose in each dimension, choosing the best new pose as the next starting point. When the result no longer improves the step size in each dimension is reduced to refine the pose. This results in a locally optimal pose, however this is sufficient with a precise initial guess. Since the initial guess is given by the odometry localization it is sufficiently precise to only require minor refinement. An example of a result of the LiDAR scan before and after pose refinement is shown in Fig. 4.17.

The refined pose estimate is then used to correct the state estimate of the Kalman filter. As the resulting pose $\bar{T}^{V \to N}$ is the same pose estimated in the Kalman filter (although reduced to two dimensions) the measurement function is simple to define. First the measurement is given by

$$z_t^{lo} = \begin{bmatrix} r_t^x \\ r_t^y \\ \psi_t \end{bmatrix}, \tag{4.48}$$

where $x_t \in \mathbb{R}$ and $y_t \in \mathbb{R}$ are the x-and y-position of the vehicle and $\psi_t \in \mathbb{R}$ is the orientation. As the measurement is given in 2D the measurement function performs a

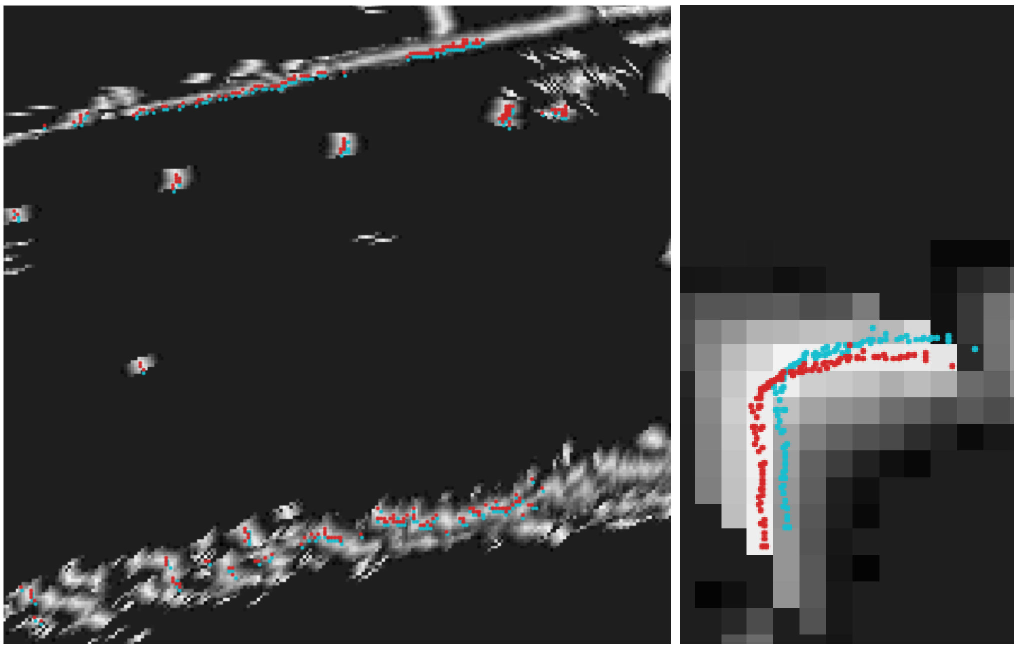


Figure 4.17: Two example results of TSDF scan matching. In greyscale the (absolute) truncated distance to previous measurements is shown, while the current measurement is shown in blue and the optimized scan is shown in red.

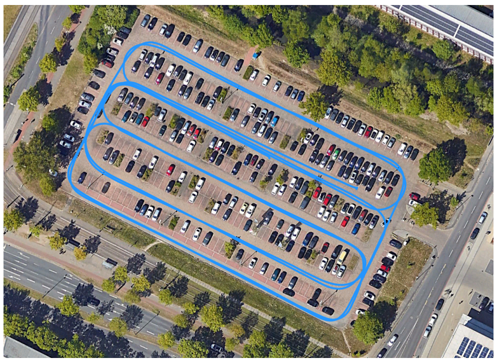
reduction in the state space. This is given by

$$h^{lo}(x_t) = \begin{bmatrix} r_t^x \\ r_t^y \\ \operatorname{atan2}\left(2 \cdot (q_{t,w} \cdot q_{t,z} + q_{t,x} \cdot q_{t,y}), 1 - 2 \cdot (q_{t,y}^2 + q_{t,z}^2)\right) \end{bmatrix} . \tag{4.49}$$

The measurement is assumed to be affected by additive, normally distributed noise with covariance Q_t^{lo} and zero mean. The presented scan matching approach does not inherently produce an uncertainty measure, and thus the covariance is determined manually. Different covariances are evaluated in the following evaluation. The derivation especially of the orientation is not trivial, however the evaluation is performed using a UKF and thus no derivations are required. To use the update with an EKF numerical derivation can be used to obtain derivatives instead.

4.5.2 Evaluation

The performance of the TSDF scan matching is evaluated similarly to the odometry localization, however only the Parking Lot data set shown in Fig. 4.8a is reused, as the other datasets do not contain LiDAR data. In addition a second data set was added containing a drive through Borgfeld, Germany, which is a suburban city scenario. The trajectories of the datasets are shown in Fig. 4.18.


(a) Parking Lot

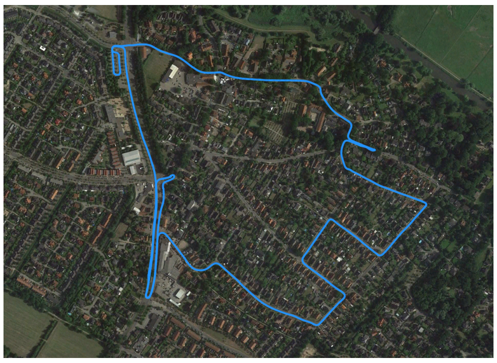
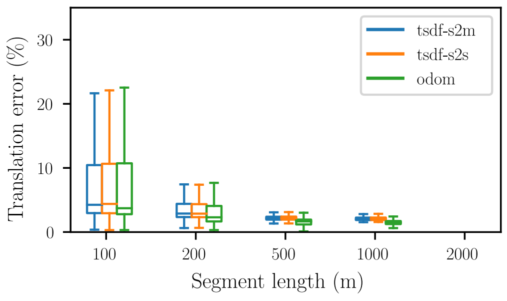

(b) Borgfeld

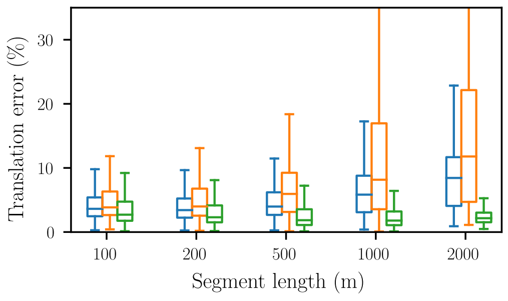
Figure 4.18: Ground-truth trajectories of the evaluation datasets. (Maps generated using Google Earth Pro.)

The datasets were recorded using the research vehicle presented in Chap. 3. However the IMU used differs between the two. The Parking Lot data was recorded using an ADIS IMU while the Borgfeld dataset was recorded using a lower cost UM7 IMU. As the focus of this work lies in the relative difference scan matching makes in the localization performance this difference in sensor equipment does not influence the validity of the results. Similar to the odometry evaluation the localization accuracy is determined by comparing to the ground truth recorded using GNSS. The evaluation was performed using a regular Moving Reference UKF as reference and the same filter additionally corrected using the TSDF matching measurement. For scan matching two approaches are compared: one where the map is continuously built over time (scan-to-map matching) and one where the map is only built on the previous scan (scan-to-scan matching). While the map built from all scans is expected to contain more accurate information it is subject to the odometry drift and over time this may deteriorate the map and with it the matching quality. Thus both approaches are evaluated. As the covariance of the resulting pose is not determined by the matching algorithm a comparison between two covariances is performed. For the first evaluation a positional standard deviation of $\sigma_r = 0.25\,\text{m}$ and a rotational standard deviation of $\sigma_a = 5\,\text{deg}$ are used. The results are shown in Fig. 4.19. The second evaluation is performed using $\sigma_r = 0.025\,\text{m}$ and $\sigma_a = 0.5\,\text{deg}$. The results of the second evaluation are shown in Fig. 4.20. The corresponding resulting trajectories are shown in Fig. 4.21 and Fig. 4.22.

As can be seen in these evaluations the scan matching does not improve the localization result and in fact in some cases even leads to significantly worse results. Especially the scan-to-map matching performed on the Parking Lot dataset shown in Fig. 4.22a fails to produce good results and fully deteriorates over time when the filter assumes the LiDAR odometry to be more precise. This is likely caused by the odometry drift which causes the TSDF map in this scenario to be cluttered with noise. Thus in that scenario scan-to-scan matching performs better. This is however only the case in this specific scenario. Overall the scan-to-scan matching performs significantly worse than the scan-to-map

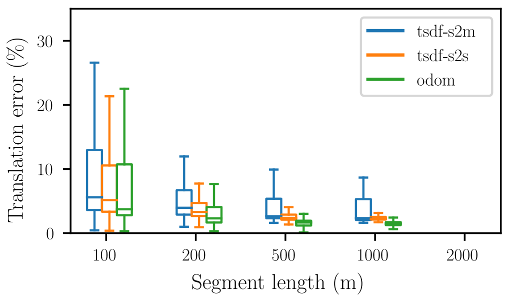


(a) Parking Lot

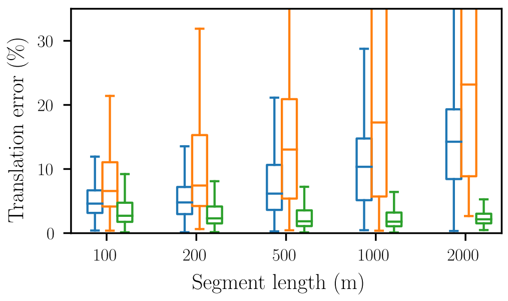


(b) Borgfeld

Figure 4.19: Evaluation with $\sigma_r = 0.25\,\text{m}$ and $\sigma_a = 5\,\text{deg}$.

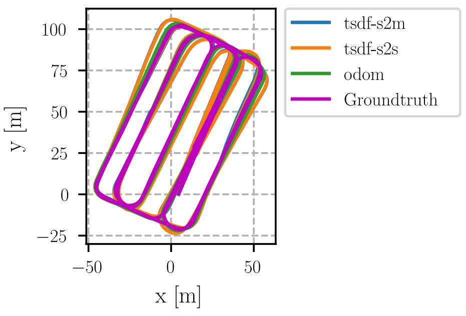


(a) Parking Lot

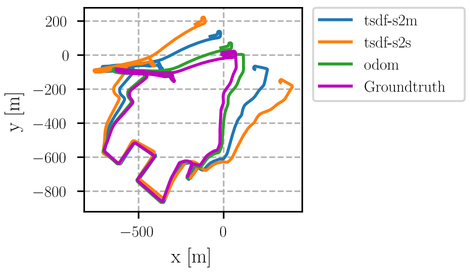


(b) Borgfeld

Figure 4.20: Evaluation with $\sigma_r = 0.025\,\text{m}$ and $\sigma_a = 0.5\,\text{deg}$.

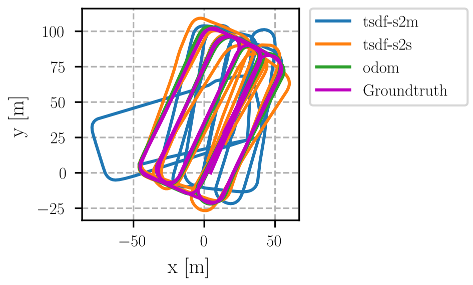


(a) Parking Lot

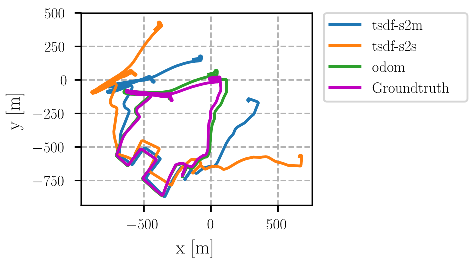


(b) Borgfeld

Figure 4.21: Resulting trajectories of the first evaluation with $\sigma_r = 0.25m$ and $\sigma_a = 5\,\text{deg}$.



(a) Parking Lot



(b) Borgfeld

Figure 4.22: Resulting trajectories of the second evaluation with $\sigma_r = 0.025m$ and $\sigma_a = 0.5$ deg.

approach. This indicates that the added information gained by accumulating past scans in the TSDF map provides significant improvements to the scan matching results. It also indicates that the scan matching relies heavily on a complete representation of the environment, which is not immediately provided by the available LiDAR.

For both evaluations the moving reference UKF without scan matching strongly outperforms even the scan-to-map matching. In fact when the filter believes the measured scan matching pose to be more accurate the resulting localization becomes less accurate. Due to the very high accuracy achieved by the moving reference filter it is however not unexpected that the LiDAR was not able to improve the results. The data obtained by the LiDAR is relatively sparse due to the small vertical field of view and small changes in the vehicle pitch or roll can change where the LiDAR measures points, which would result in a incorrect pose estimate even for an optimal scan matching result. Taking these changes into account with full 3D matching is however not feasible with the available sparse LiDAR data. While Fig. 4.17 shows a matching result where the initial guess was significantly offset from the correct estimate, this was only chosen for easier visualization purposes. A further example is shown in Fig. 4.23 to visualize how accurate the initial guess obtained by the odometry is and how difficult it can be to find the optimal solution.

As is shown here the optimal solution is not clear and the initial guess is already close to or on the seemingly best estimate. The scan matching only results in minor updates of the vehicle pose. On average the scan matcher is however not able to find a more accurate solution than the one calculated by the moving reference filter. In the original paper, TSDF scan matching was used as a tool to perform LiDAR SLAM. This relies much less on a highly precise result in every step as long as a loop closure is performed regularly. During loop closure accumulating errors on the trajectory are eliminated, which is not possible in this work. However SLAM is not usable to aid the odometry localization as the loop closure introduces jumps in the state estimate.

Overall it was therefore not possible to improve the odometry localization accuracy using TSDF-based scan matching. Therefore the results shown in [Serov et al., 2022] on camera-based visual odometry were confirmed on LiDAR data as well. With the available combination of IMU, speed and steering measurements the precision of the

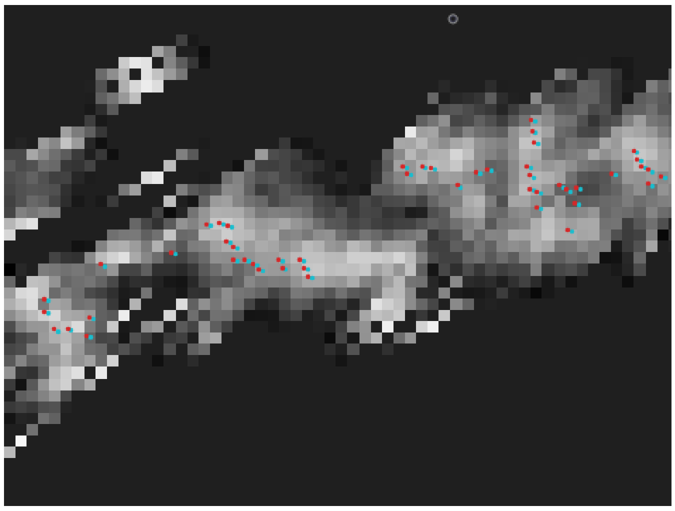


Figure 4.23: Difficult TSDF matching example where bushes are observed. The initial guess is shown in blue, the scan matching result is shown in red. No clear optimal solution is visible.

localization is high and difficult to improve, especially with the available 4-layer LiDAR that only allows for 2D scan matching. Results on the public Visual Odometry/SLAM dataset in the KITTI Vision Benchmark Suite [Geiger et al., 2012] indicate that with a 3D LiDAR significantly higher accuracies can be achieved. Thus the use of LiDAR odometry will be evaluated in future work using a 3D LiDAR with a higher vertical FOV and more layers.

5 Mapping

Autonomous driving requires a high level of understanding about the vehicles surroundings to enable safe driving in often complex and dynamic areas. This task can be divided into one where the static environment is estimated and one where all other traffic participants that may be moving around the ego-vehicle are detected and tracked. In this chapter the algorithm for the estimation of the static environment is presented, which is referred to as mapping in the following. This is used by subsequent algorithms to determine drivable areas in the surroundings, preventing collisions with any unmoving obstacle. The second task of detecting and tracking other traffic participants is discussed in Chap. 6, however its results influence the mapping to a certain extent. The sensor data generally contains both dynamic and static parts of the environment and needs to be filtered using the tracked traffic participants to avoid integrating dynamic parts into the static environment map.

For static environment estimation a grid-mapping approach as introduced in Sect. 2.3 is used in order to estimate an occupancy map. This is built up using scan points obtained from the six LiDARs built into the research vehicle as shown in Chap. 3. The grid map discretizes the environment in $20\,\text{cm} \times 20\,\text{cm}$ cells with each cell estimating whether the corresponding space in the world is occupied or free. An example of such a map can be seen in Fig. 5.1. The mapping performed here is mapping with a known pose, which stands in contrast to SLAM approaches where the mapping and localization problem are seen as a joint problem to be solved at the same time, influencing each other. The decision to perform mapping with a known pose was made to avoid jumps in the localization estimate that occur during loop closure of SLAM approaches. However it influences how the mapping is performed, as a SLAM approach would ensure a consistent map even over longer periods of time. When performing mapping with known pose the performance of the mapping approach strongly relies on the underlying localization. In this case the odometry estimation proposed in Sect. 4.2 is used. While this localization is accurate over short distances, it still introduces drift over time, which is reflected in the map, especially after revisiting places that were previously estimated. To avoid drift the global localization could be utilized, however this suffers from jumps which again would influence the mapping performance as the vehicle might shift closer or further from previously detected obstacles due to the jumps. This may produce dangerous behavior caused by the vehicle control. In autonomous driving the map can however be estimated only for a limited time window instead of building a full map of the entire traversed environment. This is due to the environment being highly dynamic and quick to change, which means old information becomes obsolete after a short time. Therefore the drift

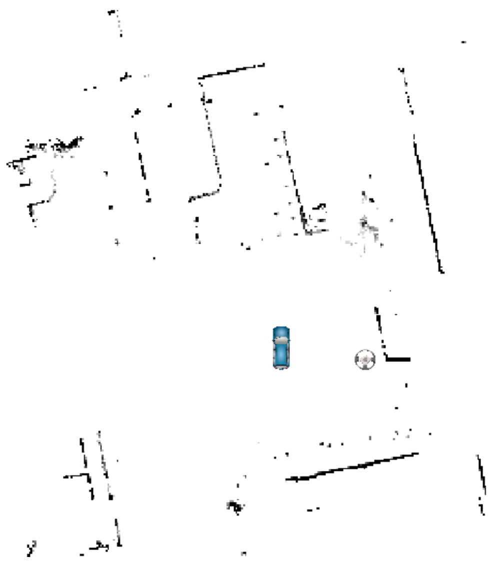


Figure 5.1: Example of an occupancy grid map. The occupancy of each cell is shown in grayscale, with the current pose of the ego vehicle indicated by the blue vehicle.

introduced by the odometry localization does not influence the mapping performance. To perform mapping over a limited time window LiDAR scans are added when they first arrive and then substracted from the map as they become too old. The result is a short-term map which accurately reflects the current state of the environment while not taking into account more than a fixed amount of previous data.

Grid mapping is often utilized in robotics with limited space that needs to be mapped or in autonomous driving scenarios with a limited range. This allows for pre-allocation of the memory required to store the entire map, which speeds up map operations significantly. However for a generalized mapping framework in autonomous driving with a large range, additional measures are needed to allow processing over these potentially very large areas. As the mapped area becomes larger the memory required for the map grows, and at some point it becomes impossible to allocate a map in memory which is large enough to cover the entire area. An alternative approach is to use a map representation which does not rely on any pre-allocation of memory and instead allocates it on demand, however this usually comes at a significant disadvantage when it comes to processing speed. In this work a system is used which still utilizes pre-allocated memory, however this is only done in a limited area. For this a moving-window approach is used as published in [Wellhausen et al., 2021] where the mapped area is restricted to a window roughly around the current position of the ego-vehicle.

In the following, first an overview of the current state of the art in mapping systems is given. Secondly the implementation of the short term grid map is explained and lastly the moving-window approach is introduced and evaluated.

5.1 State-of-the-art

Mapping the environment is part of virtually every autonomous driving stack, with grid maps being used as the de-facto standard for representing the environment. The information that is estimated in this map is however different depending on the use-case, the equipped sensors and the information required by subsequent algorithms. The standard occupancy grid map [Elfes, 1989] has been in use for many years already, where within each cell the probability is estimated for the space in the world to be occupied. There are however many extensions to this approach, where additional information is estimated within each cell, or where the map is structured differently. One example of an extension to the classical occupancy grid map is the evidential grid map which makes use of the Dempster-Shafer theory [Dempster, 1967; Shafer, 1976]. This was initially utilized in autonomous driving in [Pagac et al., 1998], however it has since been utilized in a number of publications. In [Capellier et al., 2018] evidential maps are used to fuse LiDAR and Camera data into a grid map, utilizing semantic labels to improve handling of dynamic data. In [Richter et al., 2020] it is utilized for creating a semantic grid map estimating the type of object contained in different parts of the environment, which is performed solely on stereo camera data. In [Tanzmeister and Wollherr, 2016] evidential maps are utilized to estimate the dynamic properties of the environment, determining a velocity for each part of the environment and differentiating between static and dynamic cells. Overall many use cases exist for evidential maps and as such this is considered in the design of the mapping system, allowing easy integration of different cell types as well as mapping algorithms.

Another point of interest in current research is the map organization. One main issue in utilizing grid maps is the large amount of cells as well as the high memory consumption that results from discretizing the surroundings in a fine grid of cells. In [Jungnickel et al., 2016] and [Garcia et al., 2014] this is tackled by using dynamic grid resolutions. Instead of representing the entire environment using a uniform cell size a choice can be made to represent more interesting areas with a higher resolution, while others are represented only roughly. While this significantly reduces memory consumption it comes at the cost of additional upkeep when the map resolution is re-evaluted. Additionally cell access becomes more difficult when the cell structure is not previously known, again raising computational demand. In the work of [Jo et al., 2018] a quadtree is used to subdivide the globe into areas until the contained area is sufficiently small. When an area is observed a node is created in the quadtree which contains the grid map for that area. This is done to enable updates of a global map between multiple vehicles. While the global representation in a quadtree is not required in this work, the idea of using an additional data structure to control where the grid map should be extended is very interesting and is further explored in this work. In [Buerkle et al., 2020] a mapping approach is presented where submaps are utilized, however here the submaps do not have a uniform cell size. Instead submaps that are further from the vehicle have a lower resolution. In addition the map is not estimated in the world frame but in the vehicle frame. This leads to a system where only the direct surroundings are mapped. The memory is handled very efficiently using the dynamic resolution however additional computations are required to transform the map between measurements as the contents within the world shift according to the movement of the vehicle. Another approach utilizing submaps is presented in [Stoyanov et al., 2013]. In this a 3×3 grid of submaps is used, which is roughly centered around the vehicle. Each of these submaps contains the map of the corresponding space in the world. As the vehicle moves towards new areas submaps are discarded behind the vehicle, while new submaps are created in the front. This allows for efficient data storage of only the relevant parts of the environment while not compromising on computation time by using complex data structures. In addition the map is estimated in the world coordinate system and does not require shifting cells between measurements. In this work a similar approach is utilized. The main contribution of this thesis is an evaluation of different data structures that can be used for a similar mapping system. By using different combinations of data structures the memory consumption and computation time can be tuned for different tasks. By evaluating the mapping with different data structures on multiple relevant tasks a suggestion is given which combinations are suitable under certain circumstances.

5.2 Short-term Grid Mapping

In this work a short-term grid map is utilized which integrates LiDAR measurements as they become available, and removes them from the map after a certain time. This is done to counteract the influence of the drift caused by the odometry localization, which would result in an inaccurate map upon revisiting an area. The short-term map is estimated by determining the fullness $\theta_i \in [0, 1]$ of each cell as explained in Sect. 2.3. This approach works under the assumption that the noise produced by the LiDAR measurement is significantly smaller than the error introduced by the discretization of the grid map, which is a reasonable assumption for modern LiDAR systems. Thus the measurement noise is disregarded and instead the map estimation reduces to

$$\theta_i = k_i/(k_i + l_i), \tag{5.1}$$

with k_i indicating how often an endpoint of a scan hit the cell and l_i representing how often the cell was traversed but not hit, which would indicate that it is free. Thus the estimation of the state within each cell only depends on the counters k_i and l_i. To estimate these counters the current LiDAR scan $S^L = m_0^N, ..., m_D^N$ with $D \in \mathbb{N}$ as well as the pose of the LiDAR in the navigation frame $T^{L\to N} = T^{V\to N} T^{L\to V}$ are required. The transformations $T^{V\to N}$ and $T^{L\to V}$ are obtained by the odometry localization and the extrinsic calibration of the LiDAR respectively. With this transformation the LiDAR scan S^L is transformed to the navigation frame by

$$S^N = T^{L\to N} S^L, \tag{5.2}$$

with the transformation being performed on each point in the scan. Given the origin of the LiDAR and the positions in the world that were hit, the ray along which the LiDAR measured is known. These rays are however in the navigation frame. With the goal being to obtain all cells that are traversed by this ray, the position of the LiDAR and all endpoints where the rays hit an obstacle must be discretized to map coordinates. This is done by

$$p^M = \left\lfloor (p^N - o^N) \cdot \frac{1}{\delta} \right\rfloor , \tag{5.3}$$

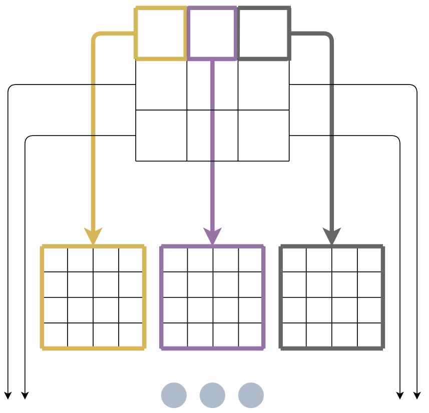
where δ is the edge length of each cell in the map and o^N is the origin of the map in the navigation frame. This is performed for all scan points as well as the position of the LiDAR in the navigation frame r^N. For each endpoint m_i^M all traversed cells are obtained using the Bresenham algorithm [Bresenham, 1998] between r^M and m_i^M. The result is a list of map coordinates of cells that were traversed, with the last cell being the hit cell. The counter l_i is increased for the traversed cells, while k_i is only increased for the hit cell. As a special case of a 4-layer LiDAR, there are multiple scan lines that measure along the same measurement angle. When one of the hits on this line is closer to the LiDAR than the others (e.g. due to a low obstacle only hit by the lowest layer) the information about the cell being hit would effectively be overwritten by the other scan lines, and only a 1/4 probability for the cell to be occupied would result. To avoid this issue the map is only updated as described above for the closest hit along one measurement angle. For all other hits the traversed cells are not updated, however for the hit cell the hit counter k_i is still increased to maintain the information about all measured obstacles. To later remove a scan, the updated cells are stored together with the increased counter in a queue and removed again after a certain time, reversing the performed updates.

5.3 Moving-window Grid Mapping

As discussed it is not necessary to keep old information in the map indefinitely due to changing environments and odometry drift. Instead a fixed size of $s \times s$ is defined as a window for the map which is the minimum size that should be covered, and this window is used for keeping information. It roughly moves with the vehicle and parts of the map that fall outside of the active window are discarded. Other shapes than squares could be considered, however since the vehicle may turn and the maps orientation is world-fix a square is a good choice. The work presented in this section was first published in [Wellhausen et al., 2021], however it was rewritten and partially extended to better match the scope of the thesis.

The total number of cells required to cover the desired square is calculated by $D = (s/\delta)^2$, where δ is the edge length of a single cell. In this work the parameters are chosen as $s = 350$m and $\delta = 0.2$m, which results in $D = 1750^2 = 3062500$ cells. A world-fix representation is chosen for the map. A vehicle-fix representation is not chosen here due to the added computational cost of shifting cells according to the vehicle movement and due to a loss in accuracy caused by interpolation when cells are transformed and mapped to new cells.

To implement the moving-window grid map a hierarchical submap approach is chosen with two layers as shown in Fig. 5.2a. Submaps are used to reduce computational cost


(a) Hierarchical organization into submaps.

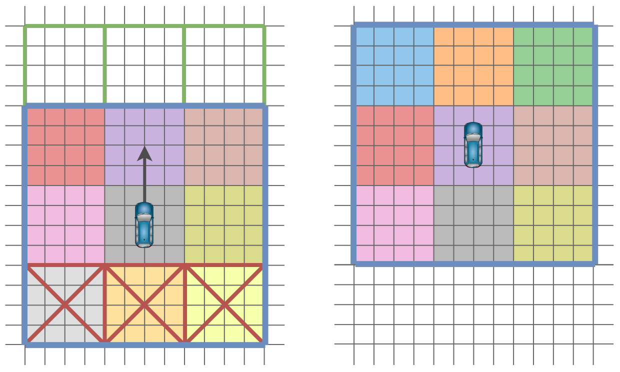

(b) Change of the considered window with the moving vehicle.

Figure 5.2: A grid map structured using submaps. (a) illustrates the hierarchical organization, where the map is divided into submaps (top) with each submap containing multiple cells (bottom). (b) shows the movement of the considered window (blue outline). Each colored square corresponds to a submap. New submaps are added (green outlined squares) as the vehicle moves, while old submaps are discarded (red crosses). (Figure adopted from [Wellhausen et al., 2021].)

when the active mapping window is moved forward. Without submaps this would require moving every single cell, while using submaps it reduces to moving much fewer submaps. On the outer layer a grid map is used to store the submaps. The inner layer stores the grid cells that estimate the state of the corresponding space in the world. As the vehicle moves through the world full rows of submaps are allocated in front of the vehicle, while rows behind the vehicle are discarded to move the mapped window forward. This is visualized in Fig. 5.2b. While the layers store very different information, the data structures used can be the same since both layers can be seen as a grid. The inner layer operates much like a regular grid map, however on the outer layer some additional handling is required in order to delegate cell access to the correct submap and to handle allocation and deletion of submaps. The specific operations are defined in the following, however first the structure of the hierarchical map is explained. The amount of submaps K allocated on the top level to cover an area of $s \times s$ meters depends on the amount of cells L within each submap in each dimension. It is calculated by

$$K = \left\lceil \sqrt{\frac{D}{L}} \right\rceil^2 , \tag{5.4}$$

which gives the total amount of submaps in the outer layer. The true amount of required submaps in a single dimension $\sqrt{\frac{D}{L}}$ is rounded up as no partial maps can be allocated, which would make submap allocation and handling more difficult and add computational overhead. Therefore K submaps are unlikely to cover the minimum mapped area perfectly, but is guaranteed to cover at least the defined size. As an example, choosing

$L = 256^2$ each submap contains $256 \times 256 = 65536$ cells. With the total amount of cells $D = 3062500$ from above this results in $K = 7^2 = 49$ submaps. Since the actual area covered by all submaps may be larger than the minimum size s, the number of cells $\hat{D}$ will usually differ from the theoretical number D that is needed to cover the area perfectly. The true amount $\hat{D}$ is obtained by $\hat{D} = L \cdot K$. In this example this leads to a true amount of $\hat{D} = 1792^2 = 3211264$ cells in the map, which covers a length of $\sqrt{K} \cdot L \cdot \delta = 7 \cdot 256 \cdot 0.2\text{m} = 358.4\text{m}$ in each dimension. The size of the entire map expressed in terms of cells in each dimension is therefore given by $\sqrt{\hat{D}}$. The size of each submap is given by $\sqrt{L}$, while $\sqrt{K}$ gives the number of submaps in each dimension.

With the structure of the map defined next the access of individual cells withing the map is explained. Given a position $p^N \in \mathbb{R}^2$ in the navigation frame the corresponding cell index in the map is computed by

$$p^M = \left\lfloor \frac{p^N}{\delta} \right\rfloor . \tag{5.5}$$

This assumes that the origin of the map and the navigation coordinate system are the same. Note that while the active mapping window moves around, the origin of the map is not moved with it but stays constant. To access the cell with index p^M first the correct submap index $s \in \mathbb{R}^2$ is needed, which is calculated by

$$s = \frac{p^M - p^M_{\text{orig}}}{\sqrt{L}}, \tag{5.6}$$

where p^M_{orig} represents the origin, or bottom left corner, of the currently active window in global map coordinates. With this submap index the correct submap is obtained from the outer layer, however within this submap the global index p^M is meaningless as each submaps indices start at zero. The index of the point within the submap is obtained by

$$p^C = p^M - p^M_{\text{orig}} - s \cdot \sqrt{L}. \tag{5.7}$$

Accessing a cell using a coordinate in the navigation frame is the most common operation on the map, however occasionally the position in the navigation frame that corresponds to a cell index is needed. In order to compute this, (5.5) to (5.7) can simply be inverted. Note that this will however yield the position of the bottom left corner of the cell. When the center of the cell is desired, $\frac{\delta}{2}$ must be added to the resulting position in each dimension.

As shown in Fig. 5.2b the window needs to be moved according to the movement of the vehicle whenever its distance to the center of the current window becomes too large. As the window is always shifted in rows of submaps the distance threshold is the length of one submap in world coordinates, i.e. $\sqrt{L} \cdot \delta$. After the shift the vehicle is again roughly in the center of the window. To perform the shift all submaps in the outer layer need to be shifted within the grid map according to the movement of the vehicle. This operation is done efficiently by storing the submaps as pointers in the outer layer, removing the

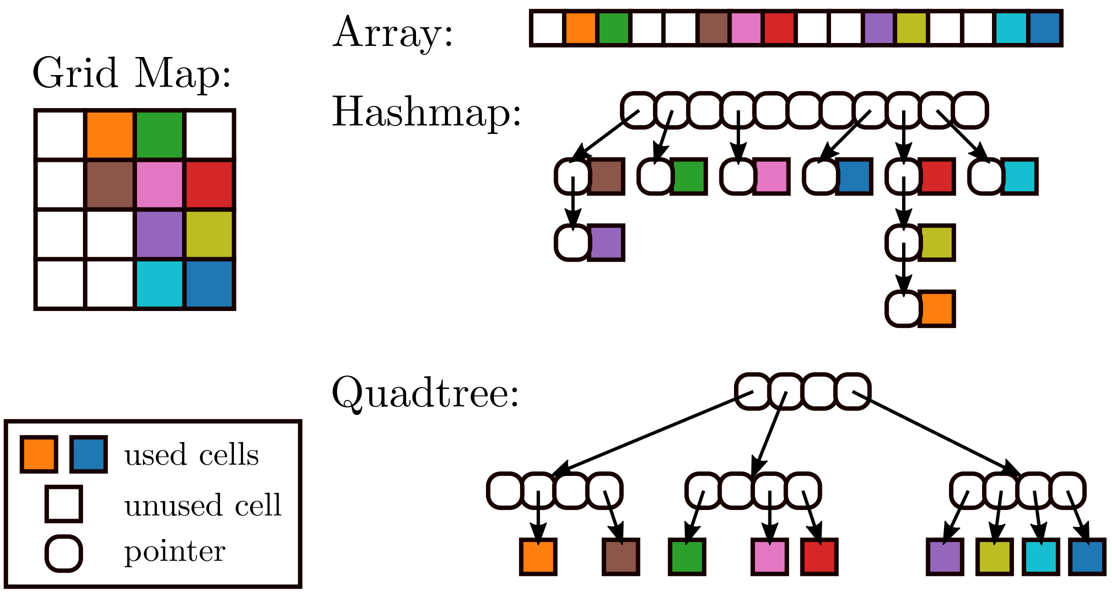


Figure 5.3: Schematics of the different underlying data structures for a partially filled grid map. The array stores all rows one behind the other. The hashmap uses a bucket array with linked lists for collision resolution. The quadtree partitions the map into hierarchically organized 2×2 blocks. (Figure adopted from [Wellhausen et al., 2021].)

need to copy the entire submap to a different cell. Submaps that would fall outside of the window after moving them are discarded, while depending on the data structure new submaps may be allocated in new areas within the window. These data structures are introduced in the following.

5.3.1 Data Structures

The used data structures have a strong influence on the behavior of the system. To select suitable ones the main requirement is a sufficiently fast random cell access using the index calculated above. In this work 2D arrays (ar), hashmaps (hm) and quadtrees (qt) are evaluated. In Fig. 5.3 the data structures and their memory management is visualized. To implement a 2D array the intuitive structure is to use an array for each row or column in the grid. This however leads to the memory being fragmented, which has downsides in allocation and deallocation as well as other memory management such as copying. Instead a single array is used which stores the grid row-wise back to back in a single memory block. This enables cell access by calculating the memory index from the map index using

$$a(x, y) = x \cdot r + y, \tag{5.8}$$

where x and y are the x- and y-index of the cell within a submap or of the submap within the outer grid and r denotes the number of cells in each row of the grid. Note that for this cell access to work the entire map must be allocated fully in memory, even including unused cells. While this allocates memory for a lot of unused space the cell access becomes very efficient.

Hash maps [Cormen et al., 2009, Chap. 11] store its elements in buckets, where the bucket index is computed by a hash function. This hash function converts a complex value such as the x and y index of the cell to a single index of the bucket. When designing the hash function ideally it should distribute the incoming data evenly on the available buckets, as collisions generally lead to slower cell access. In this work the hash function proposed in [Teschner et al., 2003] is used, which is defined as

$$b(x, y) = (73856093 \cdot x) \dot{\vee} (19349663 \cdot y) \bmod n\,, \tag{5.9}$$

where n represents the number of buckets used in the hash map and $\dot{\vee}$ is the bit-wise XOR operation. The amount of buckets is usually chosen to be smaller than the maximum number of elements and can be increased over time when needed. This reduces memory consumption, however it leads to hash collisions where two or more different elements are mapped to the same bucket. To handle this buckets usually contain linked lists of all elements mapped to the bucket. Due to this structure memory is mostly allocated on demand with the exception of some empty buckets being pre-allocated. Due to hash collisions cell access becomes slower however, as in some cases a linked list must be traversed.

Quadtrees [de Berg et al., 2008, Chap. 14] use a tree structure to store its elements. This representation is suitable whenever a spacial distribution of the data is possible. Each node in the tree has 4 children until the lowest level of the tree is reached, where the nodes are called leafs and do not have children. For representing a grid map the map is repeatedly partitioned into 2×2 blocks until the area covered by each leaf matches the defined cell size or submap size depending on the layer where this representation is used. Quadtrees allow dynamic memory allocation, with children only being allocated on demand. Thus the representation does not allocate unnecessary memory, however it comes at a cost of slower cell access. To reach a certain cell the tree needs to be traversed from the root node to the leaf. The cost of this operation depends on the depth of the tree.

5.3.2 Theoretical Analysis

Having introduced all evaluated data structures this section gives a theoretical analysis of their time and space complexity. This is done for each data structure individually as well as for their use in a hierarchical structure as proposed in this work. An overview of the results is given in Tab. 5.1, Tab. 5.2 and Tab. 5.3. Additionally the overhead caused by the practical implementations of the data structures are discussed. While the array fully allocates all cells, the hashmap and quadtree only allocate used cells. For the whole map the number of used cells is denoted as $\tilde{D}$ while for a single submap it is denoted as $\tilde{L}$. The number of used submaps is represented as $\tilde{K}$ while $\bar{L}$ refers to the average number of cells over all submaps.

First the time complexity is evaluated. For an array, the position of a cell in the

Table 5.1: Complexity of the different data structures without hierarchical organization. (Table adopted from [Wellhausen et al., 2021].)

	array	**hashmap**	**quadtree**
Time	$\mathcal{O}(1)$	Best: $\mathcal{O}(1)$ Worst: $\mathcal{O}(\tilde{D})$	$\mathcal{O}(\log D)$
Space	$\mathcal{O}(D)$	$\mathcal{O}(\tilde{N})$	$\mathcal{O}(\tilde{D})$

Table 5.2: Time complexity for different combinations of data structures. (Table adopted from [Wellhausen et al., 2021].)

		inner		
		array	**hashmap**	**quadtree**
outer	**array**	$\mathcal{O}(1)$	Best: $\mathcal{O}(1)$ Worst: $\mathcal{O}(\tilde{L})$	$\mathcal{O}(\log L)$
	hashmap	Best: $\mathcal{O}(1)$ Worst: $\mathcal{O}(\tilde{K})$	Best: $\mathcal{O}(1)$ Worst: $\mathcal{O}(\tilde{K}+\tilde{L})$	Best: $\mathcal{O}(\log L)$ Worst: $\mathcal{O}(\tilde{K}+\log L)$
	quadtree	$\mathcal{O}(\log K)$	Best: $\mathcal{O}(\log K)$ Worst: $\mathcal{O}(\log K+\tilde{L})$	$\mathcal{O}(\log K+\log L)$

memory is computed directly from the cell index. Therefore the time complexity is $\mathcal{O}(1)$. Additionally there is an overhead caused by calculating the memory position using (5.8), however this is minor. No additional overhead is caused by accessing new elements as the map is fully allocated initially.

For a hash map the best-case time complexity for accessing a cell is $\mathcal{O}(1)$ as well. The bucket index is computed using (5.9) and in the best case the element is the first in the contained linked list. However when hash collisions occur a single linked list may

Table 5.3: Space complexity for different combinations of data structures. (Table adopted from [Wellhausen et al., 2021].)

		inner		
		array	**hashmap**	**quadtree**
outer	**array**	$\mathcal{O}(K\cdot L)$	$\mathcal{O}(K\cdot\bar{L})$	$\mathcal{O}(K\cdot\bar{L})$
	hashmap	$\mathcal{O}(\tilde{K}\cdot L)$	$\mathcal{O}(\tilde{K}\cdot\bar{L})$	$\mathcal{O}(\tilde{K}\cdot\bar{L})$
	quadtree	$\mathcal{O}(\tilde{K}\cdot L)$	$\mathcal{O}(\tilde{K}\cdot\bar{L})$	$\mathcal{O}(\tilde{K}\cdot\bar{L})$

contain many elements. In the worst case all elements are sorted into the same bucket, resulting in a linked list containing all elements. This linked list must be traversed until the correct element is found, resulting in a worst case complexity of $\mathcal{O}(\tilde{D})$. There are some additional overheads in the implementation of a hashmap. Similar to the array the bucket index is calculated, which is however a simple computation. When performing a cell access, during the traversal of the linked list a check must be performed to evaluate whether the current element is the one that should be accessed. This again depends on the number of elements in the list. Finally during insertion memory needs to be allocated which adds overhead in itself. Additionally when too many collisions occurred the amount of buckets n may be increased. For this all bucket pointers must be moved and the bucket indices recomputed as (5.9) depends on n.

For a quadtree, cell access is achieved by traversing the tree from root node to the requested leaf node. This operation has a fixed complexity of $\mathcal{O}(\log D)$ and does not depend on the amount of cells $\tilde{D}$ that have been allocated but only on the maximum number of possible cells D. This is due to the map having a fixed depth that is traversed each time. Some overheads exist in the quadtree implementation. In each node four child nodes exist and the arithmetic operation to compute the correct child adds some overhead. During insertion of new elements additional overhead is caused by allocating the new leaf node as well as all branch nodes that may not have been created previously. The complexity of this is bounded by $\mathcal{O}(\log D)$, however as the tree is filled over time, on average much fewer nodes need to be allocated.

Next the space complexity of each individual data structure is evaluated. For the 2D array there are clear disadvantages regarding the space requirements. With the entire grid being fully allocated the space complexity is given by $\mathcal{O}(D)$. This includes both used cells and unused cells, thus allocating a large amount of space that is not required in theory. Hashmaps on the other hand allocate space on demand. Cells that have not been observed are not allocated, which potentially saves a large amount of memory. The space complexity is thus given by $\mathcal{O}(\tilde{D})$. Some overhead exists due to buckets being allocated as well as pointers for storing successors in the linked list. Lastly, quadtrees behave similarly to hashmaps in terms of memory consumption. Cells are allocated on demand, while unused cells are not represented, resulting in a space complexity of $\mathcal{O}(\tilde{D})$. Some overheads are caused by the tree structure, as the branch nodes and root node are allocated while not representing actual cells in the map. Each branch node as well as the root node contains four extra pointers for their successors, where the amount of extra branch nodes depends on how many cells are allocated in the map. The approach benefits from the usual structure of points in LiDAR scans however, with points usually being clustered together, while other areas are empty and would not be represented in the tree.

Lastly the combined data structures are compared when used in a hierarchical map with an outer and an inner layer. For the time complexity this structure results in an additive complexity, as shown in Tab. 5.2. For space complexity the combination results in a multiplicative increase as shown in Tab. 5.3. In this comparison the amount of elements in the outer layer is given by the amount of submaps K, while on the inner level it is given by the amount of cells in one submap L. For hashmaps in particular

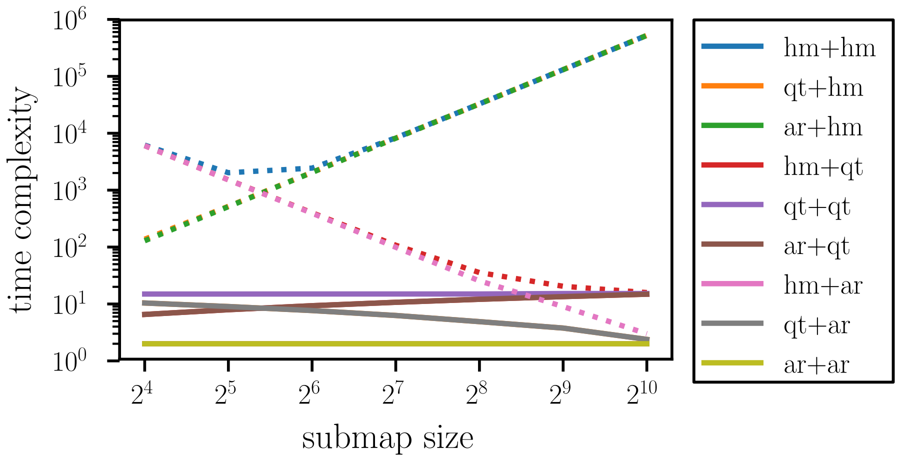


Figure 5.4: Time complexity of different combinations of data structures with varying submap sizes $\sqrt{L}$ for a fixed desired total number of cells $D = 1\,750^2$. The actual number of cells $\hat{D}$ may vary. A filling level of 50% is assumed for the outer and inner data structures resulting in $\tilde{K} = 0.5K$ and $\tilde{L} = 0.5L$. For combinations with hashmap, the solid lines show the best case and the dotted lines show the worst case. Some combinations have the same or almost the same time complexity and thus overlap in the plot. Those are ar+ar, hm+hm, hm+ar, and ar+hm (each best case); ar+hm and qt+hm (both worst case); qt+ar and qt+hm (best case); as well as ar+qt and hm+qt (best case). (Figure adopted from [Wellhausen et al., 2021].)

the worst case time complexity does not depend on the maximum amount of submaps or cells but instead on the used submaps $\tilde{K}$ and cells $\tilde{L}$. For the space complexity of the hashmap and quadtree the number of used cells is again relevant, which results in $\tilde{K}$ for the outer layer. On the inner layer however all submaps must be considered. Thus the average number of used cells over all instantiated submaps $\bar{L}$ is used.

The time complexity when combining the data structures in a hierarchical map depends on the number of submaps as well as the number of cells per submap. Depending on the data structure the number of submaps is given by K or $\tilde{K}$ while the number of cells is given by L or $\bar{L}$. When a large submap size is used, with only very few total submaps, the complexity depends largely on the inner data structure. On the other hand this relation is reversed when a large number of small submaps are used. The theoretical complexity both in the best case and the worst case is given in Fig. 5.4 for different sizes of submaps $\sqrt{L}$, while the size of the map $\sqrt{D}$ is kept constant. Both the hash map and the array have a best case complexity of $\mathcal{O}(1)$, thus any combination of these data structures will also have the same complexity. The quadtree complexity depends on the depth of the tree, which is evened out in the qt+qt combination, regardless of submap size. Both arrays and hash maps in the best case do not depend on the amount of elements in the map. This results in increasing time complexity for larger submap sizes whenever either of these is used on the outer layer, while the inner layer is size-dependent. This is due to the complexity of the inner layer becoming more relevant with larger submap sizes. On the other hand the time complexity decreases when a size-dependent data structure is used on the outer layer with a size independent structure on the inner layer. One interesting result can be observed in the combination of hashmap+hashmap in the worst case. Here the optimal complexity is reached at a submap size of 2^5, and becomes worse with both smaller and larger submap sizes. The theoretically optimal submap size is at $L = K = \sqrt{D}$.

Overall a clear relationship between time complexity and space complexity emerges. High space complexity leads to lower computational demand and vice versa. This also holds for combinations of data structures in a hierarchical map. For the data structure used on the inner layer it is largely irrelevant that the overall map structure is hierarchical. The performance is similar to using a single non-hierarchical map. However especially on the outer layer an interesting behavior can be seen when using hashmaps or quadtrees. The resulting map has large parts that are completely unallocated, where full submaps have not been used yet. Especially in combination with an array on the inner layer the performance may be interesting to observe. Here the speed of the cell access that the array offers is combined with the space-saving properties of the hashmap and quadtree. All proposed combinations of data structures and their runtime and memory requirements are evaluated in the following on real-world data obtained by the research vehicle.

5.3.3 Evaluation

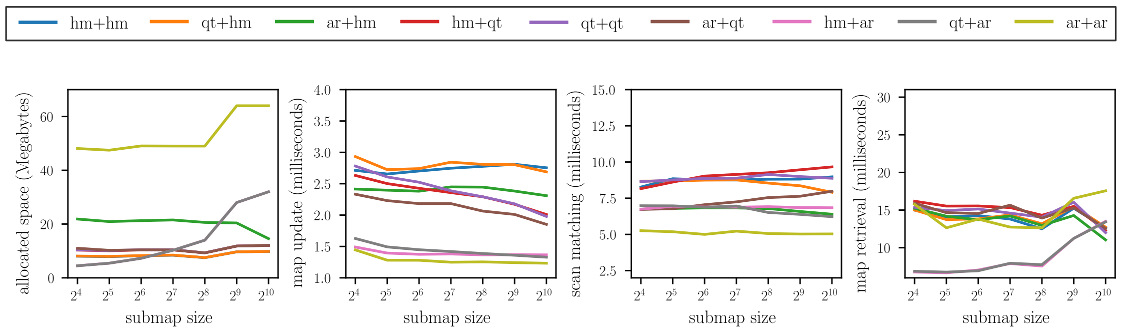
The evaluation of the different data structures and their combination in a hierarchical map was performed on different real-world datasets that showcase different environments. A highway dataset shows the performance on a short stretch of 7.8km of highway, with speeds of up to 93 km/h. The observed environment contains few features, which is reflected in the LiDAR data and the resulting map. The Braunschweig dataset contains data from a rural area. It covers 15km which were driven at medium speeds of up to 66km/h. The surroundings have differing amount of structures that are measured by the LiDAR. Finally the Borgfeld dataset contains a drive through a city scenario. The dataset covers 4.5 km at lower speeds of up to 51 km/h. The surroundings have rich structures and produce comparably dense LiDAR scans. The choice to use self-recorded data was made for commonality with the work presented in Chap. 4 and Chap. 6. However as the evaluation shows the results do not depend on the evaluated scenario, indicating that the results are applicable to other datasets with similar sensor setups as well.

All data was recorded using the demonstrator vehicle presented in Chap. 3. To provide the state estimation of the vehicle the odometry localization presented in Sect. 4.2 is used as a UKF. For the implementations of the data structures the array is realized using the vector implementation of the standard template library (STL), while hashmaps are implemented using an unordered map, also from the STL. The quadtree does not have a standard implementation and is thus self-implemented. All evaluations were performed on an Intel Core i7-7700K CPU with 32GB of RAM. The algorithms run single threaded. The size of the map is set to $s = 350$m and the cell resolution is chosen as $\delta = 0.2$m. The size of the submaps is varied in the evaluation as it is a driving factor in the differences between the different combinations. The data provided by the LiDAR provides data on three layers at once in 3D, the vertical field of view is however very small and as such the 3D information is discarded in this work by projecting the points to the horizontal 2D plane. In the following the evaluated map operations are presented, followed by the evaluation on the hierarchical map structure presented here. Finally an additional evaluation is performed for comparison, where the presented data structures are used as a single map holding all cells at once.

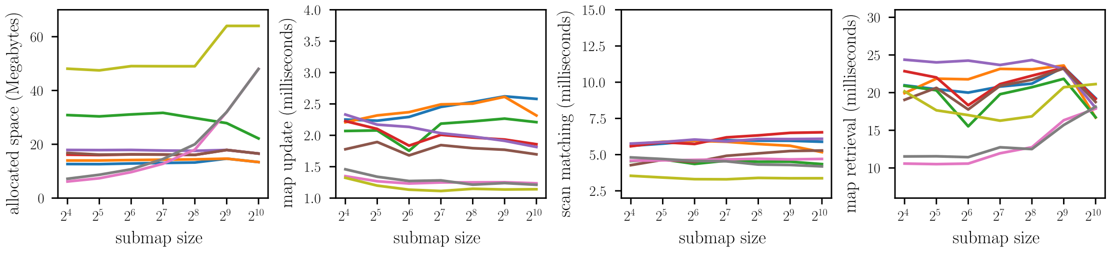
The evaluated operations on the map are meant to reflect typical tasks in autonomous driving. The incoming LiDAR data is first used in a scan matching step, which is meant to align the LiDAR data with the map in order to refine the state estimation. For this a gradient search with a beam endpoint model is used [Thrun et al., 2005, Sect. 6.4]. In this evaluation the scan matching is only used to obtain the performance measures and is discarded afterwards without influencing the state estimation. The second evaluated task is a map update using the new scan. This is done as described in Sect. 5.2. Moving the active submap grid is seen as part of the map update in the evaluation. For the array this is simply done by swapping all submap pointers to their new position and clearing the submaps at the border, erasing any need for costly memory allocation. For the hashmap and quadtree the outer storage needs to be rebuilt as their internal structure changes significantly with the shift. The last evaluated operation is the map retrieval. This is the task of obtaining all used cells from the map. Unused cells are not retrieved as they do not contain any information. Each step is timed and the runtime compared. Additionally the memory consumption of the different data structure combinations is calculated. First the evaluation is performed on the hierarchical map. The results are shown in Fig. 5.5.

As expected the use of arrays leads to a very high memory consumption. On the other hand they excel in the map update and scan matching task. Especially the use of arrays as the inner data structure where the cells are stored results in fast processing in these two tasks. When combining arrays on both layers a very fast map update and scan matching is performed, however the map size is significantly larger compared to all other combinations. The sudden increase in allocated space between submap sizes 2^8 and 2^9 is caused by the ceil function in (5.4), which results in a large difference between D and $\hat{D}$ in this case. While this combination is very fast for the map update and scan matching, map retrieval is comparably slow. This is caused by the full allocation of the map. While only the used cells are retrieved, in the array all cells are allocated. To decide whether they are used each cell must be accessed, slowing down the map retrieval significantly. To reduce the map size and allocate memory in a more sensible matter hashmaps and quadtrees can be used, where only relevant cells or submaps are allocated, depending on the layer where the data structure is used. This is visualized in Fig. 5.6. The reductions in memory consumption are clearly visible in Fig. 5.5, however the resulting processing speed is reduced compared to using an array.

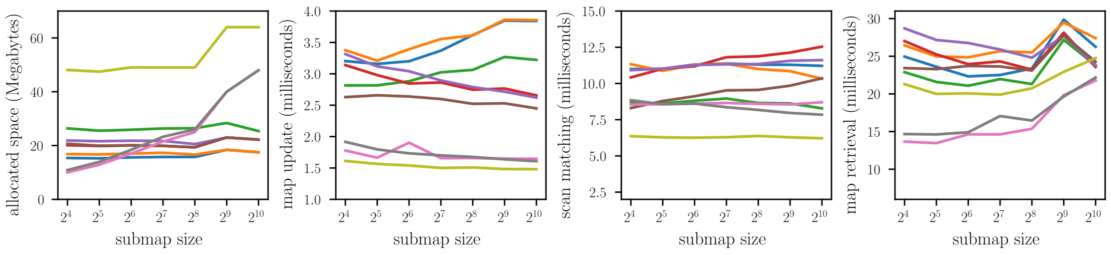
Especially the combinations of quadtrees and hashmaps on the outer layer with arrays on the inner layer were predicted to be promising combinations. They combine fast cell access on the lower level with a deliberate allocation of the used submaps, which saves a lot of space depending on the submap size. The evaluation shows that ideal behavior is achieved around a submap size of 2^6 where the small submaps lead to an almost ideal allocation of cells with little unused overhead. Here the speed of the map update and



(a) Highway



(b) Braunschweig



(c) Borgfeld

Figure 5.5: Allocated space as well as map update, scan matching, and map retrieval speed for different datasets. Each plot shows the median value of all time steps. (Figure adopted from [Wellhausen et al., 2021].)

scan matching are close the the array+array combination, although slightly worse, which is expected. Especially in the map retrieval the combination shows its strengths, which is much faster compared to all other combinations when small submap sizes are used. This is caused by the fast cell access on arrays on the inner layer, while only having allocated a small amount of unused cells that need to be iterated, significantly reducing the overhead compared to the array+array combination.

As a comparison the mapping was evaluated on the same scenarios by using a single-layer map where the entire map is stored in one data structure without the use of submaps. The evaluation was performed on the Borgfeld dataset, as the other two datasets cover an area that is too large to represent in a single map. For the Borgfeld dataset a map of $D = 16384^2 = 268435456$ cells was used, with a cell size of $\delta = 0.2$m.

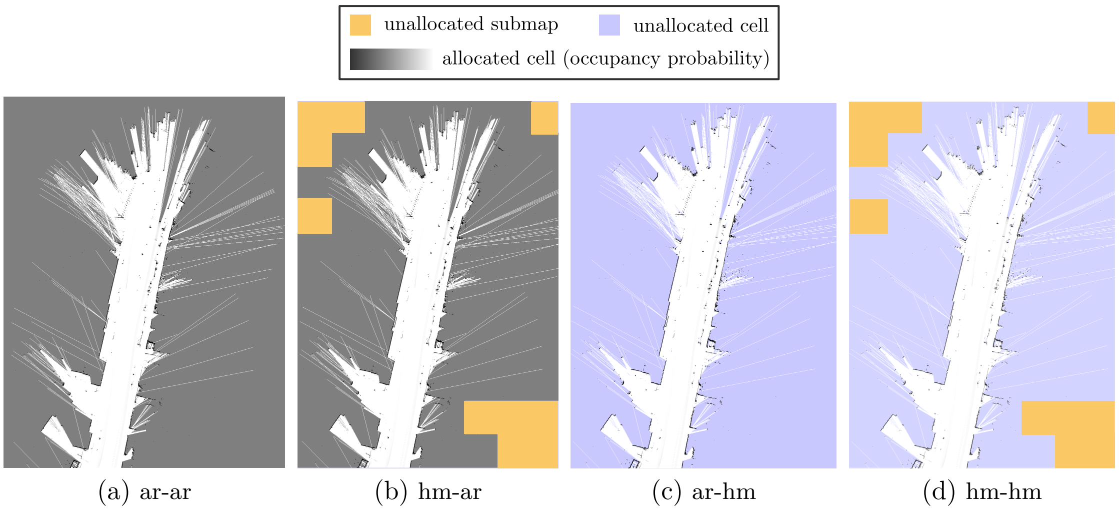


Figure 5.6: Comparison of allocated submaps and cells with different data structures. For better visualization, the figures only show a section of the map and not the whole square window. In case of allocated cells, black corresponds to a high and white to a low occupancy probability. (Figure adopted from [Wellhausen et al., 2021].)

Table 5.4: Results for a single map instead of a hierarchical one for the Borgfeld dataset (median values over all time steps). (Table adopted from [Wellhausen et al., 2021].)

	array	**hashmap**	**quadtree**
allocated space (MByte)	4 096	171.67	221.57
update (ms)	1.49	4.09	2.65
scan matching (ms)	5.21	8.47	10.41
map retrieval (ms)	708.73	297.16	255.89

This roughly covers an area of 3km^2. The results are shown in Tab. 5.4. The performance of the array as a single map is comparable with the performance in a hierarchical map. This is likely due to the array access using close to no computational resources. The driving factor here is likely other computational overhead, and thus no differences are visible.

For the quadtree the computation time is comparable to using quadtrees in a hierarchical map with larger submaps. In the hierarchical map the outer layer needs to be rebuilt whenever the window is moved. This produces a significant overhead which is reduced for larger submaps, thus resulting in more similar execution time for larger submaps. Overall the submap approach slightly outperforms the single map approach due to the lower depth of the quadtrees in this case.

For the hashmap the map update speed is slower in a single map organization, while scan matching is performed faster. In a single hashmap simple cell access would be expected to be faster as only one access to the hashmap is required as opposed to two. However the map update not only accesses the map but also allocates new cells, which may result in allocation of new buckets in the hashmap. This causes elements to be redistributed between buckets, which is computationally expensive for large maps. On the other hand the scan matching does not allocate new cells. This operation is thus faster in a single map approach.

In addition Tab. 5.4 gives the memory allocated in the final map as well as the average time for map retrieval. These values are however largely meaningless in this evaluation as the map is much too large to work with and are only given for completeness. Among the evaluated data structures the hashmap is the only one that could in theory be used for arbitrarily large areas as it is automatically extended when new elements are inserted. Both the array and the quadtree have pre-defined maximum sizes. However as the single map evaluation shows the hashmap approach becomes less efficient especially in the map update with a larger map size and should thus also not be used as a single map.

5.3.4 Recommendation

From all evaluated data structure combinations three of them stood out to be useful in certain tasks. The combination of arrays on both layers outperforms all other approaches in simple processing speed of new data. This comes at the cost of a high memory demand and a resulting slow map retrieval. Using quadtrees or hashmaps on the outer layer instead with arrays on the inner layer creates a compromise between processing speed and map retrieval speed, while also using significantly less memory. The usability of these different combinations depends on the use case in which the map is utilized. In most cases a map is built which is retreived and sent to subsequent algorithms for further processing. In this case a combination of a quadtree or hashmap with an array on the inner layer is preferable. If the mapping is meant to be performed without regular retrieval of the map, for example for building a reference map for later use, using arrays on both layers produces the best performance.

While these findings were generated without any parallel processing, in general the read-only cell access performance of the different data structures would be similar in a parallelized mapping approach. Write operations when using parallelization need to ensure synchronized access of the cell as well, which again does not change the findings. There are however advantages of using arrays for parallelization, especially when the cells accessed by each thread are disjunct. First, the required memory in an array is already allocated, while hashmaps and quadtrees allocate memory on demand which needs to be synchronized between threads, resulting in potential delays in execution. Additionally in an array the memory is allocated in a single block for each submap. This speeds up data transfer between the CPU and GPU, which is often a limiting factor in GPU programming. To further decrease the impact of the data transfer a larger submap size should be used to increase the size of the data blocks that are transfered at once. A combination of hashmaps or quadtrees on the outer layer with arrays on the inner layer may additionally reduce the overhead caused by the data transfer, as it causes less unused submaps to be allocated, which do not need to be transferred.

5.3.5 Conclusion

In this work different data structures for grid mapping in large-scale environments were evaluated, with a focus on applications in autonomous driving. To enable mapping in large scenarios a moving-window approach was chosen which only focuses on the immediate surroundings of the vehicle. A world-fixed map is created, however the map is only calculated and kept in a fixed window, which is moved with the vehicle roughly in its center. This is done by utilizing a hierarchical map structure where the map is divided into submaps. As the vehicle moves, submaps behind the vehicle are continuously discarded, while new ones are allocated in the front of the vehicle.

All coordinate transformations between the navigation frame, the global grid map storing submaps and the submaps storing the cells are given in this work. In the evaluation three different data structures are compared for use in this hierarchical map, namely a 2D-array, a hashmap and a quadtree. As its main contribution this thesis evaluates the performance of each combination of these data structures in a hierarchical map, both on a theoretical level and in practical use. The theoretical analysis of the different data structures covers their time and space complexity as well as possible overheads, both for isolated use and for combinations in a hierarchical map. The practical evaluation was performed on multiple real-world datasets and evaluates the performance of all different combinations of data structures in a hierarchical map. This was evaluated on typical map operations that are relevant to autonomous driving, namely the map generation, scan matching and map retrieval. In addition the memory consumption was evaluated. The results showed multiple interesting combinations of data structures, with arrays being the best for the inner layer. On the outer layer arrays perform best when the map is not retrieved regularly. However when this is required the hashmap and the quadtree outperform arrays on the outer layer as long as small submaps are used. The moving-window grid mapping approach with the proposed data structures is actively used in the autonomous driving stack on the research vehicle presented in Chap. 3.

6 Object Detection and Tracking

Another crucial functionality required for autonomous driving is the detection and tracking of other traffic participants within the vicinity of the vehicle. While the map created in Chap. 5 determines whether areas are traversable or not, no information is contained about the underlying dynamics of the surroundings. In fact, most approaches do not consider dynamics at all and instead make the assumption of a static environment, leading to inconsistent estimates when dynamic objects are measured [Reineking and Clemens, 2014]. Therefore dynamic objects are handled separately in this work.

For handling dynamic objects two tasks need to be solved. First, the objects need to be detected. This can be done in any number of different sensors, where each sensor measures different aspects of the object. In this work, object candidates are generated using LiDAR, Radar, Camera and V2X data. While for the first part of this section the object detection will be seen as a black box, with each sensor detecting objects independently, in Sect. 6.3 different object detection algorithms for the LiDAR are explored to avoid the reliance on the black box detections.

After detection the object candidates with their different properties need to be fused into a single consistent estimate and their object state including the pose, size and dynamics must be estimated. This is also referred to as object tracking or multi-object tracking. There are multiple stages at which this fusion may take place. The simplest case is a so called late fusion, where the objects are detected completely independently and then combined and fused together. No information is exchanged between the sensors during object detection, making the detections independent from each other. The fusion is performed at the latest possible stage in the sensor processing chain, thus coining the name late fusion. On the other end of the spectrum an early fusion may be used for object detection and tracking. Instead of detecting objects with each sensor individually, the data from multiple sensors is combined directly within a common representation. Using this approach the object detection may leverage much richer features that were recorded using multiple sources. This can lead to more accurate object detections, which in turn improves tracking performance [Gadzicki et al., 2020]. The distinction between early and late fusion is however not rigid. A mixed approach can be used in order to allow accurate object detection on a richer representation while other sensors may perform object detection separately [Malawade et al., 2022]. In this work however a late fusion is utilized. This allows for fusion of arbitrary sources of information, whereas in an early fusion including new information sources would often require significant changes to the fusion algorithm. However a mixed approach is evaluated in Sect. 6.3.2, where the LiDAR points from different sensors are fused into a joint representation early and Radar data could be included as well.

In the following first the state-of-the-art of object detection and tracking is given in Sect. 6.1. Secondly the late fusion approach is presented in Sect. 6.2. Finally the object detection approaches are given in Sect. 6.3.

6.1 State-of-the-art

Object detection and tracking has been an important field of research since long before the emergence of autonomous vehicles. In recent years the interest in this field has grown further especially with recent improvements in deep learning technology. While the focus of this work is on classical approaches, in the following an overview will be given on both classical and deep-learning based approaches. First the current state-of-the art on tracking approaches is presented, while recent advancements for LiDAR object detection will be given thereafter. The overview focuses on LiDAR-based algorithms as the vehicle is equipped with six LiDARs that allow full coverage of the environment, making this the main sensor for object detection. Camera-based detection is out of the scope of this thesis and only seen as a black-box instead.

6.1.1 Multi-Object Tracking

The field of multi-object tracking has been researched thoroughly and is a comparably mature field of research in autonomous driving, with many of the utilized technologies such as Kalman filtering [Kalman, 1960] or Monte Carlo sampling-based approaches [Liu and Chen, 1998] being used for many years. However there are still advancements in the field. An overview over multi-object tracking approaches is given in the following, with a focus on extended object tracking, where the tracked object is represented to have a spacial extent as opposed to point tracking where only single points are considered. Especially in autonomous driving the extent of the tracked objects is a crucial property, making extended object tracking the more suitable approach.

In [Luo et al., 2021] an overview over multi-object tracking approaches is given. This work focuses on probabilistic approaches, as uncertainty in sensor measurements is a crucial information and including this in the state estimation has significant advantages when the noise is estimated well. In addition probabilistic approaches give an uncertainty measure for the resulting state estimate which can be used by subsequent algorithms and provides very valuable information. Variants of the Kalman filter are among the most frequently used methods in object tracking and have been used in a large number of works surrounding object tracking in autonomous driving. In [Chen and Shao, 2021] an EKF is utilized for visual tracking of vehicles in an image. In this work intersection over union is used for determining which detected vehicle belongs to which tracks, while a Kalman filter is used to predict the state of the vehicle forward. Upon successful association the state estimate is corrected using the associated measurement. In [Guo and Zhao, 2022] an adaptive cubature Kalman filter (CKF) is used for tracking in autonomous driving, where the objects are detected using deep learning on an image. After detection the object is fused with the current state estimate to obtain a smooth estimate which takes into account the motion model of vehicles. In fact there are countless object tracking approaches that utilize different variants of the Kalman filter, such as [Steyer et al., 2017; Mobus and Kolbe, 2004; Zhang et al., 2022] to name a few more. The Kalman filter allows for both the motion of the vehicle as well as the measurement of the vehicle state to be modeled precisely. Both of these processes are well defined and as such they can be modeled with high precision, making the Kalman filter a precise tool for object tracking. Due to this they are utilized in this work as well.

There are however other approaches also suitable for multi-object tracking, such as the particle filter. This is for example used in [Fang et al., 2019] for tracking of vehicles on a road and [Santos et al., 2019] for tracking of an unmanned aerial vehicle. However a particle filter is generally less computationally efficient than a Kalman filter which is a significant consideration in autonomous driving where computational power is limited. With the Kalman filter already being suitable for efficient state estimation the particle filter is therefore not used in this work.

Further advancements in the field of multi-object tracking were achieved through the use of random finite sets (RFS) [Mahler, 2007] which allows modeling the uncertainty of associations between object tracks and measurements. This is used in the Probability Hypothesis Density (PHD) filter [Mahler, 2007], where the PHD is the first order moment of an RFS and models an intensity function over the state space. The PHD is propagated in time, allowing track association in dynamic and cluttered environments. This approach is highly useful in these scenarios when point tracking is performed, where track association is a highly difficult task. However in this thesis extended object tracking is considered, where the object association problem is significantly simpler to solve as the extent of the objects carries additional information which can be utilized for association. In addition in autonomous driving there is a natural distance being kept between traffic participants which further simplifies the association problem. Therefore the PHD filter is not considered in this thesis.

6.1.2 LiDAR Object Detection

While LiDAR object detection was traditionally often done in 2D due to the available LiDAR sensors measuring only in one dimension, this has shifted in recent years due to emerging LiDAR technologies enabling measurements on many layers. As such recent research on 2D object detection is limited when compared to 3D object detection, however it is still utilized in many scenarios due to reduced computational demands and reduced sensor cost. In [Chen et al., 2020] a deep-learning based 2D detection algorithm is proposed which uses a Region proposal Convolutional Neural Network (RCNN) for detection. In the approach a pseudo-image is generated from the point cloud by projecting it to the ground plane and discretizing it into an image. On this image regions of interest are detected by a CNN, which correspond to other traffic participants. In addition the authors propose a second object detection network which does not rely on the pseudo image representation but rather uses the raw LiDAR points, which is much faster to process due to the sparseness of the data. It is based on ResNet [He et al., 2016], although with some changes to achieve real-time performance. The models in this work were trained on a self-recorded dataset, as no suitable public dataset is available for 2D object detection on LiDAR data.

For 3D object detection using deep learning significantly more research exists, which is driven by a number of available public datasets such as KITTI [Geiger et al., 2012], Waymo [Sun et al., 2020] and nuScenes [Caesar et al., 2020]. A comprehensive overview of deep-learning based approaches for 3D object detection is given in [Zamanakos et al., 2021]. Different representations for the point clouds are used to reduce computational load, although some approaches such as [Shi et al., 2019; Yang et al., 2019; Yang et al., 2020] utilize the raw or down-sampled point clouds directly for detection. Other approaches such as [Zeng et al., 2018; Barrera et al., 2020] project the data to a plane in order to reduce dimensionality of the data for detection, while the 3D information is recovered retroactively. These approaches could in theory also be utilized for 2D object detection, as they do not directly utilize the 3D data for detection, however the data provided by 3D LiDARs is still denser even when projected to a plane, giving the detection algorithm more features to work with. Additional approaches such as [Zhou and Tuzel, 2018; Shi et al., 2023; Lang et al., 2019] utilize voxels or pillars for geometric abstraction of the point cloud data in order to reduce computational load. Additionally in [Team, 2020] an object detection tool is given for easy evaluation of different object detection implementations which supports multiple state-of-the-art implementations and supports multiple public datasets. As this overview shows much research is done in the field of 3D object detection and this field is still evolving quickly. However even with many publicly available datasets the training data available is limited and most object detection algorithms restrict the amount of different object labels that are supported. To ensure high detection performance in all driving situations a vast amount of training data is required, while only limited data is currently publicly available. In combination with the lack of explainability and with no comprehensive data sets available of 4-Layer LiDARs built into the sides, front and rear of the vehicle a conscious choice was made to not utilize either 2D or 3D object detections based on deep learning.

While current research is heavily focused on 3D object detection using deep learning, some recent works still utilize classical approaches for this task, both in 2D and in 3D. For 3D object detection [Sualeh and Kim, 2019] propose to use ground removal, clustering and box fitting in order to detect other vehicles in the environment. The work utilizes multiple 3D LiDARs mounted on various locations on the top of the vehicle. The object detection is paired with a tracking approach using the interacting motion model unscented Kalman filter (IMM-UKF) to track the vehicle. By utilizing an IMM different models can be assumed for different motion patterns, increasing the tracking accuracy. In [An et al., 2019] a 4-layer LiDAR is utilized, however the data is reduced to two dimensions and detection is performed on contour information in the point cloud data. For this a contour is extracted from the measured points and made into object hypotheses, which are then tracked by using dynamic time warping in order to find associations between contours from frame to frame. While the shows promising results, it is developed for the detection of other cars. However especially in urban driving the detection of bikes and pedestrians is a crucial task, both of which do not have clearly defined contours when measuring them with a LiDAR. As such an approach solely relying on contour information is unsuitable for this task.

In [Guo et al., 2019] a feature-based approach is proposed. For this a clustering is performed on the available 4-layer LiDAR data, which is again projected to a 2D plane. On these clusters one or two lines are fitted and depending on the number and orientation of the lines the objects are either classified as a vehicle, a bicycle or a pedestrian. While this approach performs well on the evaluation data, it strongly relies on a large number of fine-tuned parameters and it is unclear if it would be possible to add more object classes in the future. As the resulting object detection framework would be very difficult to extend this approach is not explored further in this work despite its strong performance on the evaluation data set.

In [Steyer et al., 2017] a 2D object detection approach is presented which utilizes evidential dynamic maps that estimate the dynamics of the surroundings of the ego vehicle and use this information for generating object candidates. For this both LiDAR and Radar data is utilized in order to estimate the dynamics for each cell in a grid map. This is done using particles, where each particle represents potential dynamics of the cell. The inclusion of dynamics in the object detection task is an interesting approach, as this contains much information about the object and may aid in distinguishing between objects and other structures, which is often not a trivial task on 2D LiDAR data. Therefore this approach is further investigated for the use when only LiDAR data is available in Sect. 6.3.

6.2 Object Tracking using Late Fusion

Object tracking is performed using a $\boxplus$-Kalman filter similar to the filter introduced in Sect. 4.2. In fact the underlying estimation problem is very similar in nature, with the goal being the estimation of a vehicle pose as well as its dynamics. Therefore, some similar models are used, with the distinction that the state of the vehicle is strictly measured from external sensors. The state to be estimated is defined as

$$x_t = \begin{bmatrix} r_t \\ \psi_t \\ s_t \\ v_t \\ a_t \\ \omega_t \end{bmatrix}, \tag{6.1}$$

with $r_t \in \mathbb{R}^2$ denoting the position of the vehicle in 2D and $\psi_t \in SO(2)$ representing the rotation around the z-axis of the vehicle (yaw). The symbol $s_t \in \mathbb{R}^2$ represents the size of the object represented by a rectangular bounding box, while $v_t \in \mathbb{R}^2$ denotes the body-fix velocity of the vehicle. Lastly, $a_t \in \mathbb{R}^2$ and $\omega_t \in \mathbb{R}$ denote the body-fix acceleration and turn rate of the vehicle respectively. The state space is thus defined as

$$\mathcal{S} = \mathbb{R}^2 \times SO(2) \times \mathbb{R}^2 \times \mathbb{R}^2 \times \mathbb{R}^2 \times \mathbb{R}. \tag{6.2}$$

Note that a choice was made for the object tracking to estimate the vehicle state in 2D instead of using the 3D state as introduced for localization. This choice was made based on the available data for object detection. With the LiDAR only measuring with a vertical FOV of 3.2 degrees, very limited 3D information is available. This is however not enough to confidently measure an object height. In fact for most tasks no 3D information is required. With the other traffic participants moving on a roughly flat road, they can be assumed to always be moving at the same height as the ego vehicle.

Another change to the state vector is the addition of the size of the vehicle, which is unknown when estimating other traffic participants. A case can be made that this is not a property that needs to be filtered, as this is a fixed value and once the vehicle was fully observed it is known and does not change. However, with the LiDAR giving comparably sparse measurements it is not always possible to determine the correct size initially. Instead it is viewed as an uncertain property and filtered over time. However, some measures need to be taken in order to avoid shrinking objects. This is discussed in Sect. 6.2.3. For the Kalman filter again the motion model as well as the measurement models are presented in the following. The motion model is defined as

$$f(x_t) = \begin{bmatrix} r_t + R_t v_t \cdot \Delta t \\ \psi_t \boxplus \omega_t \cdot \Delta t \\ s_t \\ v_t + a_t \cdot \Delta t \\ a_t \\ \omega_t \end{bmatrix}, \tag{6.3}$$

with R_t describing the 2D rotation matrix resulting from the $SO(2)$ rotation defined by ψ_t and Δt denoting the time since the last update. The state transition noise M_t is defined as

$$\bar{M}_t = \Delta t \begin{bmatrix} \hat{r}_t \\ \hat{\psi}_t \\ \hat{s}_t \\ \hat{v}_t \\ \hat{a} \\ \hat{\omega} \end{bmatrix}, \tag{6.4}$$

with $\bar{M_t}$ describing the principal diagonal of the covariance M_t. The corresponding Jacobian for the motion model is given by

$$J = \frac{\partial f}{\partial x} = \begin{bmatrix} I_{2\times2} & \frac{\partial f_r}{\partial \psi} & 0_{2\times2} & \frac{\partial f_r}{\partial v} & 0_{2\times2} & 0_{2\times1} \\ 0_{1\times2} & 1 & 0_{1\times2} & 0_{1\times2} & 0_{1\times2} & \Delta t \\ 0_{2\times2} & 0_{2\times1} & I_{2\times2} & 0_{2\times2} & 0_{2\times2} & 0_{2\times1} \\ 0_{2\times2} & 0_{2\times1} & 0_{2\times2} & I_{2\times2} & I_{2\times2}\Delta t & 0_{2\times1} \\ 0_{3\times7} & & & & I_{3\times3} & \end{bmatrix}. \tag{6.5}$$

In this Jacobian the non-trivial derivations are given by

$$\frac{\partial f_r}{\partial \psi} = \begin{bmatrix} -\sin(\psi)\cdot v_x - \cos(\psi)\cdot v_y \\ \cos(\psi)\cdot v_x - \sin(\psi)\cdot v_y \end{bmatrix} \cdot \Delta t \tag{6.6}$$

$$\frac{\partial f_r}{\partial v} = R_t \cdot \Delta t, \tag{6.7}$$

where $R_t \in SO(2)$ is the 2D rotation matrix corresponding to the rotation defined by ψ_t. With the motion model defined, the estimated object can now be predicted forward in time according to (6.3). Note that this model is not fully applicable to pedestrians who may present more unpredictable behavior. Modeling pedestrian behavior is however out of the scope of this work.

To now fuse different sources of information about the tracked objects into one consistent estimate, the measurement models for different information sources are given in the following. For such a fusion to work it is however important that the state estimation and the detection algorithms are aware with respect to which reference point the objects are described. The handling of the reference point is thus explained first.

6.2.1 Reference Point

The choice and handling of the reference point of a detected or tracked object is crucial in the object state estimation task. An incorrect choice of a reference point may lead to jumping position estimates, incorrect sizes and incorrect dynamics estimates. An example of the fusion of objects with different reference points is given in Fig. 6.1, where it can be seen that an incorrect choice of reference points may lead to an incorrect shift in the position estimate. In this example an object is observed fully by the red box and partially by the green box. If the reference point is chosen in the center the fused estimate takes into account both reference points, with the new estimate lying between the two reference points. This falsely results in a shift in position, with the blue box now covering areas that were observed by neither measurement. Choosing the bottom right as a reference point instead the result is positioned correctly.

It is therefore important to specify a reference point whenever a 2D object is repre-

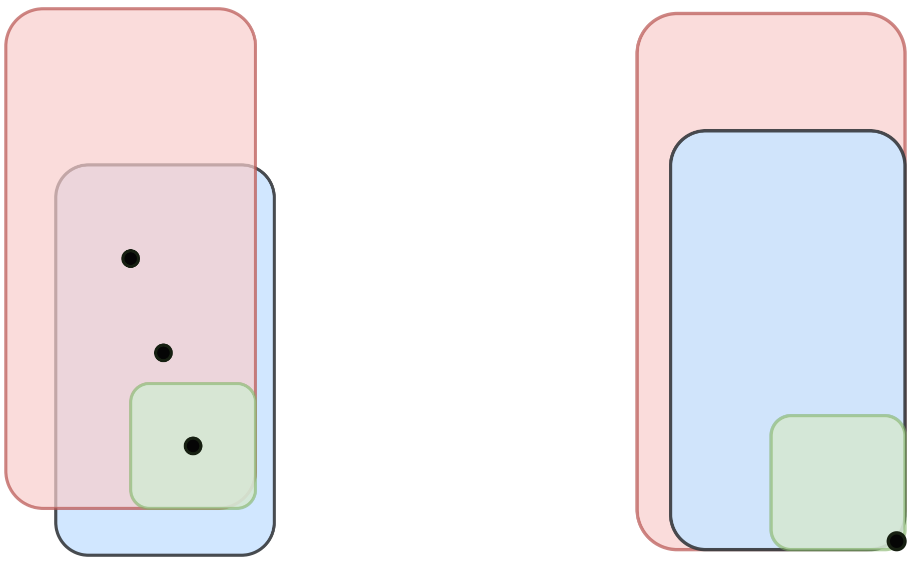


Figure 6.1: Fusion of object estimates with incorrect reference points (left) and with correctly set reference points (right). Two measured objects (red and green) are fused to obtain a combined estimate (blue). Reference points are shown as black dots.

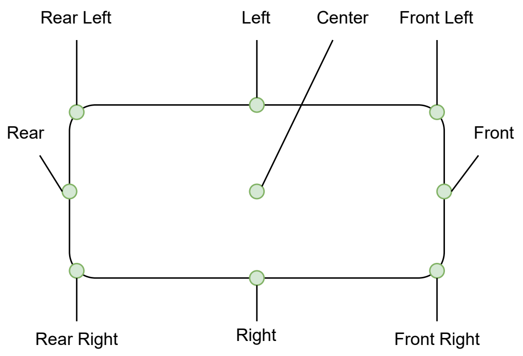


Figure 6.2: Possible reference points on a 2D bounding box.

sented. The choice of the best reference point is always dependent on the current scene and what parts of the object are observed. Therefore multiple options must be available to choose from for the reference point. These are shown in Fig. 6.2. To choose from these points, many approaches exist. In the simplest case the distance of the sensor to each reference point on the object is calculated and the point with the smallest distance is chosen as reference point. The idea behind this is that the closer the point is to the sensor the better it is observable. This is however not the case when occlusion is considered. In that case it may happen that closer parts of an object are partially occluded, making the chosen reference point unreliable. Instead the choice of the reference point can be aided by the mapping approach presented in Chap. 5. To determine whether measured object borders around a potential reference point reflect the real object boundaries or whether they are estimated incorrectly due to occlusion, a check can be performed on the cells adjacent to the perceived object boundaries. Since the occupancy grid map contains information about how often a cell was traversed or measured recently, it can be used to determine whether the area next to the boundary was measured to be free. If it was measured in the LiDAR scan to be free the object boundary can be observed in that position and it can be used as a reference point. If it was not measured it is currently occluded and a different reference point must be chosen. Using this method a correct reference point can be chosen for an object detected by a LiDAR.

6.2.2 LiDAR Object List

Using a late fusion approach any object source can be fused into the state estimate, as long as the measurement model can be formulated. This includes detections that are not generated as part of the sensor fusion stack but in external software as a black box. This is the case for the Scala LiDAR sensors built into the research vehicle as shown in Sect. 3.4. The built-in object detection measures full object state estimates including a pose, size and velocity estimate. These estimates are meant to be used as is and are specifically designed and optimized for object detection on large roads at higher speeds, such as highways. The detection performance in urban areas is therefore significantly impaired. In addition to the low detection performance of a single scanner, no inherent way of combining the estimates from multiple scanners is offered by the scanners. This feature is available for purchase which includes building additional hardware into the vehicle. This is however not included in the demonstrator vehicle setup and as such the combination of the different measurements must be performed in the sensor fusion. To overcome these weaknesses of the detected objects, the estimated LiDAR object states are fused using a late fusion. It allows to use all objects measured by any scanner as measurements, and by combining them strengthens the detection performance, as some scanners may generate better estimates than others. For this first an object association is required, which is described in the following.

Object Association The objects given by the scanners are already tracked and the measurement contains a unique tracking id. This is however only unique within one scanner. When data from multiple scanners is combined, a mapping $m : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ is required, which maps from a scanner id and the unique object id within that scanner to a unique filter id within the sensor fusion stack. In this mapping, multiple pairs of scanner ids and object ids can map to the same filter id, in order to associate different tracks from scanners to the same filter. The mapping is obtained by first checking the intersection of the two objects. If either object is sufficiently covered by the other one they are associated to the same filter id. To keep computations simple this is checked by evaluating whether the center point of either object lies within the bounding box of the other. This effectively means that at least a quarter of the object is contained within the other object, which is a reasonable threshold to assume that the object can be associated. To check whether the center point $c1^V$ of the first object $O1$ is contained in the bounding box of the second object $O2$, the center point is first transformed from vehicle coordinates to the coordinate system of $O2$. This is done by the simple transformation

$$c1^{O2} = T^{V \to O2} c1^{V}, \tag{6.8}$$

where $T^{V \to O2}$ is the transformation obtained by combining r_t and ψ_t of $O1$ into a transformation matrix. With the center point of $O1$ now expressed in the coordinate system of $O2$ the check whether it is within the object boundaries simply reduces to

$$\text{inside}(c1^{O2}, O2) = \left(-\frac{s2_x}{2} > c1^{O2}_x > \frac{s2_x}{2}\right) \wedge \left(-\frac{s2_y}{2} > c1^{O2}_y > \frac{s2_y}{2}\right), \tag{6.9}$$

where $s2$ is the size estimate of the second object. This check is performed in both directions to cover cases where a smaller object is contained within a significantly larger object. There are however cases where the size of one or both objects could not be determined correctly by the detection algorithm, for example when only the front was visible in the sensor data. In this case the size in one dimension will be close to zero and overlap is less likely. To still associate objects in these cases a distance threshold is applied on the center point. Any objects closer than the threshold are associated

$$\text{close}(c1, c2) = \|c1 - c2\| < t_{cl}, \tag{6.10}$$

with t_{cl} being the distance threshold. The choice of threshold is difficult as there is a significant difference between reasonable values for vehicles and pedestrians. As with the motion model it is however not as important to perfectly model pedestrians, partially because they move comparably slow and are easy to avoid without estimating them perfectly. Another reason is that detecting pedestrians in the sparse data available from the LiDAR is extremely challenging and as such the focus is shifted away from this task. Therefore some wrong associations of pedestrians in a group are tolerated. As a consequence for this thesis a threshold of $t_{cl} = 1\text{m}$ was chosen. To determine whether two objects are associated, both checks are performed as

$$\text{associated}(O1, O2) = \text{close}(c1^{V}, c2^{V}) \vee \text{inside}(c1^{O2}, O2) \vee \text{inside}(c2^{O1}, O1). \tag{6.11}$$

When no association is found a new Kalman filter is initialized with the current LiDAR measurement as its estimate. When an association to a filter is found the measurement is instead fused into the existing state estimate which is described in the following.

LiDAR Object Fusion To fuse objects into the state estimate, first the measurement is defined by

$$z_t^o = \begin{bmatrix} r^V \\ \psi^V \\ s^O \\ v^O \\ a^O \\ \omega^O \end{bmatrix}, \tag{6.12}$$

where r^V represents the position of the reference point, while ψ_t^V represents the orientation, both given in the vehicle frame. Additionally, s^O, v^O, a^O and ω^O represent the size, velocity, acceleration and turn rate in the object frame respectively. To fuse the measurement it is important that the reference point of the measured object and the state estimate are the same. Therefore, the reference point of the state estimate is shifted to match the reference point of the measurement. It is important to perform the change in this direction, as this ensures that the reference point is observable in the current scan. To perform this change on the object state it is passed through the function

$$h^c(x_t, t_t) = \begin{bmatrix} T_t \cdot t_t \\ \psi_t \\ s_t \\ v_t \\ a_t \\ \omega_t \end{bmatrix}, \tag{6.13}$$

where T_t is the transformation matrix corresponding to the estimated object pose, while t_t is the offset from the previous reference point to the new one. Here the covariance is zero as there is no uncertainty added during the process and the Jacobian is given by

$$J^c = \frac{\partial h^c}{\partial x} = \begin{bmatrix} I_{2\times 2} & \frac{\partial f_r}{\partial \psi} & 0_{2\times 2} & 0_{2\times 2} & 0_{2\times 2} & 0_{2\times 1} \\ 0_{1\times 2} & 1 & 0_{1\times 2} & 0_{1\times 2} & 0_{1\times 2} & \Delta t \\ 0_{7\times 2} & 0_{7\times 1} & I_{7\times 7} & & & \end{bmatrix}. \tag{6.14}$$

The derivation $\frac{\partial f_r}{\partial \psi}$ is given by

$$\frac{\partial f_r}{\partial \psi} = \begin{bmatrix} -\sin(\psi) \cdot v_x - \cos(\psi) \cdot v_y \\ \cos(\psi) \cdot v_x - \sin(\psi) \cdot v_y \end{bmatrix}. \tag{6.15}$$

With both the measurement and the state estimate expressed in the same reference point they can be combined in the filter. The measurement function $h^o : \mathcal{S} \rightarrow \mathcal{S}$ is defined as

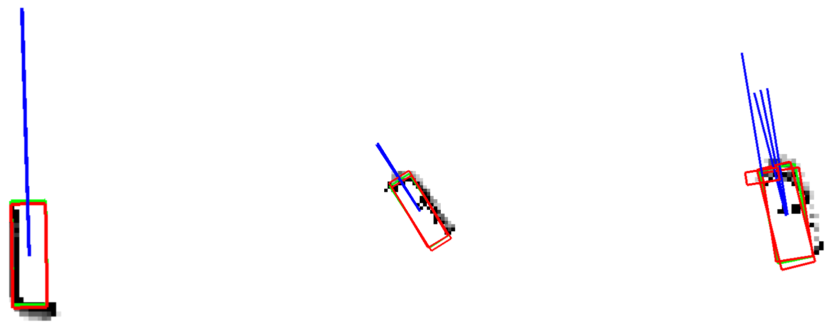
$$h^o(x_t) = \begin{bmatrix} T^{V\rightarrow N} \cdot r_t \\ \psi_t \boxminus \|R_z^{V\rightarrow N}\| \\ s_t \\ v_t \\ a_t \\ \omega_t \end{bmatrix}, \tag{6.16}$$

where $T^{V\rightarrow N}$ is the transformation from navigation to vehicle frame obtained by the odometry localization presented in Sect. 4.2 and $\|R_z^{V\rightarrow N}\|$ is the 2D rotation angle which corresponds to the rotation around the z-axis of the ego vehicle in the navigation frame. The measurement is assumed to be affected by additive, normally distributed noise with covariance Q_t^o and zero mean, which is estimated by the LiDAR. The Jacobian of the measurement function is given by

$$J^o = \frac{\partial h^o}{\partial x} = \begin{bmatrix} R_z^{V\rightarrow N} & 0_{2\times 8} \\ 0_{8\times 2} & I_{8\times 8} \end{bmatrix}, \tag{6.17}$$

where $R_z^{V\rightarrow N}$ is the 2D rotation matrix representing the rotation of the ego vehicle in the navigation frame. With the motion model and the measurement model for LiDAR objects defined the object tracking framework is already functional. In fact it can be used with any detection algorithm that measures the object state directly and is not reliant on a specific object detection. However for a first test the fusion is performed on the detected objects given by the black box implementation of the Scala LiDARs.

The evaluation is performed qualitatively to showcase scenarios in which the object detection and tracking work well and also where they fail to produce good results. In Fig. 6.3 the fusion of three detected objects into a state estimate is shown. In this figure a vehicle is detected by multiple LiDAR sensors, each estimating a different object state. All detections are fused into the state, with the resulting estimate shown in green. In Fig. 6.3a the simplest case is shown where only a single LiDAR measures the object. this is fused with the previous estimate, creating the current state estimate. In Fig. 6.3b a slightly more complex case is shown, where two LiDARs measure the same object. As can be seen in this figure, both detections are correctly associated with the same filter, leading to a single estimate for both measurements. In Fig. 6.3c a case is shown where the object is measured by three different sensors, however one sensor only partially observes the object. As can be seen all objects are correctly associated to the same state estimate with no second state estimate being generated for the partial measurement. The resulting state estimate reflects measured properties from all three measurements. The partial measurement however does not reflect the actual size of the object. This may happen when only a part of the object is visible, for example when



(a) One object fused into the estimate (b) Two objects fused into the estimate (c) Three objects fused into the estimate

Figure 6.3: Object detections and resulting state estimate. Detected object in red, tracked object in green. Note that the green bounding box is partially covered by the red bounding boxes due to similar values in the measurement and the result.

it is measured on the edge of the FOV of the LiDAR. In the naive model presented above this would lead to a reduction in the estimated object size, which is however not a logical explanation for the measurement, as the object was previously measured to be larger and no traffic participants that are to be estimated in this work can shrink. Therefore, significant reductions in the size of the measurement compared to the estimate are handled separately in the filter. This is explained in detail in the following.

6.2.3 Object Size

As mentioned in Sect. 6.2 the size of an object is seen as part of the state in this work and as such is estimated probabilistically by the Kalman filter. Including the size in the state has clear advantages. The size of an unknown traffic participant is an uncertain variable and is measured with some uncertainty which can be taken into account in a probabilistic state estimator. Additionally by including it in the state no extra mechanism is required for size estimation, which simplifies the process as well as the maintainability. Difficulties arise when an object is measured twice with significantly different sizes, as shown in Fig. 6.3c. In a standard Kalman filter, the probabilistic combination of the two measurements would result in an estimate with a reduced size. With the object size specifically this is however not the desired result, as objects are not expected to shrink and the largest measured object size is the best current estimate excluding measurement noise. To reflect this within the filter the measurement model is modified in cases where a measurement is significantly smaller than the estimate. For this a maximum allowed shrinkage $\theta_s \in [0, 1]$ is defined, which determines how much smaller the measurement can be compared to the state. This is given by

$$\text{smaller}(x, z) = x_x \cdot \theta_s > z_x \wedge x_y \cdot \theta_s > z_y, \tag{6.18}$$

where x_x and x_y refer to the x and y size of the current state, while z_x and z_y denotes the size of the current measurement. Note that even if one dimension is large enough to be included in the measurement the check still rejects the entire measurement. This is due to the fact that measurements that measure an incorrect size in one dimension are generally less reliable in estimating the other dimension as well.

To exclude the size from the state estimation problem, first a reduced measurement is defined as

$$z_t^{or} = \begin{bmatrix} r^V & \psi^V & v^O & a^O & \omega^O \end{bmatrix}^T . \tag{6.19}$$

Secondly, the size is removed from the measurement function, which is defined as

$$h^{or}(x_t) = \begin{bmatrix} T^{V \to N} \cdot r_t \\ \psi_t \boxminus \|R_z^{V \to N}\| \\ v_t \\ a_t \\ \omega_t \end{bmatrix} , \tag{6.20}$$

Lastly, both the covariance and the Jacobian are reduced to reflect the changes in the measurement. For this, the matrix $M^R \in \mathbb{R}^{8\times10}$ is defined as

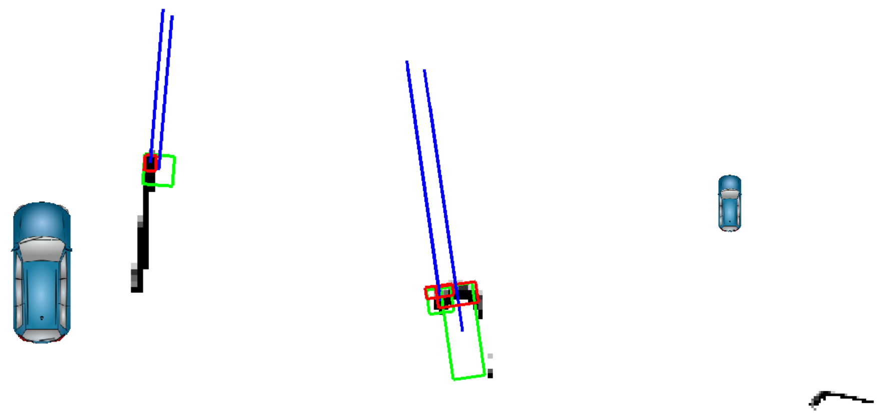
$$M^R = \begin{bmatrix} I_{3\times3} & 0_{3\times2} & 0_{3\times5} \\ 0_{5\times3} & 0_{5\times2} & I_{5\times5} \end{bmatrix} , \tag{6.21}$$

with the reduced covariance defined as $Q_t^{or} = M^R Q_t^o (M^R)^T$ and the Jacobian defined as $J^{or} = M^R J^o$.

With this version of the measurement update the size is disregarded, while the pose, velocity, acceleration and turn rate are still updated. This allows the tracking system to handle cases as shown in Fig. 6.3c, where one of the associated detections was only partially observed. Often such partial observations have a lower detection quality compared to detections where the full L-shape of the vehicle is visible. Excluding such partial measurements is however not desired, as this would often lead to tracks being lost when the object is partially observed for an extended period of time.

The object tracking system that results from the given motion and measurement models performs well in cases where the objects can be well observed by the LiDAR. Especially when the L-shape is visible a detection is given with a very high accuracy. There are however many cases where the object detection built into the LiDAR fails to detect objects, misdetects their properties or gives false positives or negatives.

Some such cases are shown in Fig. 6.4. In Fig. 6.4a an example is shown where the downside of a late fusion comes into play. The object that is to be detected is right next to the ego vehicle, and its entire side is visible. With the detection being performed on each LiDAR individually the object is however not seen as a whole in any scan. Instead it is split, resulting in a small object detected at the front, and no detection at all in the



(a) One partially detected object, resulting in incorrect object bounds

(b) Two detected objects without sufficient overlap, resulting in two tracked objects

(c) An object behind the car that is not detected by the LiDAR

Figure 6.4: Examples of misdetections made by the LiDAR. Detected object in red, tracked object in green.

back. In Fig. 6.4b an example is shown where the object association was unsuccessful. Instead of obtaining a single track for both detections, two tracked objects are created. This is a drawback of the association function as defined in (6.11). Neither center point is contained in the other track, and the distance threshold is not met either. The result is however not critical to the systems safety, as both objects have correctly estimated dynamics. Over time as the tracks overlap more they will be merged and create a single consistent object again. Lastly, Fig. 6.4c shows one of the cases where an object is seen fully but not detected. This shows a significant drawback of the presented approach, as it fully relies on a blackbox object detection with no way of including missed detections like this. While the presented approach gives a good baseline to work with, to improve tracking performance an improved object detection implementation is required which is explored further in Sect. 6.3.

In addition to an improved object detection a multi-sensory approach is explored in this work. In the following the additional sensors are introduced and their measurement models given. This includes a front-facing camera, a radar and information obtained by V2X communication.

6.2.4 Camera

A camera is the ideal sensor to use in combination with LiDARs to obtain additional information about surrounding traffic participants. While a LiDAR gives highly precise location and shape information in a 3D world frame, it is difficult to classify objects solely based on the geometric data contained in a scan. While object classification based

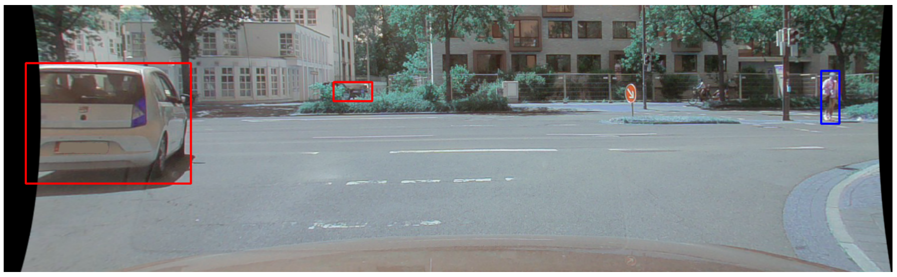


Figure 6.5: Example output of the camera-based object detection overlaid onto the image. Red boxes represent cars while blue boxes show pedestrians.

on LiDAR data is out of the scope of this thesis, the camera provides this information for objects within its field of view, which gives subsequent algorithms valuable additional information. In addition to difficulties with object classification, object bounds are often not clear in LiDAR data. When two objects are close together it becomes impossible to determine whether a single or multiple objects are contained in a scan, as there is no color information and as such no texture available from which to derive boundaries. A camera excels in these tasks. Object detection and classification in camera images is a topic that has been very thoroughly investigated and is still an active area of research. With the recent surge of deep learning-based approaches the performance of camera based object detection and classification has greatly improved, making them essential in the recent advancements in the field of autonomous driving. While the camera-based object detection is out of the scope of this work, the fusion of such objects into the state estimate via late fusion is investigated. By fusing the camera estimate with other estimates such as the LiDAR detections, the aforementioned strengths of the camera are utilized, while its main weakness of not containing 3D information is mitigated by the LiDAR detection. Similar to the LiDAR object detection, for the late fusion the camera detection is seen as a blackbox. The result is a 2D bounding box in the camera image for each object, including a classification of the object. An example of an object detection can be seen in Fig. 6.5. In addition a confidence is given for each detection to be an object. The contained information is highly valuable for object tracking, however to make it usable for the filter is difficult as the camera measures in 2D on the image plane. Connecting this to the vehicle coordinate system requires a calibration of the camera both intrinsically and extrinsically, which is described in the following.

Intrinsic Calibration The intrinsic calibration determines the distortion parameters to correct distortion in an image. These effect are introduced by the light entering the camera through a lens, which is called radial distortion or fish-eye effect, and by the sensor not being aligned with the lens, called tangential distortion. In addition to the

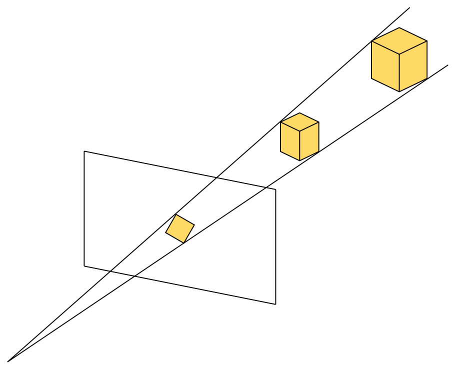


Figure 6.6: Visualization of the ambiguity of the size of an object seen in an image.

distortion parameters a camera matrix C is determined, which is defined as

$$C = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} . \tag{6.22}$$

This matrix contains four parameters. The parameters f_x and f_y represent the focal length (or more precisely the image width) which give the distance of the image plane to the optical lens in pixels. For square pixels it will hold that $f_x = f_y$, while any non-uniform pixels will result in different focal lengths in each dimension. The parameters c_x and c_y on the other hand describe the center point of the image, which determines where the focal point lies on the image sensor. This value is given in pixels as well. Using this camera matrix, a point in 3D is projected to the 2D image plane by

$$\begin{bmatrix} p_x \\ p_y \\ w \end{bmatrix} = C \cdot \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} , \tag{6.23}$$

where p_x and p_y are the resulting pixel coordinates and w is a scale factor. As this is a projection from a 3D space to a 2D space, information is lost which is impossible to recover without additional information. This in turn means that information given in the 2D space, such as object bounding boxes, can not be converted to 3D space directly without making some assumptions. Instead, a pixel on the camera plane can be seen as a ray in the 3D world. Somewhere along this ray lies something that reflected the light which was measured by the sensor. This means that scale in the world can not be determined from a camera image. This is visualized in Fig. 6.6. This shortcoming makes working with camera detections less intuitive than e.g. LiDAR detections, where the object state is measured directly. There exist many implementations of an intrinsic camera calibration, with one of the most commonly used implementations being the

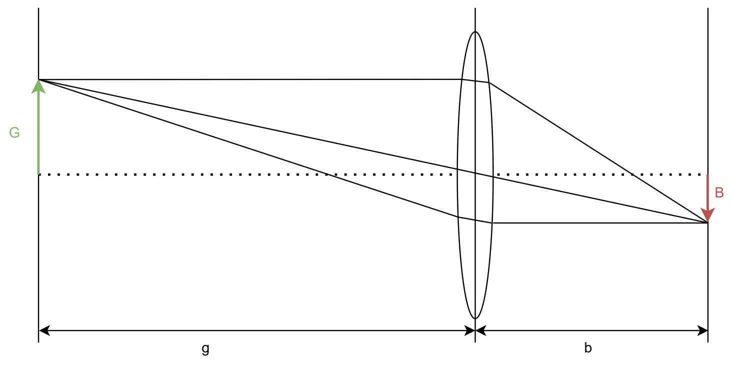


Figure 6.7: Visualization of the parameters of the lens equation.

camera calibration in OpenCV [Bradski, 2000], which determines all the aforementioned parameters.

Extrinsic Calibration In addition to the intrinsic calibration an extrinsic calibration is needed, which determines the pose of the camera within the vehicle. Such a calibration is required for the LiDAR sensors as well, however in the demonstrator the LiDAR sensors' poses were determined by the manufacturer. Determining the pose of the camera is a difficult task. Simply measuring it is not an option as even minor errors in the orientation would lead to large positional errors when measuring distant objects. This is caused by the lever-arm between the camera and the object, which is highly sensitive to rotational changes. Instead a calibration process is needed in order to obtain an accurate camera pose. This is however made difficult by the fact that the camera measures significantly different data than other available sensors with a known pose like the LiDAR. One way of combining the two sensors is to detect a common object in both sensors. This gives the transformation from the LiDAR to the camera frame by solving the equation $T^{L \to C} = \left(T^{C \to O}\right)^{-1} T^{L \to O}$. However, with the camera measuring in 2D and the LiDAR measuring in 3D, detecting a common object in 3D requires a specially prepared object. While it is not generally possible to retrieve 3D information from a 2D image, this can be circumvented by detecting objects of a known size in the world. The reason this is possible is shown in figure Fig. 6.7. When measuring an object in the world generally only the image distance b and the size of the object in the image B are known. On the other hand the distance of the object from the camera g and the size of the object in the world G are unknown. From the intercept theorems [French, 2004, Chap. 7], the relationship between these values can be described by

$$\frac{G}{g} = \frac{B}{b}. \tag{6.24}$$

Thus if the size of the object in the world is known, the distance of the object to the sensor can be determined. This can be utilized to determine the location of points in the world, and in fact, when using certain markers like ArUco markers [Garrido-Jurado et al., 2016], a full 3D pose can be estimate in a camera image. Therefore, for detecting the objects pose from an image, an ArUco marker is attached to a known position on the object and detected, yielding the transformation $T^{C \to O}$.

On the other hand the object needs to be detected in LiDAR data as well. To perform this detection the object is placed on a flat surface. A cube is used as an object, which means that when estimating the pose of it on a flat surface, only the yaw needs to be determined, as it inherently does not have any roll or pitch on a flat surface. For yaw estimation a line is fitted on the measured points of the object, which has one side facing the sensor. This gives the full orientation of the object, while the position is given by the leftmost measured point on the object front, with the z-position being zero. With this algorithm the pose of the object is detected in the LiDAR frame, given the missing transformation $T^{L \to O}$. The full transformation $T^{C \to L}$ can now be used to make the camera information usable by the tracking algorithm. In addition, as the transformation $T^{L \to V}$ is known, this calibration indirectly connects the camera frame to the vehicle frame through the transformation $T^{C \to V} = T^{L \to V} T^{C \to L}$.

Camera Fusion There are two options for using 2D camera detections to update the filter. The first option is to estimate an object in the 3D world from its 2D bounding box representation. While this is the more intuitive solution it requires assumptions that introduce large uncertainties in the filter update. The second is to project the object track into the image plane using the camera matrix to estimate how it should look within an image. While this is more unintuitive, this fits well in the Kalman filter framework, as the measurement function transforms the current state estimate to the measurement space, which is exactly what is done for the transformation from object tracks to the image. The filter thus calculates what the camera would have measured if the current estimate was correct, using the deviation from the actual measurement to update the state estimate. Therefore in this work this approach is used.

As the object has an unknown and possibly complex shape, determining its 2D bounding box in an image cannot be done exactly. As an approximation a 2D bounding box in world coordinates is used, which is also the representation of the object in the filter. This shape fits well for other vehicles, while it only roughly approximates pedestrians or bikers, however this is acceptable. To update the filter using this method the measurement function is explained in the following. As with the LiDAR object update, first an object association is performed in order to determine which measurement by the camera corresponds to which object track. To perform this association a measure for similarity is required for comparing 2D bounding boxes with the estimated object tracks. While for the LiDAR object association this could simply be performed by detecting overlapping objects or checking simple closeness, for the camera it becomes more difficult as there are more ambiguities and the measurement is in a different space. Therefore, an association score is calculated for every pair of camera detection and object track. From all combinations the best are selected as associations. The score is calculated by projecting the tracked object into the camera image and calculating how similar its bounding box would be to the measured box. For this the corners of the object are transformed to the camera coordinate system using the extrinsic calibration by

$$c_i^C = T^{V \to C} c_i^V , \tag{6.25}$$

with $i \in [1, .., 4]$ determining the corner of the object. The corner is then projected to the 2D image by

$$c_i^I = f \cdot \begin{bmatrix} c_x + \frac{c_{i\,x}^C}{c_{i\,z}^C} \\ c_y + \frac{c_{i\,y}^C}{c_{i\,z}^C} \end{bmatrix} . \tag{6.26}$$

The equations for determining the left, right, top and bottom boundary of the 2D bounding box is given by

$$\begin{aligned} bb_l &= \min(c_{1x}^I, ..., c_{4x}^I), \\ bb_r &= \max(c_{1x}^I, ..., c_{4x}^I), \\ bb_t &= \min(c_{1y}^I, ..., c_{4y}^I), \\ bb_b &= \max(c_{1y}^I, ..., c_{4y}^I). \end{aligned} \tag{6.27}$$

With the boundaries defined, a similarity can be calculated. For this the overlap between two bounding boxes is calculated. However, as the height of objects is unknown and cannot be estimated correctly given the available LiDAR data, the height of the bounding box estimated in (6.27) will be incorrect. On the other hand the estimated width will reflect the object well. Therefore, the overlap between two objects $o1$ and $o2$ is only calculated along the x-axis by

$$\text{intersection}(o1, o2) = \max(0, \min(o1_r, o2_r) - \max(o1_l, o2_l)), \tag{6.28}$$

where $\times_l$ is the left pixel boundary of the object while $\times_r$ is the right one. The overlapping area is however not a suitable score to determine the similarity of two bounding boxes. Instead, the overlap is compared to the total area as a percentage of overlap. The total area is calculated by

$$\text{totalArea}(o1, o2) = \max(o1_r, o2_r) - \min(o1_l, o2_l), \tag{6.29}$$

which is then used to calculate the percentage overlap by

$$\text{score}(o1, o2) = \frac{100}{\text{totalArea}(o1, o2)} \cdot \text{intersection}(o1, o2). \tag{6.30}$$

To now obtain the best object association any optimization technique may be used. Here, every filter may be assigned one or zero detected bounding boxes with which the filter is updated. The total score over all associations is calculated and maximized. This can either be done by optimization or simply by calculating the score for all options. An example of an association result is shown in Fig. 6.8. Note that while there are three vehicles in the scene the black box LiDAR object detection was only able to detect the

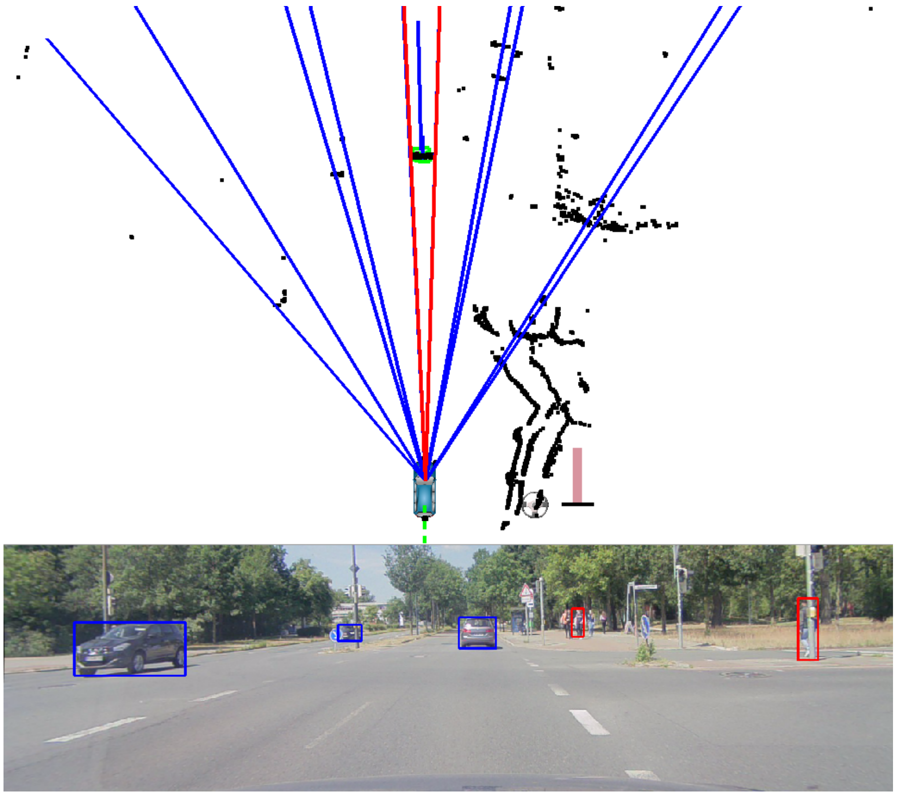


Figure 6.8: Camera and LiDAR object detections of the same scene and their association. Tracked object shown as a green bounding box in the top. Detected camera objects are shown as bounding boxes in the camera image as well as projected to the world in the top, visualized as angular ranges by blue lines. Red lines visualize the angular range where an object association was found.

vehicle directly in front. Thus only one object association was found as there was only one tracked object in the scene.

After obtaining the object association, the detected bounding boxes are used to update the filter. For this, the measurement z_t^c is defined as

$$z_t^c = \begin{bmatrix} p_l & p_r \end{bmatrix}, \tag{6.31}$$

where p_l and p_r are the left and right pixel coordinate of the bounding box in the image. The measurement function is calculated very similarly to the object association. It projects the estimated state into the camera image and calculates the image bounding box from them. Accordingly, the measurement function is not trivial to define. It iterates over all corners, projects them onto the image and calculates the bounding box according to (6.25) - (6.27). The measurement is assumed to be affected by additive, normally distributed noise with covariance Q_t^c with zero mean. Determining the Jacobian J^c for this measurement function is not trivial as there are many parts to the measurement function. In fact only parts of the measurement function are derivable due to the usage of the min and max function. To obtain this derivative, numerical derivation around the current state x_t is performed to obtain the Jacobian.

The result is a tracking system which is able to use the data from a camera for two purposes. Firstly, by associating the camera data with the available tracked objects the classification performed on camera data can be used for them, which provides valuable information for subsequent algorithms. Secondly, the state estimate can be updated using the camera data. Upon evaluating the tracking result clear downsides of the measurement update became apparent however, and as such it is not actively used as part of the sensor fusion stack. The update allows for changes in the state in all dimension, which gives the filter too much freedom in updating the state to match the measurement. As an example, the size of an object may be falsely estimated, however the filter instead moves the reference point of the state closer or further away from the camera in order to change its apparent size in the image. To circumvent this the update could be constrained to the size, however even here caution must be taken as the dimension in which the size can be increased must be constrained as well. An object that is estimated too small can have its size altered either in x- or y-direction and both will eventually cause the object to appear the same size as the measurement. The only direction in which the object size should be increased is perpendicular to the viewing direction of the camera, which matches the orientation of the image plane. However constraining the Kalman filter update to this dimension is out of the scope of this work and will be considered in future work instead.

6.2.5 Radar

The Radar is a very useful sensor in the context of autonomous driving. The data it provides complements that of LiDARs and Cameras well, as it provides information about the dynamics of the sensors surroundings. Combining this with the highly precise positional information provided by the LiDAR and the detailed texture information obtained by the camera allows for very detailed estimation of the entire object state. An ideal sensor setup for autonomous driving would include all three sensors, and the demonstrator used for this work includes a Radar as well, however it only provides limited information compared to currently commercially available Radars. The Radar is built in by the car manufacturer and no access is available to the sensor data. Instead only preprocessed information is available containing up to four objects in the vicinity as well as their velocity in x-direction of the ego vehicle. This data is fused with the tracked objects as part of the late fusion, however the Radar system is designed for highway scenarios and as such objects often remain undetected. In Sect. 6.3 an attempt is made to recover the dynamics information that would be obtained by a Radar only from LiDAR point clouds, however directly measuring these properties is beneficial in any case. In the following the measurement function for the preprocessed Radar data as provided by the demonstrator vehicle is given.

The Radar measures the point in the world $p_t \in \mathbb{R}^2$ where an object was detected and its relative velocity $\hat{v}_{t,x} \in \mathbb{R}$ along the x-axis of the ego vehicle. As such the measurement is defined as

$$z_t^r = \begin{bmatrix} p_t \\ \hat{v}_{t,x} \end{bmatrix}. \tag{6.32}$$

The measurement is assumed to be affected by additive, normally distributed noise with covariance Q_t^r with zero mean. For the measurement function it is not fully defined which point on the tracked object is measured by the radar, as the state estimate contains a full bounding box while the radar only measures a point and a velocity. As such the closest point on the rectangle describing the tracked object is calculated and a large uncertainty is assumed for this part of the measurement. To obtain the closest point, all positions of corners on the tracked object are calculated and their distance to the origin of the ego vehicle is calculated. The corner with the lowest distance is kept and its position in x-direction is used as the position in the measurement function. To obtain the set of corners $\mathcal{C} = \{c_1, ..., c_4\}$ the following function is used, which gives them in the vehicle coordinate system:

$$\mathcal{C} = \text{corners}(x_t) = \left\{ T_t^{O\to V} \begin{bmatrix} s_x/2 \\ s_y/2 \end{bmatrix}, T_t^{O\to V} \begin{bmatrix} s_x/2 \\ -s_y/2 \end{bmatrix}, T_t^{O\to V} \begin{bmatrix} -s_x/2 \\ -s_y/2 \end{bmatrix}, T_t^{O\to V} \begin{bmatrix} -s_x/2 \\ s_y/2 \end{bmatrix} \right\}. \tag{6.33}$$

From this the closest point is simply determined by

$$\text{closest}(\mathcal{C}) = \min(\{\|c_1\|, .., \|c_4\|\}) \tag{6.34}$$

Note that while in theory the closest point of the object could lie between two corners, this is irrelevant as it is used to determine the minimal x-coordinate which will always be found in a corner. The measurement function is therefore defined as

$$h^r(x_t) = \begin{bmatrix} \text{closest}(\text{corners}(x_t))_x \\ \left(T_t^{N\to V} T_t^{O\to N}\right)_y \\ \left(T_t^{N\to V} v_t\right)_x - \bar{v}_{t,x} \end{bmatrix}, \tag{6.35}$$

where $\left(T_t^{N\to V} T_t^{O\to N}\right)_y$ gives the y-position of the center of the object in vehicle coordinates, v_t is the currently estimated velocity of the tracked object and $\bar{v}_{t,x}$ is the velocity in x-direction of the ego vehicle obtained by the localization. As with the measurement function for the camera measurements, this function has derivable parts, but it is not derivable analytically. Therefore, again numerical derivation is used to obtain the derivative around the current object state x_t.

The result is a method which directly updates the dynamics of the tracked object. An example of this can be seen in Fig. 6.9. In this scene three vehicles are tracked by the late fusion, however the radar only detected the one in front of the ego vehicle. The detected velocity is used to update the object, and as can be seen in the figure, the resulting velocity estimate closely resembles the dynamics estimated by the radar. Note

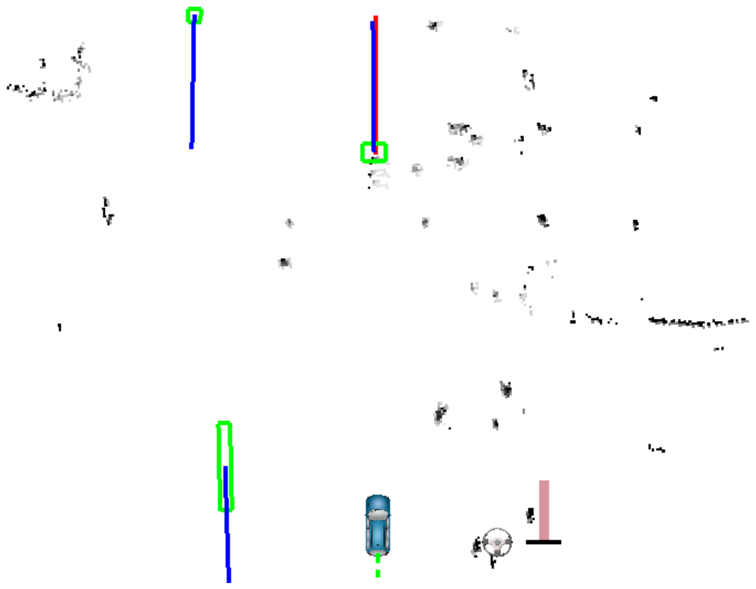


Figure 6.9: Tracked objects shown as green bounding boxes with their estimated velocity shown in blue. The velocity detected by the radar is shown as a red line originating from the point where it was detected.

that the size of both objects in front of the vehicle is not estimated correctly as they are only seen from the front or back and as such no length information is available.

6.2.6 V2X

One emerging approach to aid in object detection and tracking is to enable communication between the ego vehicle and other vehicles with varying degrees of intelligence as well as with surrounding infrastructure. This is made possible by V2X communication, which is introduced in Chap. 3. By enabling communication with any nearby V2X-capable objects, this technology offers a chance to make autonomous driving safer in the future. In fact many car manufacturers such as Volkswagen, Volvo and Mercedes are now building vehicles equipped with V2X technology [Volkswagen, 2020; ADAC, 2020]. As more and more manufacturers make use of this technology, the benefits for the safety of autonomous driving grow. While V2X communication cannot replace a robust and reliable object detection and tracking, any object that communicates its own state is a potential source for errors eliminated. Robust object detection especially remains relevant while this technology is not mandatory in all vehicles, however it also provides a layer of redundancy that would likely remain in autonomous vehicles after widespread adoption of V2X technology. The information being communicated as well as its quality varies depending on the sensors as well as the algorithmic implementation on the vehicle. It may vary from standard GNSS solutions with only meter-accurate positioning to a state estimate similar to the one presented in Chap. 4 which achieves centimeter-accurate positioning. Especially when standard GNSS is used, the accuracy can be low enough that further processing is required in order to correctly consider the other traffic participant in the ego vehicle control. Therefore, the data obtained from V2X is fused as part of the late fusion into the state estimate of the corresponding traffic participant. The corresponding measurement model is given in the following.

As mentioned before, the contents of the V2X message strongly depends on the sensor equipment of the vehicle. However, in the following first the case is presented with another vehicle with a full sensor setup which estimates its own full vehicle state. The reduced measurement update is given thereafter. The measurement z_t^v is defined as

$$z_t^v = \begin{bmatrix} \bar{r}_t \\ \bar{\psi}_t \\ \bar{s}_t \\ \bar{v}_t \\ \bar{a}_t \\ \bar{\omega}_t \end{bmatrix} . \tag{6.36}$$

To fuse the measurement into the existing state estimate, first an association is performed to find the corresponding tracks for each measurement. This is done using the closeness and overlap check in (6.11). Since objects obtained from V2X communication have a known source this association is only performed once, after which the filter and the V2X messages from that vehicle are permanently associated. If no association is found however, a new filter is created.

Before the measurement can be fused into the state the reference point of the filter is shifted to match the measurement. This is done as explained in Sect. 6.2.1.

As the measurement contains all parts of the state directly, the measurement function is simply defined by

$$h^v(x_t)X = x_t, \tag{6.37}$$

where the covariance Q_t^v is determined by the sender vehicle and zero mean is assumed. The Jacobian $J^v \in \mathbb{R}^{10\times 10}$ for the measurement function is simply given by the Identity matrix $J^v = I_{10\times 10}$.

To enable fusion of vehicles with less sophisticated equipment the measurement is reduced to only include a pose and the size of the object. This can be estimated by any vehicle simply using a GNSS antenna which is standard equipment for any V2X-capable vehicle, although the size of the vehicle must be configured.

The reduced measurement is therefore defined by

$$z_t^{vr} = \begin{bmatrix} \tilde{r}_t \\ \tilde{\psi}_t \\ \tilde{s}_t \end{bmatrix} . \tag{6.38}$$

Accordingly the measurement function is given by

$$h^{vr}(x_t) = \begin{bmatrix} r_t \\ \psi_t \\ s_t \end{bmatrix}. \tag{6.39}$$

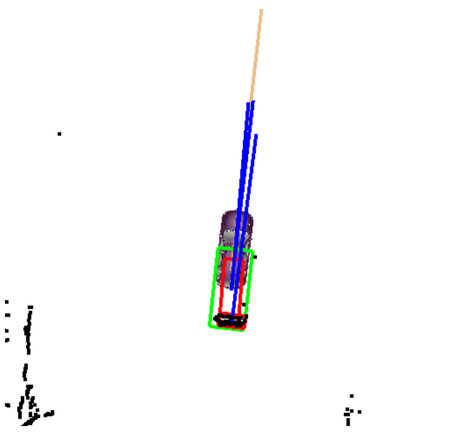
Here the covariance is not estimate by the other vehicle and as such the covariance Q_t^v is seen as a tuning parameter as no information is available about the quality of the solution. It is however assumed to have zero mean. The Jacobian is given by

$$J^{vr} = \begin{bmatrix} I_{5\times5} & 0_{5\times5} \end{bmatrix}. \tag{6.40}$$

With these measurement models V2X communication can be included in the late fusion seamlessly. Especially when using the reduced measurement model where the other vehicle is only equipped with a simple inaccurate GNSS antenna the multi-sensor fusion shows improvements in object tracking accuracy. This is shown in Fig. 6.10 where two examples are shown where a vehicle is detected by the LiDAR sensors, while it also publishes its own state using V2X communication. In Fig. 6.10a an example is shown where the LiDAR estimates the vehicle well, but the size is slightly too small. The object received using V2X communication on the other hand is not positioned correctly, while the size is correct. Through fusion a solution is found which better reflects all parts of the vehicle state. A similar behavior is seen in Fig. 6.10b, however here the LiDAR detection only detected the side of the vehicle and as such the size would not be reflected well using only the LiDAR measurement. The V2X position is again significantly shifted, however through fusion again an accurate object estimate is obtained. Another advantage of V2X communication is being able to detect objects that are not visible to the vehicle, such as objects behind a sharp turn or just below the highest point of an incline. Neither the Camera nor the LiDAR is able to detect such an object, as both require line of sight. This is not the case for V2X communication, and as such the objects would be considered in vehicle control even before they become visible, which significantly improves driving safety in certain areas. Overall the usage of V2X data for object tracking is a highly promising approach to improve reliability of the object tracking system. As the technology continues to improve and becomes more widely accepted, it has the potential to aid in the emergence of full self driving vehicles.

6.2.7 Conclusion

In this section a late fusion approach for multi-object tracking was presented. It estimates the state of each traffic participant using a $\boxplus$-Kalman filter. Motion and measurement models are presented including the corresponding Jacobian to allow implementation in both an EKF and a UKF. While the presented approach enables the fusion of many different sources of information in a modular and simple framework, it strongly relies on the quality of the underlying object detection from each sensor. This object detection was


(a) Fusion result with multiple slightly incorrect LiDAR objects and an incorrect V2X object position

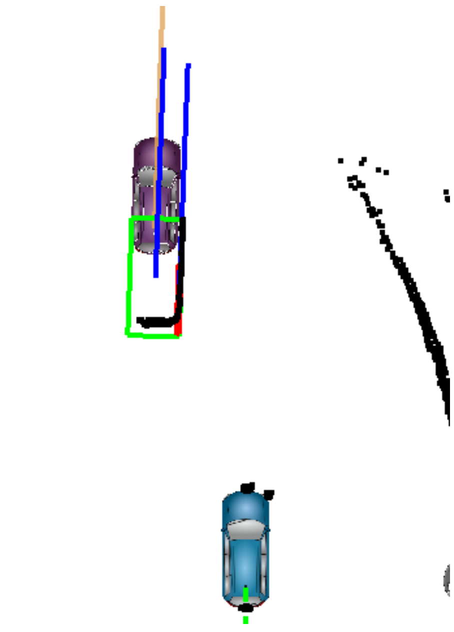
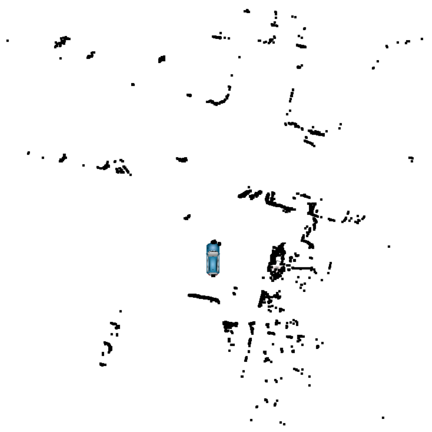

(b) Fusion result with an incorrect LiDAR object size and an incorrect V2X position

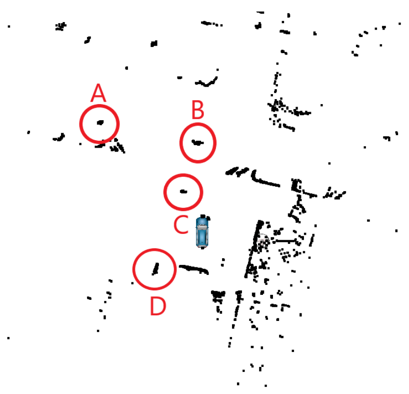
Figure 6.10: Tracking result by fusing V2X objects with LiDAR objects. Tracked objects shown in green, detected objects in red and the V2X objects are visualized as a purple car while the ego vehicle is shown as a blue car. The current LiDAR scan is shown in black as a reference.

seen as a blackbox, with the detections being directly used for tracking. This approach is acceptable for the Camera and Radar data as they are only used to refine the object tracks but not for initialization due to the nature of data contained in the measurements. However in the case of the LiDAR object detection, the overall performance of the system is strongly reliant on a high accuracy. Since the LiDAR objects are used to initialize object tracks, false positives and false negatives have a strong influence on the system performance. Unfortunately the built-in object detection of the LiDARs only performs well in certain scenarios, while many scenarios are not well covered by it. Therefore in the following section different algorithms are evaluated to improve the quality of the LiDAR object detection. The presented late fusion for multi-object tracking is actively used as part of the autonomous driving software stack on the research vehicle.

6.3 Dynamics and Object Detection

Object Detection on sparse 3D point clouds with only very few layers or even 2D point clouds is a difficult task even for humans. The LiDAR takes only a slice from the environment and represents it as a point cloud, which leads to many ambiguities and structures that are very hard to identify. One example of this is shown in Fig. 6.11a. While human


(a) Scene in which it is difficult to determine other traffic participants



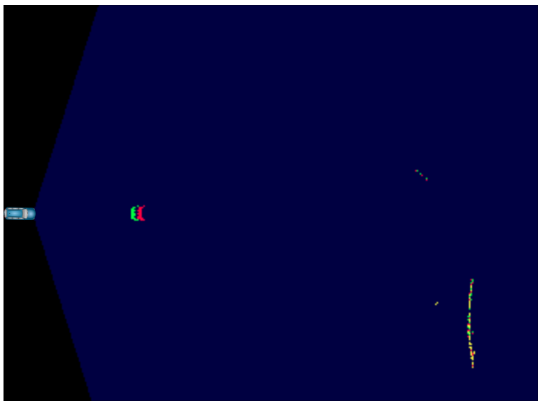
(b) Same scene a few frames later, dynamic objects circled in red

Figure 6.11: Visualization of the difficulty of object detection on the available LiDAR data. A and B are cars, while C is a pedestrian and D a bicycle.

performance is not a perfect comparison, the difficulty to detect these object for humans indicates that this problem is difficult to solve algorithmically as well. With very little structure available to use for classification it is difficult to distinguish traffic participants from static structures. Therefore a one-shot detection on a single LiDAR point cloud was not attempted in this work as it offers little chance of success. Instead this work focuses on determining areas of motion in the environment and using this additional information for object detection. To get an idea of the benefit of estimating motion in LiDAR scans, the scene from Fig. 6.11a is shown again in Fig. 6.11b after some time has passed. As can be seen most of the scene remained static between the scans excluding sensor noise, however there are some areas where movement can be observed, which is circled in red. For the vehicle shown in B the movement is very noticeable, while the movement of the pedestrian in C is already rather subtle, although still noticeable. Especially for the car in A and the bicycle in D it is noticeable that they are dynamic as they were not measured previously. As such, by taking into account multiple LiDAR scans motion can be detected directly from the scan data which in turn enables the detection of objects. In this work three different LiDAR-based approaches for dynamics estimation and object detection are presented and their applicability to the scenario of urban autonomous driving is evaluated.

6.3.1 LiDAR Optical Flow

To estimate motion in the environment of the vehicle a frequently utilized approach is to estimate optical flow on a camera image. By estimating how pixels in the image moved between two frames it can be inferred how the vehicle moved between the frames and how dynamic objects in the view of the camera moved. Optical flow on camera images is widely adopted in many different applications [Zhai et al., 2021], however for object


(a) Transformed scan points in the same coordinate system. Old scan shown in green, current scan in red. Associated points present in both scans shown in yellow

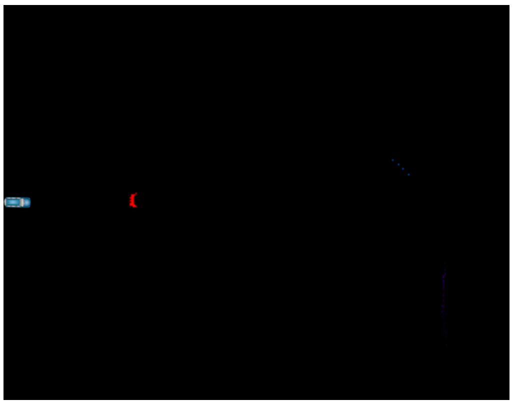

(b) Optical Flow calculated on LiDAR images. Color determines the direction while intensity shows the estimated velocity

Figure 6.12: Two LiDAR scans and the resulting optical flow between them. The ego vehicle position is shown as a blue car.

detection and tracking in autonomous driving it has some major drawbacks. First, it is difficult to differentiate between ego motion of the vehicle and motion of other traffic participants. Estimating how pixels are expected to move from frame to frame given the ego motion is possible to a certain extent. However, since the distance of objects in the image is unknown a certain inaccuracy is expected. Second, the motion is estimated in pixel coordinates in an image. This can be used to reconstruct movement in the world either by utilizing two frames to imitate a stereo camera or by making assumptions about the world, however both approaches add a large amount of uncertainty to the estimation process.

In this work the optical flow is therefore estimated on LiDAR data instead. For this the LiDAR data is mapped using a grid map with static cell size as well as a static map size for each frame. For the optical flow calculation the grid map is converted to an image. For this each cell is represented as a pixel with a binary value that determines whether the corresponding cell was hit by a scan point. This is done for two consecutive scans, resulting in two images that represent the current state of the vehicles surroundings. In contrast to the optical flow on camera data the ego motion can be compensated almost completely in this approach. Using the localization from Sect. 4.2 a precise transformation can be obtained between the two scans, resulting in static areas having no estimated flow. Moving traffic participants on the other hand will be tracked by the flow, resulting in high flow in areas where dynamic objects are contained. This is visualized in Fig. 6.12. As can be seen there are static areas in the surroundings that show little to no flow, while the objects that are moving in the scene are clearly visible. In the following the calculation of the flow and subsequent measures to extract object candidates and their properties are presented in detail.

First, in order to create images from the LiDAR scans, the previous scan $S_{t-1}^{L_{t-1}} = \{m_1, ..., m_N\}, N \in \mathbb{N}$ in the previous LiDAR frame L_{t-1} is transformed to the vehicle frame V_t at the time where the new scan was taken. This is done by the following transformation using the LiDAR extrinsic calibration as well as the ego-motion estimated by the localization

$$S_{t-1}^{V_t} = T^{N \to V_t} T^{V_{t-1} \to N} T^{L \to V} S_{t-1}^{L_{t-1}}, \tag{6.41}$$

where the transformation is applied to the scan by transforming each point individually. Second, the new scan is transformed from the LiDAR frame to the vehicle frame by

$$S_t^{V_t} = T^{L \to V} S_t^{L_t}. \tag{6.42}$$

After transformation both scans are in a common coordinate system as visualized in Fig. 6.12a.

After transforming the scan points they are converted to an image. For this an image of a static size is created. In this image each pixel represents an area of $o \times o$ centimeters. Depending on the resolution of the LiDAR this can be tuned, however for this implementation it was set to 20×20 centimeters. The total considered area of $w \times h$ meters is fixed depending on the needs of the object detection, where in this thesis an area of 70×55 meters was used. From these values the size of the image is calculated by

$$\begin{bmatrix} s_x \\ s_y \end{bmatrix} = \begin{bmatrix} \frac{w}{o} \\ \frac{h}{o} \end{bmatrix}. \tag{6.43}$$

In this image the vehicle frame is fixed in $\begin{bmatrix} 0 \\ \frac{h}{2} \end{bmatrix}$ with the x-axis of the vehicle frame aligned with the image x-axis. The scan points of both scans, which are now all expressed in the same vehicle coordinate system, are projected to their own image by discretization using the cell size o.

Next the area in which an optical flow is calculated is reduced to the area visible in both scans. This is calculated from the opening angle of the LiDAR and is also visible as a blue mask in Fig. 6.12a. It is calculated by drawing lines along the opening angle of the LiDAR using the Bresenham algorithm for line drawing [Bresenham, 1998]. Afterwards the rest of the area is filled using flood fill to obtain a mask for the observed area. This mask is used to discard any scan points that are not observable in both scans. Without their removal the flow estimation would attempt to match points from one scan where its counterpart in the second scan is not observed anymore, resulting in incorrect flow around the edges of the FOV of the LiDAR.

From the resulting images the flow is calculated using the Gunnar Farneback's algorithm [Farnebäck, 2003] as implemented in the image processing library OpenCV [Bradski, 2000]. The result can be seen in Fig. 6.12b. For each pixel a vector is calculated which represents its estimated movement. The direction as well as distance of movement in pixels is represented in this vector. From this information the velocity and

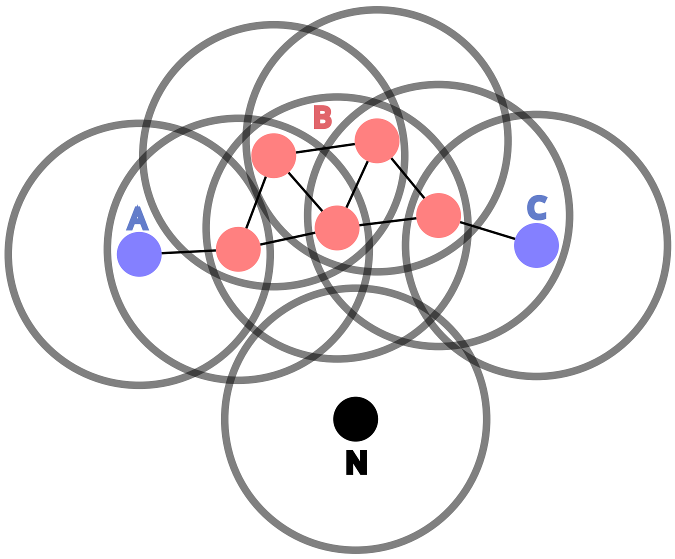


Figure 6.13: Neighborhood relations in DBSCAN with a minimum number of neighbors of 3 for core points.

direction of travel of the corresponding object can directly be inferred. This becomes more apparent in Fig. 6.14c where two vehicles travel in the same direction and are thus colored similarly, however one with less intensity due to its slower movement.

After obtaining the flow image containing information about movement of all parts of the surroundings, object detection can be performed. The flow already contains a lot of information which the object detection can utilize to create suitable object candidates. One modular approach is to identify clusters of points in the image with certain shared characteristics. In the simplest case this may be points that are close to each other, however some clustering algorithms allow for additional properties to be checked in order to define closeness of points. One such algorithm is the DBSCAN algorithm [Ester et al., 1996]. It is a density-based clustering algorithm which builds clusters wherever dense points are located. For this the distance between all points is checked and a threshold determined below which points are seen as neighbors. The algorithm is visualized in Fig. 6.13. When a point has enough neighbors it is flagged as a core point, marked in red in the image, and all points in its vicinity are added to the cluster. Points that are directly reachable from a core point but do not have enough neighbors are added to the cluster but not as core points, which is the case for A and C in the figure. Therefore the cluster is not extended from these points. Points that are not a neighbor of any core point are marked as outliers.

While in this example the neighborhood is visualized as spacial proximity it can be defined as any function. To utilize the dynamics estimated in the optical flow in this work neighborhood is defined on proximity as well as direction of the flow vector. The absolute velocity is not considered in the clustering as there are many cases where this is inaccurate, which would lead to an incorrect neighborhood calculation. Therefore the

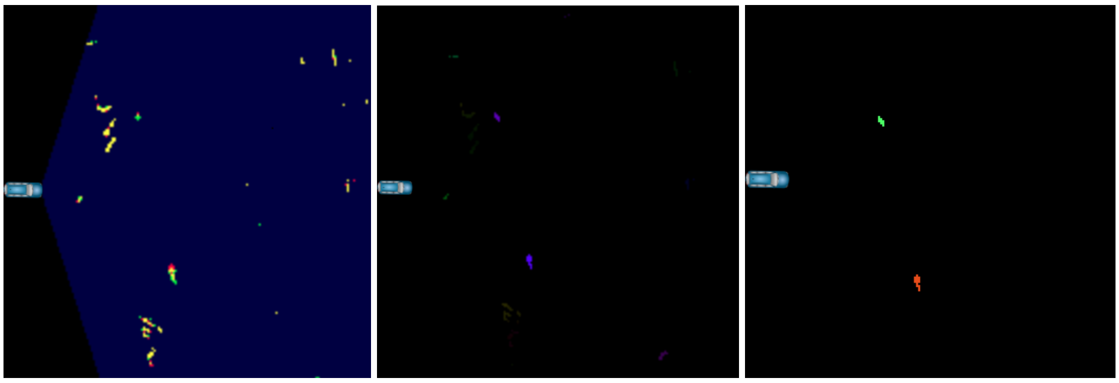

(a) Transformed scan points in the same coordinate system (b) Optical Flow calculated on LiDAR images (c) Resulting clusters, one color for every cluster

Figure 6.14: Result of the optical flow and clustering on a simple example.

neighborhood functions are defined as

$$\text{closeness}(p1, p2) = \|p1 - p2\| < t_c, \tag{6.44}$$

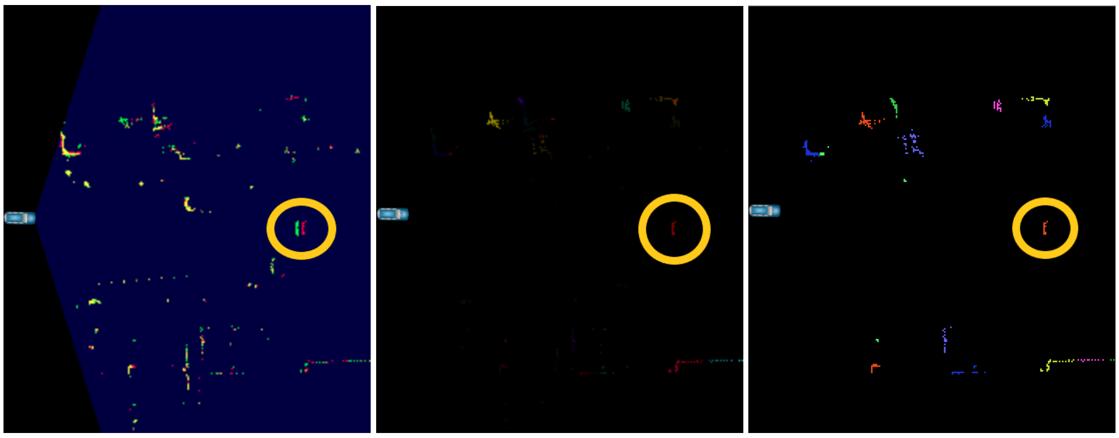
$$\text{direction}(v1, v2) = \|\text{atan2}(v1) \boxminus \text{atan2}(v2)\| < t_d, \tag{6.45}$$

$$\text{neighbor}(m1, m2) = \text{closeness}(m1_p, m2_p) \wedge \text{direction}(m1_v, m2_v), \tag{6.46}$$

where t_c is the threshold determining closeness and t_d the threshold for the direction and pi, vi and mi represent the position, velocity and a combined representation respectively. This check is performed for all pairs of pixels in the flow to obtain a list of neighbors for each pixel. With the neighborhoods defined, the DBSCAN can be performed to obtain clusters. An example is shown in Fig. 6.14. As can be seen in the results the clustering separates areas that are close together with similar dynamic properties. The flow clearly shows areas dynamic properties, in this case there are two bikers riding in front of the vehicle. The clustering automatically discards areas without enough dynamics or where the dynamics of close points does not match. The result are two clusters containing each of the two bikers.

However, in the example shown in Fig. 6.15 it becomes apparent that the flow calculated on the LiDAR images is also often inaccurate. In Fig. 6.15a it is visible that there is only a single dynamic object in the view of the scanner, which is marked in orange. In the flow in Fig. 6.15b there are however many areas that were detected to have movement, leading to a large number of clusters in Fig. 6.15c. This makes working with the clustering result very difficult as there may be a very large number of object candidates that are invalid.

To overcome this issue the detections are filtered over time and used for object tracking, which is explained in the following. For this all required information is already contained in the clusters. A bounding box is calculated that includes all cluster points, where the orientation is given by the average flow. The velocity is estimated from the average



(a) Transformed scan points in the same coordinate system (b) Optical Flow calculated on LiDAR images (c) Resulting clusters, divided by color. Amount of colors are limited, causing some colors to be reused for multiple clusters

Figure 6.15: Result of the optical flow and clustering on a more challenging example where the flow contains a lot of noise.

flow vectors, which are converted from pixel coordinates to world coordinates. First the orientation is determined from the direction of travel by

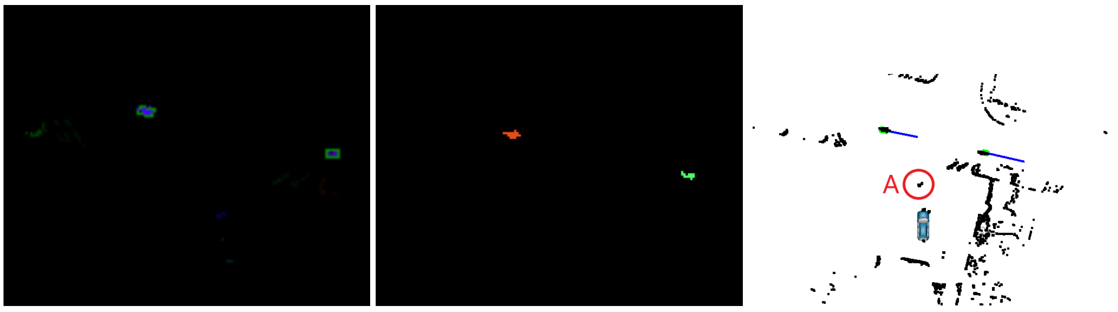
$$\psi = \operatorname{atan2}\left(\left(\sum_{i=0}^{N} v_i\right) \cdot \frac{1}{N}\right), \tag{6.47}$$

where N is the number of points contained in the cluster. After obtaining the orientation of the object the cluster points p_i^V are rotated to be aligned with the object coordinate system and the object bounds are calculated.

$$p_i^O = R_\psi p_i^V, \tag{6.48}$$
$$\text{top} = \max(p_{1,x}^O, ...p_{N,x}^O), \tag{6.49}$$
$$\text{bottom} = \min(p_{1,x}^O, ...p_{N,x}^O), \tag{6.50}$$
$$\text{left} = \max(p_{1,y}^O, ...p_{N,y}^O), \tag{6.51}$$
$$\text{right} = \min(p_{1,y}^O, ...p_{N,y}^O), \tag{6.52}$$

where R_ψ is the rotation matrix constructed from the angle ψ. The position of the cluster is set to the center, calculated by

$$\begin{bmatrix} r_x \\ r_y \end{bmatrix} = R_\psi^{-1} \begin{bmatrix} \frac{\text{top}+\text{bottom}}{2} \\ \frac{\text{left}+\text{right}}{2} \end{bmatrix}. \tag{6.53}$$



(a) Optical flow and resulting bounding box (b) Clusters calculated from the optical flow (c) Resulting tracked objects

Figure 6.16: LiDAR optical flow used for tracking. The two vehicles driving in front of the ego vehicle are correctly identified and tracked.

The velocity is estimated by

$$v = R_{\psi}^{-1} \left(\left(\sum_{i=0}^{N} v_i \right) \cdot \frac{1}{N} \right) \times \frac{o}{N \cdot \Delta_t}, \tag{6.54}$$

where Δ_t is the passed time between processed LiDAR scans. Finally, the size is estimated by

$$\begin{bmatrix} s_x \\ s_y \end{bmatrix} = \begin{bmatrix} \text{top} - \text{bottom} \\ \text{left} - \text{right} \end{bmatrix}. \tag{6.55}$$

With the attributes of the object determined it can be used for object tracking. To avoid generating objects for a large number of false positives as seen in Fig. 6.15 the detected objects are not used directly for tracking. Instead they are created as object candidates. Each object candidate needs to be observed a certain number of times in order to be accepted as a valid detection, where in this thesis three observations are required. For this each object candidate found in previous frames is predicted to the current time according to its estimated velocity. Afterwards an object association is performed using (6.11). When an association is found the known object is replaced by the new detection and a counter increased. After the object candidate is confirmed by enough measurements it is used in the late fusion to update existing object tracks or initialize a new one as explained in Sect. 6.2.2. If an object is not observed for a certain time frame it is erased from the object candidates.

The object tracking performance on the resulting detections is evaluated on a selection of scenes obtained from the demonstrator vehicle. The first scene is a scenario where the vehicle is standing at a junction with two vehicles and a pedestrian crossing in front of the ego vehicle. In the flow the vehicles are estimated well and thus the clusters represent the dynamics, pose and size of the objects well. The pedestrian is however not estimated well. It is marked in red in Fig. 6.16c which has insufficient detected flow in Fig. 6.16a. This is a general problem of the approach, which is only suitable for detecting sufficiently

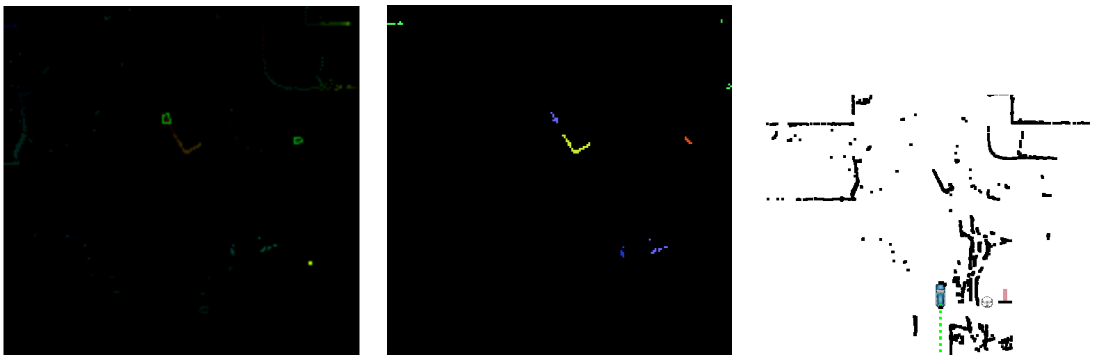

(a) Optical flow and resulting bounding box (b) Clusters calculated from the optical flow (c) No resulting tracked objects

Figure 6.17: Optical flow used for tracking. A turning car is not identified as a full cluster, leading to a missed detection and thus no tracked object.

dynamic objects. Anything traveling at low speeds will not be detected as they would be indistinguishable from estimated dynamics created by measurement noise.

Another scenario is shown in Fig. 6.17, where the ego vehicle is driving on a street and a van merges slowly onto the street in front. Due to its slow speed as well as a high turn rate the vans movement is detected poorly in the flow. This leads to the cluster being split and no object candidate being generated as the flow is spread too far over the cluster. Consequently the object is not tracked, even though in the scan it is easily visible. In addition there are a number of false positives in the clustered image, with one of them being made into an object candidate. As this object candidate is not observed more often it is however not used to create a tracked object.

Overall the evaluation showed that this approach is not suitable for object tracking in autonomous driving. The flow estimation algorithm often does not sufficiently differentiate between slow movement, sensor noise and measurement inaccuracies introduced by the ego motion. This results in a large number of false positives, but also in false negatives which are often more dangerous in autonomous driving. Filtering out false positives by only allowing object candidates that have been observed multiple times works well to mitigate some of these shortcomings, however the false negatives remain. Additionally turning objects are not estimated well in the flow and pose a threat for the ego vehicle as shown in Fig. 6.17. Therefore additional options are explored for object detection in the following.

6.3.2 Evidential Dynamic Mapping and Tracking

Using evidential mapping a second approach is evaluated in the following to obtain an accurate description of the surrounding dynamics. This is largely based on the works [Tanzmeister and Wollherr, 2016; Steyer et al., 2018] where a LiDAR and a Radar are used in order to estimate the dynamic properties and perform object tracking. In this work this is adapted to a multi-LiDAR setup where no Radar information is avail-

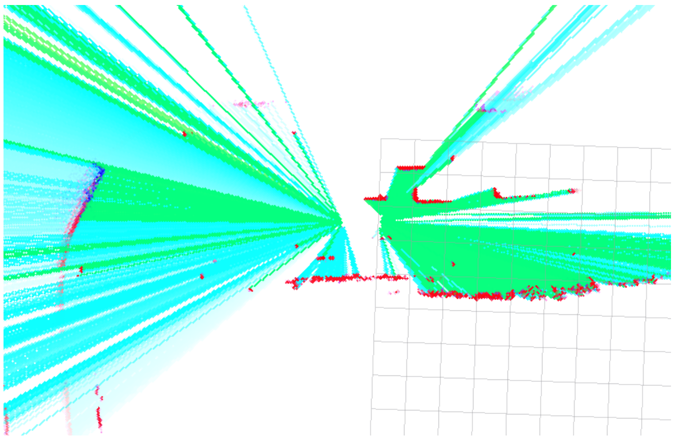


Figure 6.18: Example of an evidential dynamic map. Free areas are shown in green, statically occupied areas in red and dynamically occupied areas in red. Areas not currently observed as free that were previously free are marked in turquoise.

able. With no Radar available the dynamic properties of the environment are no longer measured directly, making the estimation problem more difficult. Dynamics are only observable over time, by generating possible dynamics for each cell and validating their correctness in subsequent scans. This is an ideal application for particle-based estimation, where multiple particles represent potential dynamics contained in a cell. This approach is presented in detail in [Tanzmeister and Wollherr, 2016] and serves as a basis for this work. The result is a grid map where in each cell the dynamics of the corresponding space in the world are estimated. This is visualized in Fig. 6.18.

The estimation of this map is based on the Dempster-Shafer theory [Dempster, 1968; Shafer, 1976]. It allows for the representation of uncertain knowledge about the environment. It especially allows the representation of multiple different properties and a combination of them. In this case the frame of discernment is defined as

$$\Theta = \{F, S, D\}\,, \tag{6.56}$$

$$2^{\Theta} = \{\emptyset, \{F\}, \{S\}, \{D\}, \{S, D\}, \{F, D\}, \{F, S\}, \Theta\}\,, \tag{6.57}$$

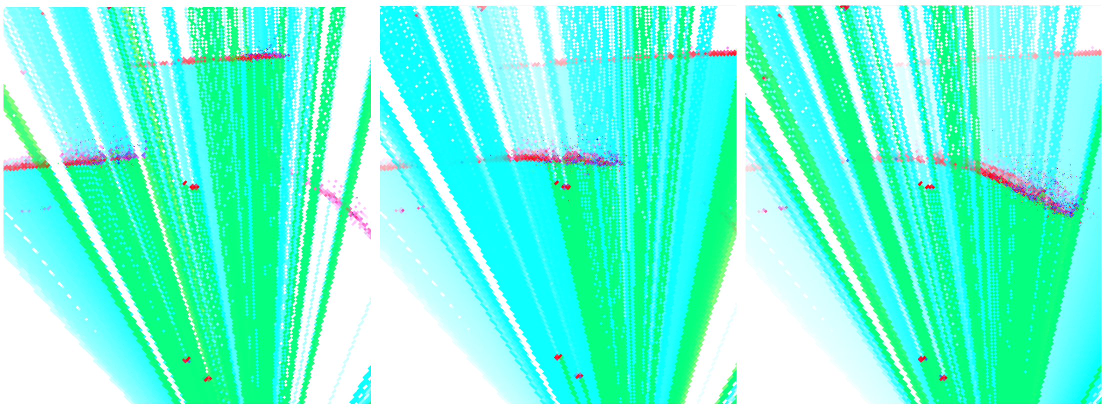
which allows for the following hypotheses:

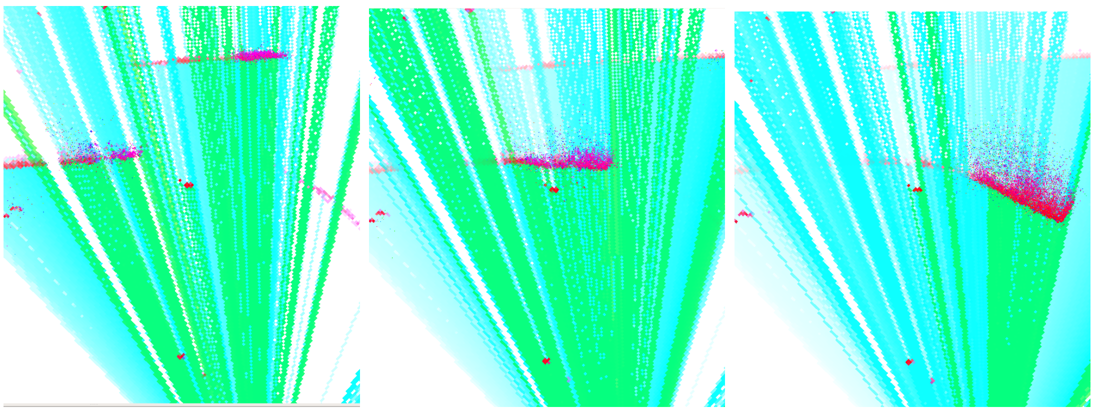
- $\{F\}$ free area
- $\{S\}$ statically occupied area
- $\{D\}$ dynamically occupied area
- $\{S, D\}$ occupied, either dynamic or static
- $\{F, D\}$ either free or dynamically occupied, may become free in the future
- $\{\Theta\}$ no information available

The hypothesis $\{F, S\}$, while part of 2^{Θ}, is not possible in reality and is therefore left out. These hypotheses are estimated using particles. In each step the currently estimated map is predicted based on its current state by propagating the particles forward depending on their estimated dynamics. After prediction the map is updated using the current measurements, where the currently estimated particles are validated by comparing them to the measurement. Finally the particles are resampled. These steps are not explained in detail in this work, however they are explained thoroughly in [Steyer et al., 2018]. One thing to note is however the lack of a Radar in this work. In the original work the update of the map is performed with a Radar in addition to the LiDAR data. Wherever the Radar measures dynamics the belief mass on $\{D\}$ in that cell is increased. This leads to more particles being kept in that cell during resampling. Additionally particles are weighted when a velocity measurement is available for their corresponding cell. This significantly speeds up the convergence of particles towards an estimate. In this work particles are initially largely initialized with random velocities and directions, leading to a large spread of particles after predicting the map. This is reduced over time as measurements restrict the potential velocities, however this is expected to be slower compared to the version where Radar is included.

For generating object candidates the approach presented in [Steyer et al., 2017] is implemented in this work. In this work the evidential map is used to create bounding box representations of contained dynamic objects. Initially all dynamic cells are clustered using the DBSCAN approach presented in Sect. 6.3.1. When there are already tracked objects estimated, the object update is however approached in the opposite way. Instead of finding suitable object candidates in the cells and updating the estimate with it, the cells are initially associated with available tracks by comparing their velocity as well as their spatial closeness to the track. From the cells associated to a track the object detection is generated, which is then used to update the filter. This is described in detail in [Steyer et al., 2017] and used in this work to evaluate object detection on evidential dynamic maps when only LiDAR data is available.

The evaluation is performed on data obtained from the demonstrator vehicle presented in Chap. 3. The calculations are parallelized on a GPU, enabling real-time performance of the adapted algorithm on this data. In this evaluation a NVIDIA RTX 3060 was used. To evaluate the convergence behavior when only using LiDAR data, the maximum amount of particles P per cell is increased between runs. First a maximum of $P = 50$ particles per cell is used as proposed by [Tanzmeister and Wollherr, 2016]. Additionally, $P = 200$ particles per cell are used for comparison, where the algorithm still performs in real-time. Finally $P = 400$ particles are evaluated to give an idea whether additional hardware would improve the performance of the approach, even though it is not real-time capable anymore on the used hardware. The real-time capability was determined in a runtime evaluation, performed for 50, 200 and 400 maximum particles, where the average and maximum computation time per LiDAR frame was determined. The results are shown in Tab. 6.1. As can be seen using 50 and 200 particles results in real-time capable processing of the LiDAR data. LiDAR data arrives at 25Hz, giving the algorithm 40ms to process it before the next scan arrives to be processed. However, using 200 particles the worst case is at 42ms, which already slightly exceeds the time between


(a) Particles and resulting dynamic map for $P = 50$.


(b) Particles and resulting dynamic map for $P = 200$.

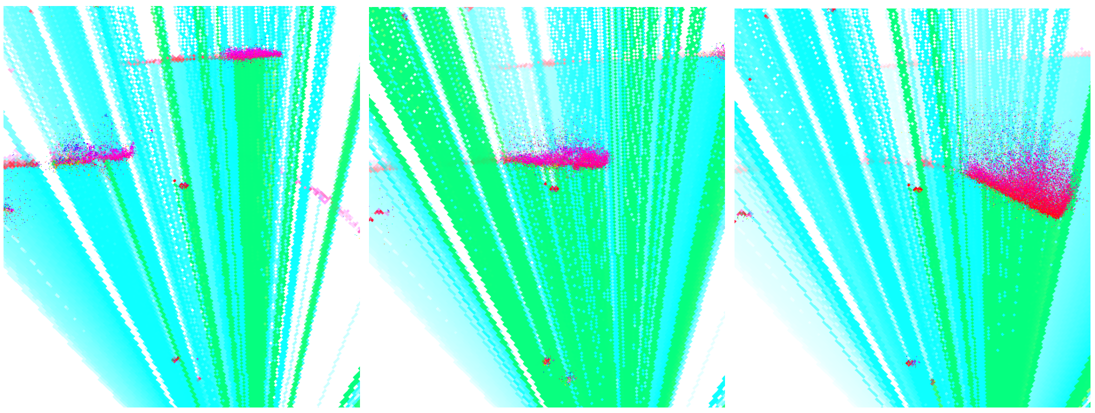

(c) Particles and resulting dynamic map for $P = 400$.

Figure 6.19: Sequence showing a vehicle turn on an intersection. Comparison of the particle distribution depending on the maximum number of particles per cell. The particle color represents the represented direction of travel.

	50 particles	200 particles	400 particles
avg (ms)	19	25	38
max (ms)	26	42	60

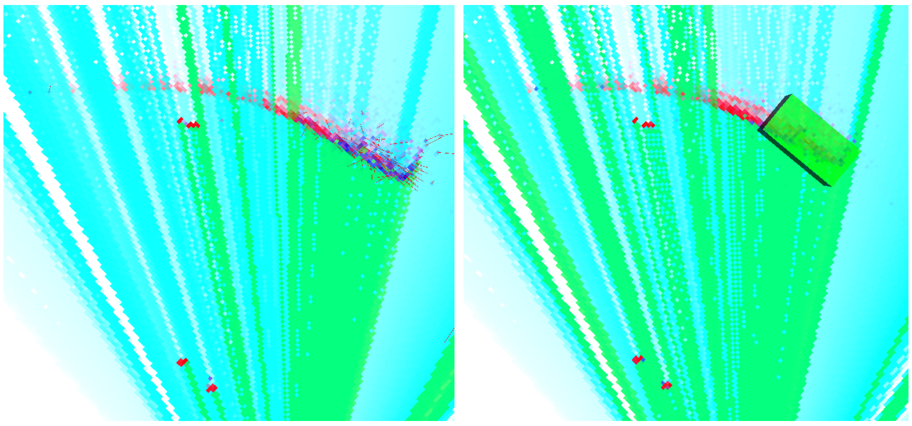
Table 6.1: Runtime evaluation for processing of a LiDAR scan.

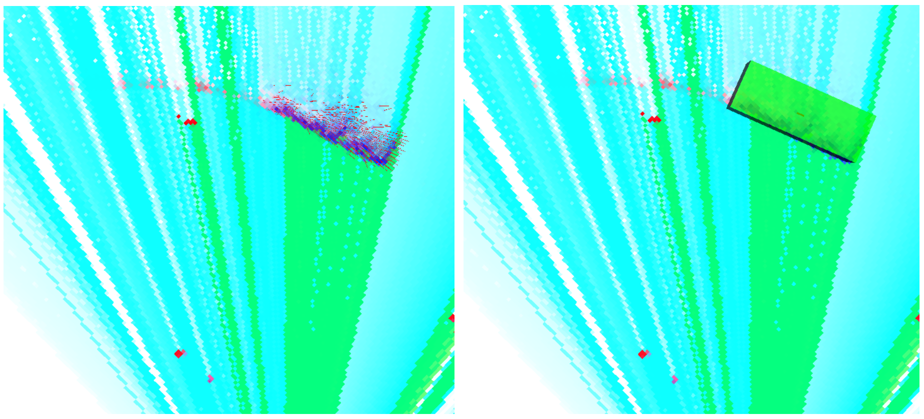
LiDAR frames. On average the 25ms are however significantly below this threshold and as such this configuration is still suitable for real-time use. The configuration with 400 particles shows an average runtime of just below 40ms, however it may increase to up to 60ms, resulting in significant delays in execution and a high usage of the GPU. This configuration is therefore not suitable for use on the vehicle.

The results of the first evaluation can be seen in Fig. 6.19. Here, a vehicle enters the FOV of the ego-vehicle from the left and performs a right turn on an intersection. As can be seen, especially when using 50 particles per cell the approach is very slow to converge on a velocity estimate. The vehicle is in the scene for a long time before the particles converge to a velocity estimate, and even in the final frame no strong convergence can be seen. The resulting velocity estimates and determined object bounding box for the third frame can be seen in more detail in Fig. 6.20a. When using a maximum of 200 particles the estimate already converges significantly quicker, with the particles being bundled around the correct dynamic area. In the first part of the sequence the particles are still spread but even in the second part the particles have converged, with the last part having a very large number of particles correctly estimating the vehicles dynamics. The resulting velocities and estimated bounding box can be seen in Fig. 6.20b. In the evaluation with 400 particles even the first scene shows a slight convergence towards the correct dynamic estimate. Both the second and third scenes have a strong convergence, with the last scene even containing particles outside of the correct object bounding box that estimate the correct velocity. This is not ideal as the resulting bounding box in Fig. 6.20c accordingly is too large, indicating that the resampling needs to be tuned differently when using so many particles. This is however out of the scope of this work as the case with 400 particles is only shown for comparison of the convergence behavior and not meant to be used for object tracking as it is not real-time capable.

While no object tracking is possible even in the last frame when only using 50 maximum particles, objects are detected significantly earlier when using 200 or 400 particles.

This is shown in Fig. 6.21-Fig. 6.23, where the estimated velocities and tracked objects are shown for the earlier two frames. Using 400 particles the particles converge sufficiently even in the first frame, which is not the case when only using 200 particles. In the second evaluated frame the objects are however estimated well in both cases. This shows that the estimation of the surrounding dynamics can be significantly improved and sped up by using a larger number of particles as expected. However even in the second frame the vehicle was visible for a long period of time before the algorithm using a maximum of 200 particles per cell was able to detect it, which may result in dangerous situations


(a) Object detection result for $P = 50$


(b) Object detection result for $P = 200$

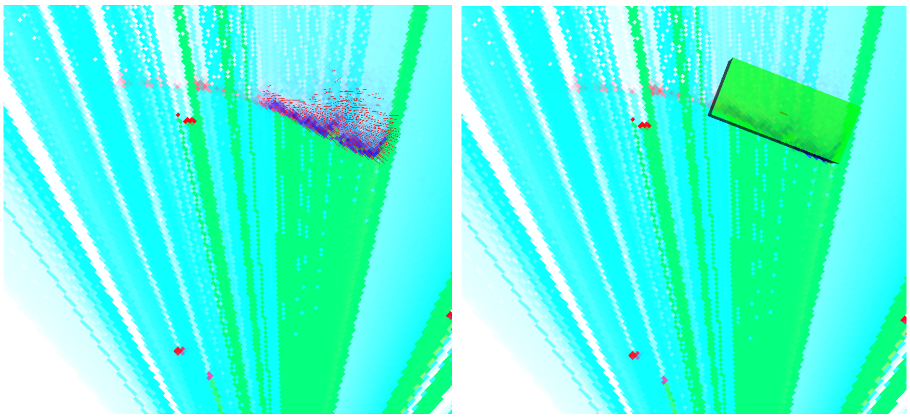
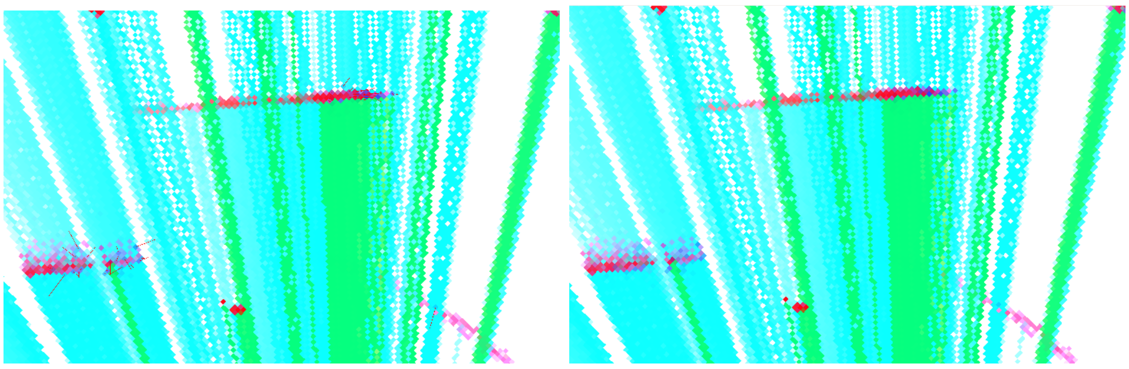

(c) Object detection result for $P = 400$

Figure 6.20: Estimated cell velocities shown as red arrows (left) and the resulting detected object shown as green bounding boxes (right) for different maximum numbers of particles per cell.


(a) Velocity and object detection on first scene

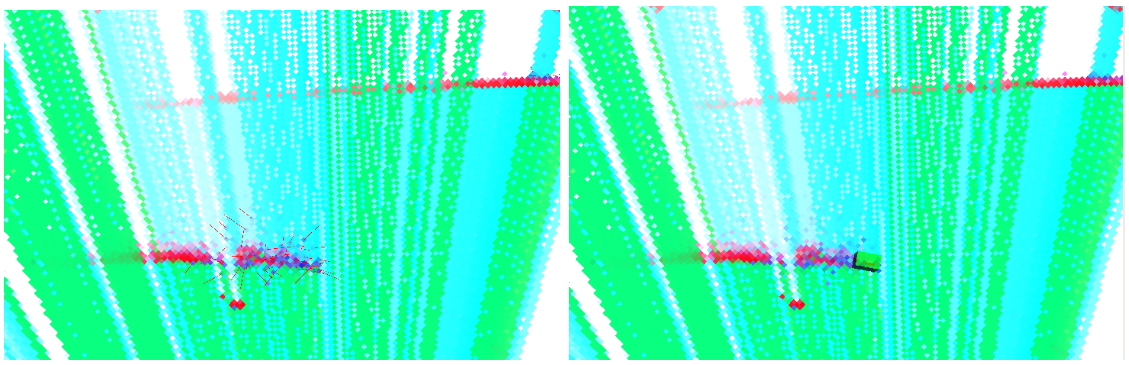
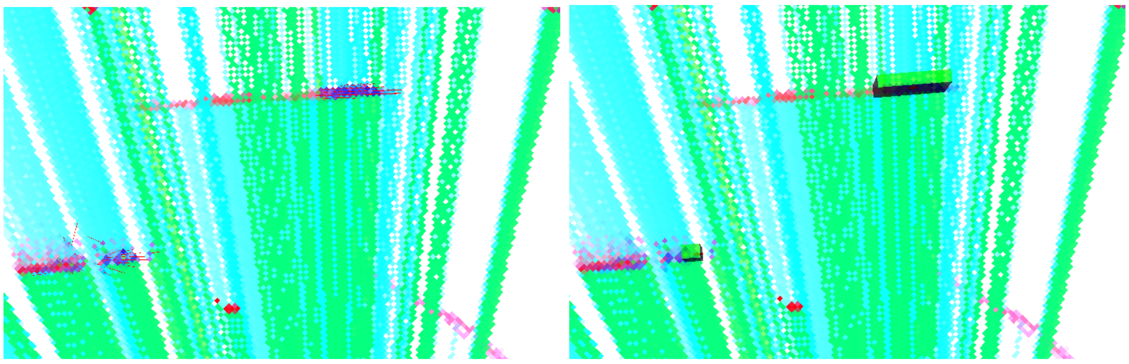

(b) Velocity and object detection on second scene

Figure 6.21: Object detection results for $P = 50$ in the first two frames where the velocity estimate had less time to converge. On the left: velocity estimate shown as red arrows. On the right: detected bounding boxes.

in autonomous driving. With 400 particles a faster convergence can be reached, however the computation time per scan cycle exceeds the real-time limit of 40ms, and on the autonomous vehicle other algorithms may use the graphics card as well. Therefore using the full capacity of the GPU is undesirable and a 400 particle setup is not used for further evaluations.

The presented case above showed a vehicle turning on an intersection, which is a comparably simple case for the algorithm, as the dynamics are easily observable. Additionally the vehicle is seen from both the side and the front during the turn, simplifying the estimation. In the following two scenarios are evaluated where the vehicle is only seen from the side or the front. This evaluation is performed with a maximum amount of particles per cell of $P = 200$. The result is shown in Fig. 6.24. Here two examples are shown where an object is observed only from the front and only the side. In the first example Fig. 6.24a the objects are only seen from the front. In this case the dynamics estimation works well and a tracked object is created accordingly for both vehicles visible in the scene. This case is ideal for the particle estimation as the particles that estimate the correct velocity will match the measurement in the next scan while the


(a) Velocity and object detection on first scene

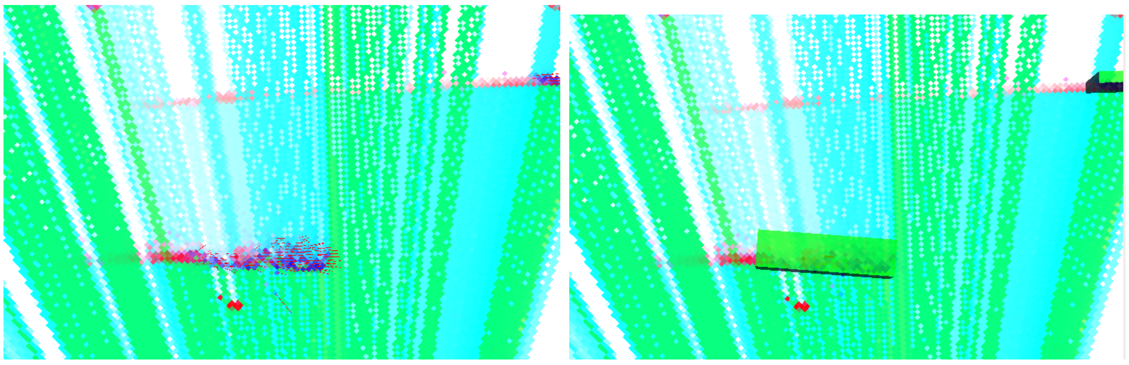
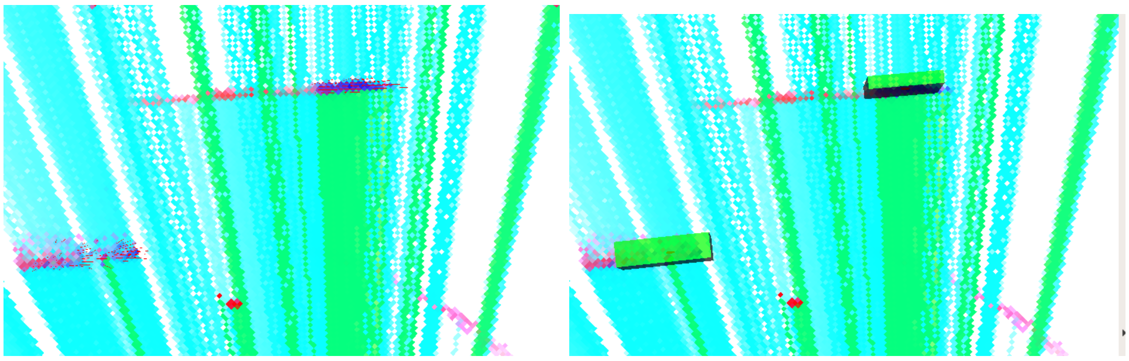

(b) Velocity and object detection on second scene

Figure 6.22: Object detection results for $P = 200$ in the first two frames where the velocity estimate had less time to converge. On the left: velocity estimate shown as red arrows. On the right: detected bounding boxes.

other particles do not. The object is only partially observed but the part that is visible is estimated well by the bounding box and the velocity indicated by the red line points in the correct direction.

In the second example shown in Fig. 6.24b a vehicle crosses from the right side of the ego-vehicle to the left. This object is not estimated well by the dynamic mapping and as a result no tracked object is created even after having almost crossed the entire intersection. There are two factors that come into play in this missed detection. First, detecting objects that are observed from the side are inherently difficult to estimate for this approach. This is because the same space is occupied for a long time by the vehicle, and measured by the scanner. A point on the front of the observed side would be measured again in the next scan, where the measurement would hit a spot further in the middle of the vehicle, until finally the vehicle passes that spot. Until that moment the vehicle could be believed to be static however, which is indicated by the red trail of static belief behind the detection. This leads to a slower convergence of the particle filter and can lead to it not converging on an accurate dynamics estimate at all when the object is not observed well. This leads to the second problem in this detection,


(a) Velocity(left) and object detection(right) on first scene

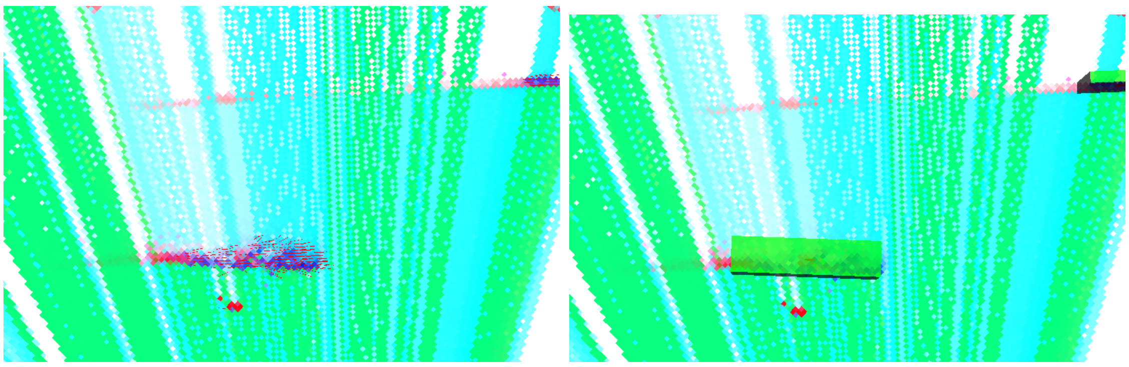
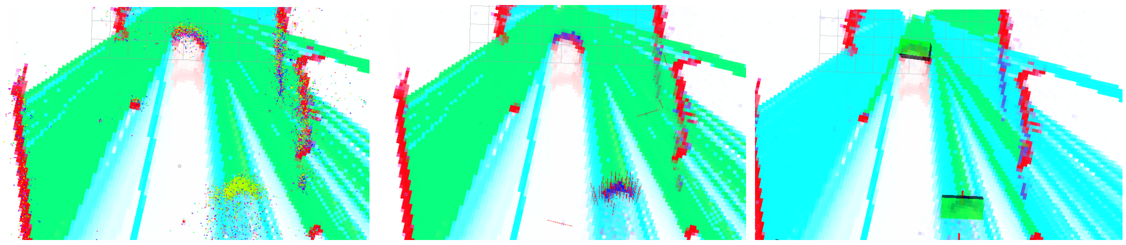

(b) Velocity and object detection on second scene

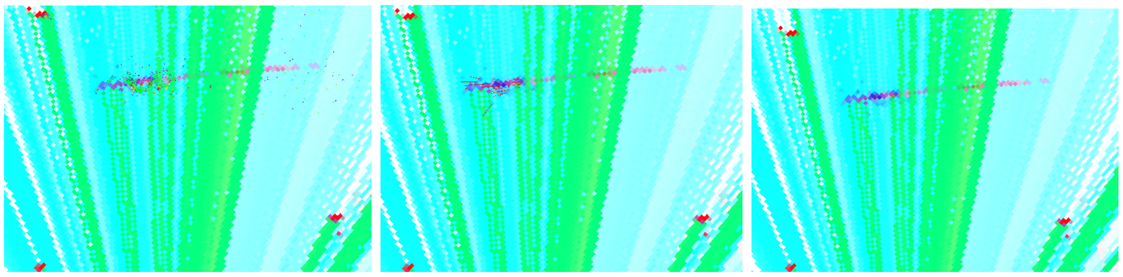
Figure 6.23: Object detection results for $P = 400$ in the first two frames where the velocity estimate had less time to converge. On the left: velocity estimate shown as red arrows. On the right: detected bounding boxes.

being that the scanner do not measure the object well. This can be inferred from the green scan lines going towards the object. There are only a few places where the scanner measured a hit on the vehicle, while the turquoise areas were not measured in the scan. This is a shortcoming of the used LiDAR which potentially has problems measuring on certain object surfaces. As a result of these two factors the object could not be tracked.

Overall the evidential dynamic mapping was able to estimate the dynamics of the surroundings of the ego vehicle more accurately compared to the LiDAR optical flow. As expected the approach is however slow to converge on an accurate estimate of dynamic objects. This can be improved by increasing the particle count, however there are still limits especially when objects are only partially observed. It also leads to an increased computational load, making the approach no longer real-time capable on the research vehicle when a maximum amount of particles per cell of $P = 400$ is chosen. From this evaluation it is clear that the approach can not be used for robust object detection with the available sensory equipment. The use of a radar would improve the convergence significantly, likely making the approach usable for object tracking on the research vehicle. Potentially a LiDAR with a higher resolution would be able to aid in



(a) Objects seen from the front



(b) Object seen from the side

Figure 6.24: Results for different object detection scenarios. Left: particles shown as colored dots. Middle: velocity estimate shown as red arrows. Right: resulting bounding boxes shown in green.

faster convergence as well, as the object might be measured fully without the missed measurements shown in Fig. 6.24b, although this depends on the surface of the measured object. While the sensor setup was fixed in the context of this work, an additional vehicle will be equipped for autonomous driving in the future which will include radars as well as higher resolution LiDARs, taking into account the findings in this work.

6.3.3 Environment Estimation using Normal Distributions Transformation (NDT)

Utilizing dynamics in the environment for object detection proved a promising direction of research, with evidential dynamic mapping producing good results especially when a large number of particles are used. However it was still too slow to converge on current GPUs. Therefore in the following an approach is evaluated which only classifies points as dynamic or static in order to reduce the complexity of the problem and speed up the estimation process. While the velocity information is valuable for object detection it can also be done simply on spatial information, taking into account only dynamic points, while the velocity is estimated over time from positional changes. In the following this classification is performed by estimating an expected distribution of measurements in the environment over time. As new measurements are included and the distributions estimated more accurately a very precise knowledge of where measurements occur is formed. New measurements in areas outside of existing distributions are therefore likely to belong to dynamic objects. This is realized by estimating normal distributions transforms

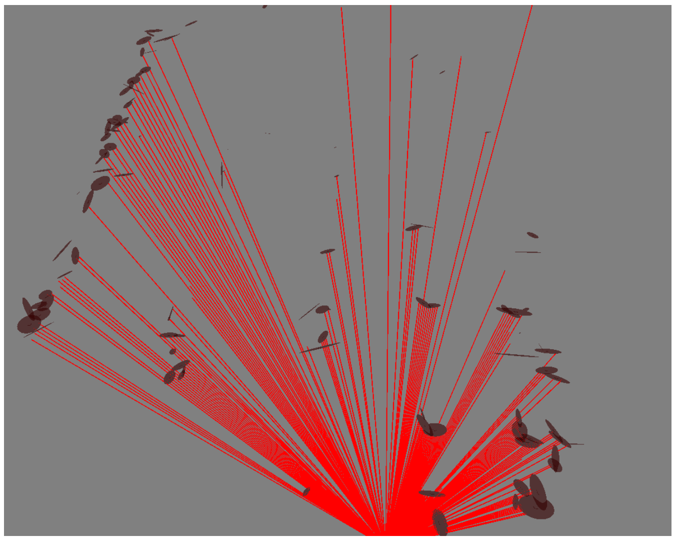


Figure 6.25: Example of a NDT map shown as ellipsoids, with the current scan shown as red lines.

(NDT) [Biber and Straßer, 2003], which were originally introduced for scan matching. In that approach a normal distribution is calculated for every cell in a grid map, estimating where points have been measured in the past. For scan matching the NDTs are used in an optimization algorithm to match the current measurement against the estimated NDT map, transforming the scan to maximize the probability for the measurement. In this work they are instead used to classify points as dynamic and static. To determine this the probability for each point is simply calculated from the existing normal distributions, where a high probability for a point means the area was previously measured as occupied as well, and is therefore likely static. Another possibility is that the area was simply not observed previously, which can however be determined in the grid map by checking whether the cell was measured previously.

To obtain a NDT representation of the environment the library ndt_map[1] is used. It estimates a 3D representation of the environment, which was modified for this work to estimate distributions in 2D. An example of such a map can be seen in Fig. 6.25. The estimation is always limited to the last five scans to avoid inaccuracies introduced by the drift of the odometry localization. In order to classify dynamic and static parts of the environment the closest cell to each point in the current scan is obtained and its likelihood calculated from the estimated normal distribution. A threshold is defined to determine what likelihood represents a dynamic point. An example of the resulting classification can be seen in Fig. 6.26. As can be seen the classification is not ideal due to strong sensor noise and many areas that are not estimated well or at all by the NDT. However for the dynamic object circled in the figure the classification is correct as the new measurement on the object fall outside of the previously estimated normal distributions. In fact the

[1] `https://github.com/OrebroUniversity/perception_oru.git`, accessed on 24.12.2023.

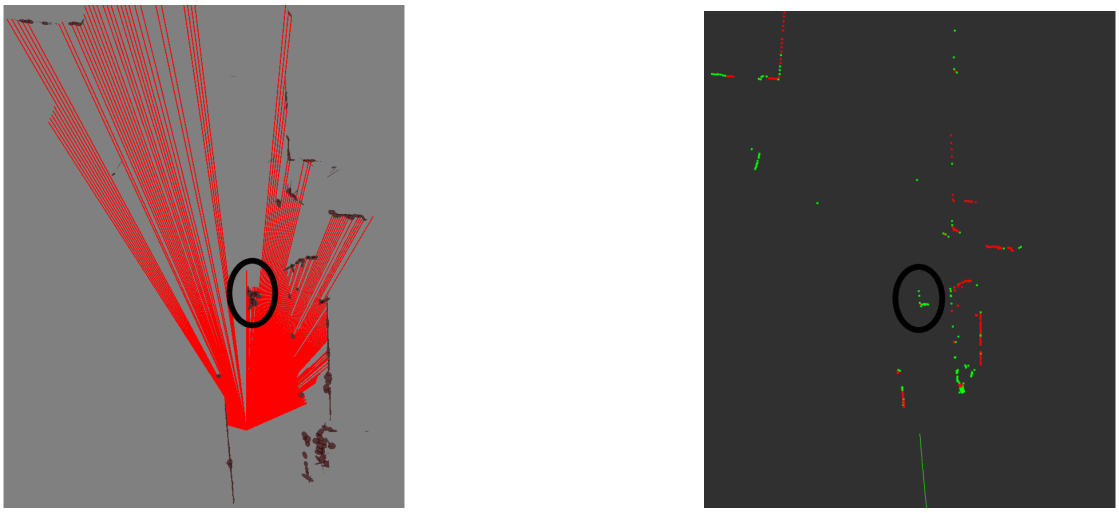


Figure 6.26: Current NDT map on the left with the resulting point classification shown on the right. Dynamic points are colored green, while static points are colored red.

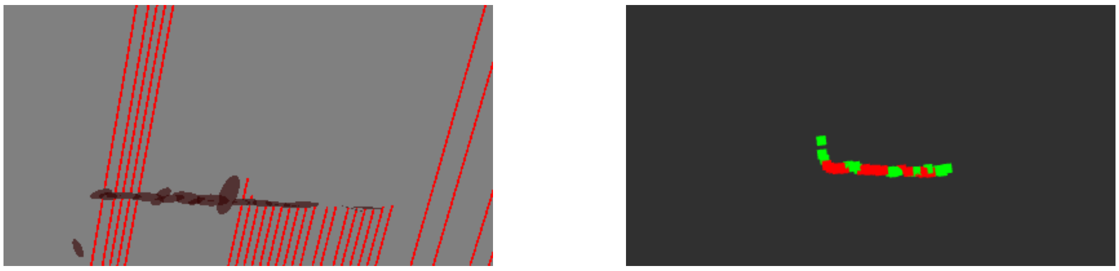


Figure 6.27: Example of a classification problem where many points are misclassified due to the difficulty of the approach with objects that are only seen from the side.

approach estimates dynamic points relatively reliably, however it has clear shortcomings in estimating many static points as dynamic. It can be seen in Fig. 6.26 on the bottom right of both images where the LiDAR measures a different part of the environment than before, which is however still static. The algorithm can not distinguish between something being seen from a different angle, therefore hitting it in a different position and object that move. The same thing holds true for measurement noise, which is difficult to distinguish from movement in some cases. This is mitigated to a certain extent by including measurement noise in the NDT estimation. However especially in areas far from the sensor the NDT is not estimated perfectly due to significantly sparser data, which results in static points falling outside of previously estimated normal distributions. This again results in many points being mistaken as dynamic by the algorithm. Lastly the algorithm is not well suited for estimating vehicles crossing an intersection from side to side. An example of this can be seen in Fig. 6.27. When the object is mostly seen from the side points on the object continuously fall into previously estimated NDTs, thus making them seem like static points. This is a systematic issue with this approach however, which is not simple to resolve and makes the approach difficult to use for scenarios including crossings.

In conclusion the algorithm is not well suited for object detection in autonomous driving. While the dynamic classification works well on vehicles driving in the same or opposite direction as the ego vehicle, there are many shortcomings in other scenarios that make it difficult to distinguish robustly between dynamic and static objects. Sensor noise, occlusion, or changes in the measurement caused by a shifting measurement origin cause a large number of misclassifications and the approach is not suited well to estimate objects that are only measured from the side.

6.3.4 Conclusion and Discussion

Overall none of the presented algorithms solve the object detection task to a degree that satisfies the robustness requirements present in autonomous driving. The reasons are plentiful and dependent on the specific algorithm, however the main issue appears to lie within the available sensor setup. With only four layers the data obtained by the LiDARs is very sparse and only shows a small slice of the world. Therefore contour information is largely unavailable with the exception of the standard L-shape which can be observed only when another vehicle is close enough and measured at a sufficient angle. An attempt was made to remedy this lack of information by taking into account not just single scans but series of scans, in order to detect objects based on moving areas in the environment. This showed some promising results especially when using evidential dynamic mapping to estimate velocities in the environment. However, the approach showed a slow convergence towards the correct velocity estimates. This may be improved by including a Radar in the sensor setup which provides raw velocity measurements instead of the currently available preprocessed dynamic objects, however this was not available in this work. Additionally denser LiDAR measurement data would have improved the convergence behavior as the objects would be measured more reliably and as such particles would be resampled using a more realistic distribution. Therefore the inclusion of a 3D LiDAR with significantly more layers and potentially a higher vertical resolution would provide a large benefit to this as well as all other approaches.

Classical algorithms were used both to maintain explainability of the results and because insufficient training data is available with a similar sensor setup. The creation of a dataset large enough to train a robust object detection that can handle any traffic participant which might be encountered in urban driving is difficult. The available public datasets are unusable for this sensor setup as they use 3D LiDARs mounted to the top of the vehicle with significantly more layers available. This shift in sensor viewpoint as well as the much denser data makes these datasets unusable. It is however possible that this object detection task could be solved better using deep learning approaches. By utilizing Long short-term memory (LSTM) networks [Hochreiter and Schmidhuber, 1997] or feeding multiple scans into the network the dynamic information could even be included in the estimation problem. In future work the performance of neural networks for object detection on the available data will be evaluated as this proved to be a difficult problem for classical methods to solve robustly. The detection depends on many factors which cannot be fully modeled in model-based approaches, while a deep-learning approach could potentially learn them, given sufficient training data. Additionally feature-based detection algorithms will be evaluated for the object detection task on single LiDAR frames. These approaches again rely on a large labeled dataset in order to train for the predefined object classes, however this is required for training neural networks regardless and as such no additional data generation is needed.

7 Conclusion

This chapter summarizes the work presented in this thesis. It gives an overview of the results that were achieved and provides an outlook on possible additions and future work on the presented topics.

7.1 Summary

In this thesis an overview of sensor fusion tasks in autonomous driving was given. Approaches were presented to solve the three main tasks in autonomous driving: localization of the vehicle, mapping of the environment and object detection and tracking of other traffic participants. All topics were solved using multi-sensor approaches, both to improve performance and to add robustness which is crucial in autonomous driving. The thesis focuses on classical approaches for these tasks as opposed to deep-learning based approaches to maintain explainability of all algorithms and to reduce the reliance on comprehensive training data set that cover all relevant scenarios which is often difficult to obtain.

For the localization of a vehicle a system consisting of two localization algorithms was proposed, one estimating an odometry, while the other provides global positioning. Both utilize the so called $\boxplus$-Kalman filter to estimate the vehicle state, where the $\boxplus$-method is used to correctly estimate the orientation of the vehicle which is part of the special orthogonal group, or rotation group $SO(3)$. The separate filters are used due to different requirements on localization systems in autonomous driving that are inherently incompatible. On one hand a smooth, jump free pose estimate is required for many subsequent tasks such as vehicle control to ensure safe driving and avoid undesired maneuvers due to a sudden change in the state estimate. On the other hand a global position free of drift in an external reference frame is required to enable route planning and consider preexisting information about street signs, lane boundaries and more. This is not possible to guarantee with current sensors as the inclusion of the GNSS data used for global positioning inherently produces jumps whenever the GNSS reception is lost for a while and then restored, leading to the sensor drift in that time frame being corrected. The odometry filter makes use of the observation that most subsequent algorithms rely on a precise relative location, while not requiring an absolute pose estimate. Therefore a moving reference Kalman filter was proposed, which estimates odometry without including GNSS measurements. Generally when only estimating odometry the position and rotation around the z-axis are unobservable, leading to growing uncertainty over those parts of the state. By moving the reference towards which the uncertainty is estimated forward in time it was shown that resulting issues from the growing uncertainty were resolved while maintaining a very high localization accuracy with low drift. For the global localization a second filter was implemented, which largely uses the same models, however this filter utilizes GNSS measurements to obtain a global positioning. Additionally it does not make use of the moving reference implementation as the state is fully observable when GNSS is available. This filter was shown to produce accurate results even in difficult areas with problematic GNSS reception. Finally the performance of LiDAR odometry using Truncated Signed Distance Function-based scan matching was evaluated on the available data recorded by the research vehicle. This evaluation however showed, that including scan matching in the state estimation did not improve the localization result given a medium-cost IMU, wheel speed and steering wheel measurements, instead introducing additional drift in many situations.

In mapping the focus of this thesis was on implementing a computationally efficient mapping system for estimating occupied and free areas in the vicinity of the vehicle. For this the observation was made that in autonomous driving there is little benefit in maintaining a map of the entire traversed area. The environments are changing constantly due to occlusion of parking cars or other traffic participants. In addition the traversed areas may be very large, making it impossible in some cases to keep the entire area in memory. Instead a moving-window implementation was explored in this work, which only keeps information in a window that is roughly centered around the vehicle while discarding everything else. For this a hierarchical map organization was utilized based on submaps, that makes it computationally efficient to move the active mapping window as the vehicle travels through an area. For the hierarchical map different data structures were evaluated on both the outer layer of the map containing the submaps and the inner layer that stores the cells. The evaluation was performed for 2D arrays, hashmaps and quadtrees. It showed different behaviors depending on the task the mapping algorithm is intended for. When the map is to be built up and extracted for subsequent algorithms regularly, which is often the case in autonomous driving, a combination of a hashmap or quadtree on the outer layer with 2D arrays on the inner layer performed best. When the only goal is to build a map of the environment however, which can be used afterwards or is only extracted occasionally a combination of 2D arrays on both layers showed the best performance, although this is comparably inefficient in terms of memory requirements.

For object detection and tracking first an object tracking framework using a late fusion was presented. This late fusion relies on object detections from multiple sensors to estimate the state of other traffic participants. For this, similar to the localization, it makes use of $\boxplus$-Kalman filters, with each traffic participant being estimated by its own filter. The late fusion supports LiDAR, Camera and Radar data. In addition the use of V2X communication in the tracking framework was explored and showed promising results. Equations as well as their derivatives for implementing the late fusion in either a UKF or an EKF were given for all measurement models as well as the motion model. Second a number of object detection algorithms based on LiDAR data were explored in order to remove the reliance on black box object detection implementations of the sensors. The explored object detection algorithms are based on the observation that within the available data the most distinct feature of an object is its movement between two measurements. Thus the focus in this thesis was put on approaches that estimate dynamics in the environment in order to generate object detections from this. First a method based on LiDAR optical flow was evaluated which showed some promising results, however in some cases movement in the environment was indistinguishable from sensor noise in the LiDAR making the result too inconsistent for robust tracking. Second an approach was evaluated which estimates a grid map where the dynamic is estimated within each cell using particles. This was originally proposed to be used with a LiDAR and Radar, however with no raw Radar available it was evaluated for the use when only LiDAR is available. While the approach showed great promise and good object detection results, this comes at the cost of high computational demand. With the currently available hardware it was not possible to obtain robust results in real-time, however when a high number of particles were used the classification performance of the dynamic grid mapping improved drastically, indicating that with better hardware or the inclusion of radar sensors this approach would likely be suitable. Finally an approach was evaluated that estimated the distribution of LiDAR measurements in the world using Normal Distribution Transforms. This was based on the idea that measurements made sufficiently outside of the estimated NDTs would likely belong to dynamic objects. While this approach was able to detect objects where the front was visible, it was unable to detect objects seen from the side as the resulting NDTs from one scan would still cover most of the side of the object in the next scan.

7.2 Outlook

While future work specific to the presented modules was presented in the corresponding chapters, an overview of future work in the field of sensor fusion in autonomous driving is given in the following.

The results of this thesis showed that the performance of the modules depends strongly on the utilized sensors. Their accuracy, the field of view of each sensor and the resolution are driving factors in the performance of the sensor fusion algorithms. While the use of multi-sensor fusion mitigates the reliance on each sensor to a certain extent, some reliance still exists and will be explored in future work. Following the results of works presented in this thesis strong evidence points towards the available sensor equipment leaving the largest room for improvements, especially in object detection. As a result an updated sensor setup for planned new autonomous vehicles was proposed based on the presented findings. This especially includes a full coverage of the environment by four Radar sensors attached to the corners of the vehicle. In addition the coverage by cameras will be increased similarly, with additional cameras being built into the sides as well as the back of the vehicle. Finally improved LiDARs will be used in a similar setup as in this thesis. These have an equal horizontal FOV, however the vertical FOV is increased significantly from 3.2 degrees to 10 degrees and the horizontal resolution reaches up to 0.125 degrees between two measurements, which is currently at 0.25 degrees. In addition these LiDARs measure on 16 layers instead of 4, generating a significantly denser 3D point cloud of the environment, although still less dense than the data contained in public datasets such as the Waymo or nuScenes dataset where a top-mounted 3D LiDAR is used. With this new equipment some of the algorithms evaluated or developed in this work will be reevaluated. Especially in object detection and tracking many shortcomings were produced by the quality of the available data. Reevaluating the performance of the evidential dynamic mapping approach given both the improved LiDARs as well as the added Radars is promising given the results produced with the current setup. Similarly the LiDAR optical flow may benefit from the denser LiDAR measurements and could be fused with Radar measurements to obtain a more robust estimate of the dynamics in the environment.

The improved sensor setup additionally enables other directions of research that will be explored in the future. With the LiDAR measuring on 16 layers over 10 degrees vertical FOV the information contained in each scan is significantly improved. While the current sensor setup provides little contour information and largely depends on detecting objects from motion in concurrent scans, this is not the case with the new LiDAR. In its data contours of the entire object in 3D are visible more clearly, enabling the use of object detection and classification methods that work on shapes. In addition to classic methods, the deep learning-based object detection on 3D point clouds has gotten a large amount of attention in the past years and thus will be evaluated for the use on the new vehicle as well. With most research utilizing a 3D LiDAR mounted to the top of the vehicle, the focus here will be put on developing an object detection that is able to make use of the six LiDARs mounted to the sides, front and back of the vehicle.

In addition to the planned work on object detection and tracking there are other fields that benefit from an improved sensor setup. While the static environment is currently only estimated as an occupancy grid map due to the spare data of the LiDARs with a small opening angle, the new setup will measure significantly larger parts of the environment vertically. This allows for more distinctions in the representation of the environment. In combination with a full coverage of the surroundings with cameras, it is possible to create a refined semantic representation. This includes information about the road boundaries, the location and height of curbstones that was previously not possible to be observed by the LiDARs, and the precise location of road signs that are identified in the camera.

Regardless of the sensor setup one future topic of research is the prediction of movement of other traffic participants. The current object tracking approach allows the prediction based on the currently estimated dynamics, however real traffic participants follow complex behavior and decision making. To improve the prediction further information about the environment and the state of the object must be considered. This includes available lanes on a road and especially on crossings. It also includes detected turn signals and skeletal data of pedestrians and bikers that can be used to predict an upcoming lane change or turn. The presented algorithms and results in this thesis serve as a solid basis for these future tasks. While many are already actively in use on an autonomous vehicle, others serve as important insights in the preparation for future developments.

Own Publications

[Clemens et al., 2020] Clemens, J., Wellhausen, C., Koller, T. L., Frese, U., and Schill, K. (2020). Kalman filter with moving reference for jump-free, multi-sensor odometry with application in autonomous driving. In *2020 IEEE 23rd International Conference on Information Fusion (FUSION)*, pages 1–9. IEEE.

[Folkers et al., 2022] Folkers, A., Wellhausen, C., Rick, M., Li, X., Evers, L., Schwarting, V., Clemens, J., Dittmann, P., Shubbak, M., Bustert, T., et al. (2022). The opa 3 l system and testconcept for urban autonomous driving. In *2022 IEEE 25th International Conference on Intelligent Transportation Systems (ITSC)*, pages 1949–1956. IEEE.

[Wellhausen et al., 2021] Wellhausen, C., Clemens, J., and Schill, K. (2021). Efficient grid map data structures for autonomous driving in large-scale environments. In *2021 IEEE International Intelligent Transportation Systems Conference (ITSC)*, pages 2855–2862. IEEE.

Bibliography

[ADAC, 2020] ADAC (2020). Welche hersteller bieten bereits car2x an? adac-umfrage 6/2020. Available at `https://assets.adac.de/image/upload/v1595919606/ADAC-eV/KOR/Text/PDF/Umfrage_Hersteller_Car2X_dl45xm.pdf`, accessed on 02.02.2024.

[Agarwal et al., 2023] Agarwal, S., Mierle, K., and Team, T. C. S. (2023). Ceres Solver. Available at `https://github.com/ceres-solver/ceres-solver`, accessed on 27.02.2024.

[An et al., 2019] An, J., Choi, B., Kim, H., and Kim, E. (2019). A new contour-based approach to moving object detection and tracking using a low-end three-dimensional laser scanner. *IEEE Transactions on Vehicular Technology*, 68(8):7392–7405.

[Bae et al., 2020] Bae, J.-K., Park, M.-C., Yang, E.-J., and Seo, D.-W. (2020). Implementation and performance evaluation for dsrc-based vehicular communication system. *IEEE Access*, 9:6878–6887.

[Barrera et al., 2020] Barrera, A., Guindel, C., Beltrán, J., and García, F. (2020). Birdnet+: End-to-end 3d object detection in lidar bird's eye view. In *2020 IEEE 23rd International Conference on Intelligent Transportation Systems (ITSC)*, pages 1–6.

[Biber and Straßer, 2003] Biber, P. and Straßer, W. (2003). The normal distributions transform: A new approach to laser scan matching. In *Proceedings 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2003)*, volume 3, pages 2743–2748. IEEE.

[BMW, 2023] BMW (2023). Bmw driver assistance. Available at `https://www.bmwusa.com/explore/driver-assistance-safety-features.html`, accessed on 02.02.2024.

[Bradski, 2000] Bradski, G. (2000). The OpenCV Library. *Dr. Dobb's Journal of Software Tools*.

[Bresenham, 1998] Bresenham, J. E. (1998). Algorithm for computer control of a digital plotter. In *Seminal graphics: pioneering efforts that shaped the field*, pages 1–6.

[Breßler et al., 2016] Breßler, J., Reisdorf, P., Obst, M., and Wanielik, G. (2016). GNSS positioning in non-line-of-sight context—A survey. In *IEEE International Conference on Intelligent Transportation Systems (ITSC)*, pages 1147–1154.

[Buerkle et al., 2020] Buerkle, C., Oboril, F., Jarquin, J., and Scholl, K.-U. (2020). Efficient dynamic occupancy grid mapping using non-uniform cell representation. In *2020 IEEE Intelligent Vehicles Symposium (IV)*, pages 1629–1634. IEEE.

[Caesar et al., 2020] Caesar, H., Bankiti, V., Lang, A. H., Vora, S., Liong, V. E., Xu, Q., Krishnan, A., Pan, Y., Baldan, G., and Beijbom, O. (2020). nuscenes: A multimodal dataset for autonomous driving. In *CVPR*.

[Campos et al., 2021] Campos, C., Elvira, R., Rodríguez, J. J. G., Montiel, J. M., and Tardós, J. D. (2021). Orb-slam3: An accurate open-source library for visual, visual–inertial, and multimap slam. *IEEE Transactions on Robotics*, 37(6):1874–1890.

[Capellier et al., 2018] Capellier, E., Davoine, F., Frémont, V., Ibañez-Guzmán, J., and Li, Y. (2018). Evidential grid mapping, from asynchronous lidar scans and rgb images, for autonomous driving. In *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*, pages 2595–2602. IEEE.

[Chen et al., 2020] Chen, G., Wang, F., Qu, S., Chen, K., Yu, J., Liu, X., Xiong, L., and Knoll, A. (2020). Pseudo-image and sparse points: Vehicle detection with 2d lidar revisited by deep learning-based methods. *IEEE Transactions on Intelligent Transportation Systems*, 22(12):7699–7711.

[Chen and Shao, 2021] Chen, S. and Shao, C. (2021). Efficient online tracking-by-detection with kalman filter. *IEEE Access*, 9:147570–147578.

[Chiang et al., 2020] Chiang, K.-W., Tsai, G.-J., Chu, H.-J., and El-Sheimy, N. (2020). Performance enhancement of ins/gnss/refreshed-slam integration for acceptable lane-level navigation accuracy. *IEEE Transactions on Vehicular Technology*, 69(3):2463–2476.

[Clemens, 2018] Clemens, J. (2018). *Uncertainty in Localization, Mapping, and Planning: Advanced Methods and Applications*. PhD thesis, Universität Bremen.

[Clemens and Schill, 2016] Clemens, J. and Schill, K. (2016). Extended Kalman filter with manifold state representation for navigating a maneuverable melting probe. In *19th International Conference on Information Fusion (FUSION)*, pages 1789–1796. ISIF, IEEE.

[Cormen et al., 2009] Cormen, T. H., Leiserson, C. E., Rivest, R. L., and Stein, C. (2009). *Introduction to Algorithms*. MIT Press, Cambridge, MA, 3rd edition.

[Curless and Levoy, 1996] Curless, B. and Levoy, M. (1996). A volumetric method for building complex models from range images. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 303–312.

[Dai et al., 2020] Dai, H.-f., Bian, H.-w., Wang, R.-y., and Ma, H. (2020). An ins/gnss integrated navigation in gnss denied environment using recurrent neural network. *Defence technology*, 16(2):334–340.

[Daun et al., 2019] Daun, K., Kohlbrecher, S., Sturm, J., and von Stryk, O. (2019). Large scale 2d laser slam using truncated signed distance functions. In *2019 IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR)*, pages 222–228. IEEE.

[de Berg et al., 2008] de Berg, M., Cheong, O., van Kreveld, M., and Overmars, M. (2008). *Computational Geometry: Algorithms and Applications.* Springer, Berlin/Heidelberg, 3rd edition.

[Dempster, 1967] Dempster, A. P. (1967). Upper and Lower Probabilities Induced by a Multivalued Mapping. *The Annals of Mathematical Statistics*, 38(2):325 – 339.

[Dempster, 1968] Dempster, A. P. (1968). A generalization of bayesian inference. *Journal of the Royal Statistical Society: Series B (Methodological)*, 30(2):205–232.

[Elfes, 1989] Elfes, A. (1989). Using occupancy grids for mobile robot perception and navigation. *Computer*, 22(6):46–57.

[Engel et al., 2014] Engel, J., Schöps, T., and Cremers, D. (2014). Lsd-slam: Large-scale direct monocular slam. In Fleet, D., Pajdla, T., Schiele, B., and Tuytelaars, T., editors, *Computer Vision – ECCV 2014*, pages 834–849, Cham. Springer International Publishing.

[Ester et al., 1996] Ester, M., Kriegel, H.-P., Sander, J., Xu, X., et al. (1996). A density-based algorithm for discovering clusters in large spatial databases with noise. In *kdd*, volume 96, pages 226–231.

[Fang et al., 2019] Fang, Y., Wang, C., Yao, W., Zhao, X., Zhao, H., and Zha, H. (2019). On-road vehicle tracking using part-based particle filter. *IEEE transactions on intelligent transportation systems*, 20(12):4538–4552.

[Farnebäck, 2003] Farnebäck, G. (2003). Two-frame motion estimation based on polynomial expansion. In *Image Analysis: 13th Scandinavian Conference, SCIA 2003 Halmstad, Sweden, June 29–July 2, 2003 Proceedings 13*, pages 363–370. Springer.

[French, 2004] French, D. (2004). *Teaching and learning geometry.* A&C Black.

[Gadzicki et al., 2020] Gadzicki, K., Khamsehashari, R., and Zetzsche, C. (2020). Early vs late fusion in multimodal convolutional neural networks. In *2020 IEEE 23rd international conference on information fusion (FUSION)*, pages 1–6. IEEE.

[Gao et al., 2018] Gao, X., Wang, R., Demmel, N., and Cremers, D. (2018). Ldso: Direct sparse odometry with loop closure. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2198–2204. IEEE.

[Garcia et al., 2014] Garcia, F. M., Kapadia, M., and Badler, N. I. (2014). Gpu-based dynamic search on adaptive resolution grids. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1631–1638. IEEE.

[Garrido-Jurado et al., 2016] Garrido-Jurado, S., Munoz-Salinas, R., Madrid-Cuevas, F. J., and Medina-Carnicer, R. (2016). Generation of fiducial marker dictionaries using mixed integer linear programming. *Pattern recognition*, 51:481–491.

[Geiger et al., 2012] Geiger, A., Lenz, P., and Urtasun, R. (2012). Are we ready for autonomous driving? the kitti vision benchmark suite. In *2012 IEEE conference on computer vision and pattern recognition*, pages 3354–3361. IEEE.

[Graeter et al., 2018] Graeter, J., Wilczynski, A., and Lauer, M. (2018). Limo: Lidar-monocular visual odometry. In *2018 IEEE/RSJ international conference on intelligent robots and systems (IROS)*, pages 7872–7879. IEEE.

[Grisetti et al., 2010] Grisetti, G., Kümmerle, R., Stachniss, C., and Burgard, W. (2010). A tutorial on graph-based slam. *IEEE Intelligent Transportation Systems Magazine*, 2(4):31–43.

[Groves, 2013] Groves, P. D. (2013). *Principles of GNSS, Inertial, and Multisensor Integrated Navigation Systems.* Artech House, 2nd edition.

[Guo and Zhao, 2022] Guo, G. and Zhao, S. (2022). 3d multi-object tracking with adaptive cubature kalman filter for autonomous driving. *IEEE Transactions on Intelligent Vehicles*, 8(1):512–519.

[Guo et al., 2019] Guo, Z., Cai, B., Jiang, W., and Wang, J. (2019). Feature-based detection and classification of moving objects using lidar sensor. *IET Intelligent Transport Systems*, 13(7):1088–1096.

[He et al., 2016] He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778.

[Hertzberg, 2015] Hertzberg, C. (2015). *3D-sensors: Axiomatization, Modeling, Calibration, Localization and Mapping.* PhD thesis, Universität Bremen.

[Hertzberg et al., 2013] Hertzberg, C., Wagner, R., Frese, U., and Schröder, L. (2013). Integrating generic sensor fusion algorithms with sound state representations through encapsulation of manifolds. *Information Fusion*, 14(1):57–77.

[Hess et al., 2016] Hess, W., Kohler, D., Rapp, H., and Andor, D. (2016). Real-time loop closure in 2d lidar slam. In *2016 IEEE international conference on robotics and automation (ICRA)*, pages 1271–1278. IEEE.

[Hochreiter and Schmidhuber, 1997] Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, 9(8):1735–1780.

[Höffmann et al., 2022] Höffmann, M., Clemens, J., Stronzek-Pfeifer, D., Simonelli, R., Serov, A., Schettino, S., Runge, M., Schill, K., and Büskens, C. (2022). Coverage path planning and precise localization for autonomous lawn mowers. In *2022 Sixth IEEE International Conference on Robotic Computing (IRC)*, pages 238–242. IEEE.

[Jazwinski, 2007] Jazwinski, A. H. (2007). *Stochastic processes and filtering theory.* Courier Corporation.

[Jin et al., 2021] Jin, R., Liu, J., Zhang, H., and Niu, X. (2021). Fast and accurate initialization for monocular vision/ins/gnss integrated system on land vehicle. *IEEE Sensors Journal*, 21(22):26074–26085.

[Jo et al., 2018] Jo, K., Cho, S., Kim, C., Resende, P., Bradai, B., Nashashibi, F., and Sunwoo, M. (2018). Cloud update of tiled evidential occupancy grid maps for the multi-vehicle mapping. *Sensors*, 18(12):4119.

[Julier and Uhlmann, 1997] Julier, S. J. and Uhlmann, J. K. (1997). New extension of the kalman filter to nonlinear systems. In *Signal processing, sensor fusion, and target recognition VI*, volume 3068, pages 182–193. Spie.

[Jungnickel et al., 2016] Jungnickel, R., Köhler, M., and Korf, F. (2016). Efficient automotive grid maps using a sensor ray based refinement process. In *2016 IEEE Intelligent Vehicles Symposium (IV)*, pages 668–675. IEEE.

[Kalman, 1960] Kalman, R. E. (1960). A new approach to linear filtering and prediction problems.

[Lang et al., 2019] Lang, A. H., Vora, S., Caesar, H., Zhou, L., Yang, J., and Beijbom, O. (2019). Pointpillars: Fast encoders for object detection from point clouds. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 12697–12705.

[Lepetit et al., 2005] Lepetit, V., Fua, P., et al. (2005). Monocular model-based 3d tracking of rigid objects: A survey. *Foundations and Trends® in Computer Graphics and Vision*, 1(1):1–89.

[Liu and Chen, 1998] Liu, J. S. and Chen, R. (1998). Sequential monte carlo methods for dynamic systems. *Journal of the American statistical association*, pages 1032–1044.

[Luo et al., 2021] Luo, W., Xing, J., Milan, A., Zhang, X., Liu, W., and Kim, T.-K. (2021). Multiple object tracking: A literature review. *Artificial intelligence*, 293:103448.

[Mahler, 2007] Mahler, R. (2007). *Statistical multisource-multitarget information fusion.* Artech.

[Malawade et al., 2022] Malawade, A. V., Mortlock, T., and Al Faruque, M. A. (2022). Hydrafusion: Context-aware selective sensor fusion for robust and efficient autonomous vehicle perception. In *2022 ACM/IEEE 13th International Conference on Cyber-Physical Systems (ICCPS)*, pages 68–79. IEEE.

[Mercedes-Benz, 2023] Mercedes-Benz (2023). Mercedes-benz drive pilot. Available at `https://www.mercedes-benz.de/passengercars/technology/drive-pilot.html`, accessed on 02.02.2024.

[Mobus and Kolbe, 2004] Mobus, R. and Kolbe, U. (2004). Multi-target multi-object tracking, sensor fusion of radar and infrared. In *IEEE Intelligent Vehicles Symposium, 2004*, pages 732–737. IEEE.

[Mur-Artal and Tardós, 2017] Mur-Artal, R. and Tardós, J. D. (2017). Orb-slam2: An open-source slam system for monocular, stereo, and rgb-d cameras. *IEEE transactions on robotics*, 33(5):1255–1262.

[Otegui et al., 2020] Otegui, J., Bahillo, A., Lopetegi, I., and Diez, L. E. (2020). Performance evaluation of different grade imus for diagnosis applications in land vehicular multi-sensor architectures. *IEEE Sensors Journal*, 21(3):2658–2668.

[Pagac et al., 1998] Pagac, D., Nebot, E. M., and Durrant-Whyte, H. (1998). An evidential approach to map-building for autonomous vehicles. *IEEE Transactions on Robotics and Automation*, 14(4):623–629.

[Rajamani, 2011] Rajamani, R. (2011). *Vehicle dynamics and control.* Springer Science & Business Media.

[Reineking and Clemens, 2014] Reineking, T. and Clemens, J. (2014). Dimensions of uncertainty in evidential grid maps. In *International Conference on Spatial Cognition*, pages 283–298. Springer.

[Richter et al., 2020] Richter, S., Beck, J., Wirges, S., and Stiller, C. (2020). Semantic evidential grid mapping based on stereo vision. In *2020 IEEE International Conference on Multisensor Fusion and Integration for Intelligent Systems (MFI)*, pages 179–184. IEEE.

[Russell and Norvig, 2010] Russell, S. J. and Norvig, P. (2010). *Artificial intelligence a modern approach.* London.

[Santos et al., 2019] Santos, N. P., Lobo, V., and Bernardino, A. (2019). Unmanned aerial vehicle tracking using a particle filter based approach. In *2019 IEEE Underwater Technology (UT)*, pages 1–10. IEEE.

[Schmid et al., 2012] Schmid, K., Ruess, F., Suppa, M., and Burschka, D. (2012). State estimation for highly dynamic flying systems using key frame odometry with varying time delays. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2997–3004.

[Serov et al., 2022] Serov, A., Clemens, J., and Schill, K. (2022). Visual-inertial odometry aided by speed and steering angle measurements. In *2022 25th International Conference on Information Fusion (FUSION)*, pages 1–8. IEEE.

[Shafer, 1976] Shafer, G. (1976). *A mathematical theory of evidence*, volume 42. Princeton university press.

[Shan et al., 2020] Shan, T., Englot, B., Meyers, D., Wang, W., Ratti, C., and Rus, D. (2020). Lio-sam: Tightly-coupled lidar inertial odometry via smoothing and mapping. In *2020 IEEE/RSJ international conference on intelligent robots and systems (IROS)*, pages 5135–5142. IEEE.

[Shi et al., 2023] Shi, S., Jiang, L., Deng, J., Wang, Z., Guo, C., Shi, J., Wang, X., and Li, H. (2023). Pv-rcnn++: Point-voxel feature set abstraction with local vector representation for 3d object detection. *International Journal of Computer Vision*, 131(2):531–551.

[Shi et al., 2019] Shi, S., Wang, X., and Li, H. (2019). Pointrcnn: 3d object proposal generation and detection from point cloud. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–779.

[Sola et al., 2018] Sola, J., Deray, J., and Atchuthan, D. (2018). A micro lie theory for state estimation in robotics. *arXiv preprint arXiv:1812.01537*.

[Steyer et al., 2017] Steyer, S., Tanzmeister, G., and Wollherr, D. (2017). Object tracking based on evidential dynamic occupancy grids in urban environments. In *2017 IEEE Intelligent Vehicles Symposium (IV)*, pages 1064–1070. IEEE.

[Steyer et al., 2018] Steyer, S., Tanzmeister, G., and Wollherr, D. (2018). Grid-based environment estimation using evidential mapping and particle tracking. *IEEE Transactions on Intelligent Vehicles*, 3(3):384–396.

[Stoyanov et al., 2013] Stoyanov, T., Saarinen, J., Andreasson, H., and Lilienthal, A. J. (2013). Normal distributions transform occupancy map fusion: Simultaneous mapping and tracking in large scale dynamic environments. In *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 4702–4708. IEEE.

[Stuelpnagel, 1964] Stuelpnagel, J. (1964). On the parametrization of the three-dimensional rotation group. *SIAM review*, 6(4):422–430.

[Sualeh and Kim, 2019] Sualeh, M. and Kim, G.-W. (2019). Dynamic multi-lidar based multiple object detection and tracking. *Sensors*, 19(6):1474.

[Sun et al., 2018] Sun, K., Mohta, K., Pfrommer, B., Watterson, M., Liu, S., Mulgaonkar, Y., Taylor, C. J., and Kumar, V. (2018). Robust stereo visual inertial odometry for fast autonomous flight. *IEEE Robotics and Automation Letters*, 3(2):965–972.

[Sun et al., 2020] Sun, P., Kretzschmar, H., Dotiwalla, X., Chouard, A., Patnaik, V., Tsui, P., Guo, J., Zhou, Y., Chai, Y., Caine, B., Vasudevan, V., Han, W., Ngiam, J., Zhao, H., Timofeev, A., Ettinger, S., Krivokon, M., Gao, A., Joshi, A., Zhang, Y., Shlens, J., Chen, Z., and Anguelov, D. (2020). Scalability in perception for autonomous driving: Waymo open dataset. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*.

[Tanzmeister and Wollherr, 2016] Tanzmeister, G. and Wollherr, D. (2016). Evidential grid-based tracking and mapping. *IEEE Transactions on Intelligent Transportation Systems*, 18(6):1454–1467.

[Team, 2020] Team, O. D. (2020). Openpcdet: An open-source toolbox for 3d object detection from point clouds. Available at `https://github.com/open-mmlab/OpenPCDet`, accessed on 02.02.2024.

[Teschner et al., 2003] Teschner, M., Heidelberger, B., Müller, M., Pomerantes, D., and Gross, M. H. (2003). Optimized spatial hashing for collision detection of deformable objects. In *Vmv*, volume 3, pages 47–54.

[Thrun et al., 2005] Thrun, S., Burgard, W., and Fox, D. (2005). *Probabilistic Robotics*. MIT Press, Cambridge, MA.

[Titterton and Weston, 2004] Titterton, D. and Weston, J. L. (2004). *Strapdown inertial navigation technology*, volume 17. IET.

[Volkswagen, 2020] Volkswagen (2020). Car2x in the new golf: A "technological milestone". Available at `https://www.volkswagen-newsroom.com/en/stories/car2x-in-the-new-golf-a-technological-milestone-5919`, accessed on 02.02.2024.

[Wendel, 2007] Wendel, J. (2007). *Integrierte navigationssysteme: sensordatenfusion, GPS und inertiale navigation*. Oldenbourg Wissenschaftsverlag GmbH.

[Yang et al., 2020] Yang, Z., Sun, Y., Liu, S., and Jia, J. (2020). 3dssd: Point-based 3d single stage object detector. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 11040–11048.

[Yang et al., 2019] Yang, Z., Sun, Y., Liu, S., Shen, X., and Jia, J. (2019). Std: Sparse-to-dense 3d object detector for point cloud. In *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 1951–1960.

[Yeong et al., 2021] Yeong, D. J., Velasco-Hernandez, G., Barry, J., and Walsh, J. (2021). Sensor and sensor fusion technology in autonomous vehicles: A review. *Sensors*, 21(6):2140.

[Zamanakos et al., 2021] Zamanakos, G., Tsochatzidis, L., Amanatiadis, A., and Pratikakis, I. (2021). A comprehensive survey of lidar-based 3d object detection methods with deep learning for autonomous driving. *Computers & Graphics*, 99:153–181.

[Zeng et al., 2018] Zeng, Y., Hu, Y., Liu, S., Ye, J., Han, Y., Li, X., and Sun, N. (2018). Rt3d: Real-time 3-d vehicle detection in lidar point cloud for autonomous driving. *IEEE Robotics and Automation Letters*, 3(4):3434–3440.

[Zhai et al., 2021] Zhai, M., Xiang, X., Lv, N., and Kong, X. (2021). Optical flow and scene flow estimation: A survey. *Pattern Recognition*, 114:107861.

[Zhang et al., 2022] Zhang, G., Yin, J., Deng, P., Sun, Y., Zhou, L., and Zhang, K. (2022). Achieving adaptive visual multi-object tracking with unscented kalman filter. *Sensors*, 22(23):9106.

[Zhang and Singh, 2014] Zhang, J. and Singh, S. (2014). Loam: Lidar odometry and mapping in real-time. In *Robotics: Science and systems*, volume 2, pages 1–9. Berkeley, CA.

[Zhang and Scaramuzza, 2018] Zhang, Z. and Scaramuzza, D. (2018). A tutorial on quantitative trajectory evaluation for visual (-inertial) odometry. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 7244–7251. IEEE.

[Zhou and Tuzel, 2018] Zhou, Y. and Tuzel, O. (2018). Voxelnet: End-to-end learning for point cloud based 3d object detection. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4490–4499.