Robot Localization and Mapping in Dynamic Environments

Philipp Ruchti

Albert-Ludwigs-Universitat Freiburg im Breisgau
Technische Fakultät

Dissertation zur Erlangung des akademischen Grades
Doktor der Naturwissenschaften

Betreuer: Prof. Dr. Wolfram Burgard

2019

UNI
FREIBURG

Dekanin: Prof. Dr. Hannah Bast
Erstgutachter und Betreuer der Arbeit: Prof. Dr. Wolfram Burgard
Zweitgutachter: Asst. Prof. Dr. Joschka Boedecker
Datum der mündlichen Prüfung: 1. Juli 2019

Eidesstattliche Versicherung

gemäß § 7 Absatz 1 Satz 3 Nr. 7 der Promotionsordnung der Albert-Ludwigs-Universität für die Technische Fakultät

1. Bei der eingereichten Dissertation zu dem Thema
   Robot Localization and Mapping in Dynamic Environments
   handelt es sich um meine eigenständig erbrachte Leistung.

2. Ich habe nur die angegebenen Quellen und Hilfsmittel benutzt und mich keiner unzulässigen Hilfe Dritter bedient. Insbesondere habe ich wörtlich oder sinngemäß aus anderen Werken übernommene Inhalte als solche kenntlich gemacht.

3. Die Dissertation oder Teile davon habe ich

   (Zutreffendes bitte ankreuzen)

   [✓] bislang nicht an einer Hochschule des In- oder Auslands als Bestandteil einer Prüfungs- oder Qualifikationsleistung vorgelegt.

   ☐ wie folgt an einer Hochschule des In- oder Auslands als Bestandteil einer Prüfungs- oder Qualifikationsleistung vorgelegt:

   Titel der andernorts vorgelegten Arbeit: ______________________

   Name der betreffenden Hochschule: ______________________

   Jahr der Vorlage der Arbeit: ______________________

   Art der Prüfungs- oder Qualifikationsleistung: ______________________

4. Die Richtigkeit der vorstehenden Erklärungen bestätige ich.

5. Die Bedeutung der eidesstattlichen Versicherung und die strafrechtlichen Folgen einer unrichtigen oder unvollständigen eidesstattlichen Versicherung sind mir bekannt.

Ich versichere an Eides statt, dass ich nach bestem Wissen die reine Wahrheit erklärt und nichts verschwiegen habe.

______________________ ______________________

Ort und Datum Unterschrift

Zusammenfassung

Bereits Jahre bevor selbstfahrende Autos ein aktives Forschungsgebiet wurden, haben sich Roboter von statischen Maschinen, welche repetitiven Aufgaben an Fließbändern erledigen, hin zu mobilen Robotern entwickelt, welche in für Menschen gemachten Umgebungen vielfältige Aufgaben erledigen. Eine wichtige Fähigkeit eines mobilen Roboters ist es sich in seiner Umgebung zu lokalisieren. Bewegt sich der Roboter im Freien, kann er mittels des *Global Positioning System* (GPS) lokalisiert werden. Befindet sich der Roboter jedoch in engen städtischen Umgebungen oder in Gebäuden, so muss der Roboter relativ zu einer Karte der Umgebung lokalisiert werden. In dieser Arbeit werden verschiedene Methoden zur Lokalisierung und Kartierung mittels mobiler Roboter vorgestellt.

Zuerst präsentieren wir eine Methode, um einen Roboter auf einer Graph-basierten Karte, wie sie beispielsweise von OpenStreetMap kommt, zu lokalisieren. Hierzu ist der Roboter mit einer Radodometrie sowie einem 3D Laserscanner ausgestattet. Zur Lokalisierung verwenden wir einen Partikelfilter, welcher die Radodometrie nutzt um die Bewegung des Roboters zwischen zwei Scans zu schätzen und die Partikel zu verschieben. Die 3D Scans werden in eine 2D Repräsentation überführt und anschließend verwendet um Straßen in der Umgebung zu erkennen. Diese Repräentation wird mittels eines *Feature*-basierten *Boostings* in Bereiche, welche Straße repräsentieren und solche, die andere Umgebung zeigen, eingeteilt. In unseren Experimenten zeigen wir, dass dieses Verfahren anderen modernen Methoden zur Lokalisierung auf OpenStreetMap-Karten überlegen ist, da es nicht notwendig ist, dass der Roboter auf den Straßen fährt. Die Lokalisierung auf öffentlich zugänglichen OpenStreetMap-Karten erlaubt es dem Roboter zuvor ungesehene Bereiche zu besuchen und sich dort zu lokalisieren – ohne dass vorher eine Karte erzeugt werden muss.

Nicht immer ist es ausreichend sich auf einer Graph-basierten Repräsentation des Straßennetzwerkes zu lokalisieren, da dieses nur den groben Verlauf der Straßen abbildet und üblicherweise keine Information über die Bereiche zwischen diesen bereitstellt. Daher befassen wir uns im weiteren Verlauf dieser Arbeit mit der Erzeugung von dreidimensionalen Rasterkarten. Wenn Karten in dynamischen, beispielsweise städtischen, Umgebungen erzeugt werden sollen, müssen sich ändernde Bereiche erkannt und speziell behandelt werden. Im zweiten Teil unserer Arbeit stellen wir daher zwei Verfahren vor, um dynamische von statischen Messungen in einzelnen 3D Scans zu unterscheiden. Zuerst präsentieren wir eine Methode welche mittels eines *Random Decision Forests* und auf benachbarten Punkten berechneten *Features* Wahrscheinlichkeiten bestimmt, dass eine Messung von einem dynamischen Objekt, wie einem Auto oder Fußgänger, erzeugt wurde. Anschließend stellen wir eine Methode für die Dynamikerkennung vor, welche ohne das Design und die explizite Berechnung von *Features* auskommt. Diese verwendet hierzu einen *Deep Learning* Ansatz, welcher direkt auf der Tiefenbild-Repräsentation eines 3D Scans die Wahrscheinlichkeit, dass ein Punkt ein dynamisches Objekt repräsentiert, errechnet. Diese Methode benötigt zum Lernen mehr Trainingsdaten, ist aber schneller und präziser in der Erkennung dynamischer Objekte. In unseren Experimenten vergleichen wir beide Methoden und zeigen, dass beide Methoden sowohl sich bewegende Objekte, wie fahrende Autos, als auch Objekte, welche sich bewegen können, beispielsweise stehende Fußgänger oder geparkte Fahrzeuge, verlässlich erkennen.

Anschließend an die Erkennung dynamischer Bereiche in 3D Scans präsentieren wir ein Verfahren, welches mit Hilfe dieser berechneten Wahrscheinlichkeiten eine 3D Rasterkarte erstellt, welche lediglich die statischen Teile einer Umgebung repräsentiert. Eine solche Karte ist Hilfreich zur Lokalisierung eines mobilen Roboters, da die Karte über einen längeren Zeitraum den Bereich korrekt darstellt. In der Auswertung unseres Kartierungsansatzes zeigen wir, dass dynamische Objekte wie Fahrzeuge, welche möglicherweise beim nächsten Besuch der Umgebung ihren Standort gewechselt haben oder verschwunden sind, nicht in unserer Karte repräsentiert werden.

Im letzten Teil unserer Arbeit zeigen wir wie die Dynamikerkennung und der Kartierungsansatz erweitert werden können, um, zusätzlich zur Belegtheit einer Zelle der Karte, auch den Zustand eines darin repräsentierten Objektes zu schätzen. Hierzu modifizieren wir den *Deep Learing* Ansatz zur Dynamikerkennung insofern, dass wir statische, stehende und sich bewegende Objekte unterscheiden können. Da sich dynamische Objekte während der Kartierung bewegen können, kann sich unsere Karte während der Erstellung ändern. Um immer den neusten Zustand der Umgebung zu repräsentieren, müssen wir solche Änderungen detektieren. Wir evaluieren hierzu Methoden zur Erkennung eines Zustandswechsels in einer Folge von Messungen. Anschließend zeigen wir, dass die so erzeugten Karten zuverlässlich den neusten Zustand der Umgebung abbilden.

Zusammenfassend stellen wir in dieser Arbeit Verfahren im Bereich der Lokalisierung, Dynamikerkennung und Kartierung vor. Wir präsentieren zunächst eine Methode, um einen mobilen Roboter auf einer OpenStreetMap-Karte zu lokalisieren. Anschließend stellen wir Algorithmen vor, um dynamische von statischen Messungen zu unterscheiden und hiermit Karten der statischen Teile einer Umgebung zu erstellen. Des Weiteren zeigen wir, wie diese Methoden erweitert werden können, um neben der Belegtheit auch den dynamischen Zustand von Objekten in der Karte zu schätzen.

Abstract

Years before autonomous cars became an active research area, robots found their way from doing repetitive tasks on factory floors into becoming mobile robots deployed to environments made for humans. One key-functionality of a mobile robot is the ability to localize itself. In this thesis, we propose different methods for localization and mapping in dynamic environments using mobile robots.

In the first part of this thesis, we propose a particle filter to localize a robot equipped with a LiDAR sensor on a graph-based road map provided by OpenStreetMap. We present a road classification of 3D scans as well as a novel sensor model to be used in the particle filter. In contrast to other OpenStreetMap-based localization methods, our approach does not require the robot to travel on the roads.

Localization on a graph-based representation of the road network is not always sufficient, as the network only represents the coarse course of the roads and usually does not provide any information about the areas in between roads. Therefor, we proceed by presenting methods to build 3D grid maps. In dynamic environments, parts of the map can change due to moving objects. To detect such changes, we present in the second part of this thesis two methods to predict the probability that measurements were caused by dynamic objects. The first method uses local features while the second method employs a Deep Learning approach.

In the third part of this thesis, we show how these *dynamic object probabilities* can be used to generate 3D grid maps representing the static aspects of a dynamic environment.

In the last part of this thesis, we show how the dynamic detection and the mapping algorithm can be extended to estimate a map representing, in addition to the reflectance probability, a distribution of dynamic objects in the environment. This allows path planning or scan matching algorithms to treat dynamic areas differently. By representing dynamic objects in the map, we need to detect which parts of the environment change due to dynamics. Therefor, we present methods to detect such changing map parts.

We trained and evaluated our proposed methods on real-world data collected by our own robots and from the KITTI dataset. In extensive experiments, we show that, we can robustly localize a mobile robot on a graph-based road network and that our proposed mapping methods are well suited to learn robust maps in dynamic environments.

Acknowledgements

At this point I would like to thank all individuals who made this thesis possible.

First, I would like to thank Prof. Dr. Wolfram Burgard for the opportunity to do my Ph.D. at his great group. I am grateful for the freedom to choose my way of research and for the enjoyable work environment provided.

I would like to thank Bastian Steder and Michael Ruhnke from which I received a lot of helpful advice on how to solve my problems and answers to seemingly stupid questions saving me a lot of time.

I also thank my colleagues for their opinions, discussions and advices.
In particular:

- Johan Vertens for its help with the neural network for estimation of *dynamic object probabilities* and answers to Deep Learning related questions.
- Lukas Luft for his help developing a break point detection method to detect changing map parts.
- Ayush Dewan for LiDAR data which did not made it into this thesis and his answers on LiDAR related questions.
- My office colleagues Andreas and Daniel Kuhner for discussion on problems related to mathematics, LaTeX, C++, methods and general questions.

Last but not least I would like to thank my wife and kids for believing in me and this thesis. Especially during the writing part they were of vital support and bought me a lot of valuable time without which this thesis would not have succeeded in time.

Contents

CHAPTER 1

Introduction

In recent years autonomous cars became a field of utmost importance and help to make traffic safer and more efficient. Intelligent cars can help a tired or unobservant driver to avoid fatal crashes by detecting dangerous situations and warn the driver or even solve the situation themselves. With a rising number of fully autonomous cars traffic can become more efficient. For example, stop-and-go traffic can transform into slow but moving traffic if cars communicate with each other. Fully autonomous cars would also allow drivers to work, relax or even sleep during their ride.

While highly autonomous cars are just recently emerge in large numbers, mobile robots found their way already into other fields. Today, many exhausting tasks are already carried out by mobile robots. Autonomous forklifts are moving goods in warehouses [15, 42] while cleaning robots with different levels of intelligence are cleaning personal homes [20] or business rooms [34]. Mobile robots can also go where humans can not – including catacombs that are in danger of collapsing [92]. Autonomous underwater vehicle can dive deep in the oceans [49] or for days under the arctic ice [32]. There are even mobile robots deployed on mars [40].

Depending on their environment and application, mobile robots need to solve different tasks including localization, mapping, path and action planning, perception, reasoning and much more. In this thesis, we focus on the tasks of localization, mapping and dynamic detection. Mobile robots moving outside are often equipped with global positioning system (GPS) (or similar global navigation satellite systems such as GLONASS) receivers allowing for a global localization up to sub-meter accuracy. GPS works by triangulation of the time of flight of position signals received from at least four satellites. However, if the robot drives indoors or when large obstacles, for example buildings or close by mountains, limit the view to the sky, GPS does not work reliably. Another approach is to localize the robot relative to a map. In this thesis, we propose methods for localization and mapping in dynamic environments using a mobile robot.

There are various types of maps. On the one hand, there are maps representing the full 3D geometry of an environment, on the other hand maps representing high level informations such as the road network. A popular example of the later are road maps printed on paper which humans used for years to navigate. In Chapter 3, we propose a method to localize a mobile robot equipped with a 3D LiDAR sensor and wheel-odometry on a graph-based road map provided by OpenStreetMap. We use boosting to classify 3D scans into road and non-road cells. Building on this classification, we propose a novel sensor model to be used in the particle filter to weight particles on how well they are aligned with the road network. In contrast to other OpenStreetMap-based localization methods, our method do not force the robot to travel on the roads.

While localization on road networks is helpful to navigate towards a distant target, especially in previously unvisited locations, the localization accuracy is limited by the road network. If the task at hand requires a finer localization or if information about free and occupied parts of the environment is required, volumetric maps are a common choice. We therefor present in the second part of this thesis methods to build 3D grid maps which capture the geometry of the environment. To generate consistent maps in dynamic environments, we detect measurements caused by dynamic objects and treated these different than such created by static parts of the environment. In Chapter 4 and 5, we propose two methods to estimate probabilities that measurements were created by dynamic objects. The first method uses features computed on local point neighborhoods and a random decision forest to estimate probabilities for each point in a single 3D scan that this measurement was caused by a dynamic object, such as a car or a pedestrian. The second method uses a Deep Learning approach to predict these probabilities end-to-end without the design and explicit computation of features. While this method needs a large training set it is faster during prediction and estimates the probabilities with higher accuracy.

Dynamic objects, such as parked cars, might disappear or change their position if a robot revisits an environment after some time causing localization uncertainty. For this reason dynamic objects should not be present in a map or marked as such. In Chapter 6 we apply our probabilities that measurements were caused by dynamic objects to generate 3D grid maps representing the static aspects of dynamic environments. The maps represent per cell the probability that a beam hitting a cell is reflected by a static object.

In Chapter 7, we show how the dynamic detection and the mapping algorithm can be extended to estimate a map representing, in addition to the reflectance probability, a distribution about dynamic objects in the map. This allows path planning or scan matching algorithms to treat areas were dynamics occur differently. By representing dynamic objects in the map, we need to keep track of changing map parts. We therefor present methods to detect changes in a stream of measurements.

The methods presented in this thesis allow mobile robots to localize themselves in previously unseen environments and build maps in highly dynamic environments, such as urban traffic or pedestrian areas.

1.1 Contribution of this Thesis

In this thesis, we present methods for localization, dynamic detection and mapping in dynamic environments.

- In Chapter 3, we present a method to localize a mobile robot on a graph-based representation of the road network. We propose a classification of the robot's LiDAR scans into road and non-road cells. This classification is used in a novel sensor model to relate the scans to the road graph in our proposed particle filter. Other than related work this method does not require the robot to travel on the roads but allows it to leave the road network.
- To estimate in single 3D scans the probability that measurements were caused by dynamic objects, we propose in Chapter 4 a method based on a random decision forest. It uses manual designed features on local point neighbors. The method distinguishes measurements generated by static objects from dynamic ones while also detecting objects that can move such as parked cars.
- In Chapter 5, we propose a Deep Learning approach for the same task in an end-to-end fashion. This approach is faster and more accurate than the feature-based method.
- Using the proposed probabilities that measurements were created by dynamic objects, we introduce in Chapter 6 a mapping approach to build 3D grid maps representing only the static aspects of a dynamic environment.
- In Chapter 7, we show how these methods can be extended to build 3D grid maps representing, in addition to the reflectance probability, a distribution of dynamics objects represented in the map. As the environment might change during mapping, we propose methods to detect those changes for robust mapping.

1.2 Publications

Parts of this thesis are based on work published in the following papers.

- Philipp Ruchti, Bastian Steder, Michael Ruhnke, and Wolfram Burgard. „Localization on OpenStreetMap Data using a 3D Laser Scanner." In: *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*. Seattle, Washington, USA, May 2015
- Philipp Ruchti and Wolfram Burgard. „Mapping with Dynamic-Object Probabilities Calculated from Single 3D Range Scans." In: *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*. Brisbane, Australia, May 2018

1.3 Collaborations

The work used in this thesis was developed under the supervision of Wolfram Burgard at the Autonomous Intelligent Systems Lab (AIS).

The publication *„Localization on OpenStreetMap Data using a 3D Laser Scanner"* [63] as well as Chapter 4 was developed under additional supervision and with help of Bastian Steder and Michael Ruhnke.

For Chapter 7 about multi-class mapping Lukas Luft helped to develop the break point detection methods for streams of histogram measurements.

Chapter 2

Fundamentals

In this chapter, we present the basic concepts used in this thesis. After the introduction of abbreviations and mathematical notations, we discuss the topic of localization and the particle filter as one implementation of the Bayes filter to solve this task. Afterward, we shortly introduce the topic of mapping. We then present different learning methods used in this thesis. The chapter ends with a description of point clouds, the map data from OpenStreetMap and the employed robots.

2.1 Abbreviations

There are some abbreviations which are frequently used throughout this thesis which we define in Table 2.1.

Short	Meaning	Description
GPS	Global Positioning System	Satellite-based global localization system
IMU	Inertial measurement unit	A sensor to measure accelerations and rotations
osm	OpenStreetMap	An open license, user-generated world map
MCL	Monte Carlo Localization	A sample-based localization method
LiDAR	Light Detection And Ranging	A method to generate laser scans
IoU	Intersection over Union	A quality measure for labeling
KITTI	KITTI dataset [22]	A public dataset including LiDAR scans

Table 2.1: Abbreviations used in this thesis

2.2 Mathematical Notations

In this section, we introduce the most relevant mathematical notations. Our used notations are listed in Table 2.2 ordered by their first occurrence.

Symbol	Meaning
x	robot pose
z	measurement
t	time index
$bel(\circ)$	belief about $\circ$
$p(\circ)$	probability of an event $\circ$
$p(\circ \mid \bullet)$	probability of an event $\circ$ conditioned on the event $\bullet$
μ, σ^2	mean and variance
w	particle weight
$\langle x^i, w^i \rangle$	particle
$m, m^\star$	map and most likely map
n	map cell
$(x, y), (x', y')$	pixel positions in a range image
I	intensity
Z^t, Z^t_i	histogram measurement and histogram value
$\mathbf{Z}^t$	merged histograms
$\hat{\mathbf{Z}}^t$	merged histograms estimate
$H(\cdot)$	entropy
b	break point in a stream of measurements
H	hits during mapping
M	misses during mapping

Table 2.2: Mathematical notation, used in this thesis.

2.3 Localization

In the context of robotics, localization is the task to determine the pose of a robot, either globally or relative to the environment. If the robot navigates outdoors often the Global Positioning System (GPS) is used to globally localize a robot within a radius of a few meters. GPS works by triangulating satellite position messages by the time of flight of the signal. The accuracy of the estimate depends on the amount of satellites visible. In cases in which the clear view of the sky is occluded by buildings or trees, the accuracy drops heavily. Other than the global localization provided by GPS, a robot can also be localized relative to a map of the environment.

For this task two types of sensor informations are usually employed. First, a sensor to perceive the environment and second, a sensor to estimate the motion of the robot between two environment measurements. Based on the environment type and application of the robot, different sensors might be applied to perceive the environment. Such sensors include color or 3D cameras, LiDAR or radar sensors. To estimate the motion of the robot, the odometry of the robot can be estimated by counting wheel revolution, integrating IMU measurements or by matching sensor readings against each other to estimate the displacement of the robot.

2.3.1 Bayes Filter

No sensor is perfect [21], such that methods for localization need to deal with the resulting uncertainty. The Bayes Filter is a recursive filter to estimate the state of a dynamic system where only noisy measurements are available. In the context of localization, this state would be the pose of the robot and the measurements would be observations of the environment, such as laser scans. The Bayes filter estimates the belief $bel\left(x\right)$ over a random variable x as a probability distribution. The belief at a time t is estimated using all measurements seen up to this point:

$$bel\left(x_t\right) = p\left(x_t \mid z_1, \ldots, z_t\right). \tag{2.1}$$

This estimation gets exponential more complex with new measurements. The Markov assumption [5] states in this case, that given the state x_{t-1} is known, we no longer gain any information from the old measurements or states. We can use this to recursively compute the belief $bel\left(x_t\right)$ using only the old belief $bel\left(x_{t-1}\right)$ and the new measurement z_t. The Bayes filter updates the belief whenever it receives a new measurement by a prediction and a correction step. In the prediction step, we predict how the $bel\left(x_t\right)$ changes given the old belief $bel\left(x_{t-1}\right)$ as follows:

$$\overline{bel}\left(x_t\right) = \int p\left(x_t \mid x_{t-1}\right) bel\left(x_{t-1}\right) dx_{t-1} \tag{2.2}$$

In the correction step we use the new measurement z_t to correct this prediction:

$$bel\left(x_t\right) = \eta p\left(z_t \mid x_t\right) \overline{bel}\left(x_t\right), \tag{2.3}$$

where η is a normalization factor. In the case of robot localization, we do not only have sensor measurements z_t but also odometry readings or actions u_t. This changes our belief estimation as follows: $bel\left(x_t\right) = p\left(x_t \mid z_1, \ldots, z_t, u_1, \ldots, u_t\right)$. The actions are considered in the prediction step of the filter as they give a hint on how the state x_t evolves from the old state x_{t-1}. This results in the updated prediction step:

$$\overline{bel}\left(x_t\right) = \int p\left(x_t \mid u_t, x_{t-1}\right) bel\left(x_{t-1}\right) dx_{t-1} \tag{2.4}$$

An implementation of the Bayes filter is the Kalman filter (see [73] and [31]). The belief is represented by a Gaussian distribution: $bel\left(x\right) \sim N\left(\mu, \sigma^2\right)$. If the state can be estimated linearly from the measurements and the underlying noise is Gaussian the Kalman filter is an optimal filter.

2.3.2 Particle Filter

In some cases the estimation of a single Gaussian distribution is not sufficient. For example in the case of localization, the belief about the robot's pose might become non Gaussian or even multi-modal. To cope with this problem, we use a particle filter [75] to localize the robot. The particle filter is a Monte Carlo approximation of the Bayes filter. The belief is not represented by a closed form distribution, as in the Kalman filter, but is estimated by maintaining a set of samples. Each sample keeps a hypothesis about the belief and is weighted by how well it explains the seen measurements. In the particle filter the belief is represented by a finite set of weighted hypotheses called particles:

$$\left\{\left\langle x^i, w^i\right\rangle \mid i = 1, \dots, N\right\}, \tag{2.5}$$

where x^i represents the state hypothesis and w^i the weight of the i-th particle. Each particle represents a single hypothesis. The belief about the state can be estimated as:

$$bel(x) = \sum_{i=1}^{N} w^i \delta(x - x^i), \tag{2.6}$$

where $\delta(x)$ is the Dirac delta function which is 1 at a value of $x = 0$ and 0 elsewhere.

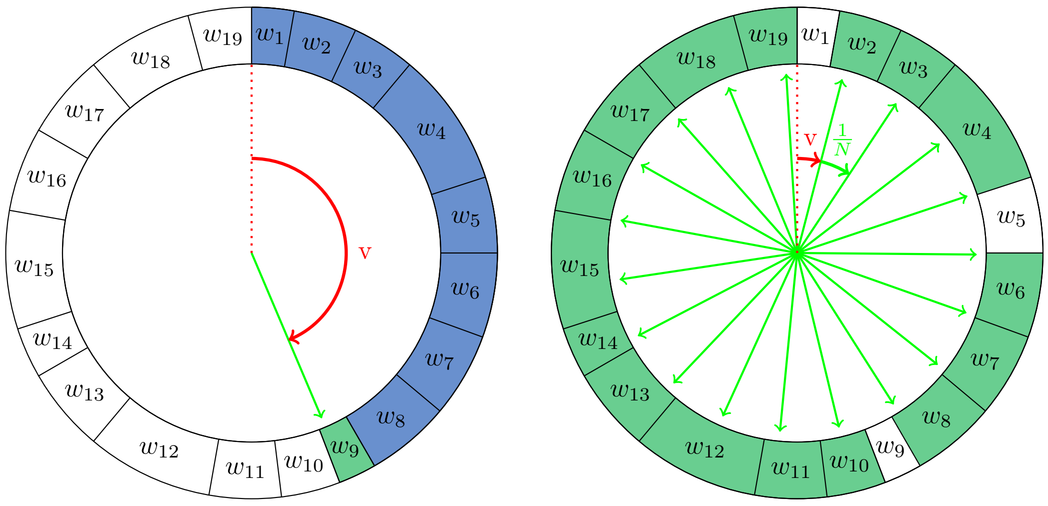
In the prediction step, for each particle the hypothesis x_t is modified by sampling from the distribution $p\left(x_t \mid x_{t-1}\right)$ or $p\left(x_t \mid x_{t-1}, u_t\right)$ if actions are given. In the correction step, the importance weight of each particle is computed based on how well the measurement z_t is explained by the state hypothesis x_t:

$$w = \eta p\left(z_t \mid x_t\right). \tag{2.7}$$

where $\eta = \left(\sum_{i=1}^{N} w^i\right)^{-1}$ is a normalization factor, such that all weights sum up to one.

As a particle filter can only maintain a finite set of particles, we need to keep the particles in areas of the distribution with a high probability density. This introduces a third step to our algorithm: the resampling. In this step a new set of particles is sampled, where the chance of survival for each particle is proportional to its weight in the old particle set. It replaces less likely particles with copies of such with high probability.

There are different methods for resampling. By sampling N-times a value v from the interval $[0,1]$ and taking the first particle $\langle x^j, w^j\rangle$ where $v \leq \sum_{i=1}^{j} w^i$, we receive particles were the survival is proportional to each particles weight. This sampling process is called roulette wheel sampling and is depicted in Figure 2.1a. The shaded area visualizes the sum of weights denoted above. The green particle w_9 is chosen. The process is repeated N times independently. This independent sampling can lead to the case where particles with high weights are not represented in the new particle set. Another method to sample a new set of particles is the stochastic universal sampling depicted in Figure 2.1b. In contrast to the roulette wheel sampling, all N particles are sampled at a time. First, a value v is sampled from the interval $[0, N^{-1}]$. As in the first sampling process the particle where



(a) Roulett wheel sampling **(b)** Stochastic universal sampling

Figure 2.1: Two different sampling methods which can be used for the resampling process in the particle filter. In the roulette wheel sampling, each particle is sampled individually. The stochastic universal sampling samples a set of N particles at once.

$v \leq \sum_{i=1}^{j} w^i$ is added to the particle set. After this, v is increased by N^{-1} and the next particle is chosen. This process is repeated N times. The stochastic universal sampling ensures that each particle with a weight of N^{-1} or higher is added to the new particle set. In both methods all particles receive a weight of N^{-1} after the sampling process.

Depending on the application, the copying of particles can be a costly process and even with the stochastic universal sampling the resampling process bears the risk of dropping particles around the true state. For this reason, the number of resampling operations should be kept as low as possible.

With an increasing number of particles, the estimate of the belief gets more robust and precise. On the other hand, more particles increase the computational load. As a result, the number of particles used is a tradeoff between computational performance and quality of the approximation.

2.4 Mapping

The task of mapping with known poses is to determine the most likely map from a set of measurements z_t were the corresponding sensor pose x_t is given:

$$m^\star = \underset{m}{\operatorname{argmax}}\, p\left(m \mid z_1, \dots, z_N, x_1, \dots, x_N\right). \tag{2.8}$$

The process of estimating the map heavily depends on the type of map generated. Feature maps store the positions of features such as trees, corners or beacons.

The mapping estimates the distributions over the position of single features. For example, each feature position can be estimated using a Kalman filter. This map representation is compact and suited for localization but does not represent the structure of the environment. On the other hand, grid maps estimate a volumetric representation of the environment. The environment is subdivided into discrete cells. These cells can either be regular or dependent on the environment as in octrees [50]. Each cell stores information about the environment in the defined space such as if a cell is free or occupied. To keep the generation of grid maps trackable the estimation of a cell is assumed to be independent of its neighbors. Using this independence assumption, we obtain:

$$p(m \mid z_1, \dots, z_N, x_1, \dots, x_N) = \prod_{n \in m} p\left(n \mid z_1, \dots, z_N, x_1, \dots, x_N\right), \tag{2.9}$$

where n represents a single map cell. The estimate for a map cell is computed with all measurements falling into this cell. Additional information, such as laser beams which cross a cell but does not ends in it, can be used. Depending on the application, the map represents different features of the environment. These include if a cell is free or occupied and the reflectance or color of the environment.

2.5 Learning Methods

In this section, we explain two methods we use during this thesis to learn which parts of a 3D scan belong to dynamic objects. First, we introduce the random decision forest, a feature-based machine learning method and the decision tree, the basic building block of the random decision forest. Both methods are based on the computation of features from the data. Instead of designing features, it is also possible to learn directly on the input, for example using a deep learning approach as presented afterward. We finalize this section by introducing the Intersection over Union (IoU) measure which we use to evaluate the result of our learned models.

The field of machine learning can be divided into two main tasks. On the one hand, the task of classification searches discrete labels for each input. An example of classification is the assignment of images of handwritten letters to the corresponding letters. On the other hand, regression searches to learn a function on the data to predict continues variables, like predicting the sale price of a car based on its age, manufacturer and color.

2.5.1 Decision Tree

A decision tree is a special tree used in machine learning either for the task of classification or regression. The tree structure used is usually a binary tree [8, 18, 35]. Each node divides the input space along an axis into two sub-regions which are then recursively represented as the left and right subtree. Each node represents a feature and a corresponding split value. If during prediction the feature value

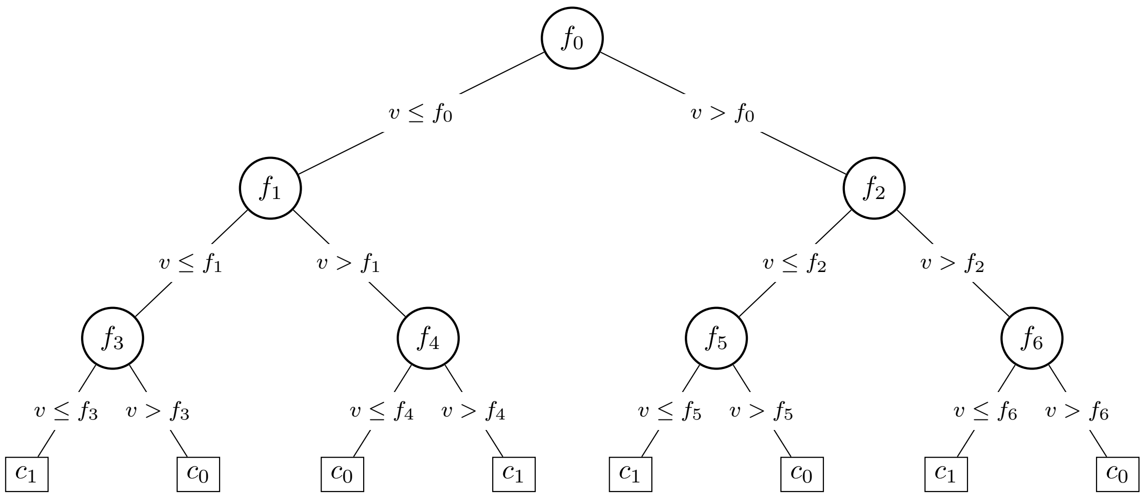


Figure 2.2: An example of a decision tree for the task of classification. Each node represents a feature which is computed on the input. Based on the result v the tree is traversed towards the left or right subtree. The leaves represent class labels.

computed on an input is smaller than the stored split value the right subtree is traversed, if it is larger the next feature is computed in the left subtree. In the case of classification each leaf stores a class label, during regression the leaves return real-valued function values. An example of a decision tree for the task of classification is shown in Figure 2.2. During learning the order of features and their corresponding split values are determined using the training data.

2.5.2 Random Decision Forest

A random decision forest [7, 28] is a set of decision trees, each trained with a randomized feature order. The random decision forest returns the combined result of the individual decision trees during prediction. By using a set of randomized decision trees the random decision forest becomes more robust to noise and reduces the problem of overfitting, which usually occurs in decision trees. During training, the randomization of the individual decision trees can be achieved by training each tree individually on a subset of the training data and features.

2.5.3 Deep Learning

Deep Learning [66] is a subfield of machine learning which is motivated by information processing occurring in the human brain. In the human brain, billions of neurons are connected by synapses to process information. Sensory input, such as the light perceived by the eye, activates neurons which then send out signals over the synapses to other neurons. An artificial neural networks, which is used in Deep Learning, consists of layers of artificial neurons. The term *Deep Learning* refers to the idea of learning neural networks with many inner or hidden layers. Each neuron of a hidden layer is connected to neurons of the predecessor and successor layer. The individual layers execute simple functions based on weights. By connecting the layers, the network is able to solve complex tasks. During training the weights of all layers are optimized to minimize a loss function on the training data. In each iteration, the computed error is back propagated through the network to update the weights. The neurons of the first or input layer are feed with the input data.

Neural networks are the method of choice for many end-to-end learning approaches on camera data in the fields of computer vision and robotic. Decision trees or random decision forests require an abstracted representation of the input, so called *features*. These features are usually designed manually which requires expert knowledge of both feature design and the task which one tries to solve and needs to be repeated every time the task changes. The quality of the prediction of a feature-based method depends heavily on the used features. On the other hand, neural networks allow to process raw data as input such as camera images. The network implicitly learns a hierarchical feature representation which is then directly used to solve the given task. On the one hand, end-to-end neural networks require no design of features, but on the other hand, require a huge amount of training data and computational power for training. Deep Learning became popular with the emerge of large datasets and powerful graphics processing units (GPU) which allow for fast and massively parallel training of large neural networks.

Deep Learning can be done on a variety of different network types depending on the application. In this thesis, we employ a Convolutional Neural Network (CNN) for the distinction of different dynamic classes in images generated from 3D scans. Convolutional neural networks consist of two parts. The encoder part generates a high dimensional feature vector by recursively breaking the input down to multiple layers of smaller sizes through convolution or pooling operations. The decoder part generates a result image by decoding the computed feature vectors recursively using deconvolutions or unpooling to obtain a single output layer. As a result of the convolutional structure of the network, the input size is not fixed allowing us to train the network on a different size of images than our images we want to annotate.

2.5.4 Intersection over Union (IoU)

The Jaccard index or intersection over union score is a measure to compare two sets. The measure returns a vale of 0 if no two elements of the sets are the same and 1 if the sets are identical. The score is computed as follows:

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|}. \tag{2.10}$$

The intuition of this Jaccard index is explained in Figure 2.3. By dividing the intersection by the union of the two sets we receive the above measure.

In the context of object detection, the Jaccard index is used under the name of intersection over union (IoU) to compare predicted bounding-boxes with their

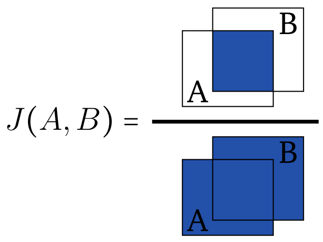


Figure 2.3: The motivation behind the Jaccard index. The intersection of both sets is divided by the union over the sets. The score represents the similarity of two sets.

ground truth. The more these boxes overlap the better the score. The measure also penalizes wrong sized bounding-boxes.

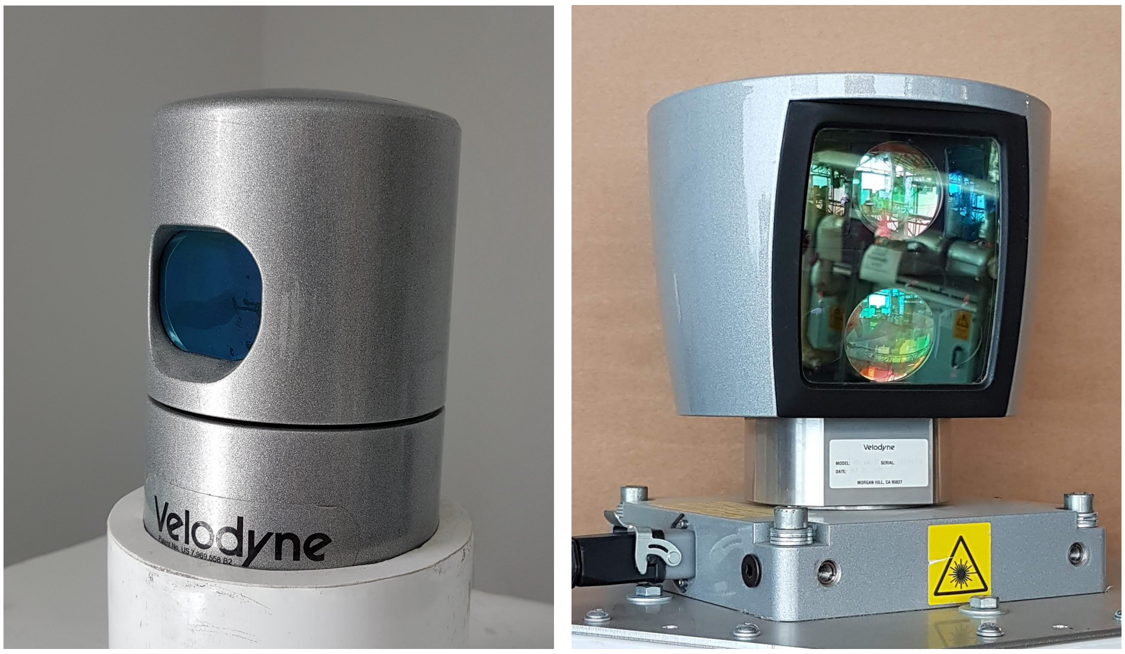
In our experiments, we compare pixel-wise predictions with pixel-wise ground truth labels. For a prediction $A = (a_1, \dots, a_N)$ and a corresponding ground truth $B = (b_1, \dots, b_N)$ we use the following equivalent notation to compute the IoU score for a class c:

$$\text{IoU}_c = \frac{\sum_{i=1}^{N} I(c, a_i, b_i)}{\sum_{i=1}^{N} I(c, a_i) + \sum_{i=1}^{N} I(c, b_i) - \sum_{i=1}^{N} I(c, a_i, b_i)}, \tag{2.11}$$

where $I(a, b, \dots)$ is the indicator function which is 1 iff all elements are the same and a_i,b_i are the predicted and ground truth value for the same pixel.

2.6 Point Cloud

A data structure used throughout this thesis is the point cloud. It is a collection of 3D points. A 3D point cloud $\boldsymbol{P} = (p_1, \dots, p_N)$ is a collection of points $p = (x, y, z, I)$, where $(x, y, z) \in \mathbb{R}^3$ defines the 3D coordinate of a point and I the returned intensity value. The point clouds used in this thesis are generated by either a Velodyne HDL-64E or Velodyne HDL-32E 3D laser scanner (see Figure 2.4). These scanners consist of 64 or 32 individual lasers arranged one above the other. By rotating the sensor head around its vertical axis, these sensors generate 3D point clouds. Each point cloud consists of all points of a full $360°$ rotation. The scanners generate data with 5 to $20\,\text{Hz}$ by adjusting the rotation speed. The speed of rotation defines the point count per point cloud as the laser fire rate does not change. The distance of the next obstacle is determined by the time of flight of the light pulse. Additionally, the intensity of the returned light is measured, which is determined by the surface reflectance. In the context of urban measurements, pavement returns almost no light while street signs and license plates return a high amount of light. An example of a point cloud from the KITTI dataset, generated with a Velodyne HDL-64E sensor mounted on the roof of a car, is shown in Figure 2.5. The sensor is located in the center of the concentric circles. The scene shows a crossing with multiple cars. The color represents the measured intensity.



(a) Velodyne HDL-32E **(b)** Velodyne HDL-64E

Figure 2.4: Velodyne LiDAR sensors. The laser scanners generate $360°$ point clouds by rotating their sensor heads equipped with 32 respective 64 individual lasers.

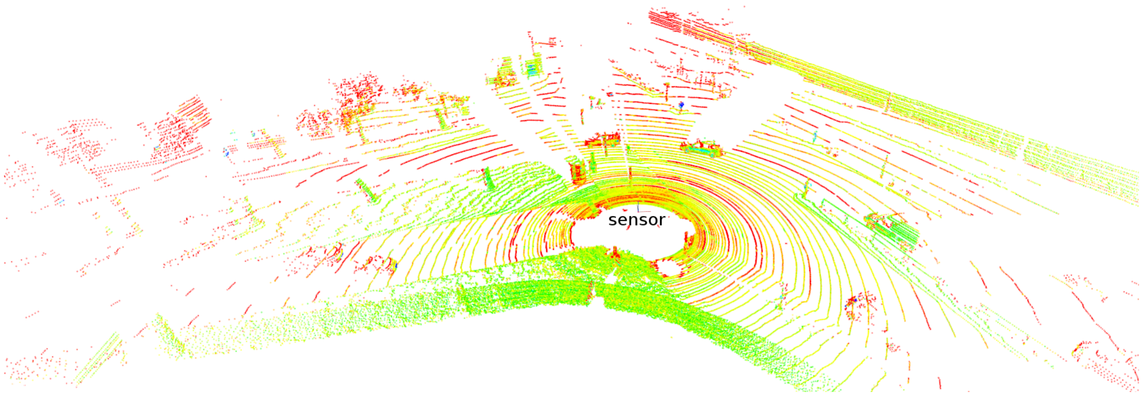
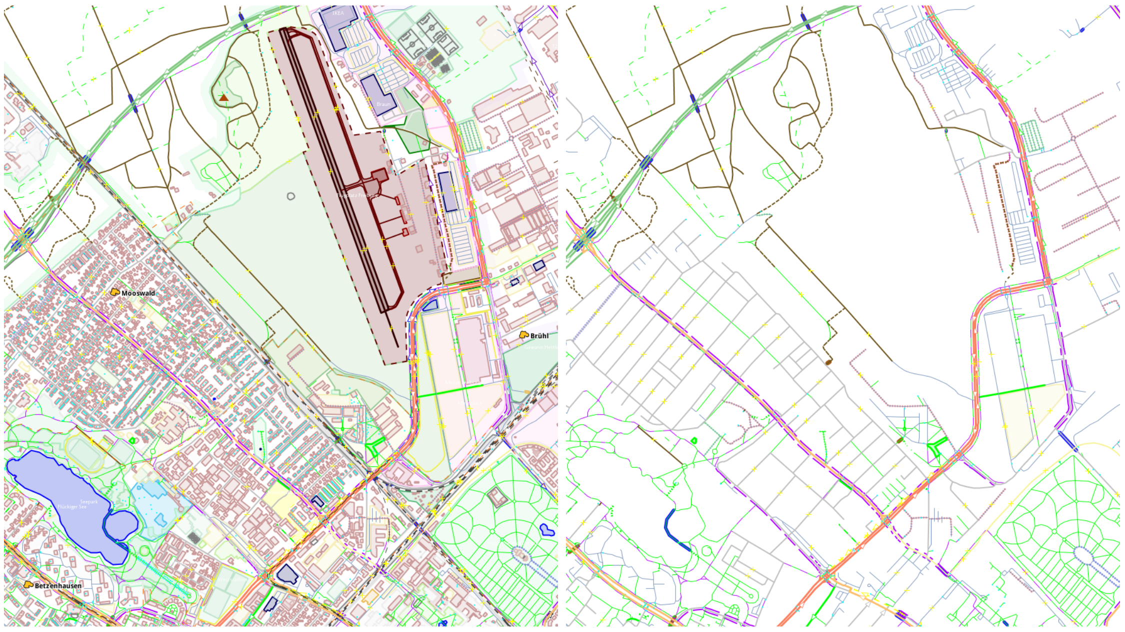


Figure 2.5: A point cloud generated by a Velodyne HDL-64E LiDAR sensor. It shows a street crossing with cars and trees. The sensor is located in the center and mounted on a car roof. Each of the concentric rings is generated by an individual laser. The color visualizes the intensity from high reflectance (blue) on license plates or the traffic sign to low reflectance (red) on the pavement and trees.



(a) OpenStreetMap **(b)** Street network

Figure 2.6: A cutout of an OpenStreetMap around our campus. The left image shows all information, while the right image depicts only the road network.

2.7 OpenStreetMap

OpenStreetMap [57] is an open license, user-generated map of the world. Everyone with a GPS-enabled device, such as a smartphone, can record data and modify the map using this data. The map consists of streets, building outlines, regions, such as forests or parks, and much more. The map type is comparable to a classic street map printed on paper. An example of an OpenStreetMap is shown in Figure 2.6a, it shows an area around the Faculty of Engineering campus of the University of Freiburg. The data we are most interested in are the streets and their connections as shown in Figure 2.6b. Streets are represented by a list of nodes and information about its type. Each node stores a global coordinate computed from GPS information. This results in a graph structure which represents the connection and course of streets. It neither represents a street width nor the shape of crossings.

2.8 Robots

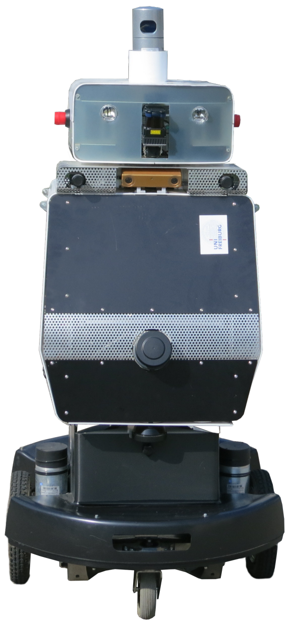
In this section, we present the two mobile robots used in this thesis. The robots are build for navigation in different environments.

2.8.1 Obelix

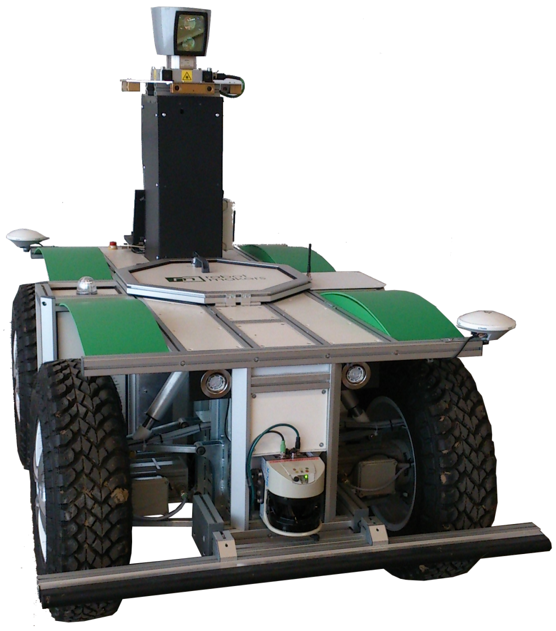
The robot used to carry out experiments in the chapter about osm localization (Chapter 3) is Obelix which is depicted in Figure 2.7a. This robot was developed by our lab, the Autonomous Intelligent Systems lab in Freiburg. It was part of the projects EUROPA [68] (EUropean RObotic Pedestrian Assistant) and EUROPA2 [70] which were funded by the European Commission. Obelix is designed to drive in urban environments on the sidewalk. He achieved to navigate like a pedestrian from our Faculty of Engineering campus to the city center of Freiburg (for more details see Kümmerle *et al.* [36]). The robot moves with a differential drive with two additional caster wheels. Its main sensors are a set of laser scanners. Two SICK LMS 151 are mounted horizontal in the front and back of the robot. A third scanner is tilted such that it is able to perceive the ground in front of the robot. The sensor we are mostly interested in is a Velodyne HDL-32E LiDAR sensor (see Section 2.6) mounted on the robots head. The motion of the robot is estimated by wheel encoders, a XSens IMU and a GPS module. The robot is also equipped with a pair of stereo cameras for object detection.

2.8.2 Viona

The second robot used in this thesis is Viona (Vehicle for Intelligent Outdoor NAvigation) which is shown in Figure 2.7b. The robot was developed for the LifeNav project [76] (Reliable Lifelong Navigation for Mobile Robots). The robot is powered by four electric motors and was built by Robot Makers. The front and back axis can be steered independently which allows to operate the robot in car like Ackermann, double Ackermann or crab steering. The later enables the robot to move sideways. We equipped the robot with a sensor tower and a set of sensors. A SICK LMS 511 LiDAR sensor is mounted on the front and back of the robot for 2D mapping and localization. The Velodyne HDL-64E 3D LiDAR described above is mounted on the top end of the sensor tower to be able to look over small obstacles. As a side effect of the high mounting of this sensor, it has a blind spot around the robot of approximately two meter radius which is reduced by a Velodyne VLP-16 LiDAR puck mounted at each side of the robot between the wheels. Below the Velodyne HDL-64E four Bumblebee2 stereo cameras are mounted. The robots is equipped with a differential GPS, an Applanix PosLV IMU and wheel encoders. Viona is designed for outdoor navigation and allows to traverse rough offroad terrain.



(a) Obelix



(b) Viona

Figure 2.7: The robots Obelix and Viona. Both robots are equipped with multiple 2D LiDAR sensors, stereo cameras, wheel encoders, GPS and IMU sensors. The sensor on top of Obelix is a Velodyne HDL-32E 3D LiDAR. On the tower of Viona a Velodyne HDL-64E 3D LiDAR is mounted.

CHAPTER 3

Localization on OpenStreetMap Data using a 3D Laser Scanner

To determine the pose of a vehicle is a fundamental problem in mobile robotics. Most approaches relate the current sensor observations to a map generated with previously acquired data of the same system or by another system with a similar sensor setup. Unfortunately, previously acquired data is not always available. In outdoor settings, GPS is a very useful tool to determine a global estimate of the vehicles pose. Unfortunately, GPS tends to be unreliable in situations in which a clear view to the sky is restricted. Yet, one can make use of publicly available map material as prior information. In this chapter, we describe an approach to localize a robot equipped with a 3D range scanner with respect to a road network created from OpenStreetMap data. To successfully localize a mobile robot, we propose a road classification scheme for 3D range data together with a novel sensor model, which relates the classification results to the road network. Compared to other approaches, our system does not require the robot to actually travel on the road network. We evaluate our approach in extensive experiments on simulated and real data and compare favorably to two state-of-the-art methods on those data.

3.1 Introduction

One essential prerequisite for autonomous navigation for cars or mobile robots is to know the pose of the vehicle in the world. For example, without this information a mobile robot would not be able to plan a path to a desired goal. The most common localization method in outdoor settings is the global positioning system (GPS). While GPS provides a global position, the accuracy of the pose estimate depends on the number and distribution of visible satellites. Especially in cities with high buildings or under tree canopies, GPS can suffer from severe outages. The goal of

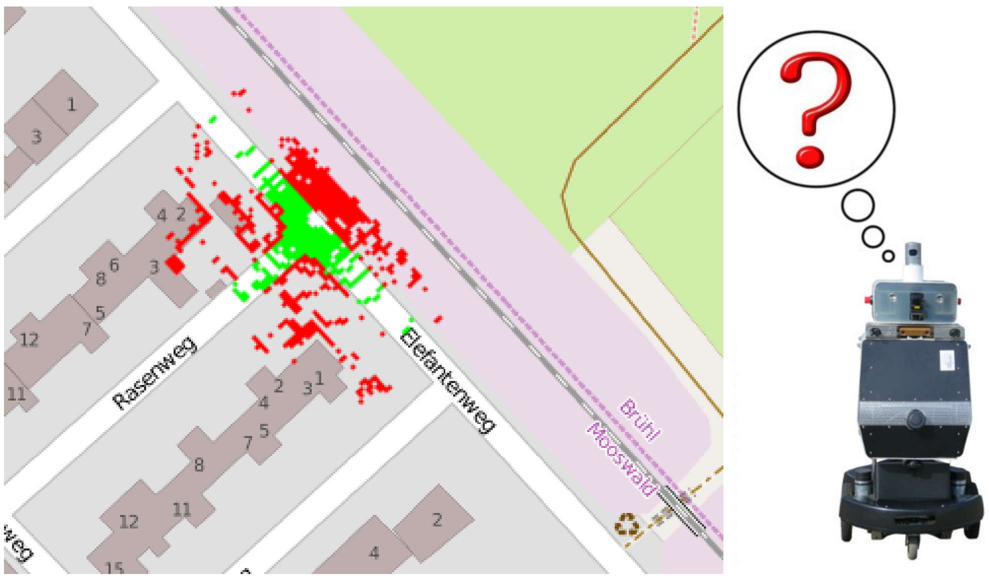


Figure 3.1: Our system localizes a mobile robot equipped with a 3D range scanner in a publicly available OpenStreeMap. The image shows a classified scan projected onto the OpenStreetMap as well as the robot Obelix.

the work presented in this chapter is to enable mobile robots to perform robust navigation even under such circumstances.

To overcome this problem, many autonomous mobile robots or self-driving cars localize themselves within highly accurate maps of the environment, built with range or vision sensors [36, 77]. While this approach yields highly accurate results, it requires a substantial effort to obtain such maps from the sensor data of the mobile robot in advance and to maintain them afterward.

In this chapter, we propose an alternative solution and present an approach to localize a mobile robot given publicly available maps, like OpenStreetMap [57], which provide a dense description of the public road network around the planet. To achieve this, our approach performs a classification of the observations obtained with a 3D laser scanner. In addition, it contains a dedicated sensor model for a probabilistic approach based on a particle filter. Compared to other approaches that perform localization on publicly available maps [9, 19], we do not make the assumption that the robot has to drive on the road network. It only has to be in a reasonable proximity to be able to observe the road network from time to time. Figure 3.1 illustrates the basic principle of our approach.

The rest of this chapter is organized as follows. First we give an overview about related work on the topic of localization on road networks. The next section presents our Monte Carlo Localization on OpenStreetMap data. We discuss our classification of scans into road and non-road cells and the sensor model to weight particles based on the classification result. After shortly discussing our method for path planning, we present our experiments on simulated and data collected with our robot to validate our approach.

3.2 Related Work

Localization in road networks has received quite some attention in the context of autonomous navigation with cars. For example during the DARPA urban challenge all teams had to localize their vehicle on a road network [33, 52, 78]. Most of the systems in this context used a fused localization estimate based on GPS, odometry, and a very accurate inertial measurement unit (IMU). Such methods typically provide locally very accurate motion estimates and can determine the global pose within a few meters but rely on very expensive, specialized hardware to do so.

One of the first methods using a Monte Carlo filter to localize a robot was proposed by Dellaert *et al.* [13]. In their experiments, they used sonar readings or laser scans to localize a robot in an occupancy grid map. Since then, many different approaches employed Monte Carlo Localization (MCL) to localize a robot [30, 61, 87]. Floros *et al.* [19] localize a robot on an OpenStreetMaps road network using visual odometry. They use a history of odometry poses to match against the road network using fast oriented chamfer matching. The authors assume that the shape of this odometry path resembles the shape of the road. While this approach enables fast and robust localization it is restricted to robots driving on roads. Brubaker *et al.* [9] provide a method for graph-based localization on OpenStreetMaps using visual odometry. They represent the robot's pose explicit on the edges of the road graph. The authors provide a probabilistic transition model to move the particles on the graph. Hentschel *et al.* [27] use OpenStreetMap data for localization, to perform path planning and autonomous vehicle control in an urban environment. In contrast to our approach, they mostly use the shape of buildings to localize the robot using 2D laser scans. This approach allows the robot to leave the road but needs an urban environment with known shapes of the buildings. Kümmerle *et al.* [37] present an approach to localize a mobile robot equipped with a 3D range scanner in an aerial image, using Monte Carlo Localization. Compared to our approach, aerial images contain richer information of the environment, e.g., buildings or trees, compared to just a road network.

In this chapter, we propose a system to localize a robot, equipped with a 3D laser scanner, with respect to a road network from OpenStreetMap. We apply a supervised classification approach to classify laser scans into road and non-road. This classification is then used in a corresponding sensor model to weight the particles of a Monte Carlo Localization. In contrast to the above-mentioned methods, which localize a robot on a road network using only odometry, our method does not require that the robot actually travels on the road network.

In the remainder of this section, we give an overview about related work on localization on OpenStreetMap which was presented *after* this work was published. Vysotska *et al.* [81, 82] integrate the building outlines from OpenStreetMap into a graph-based SLAM. This allows the algorithm to reduce the drift in the localization and improves the created maps. The error term which includes the OpenStreetMap data into the SLAM system enables the authors to guide an active exploration to reduce the uncertainty in the map. In a similar fashion Ballardini *et al.* [2] detect building facades in stereo images. By matching these detections against the building outlines from OpenStreetMap the authors can localize the robot with a lane-level accuracy. Landsiedel *et al.* [38] build hybrid maps from local 3D LiDAR scans combined with the data from OpenStreetMap. While the street map provides higher level information, such as street connections, the LiDAR measurements allow to detect local features such as street surface types or road width. Building on top of this work, Landsiedel *et al.* [39] use hybrid maps to localize a robot equipped with a 3D LiDAR based on building outlines using champfer matching. To counter inaccuracies in the OpenStreetMap data, Suger *et al.* [71] propose a method to associate tracks detected using a 3D LiDAR with streets in the street map. The method allows the robot to stay on the tracks while keeping a corresponding pose on the street graph.

3.3 MCL-based Localization on Road Networks

The goal of our system is to find the pose of a robot relative to a given road network, based on a sequence of 3D laser scans, odometry measurements, and a rough initial position of the robot (within a few hundred meters).

In the following, we will describe our Monte Carlo Localization approach, including a novel sensor model, as well as the classifier we use to distinguish road from non-road in the robot's measurements.

3.3.1 Monte Carlo Localization

In this work, we use a particle filter (see Section 2.3.2) to perform Monte Carlo Localization. A particle filter in the context of robot localization represents the probability distribution over the pose x_t of the robot. To maintain a believe about the pose of the system, our particle filter performs two steps. The first step is the prediction step, which modifies the pose hypothesis of each particle using the action u_t, the map m and the previous state x_{t-1} by sampling x_t from our motion model $p(x_t \mid u_t, x_{t-1}, m)$. It describes how the robot moved based on the old pose, the odometry measurement and the map. In the correction step, we relate the measurements to the map according to our sensor model. More precisely, we weight the particles using the measurement z_t according to our sensor model $p(z_t \mid x_t, m)$. Afterward, we resample a new set of particles from the old ones, where the chance of survival for each particle is proportional to its weight in the old particle set.

3.3.2 Our Implementation of the Monte Carlo Filter

One general problem of approximating a probability distribution with a finite set of particles is that good hypotheses might not survive the resampling step. Therefore, we use the number of effective particles to measure the quality of the particle set and resample only when this number is below a given threshold. The number of effective particles [24, 43] is computed as

$$\mathrm{N_{eff}} = 1/\sum_{i=1}^{N}\left(w^i\right)^2. \tag{3.1}$$

Resampling is only performed if $\mathrm{N_{eff}} < N/2$, where N is the number of particles. The $\mathrm{N_{eff}}$ measures the variance in the particle weights. If all particles share a similar weight the $\mathrm{N_{eff}}$ is high and no resampling is needed. For the motion update, we use odometry readings from the wheel encoders or equivalent simulated data. The measurement model uses a classified scan and a road network from OpenStreetMap to weight the particles. This model is described below.

3.3.3 Classification

Since our map encodes only road information we first need to retrieve this information from the 3D range readings. Therefor, we have to decide whether regions in range scans observe road or non-road surfaces. Throughout our experiments, we perform road classification on single scans from a Velodyne HDL-32E LiDAR, but any kind of 3D laser scanner (e.g., tilting/rotating 2D scanners) can be used with the proposed method. The method expects 3D scans that include additional reflectance values. Our classification is defined as a function mapping from discretized cells z_i to the classes *road* or *non-road*

$$c : z_i \mapsto \{\mathrm{road}, \neg\mathrm{road}\}. \tag{3.2}$$

To calculate this classification, we project the scan into a two-dimensional grid and classify the single cells independently. From all points falling into one cell, we calculate the following features:

- mean and standard deviation of the height values
- mean of the squared intensity values
- standard deviation of the intensities
- distance to a fitted plane
- normal vector of the fitted plane
- maximum difference in height values

Based on those local features we learn a classifier using boosting [4] in a supervised fashion. Cells with no or too few points to calculate features are neglected.

To train the classifier, we first collected a dataset (different from the one used in the experiments) with a robot and manually labeled the regions in the scans as being road and non-road. This leads to a classified set of features which we use to train our classifier.

3.3.4 Weighting / Sensor Model

The task of the sensor model is to determine the likelihood $p(z \mid x, m)$ of a measurement z, given the robot is at pose x in the map m. In our approach, the input to our sensor model is the road classification in form of a 2D grid map in which each cell is either unobserved, road or non-road. In the following, we explain how we relate the road classification to the provided road network.

Under the assumption that the grid cells from our sensor measurement are independent, the likelihood of a measurement z, composed of the classified grid cells $z_1, \ldots, z_N$, can be calculated as

$$p(z \mid x, m) \propto \prod_{i=1}^{N} f(z_i, m), \tag{3.3}$$

where $f(z_i, m)$ is the likelihood of one cell in the measurement given the map. We model the likelihood for each observed point as a Gaussian

$$f(z_i, m) = \mathcal{N}(\varepsilon(z_i, m), 0, \sigma_{z_i}), \tag{3.4}$$

with mean 0 and a user defined standard deviation that is individually defined for road and non-road measurements. The term $\varepsilon(z_i, m)$ is an error based on the distance to the next road in the map, which can be efficiently determined using pre-computed distance transform maps.

3.3.4.1 Cost for Road Cells

The calculation of $\varepsilon^r(z_i, m)$ for road cells is supposed to penalize measured road cells that do not correspond to the roads appearing in the map. Such regions lead to a higher distance to the road and thereby increase the error for this match. The error $\varepsilon^r(z_i, m)$ for cells classified as road is calculated for all cells z_i with $c(z_i) = road$ as follows

$$\varepsilon^r(z_i, m) = \min\left(d, |z_i - l|\right), \tag{3.5}$$

where l is the closest location on the road network and d is the maximum allowed distance to a road. The term $\varepsilon^r(z_i, m)$ therefore describes the distance between the observed cell and the closest road in the road network. Cells with a higher distance to the closest road than d will not be penalized further. The usage of the error $\varepsilon^r(z_i, m)$ in the particle filter leads to a situation in which particles are kept in regions where most of the cells that are classified as road are close to a road in the map. The effect of this error can be seen in Figure 3.2a. Removing the dotted side road results in higher errors for all cells, classified as road (green) which are in the blue ellipse. The closest road in the map is now further away, which will increase the error for this match.

On the other hand, particles tend to stick in regions with several nearby-roads or multi-lane crossings, because many roads in the vicinity tend to decreases the error for false classifications and side roads in contrast to single lanes.

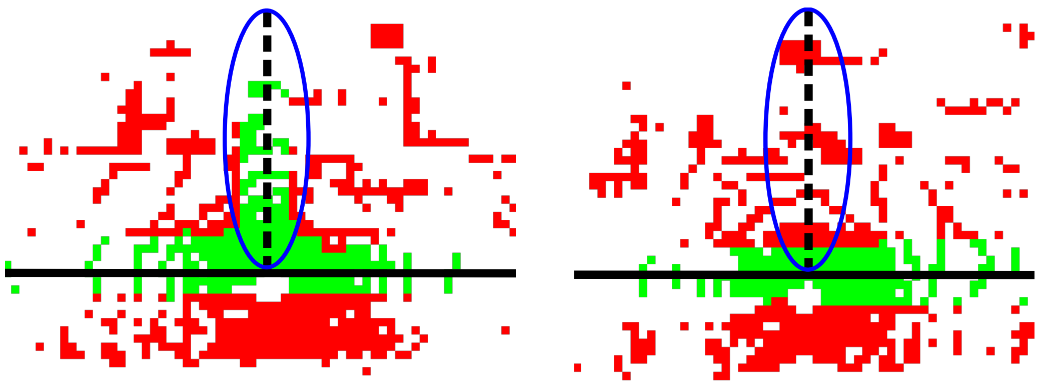

(a) Cost for road cells (b) Cost for non-road cells

Figure 3.2: Classified example scans. Red cells are classified as non-road, green cells are classified as road, and black lines show the road network. Removing the dotted road in Figure (a) increases the errors $\varepsilon^r(z_i, m)$ as the next road in the map for the cells in the blue ellipse is now further away. Adding the dotted road in Figure (b) increases the error $\varepsilon^n(z_i, m)$ as the next road in the map for the cells in the blue ellipse is now closer.

3.3.4.2 Cost for Non-Road Cells

For non-road cells the calculation of $\varepsilon^n(z_i, m)$ is supposed to penalize situations in which there are more roads in the map than can be explained by the scan, such as a side road that does not appear in the scan. Equivalently to the case for road cells we define

$$\varepsilon^n(z_i, m) = d - \min\left(d, |z_i - l|\right). \tag{3.6}$$

This term is small for non-road cells that are far away from the closest road in the road network and large for cells that are close to roads. If we have a scan with only a straight road segment as shown in Figure 3.2b, we increase the error for this match whenever we try to match it against a region of the map including an additional side road (dotted line). This is due to the fact, that non-road cells inside the blue ellipse now have a smaller distance to the next road and thereby a higher value of $\varepsilon^n(z_i, m)$.

3.3.4.3 Computing the Likelihoods

Formula 3.3 can be efficiently computed using log-likelihoods as follows

$$p(z \mid x, m) \propto \exp\left(-\sum_{i,\text{road}} \frac{(\varepsilon_i^r)^2}{\sigma_1^2} - \sum_{i,\neg\text{road}} \frac{(\varepsilon_i^n)^2}{\sigma_2^2}\right), \tag{3.7}$$

whereas $\varepsilon_i^r = \varepsilon^r(z_i, m)$, $\varepsilon_i^n = \varepsilon^n(z_i, m)$, σ_1 being the assumed standard deviation for road cells, and σ_2 being the assumed standard deviation for non-road cells. Larger values of σ_1 or σ_2 lead to more peaked, thereby stricter distributions.

3.4 Planning Paths on the Road Network

The localization relative to the road network allows us to plan paths to a goal using the road-graph representation. We therefor, implemented an A* planner for this task. The first step in our path planning procedure is the projection of the estimate about the robot's pose, reported by the particle filter, onto the road network. For this, we search the closest point on the street segments close to the robot. We also search for the closest point on the road network for the goal position in the same fashion. To find a path on the road network an A* search between the two projected points is executed. In the end, we connect the found path to the position of the robot and the goal. A screenshot of our path planner is shown in Figure 3.3.

A* Path Planning

Given a start and goal position on the road graph, we employ an A* search to find the shortest path. The A* algorithm is a graph search method proposed by Hart *et al.* [26]. The algorithm expands the next node n by choosing the one with the lowest expected cost:

$$f(n) = g(n) + h(n), \tag{3.8}$$

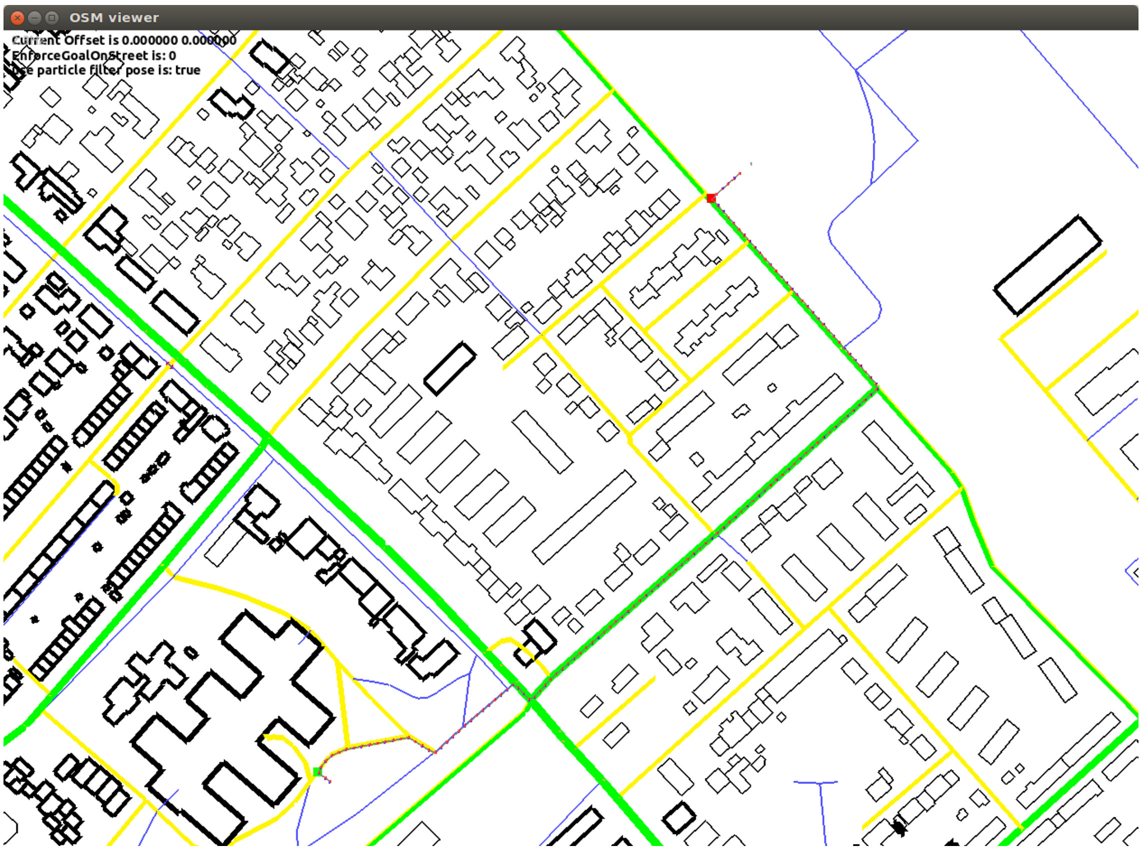
where $g(n)$ is the path length from the start to node n and $h(n)$ is a heuristic estimating the minimal cost from n to the goal. In our implementation, we use the length of the road segments to compute $g(n)$. For $h(n)$ we employ the straight-line distance as heuristic. The resulting search finds the optimal path between two points on the road network.

3.5 Experimental Evaluation

To evaluate our localization method, we performed extensive experiments on three different datasets. In the first experimental setting, we decided to evaluate our method in simulation since this provides us with ground truth, which is hard to obtain for real-world experiments. In a second experimental setting, we evaluated our method on a real-world dataset providing evidence that both the classification scheme and the sensor model are robust to the typical noise introduced by real 3D range scanners. Furthermore, we compared our method against two other state-of-the-art approaches on three different datasets. In the following, we briefly explain these methods.

3.5.1 Distance to Road

The first method we compare against calculates the weight for a particle involving its distance to the next road in the map. More precise, we define the weight as $w = \exp\left(|x - l|/\sigma\right)$, where x is the position of the particle and l is the next point on



```
Current Offset is 0.000000 0.000000
EnforceGoalOnStreet is: 0
Use particle filter pose is: true
```

Figure 3.3: A screenshot of the GUI of our A* path planner on the road network. The GUI shows the planed path in gray from the robot on the top right to a goal on the bottom left. Both positions are not on the road network and are therefore first projected onto it (red and green squares).

a road. As in our approach, σ defines the shape of the weighting distribution. This method favors regions of the map with a lot of roads close to each other. We refer to this method as *Dist2Street* in the graphs.

3.5.2 Chamfer Distance

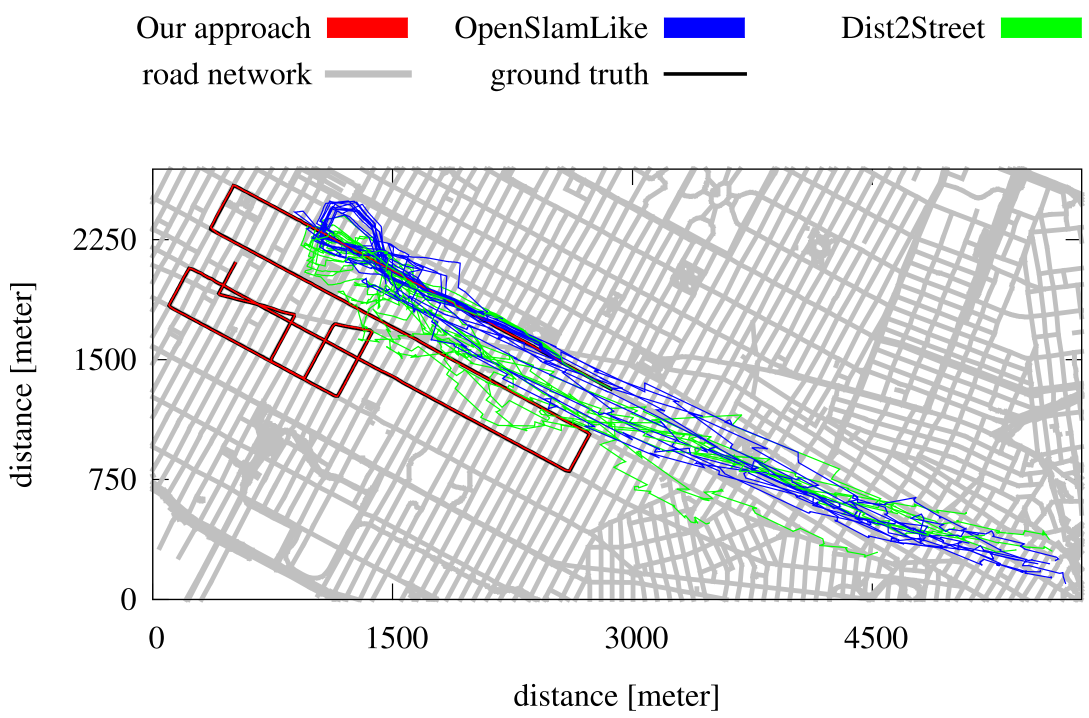
The second method is based on chamfer matching and is comparable to the method by Flores *et al.* [19]. For this method, the particle filter stores the path calculated from the odometry readings over a fixed time or distance window. To weight a particle, we calculate the distance from each point on this path ending at the particle pose to the road map. This is comparable to the chamfer matching score of the path (template image) to the road map (query image) at the position and orientation given by the pose of the particle. This method is named *OpenSlamLike* in the graphs.

3.5.3 Initialization and Parameters

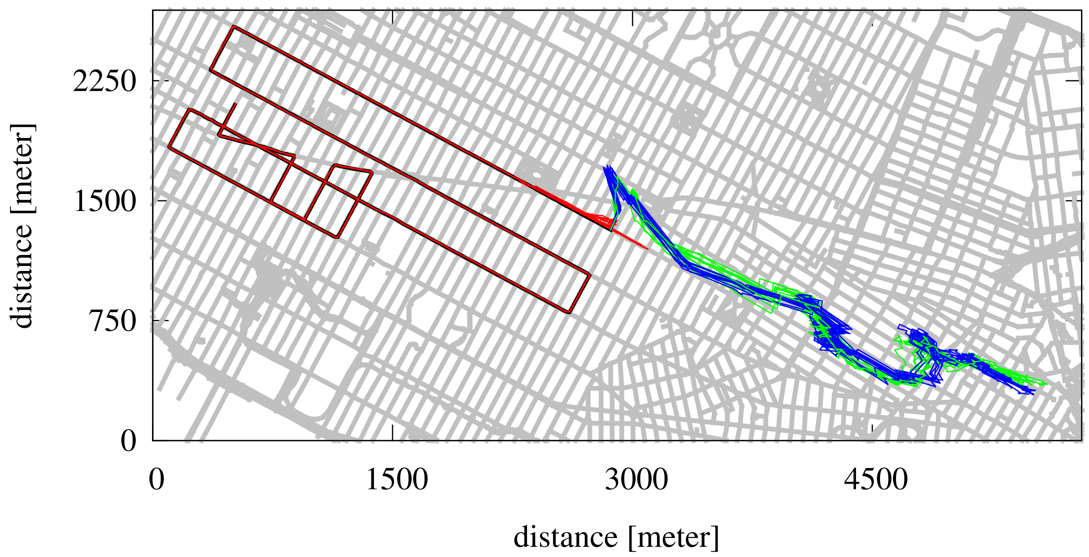
For our tracking experiments, we initialize the particle filter by sampling $2{,}000$ particles from a Gaussian with a standard deviation of $20\,\mathrm{m}$ around the true or GPS position. We then sample the orientation from a Gaussian with a standard deviation of $0.15\,\mathrm{rad}$ around the true orientation or the measurement from the compass of the robot. For the global localization we use $20{,}000$ particles which we distribute evenly in a circle with a radius of $250\,\mathrm{m}$ around the true position and sample the orientation randomly. For each experiment, we execute ten runs per method using the same parameters. To discretize the scans, we use a cell size of $1\,\mathrm{m}$. We choose $1/\sigma = 1/\sigma_1 = 0.0003$ and $1/\sigma_2 = 0.00015$. For the maximum distance to the road we use $d = 10\,\mathrm{m}$.

3.5.4 Simulation Experiments

In the simulation experiments, we want to investigate the performance of the weighting given a perfect classification. We therefor, create scans by copying the roads of an OpenStreetMap in a radius of 15 m around the requested pose. We then generate an odometry path with additional white noise and ground truth poses which we use for evaluation. In these simulation-based experiments, we want to demonstrate that a classified scan can increase the localization performance in contrast to methods using odometry only. This increase is due to the fact that we are able to observe roads that we did not (yet) drive on. These experiments are carried out on the OpenStreetMap for Manhattan. We evaluated how the methods perform on the task of tracking the position of the robot on the map. The paths created by the different methods are shown in Figure 3.4a. The trajectory estimated by our method (red) generates paths that closely resemble the ground truth path. Our method can take advantage of observations of crossings, especially with the diagonal road (running horizontal in the center of this map area), whereas

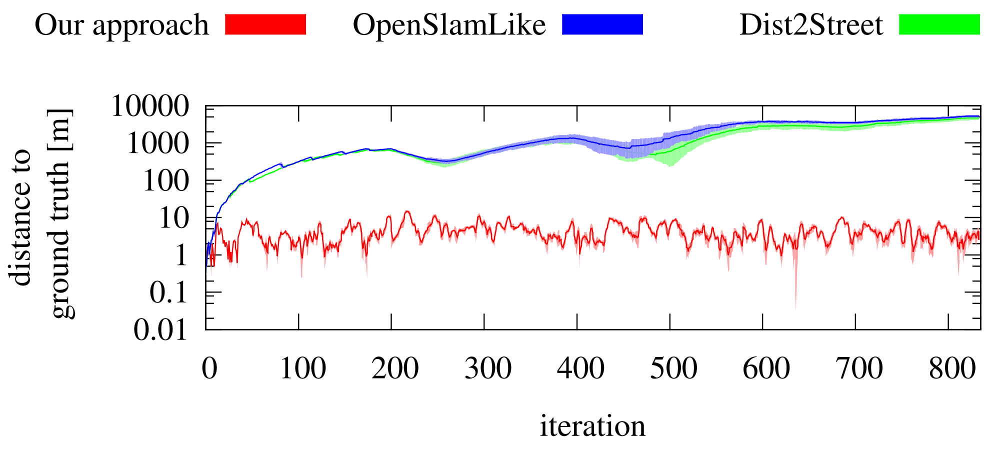


(a) Tracking paths

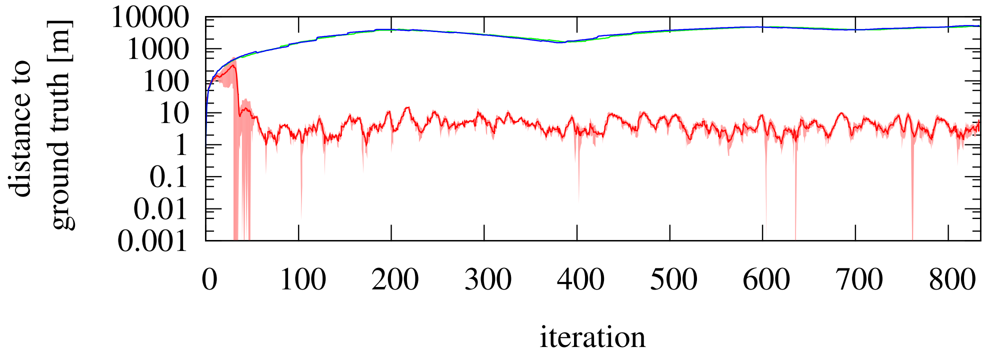


(b) Global localization paths

Figure 3.4: Resulting paths for the first simulated experiments. Ten runs were executed for each of the three methods. Figure (a) shows the paths for the ten runs created at the task of tracking while, Figure (b) shows the same for the global localization task.



(a) Tracking errors



(b) Global localization errors

Figure 3.5: Errors of the first simulated experiments. Ten runs were executed for each of the three methods. Figure (a) shows the errors with their standard deviation over the ten runs created for the tracking task, Figure (b) shows the same for the task of global localization. The errors are the differences between the weighted mean of all particles and the ground truth. The scale of the y-axis is logarithmic.

the other two methods suffer from the long straight paths and the ambiguities. Figure 3.5a shows the distance between the weighted mean of all particles to the true pose over time.

We also performed global localization on the same dataset. Figure 3.4b shows the paths, while Figure 3.5b shows the corresponding errors. In the error plot, we can see that after around 30 iterations our approach is able to find the correct road, whereas the other methods are not able to resolve the ambiguities.

In this chapter, we use a very basic classifier to distinguish between road and non-road in the scans. To see how much our method suffers from classification errors, we randomly flipped a defined amount of cells of the perfect classification and repeated the previous experiment. A plot of the resulting difference of the weighted mean of all particles to the ground truth is shown in Figure 3.6. We can see that our method performs well even if $20\,\%$ of the cells are falsely classified. Since the Particle filter only relies on the classification result and the odometry of the robot, any classification method using arbitrary sensors could be used, even if the classification is sub-optimal.

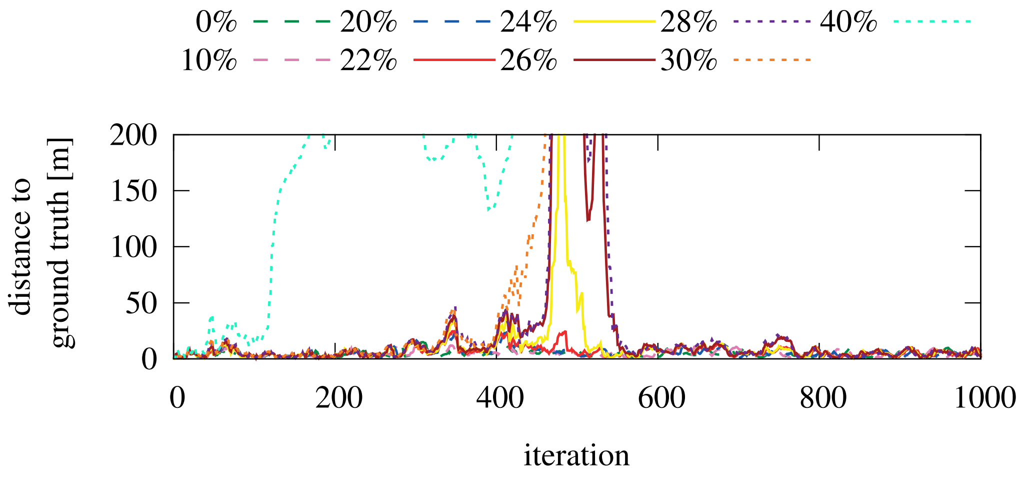
3.5.5 Simulated Offroad Data

Using observations of the environment, we are able to localize the robot even if it does not drive on the road network. In this experiment, we simulated a run on a map of Freiburg. As before, we copy a local surrounding of $15\,\mathrm{m}$ from a road network to simulate perfect classified scans. We compare our method against two other approaches that use only odometry and assume that the robot drives on the road. This assumption is violated during this experiment. On the tracking task our method outperforms the other two methods, as shown in Figure 3.7a. As expected, the other methods are able to follow the shape of the path but are always drawn towards the next roads, which leads to a higher error (see Figure 3.8a). The global localization (see Figures 3.7b and 3.8b) also performs as expected. Our method is able to converge to the correct position after roughly half of the trajectory, whereas the other two methods diverge.

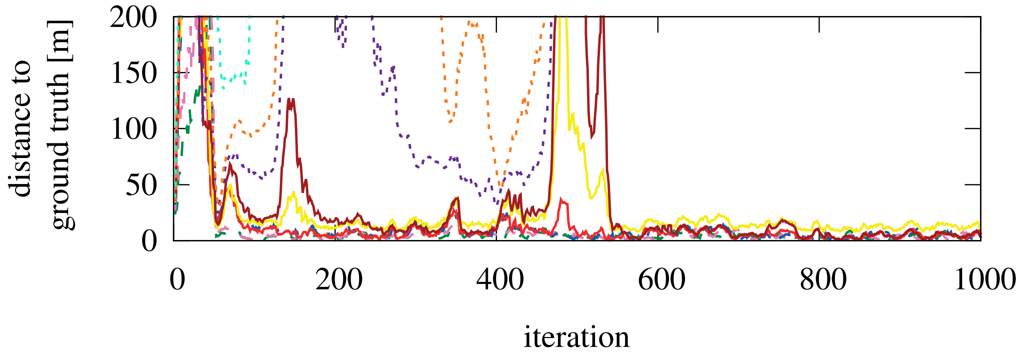
3.5.6 Robot Experiments

Furthermore, we evaluated our approach on data from one of our robots. We use the robot Obelix which is shown in Figure 3.1 and explained in Section 2.8. This robot provides odometry from wheel encoders and 3D scans with intensities from a Velodyne HDL-32E. We collected data in an urban environment. The classification was trained on a separate data set collected with the same robot. Since no ground truth is available for this real-world experiment, we use the result of a graph-based SLAM system that also incorporates GPS measurements instead.

As in the simulated setting we perform two experiments using the same data set. The first experiment evaluates the tracking performance while the second one tests the global localization. Please note that the odometry of the robot is not properly

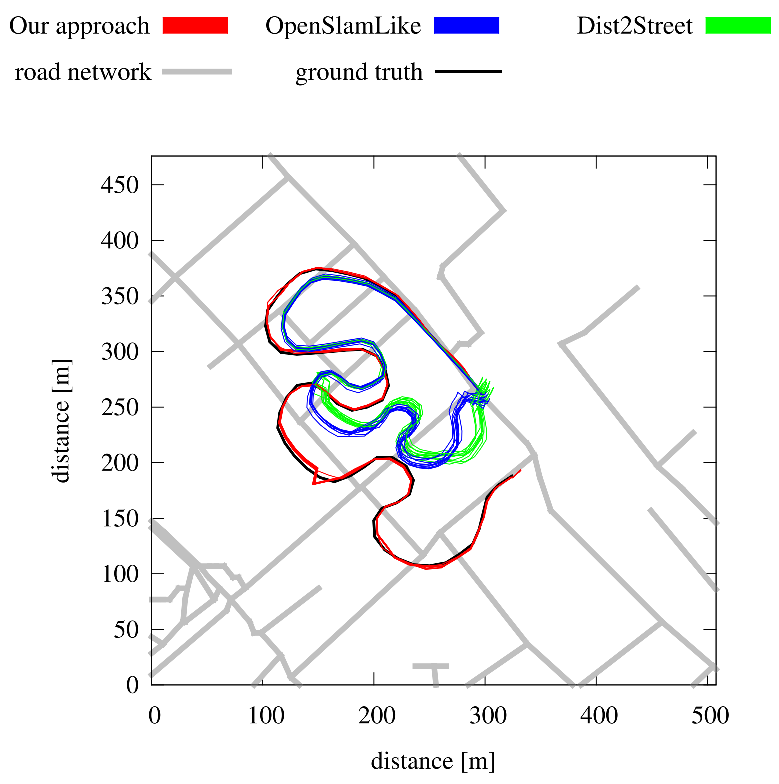


(a) Tracking errors

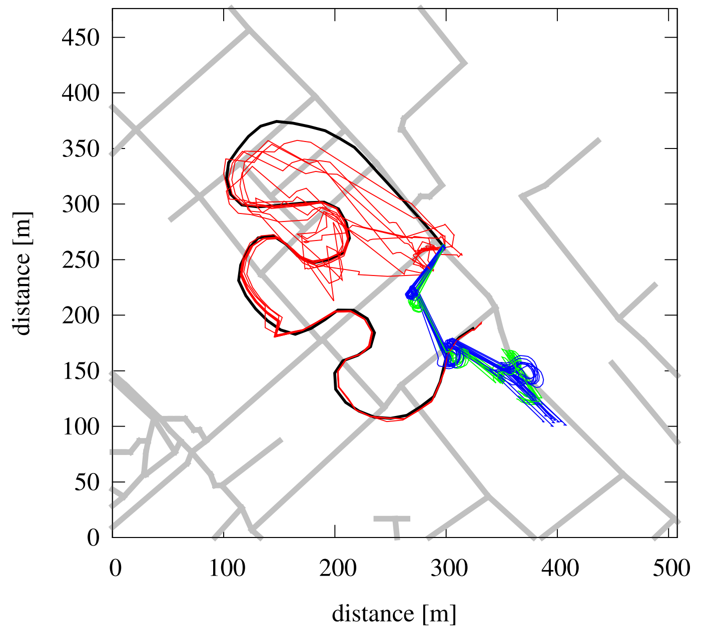


(b) Global localization errors

Figure 3.6: Position error for the weighted mean of all particles for tracking (a) and global localization (b) shown for different rates of the classification error. Our method allows up to 22 % classification error (legend above) without a substantial decrease in the localization accuracy.

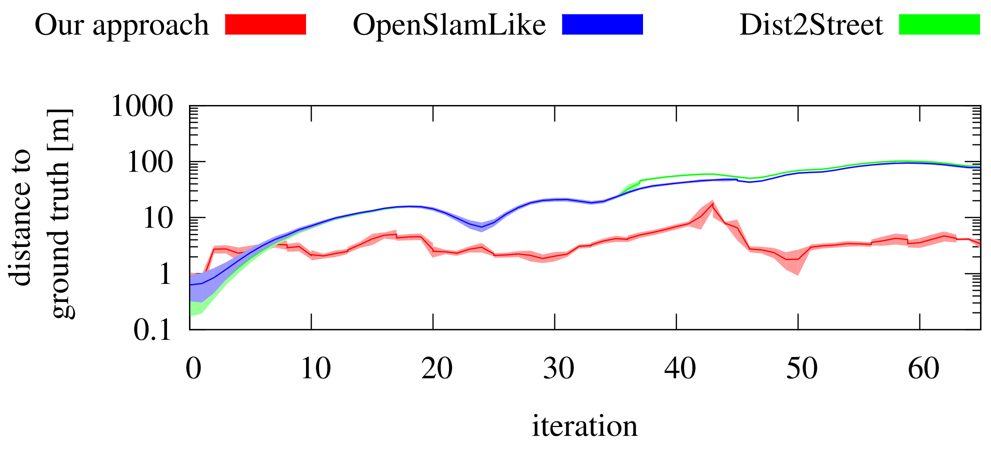


(a) Tracking paths

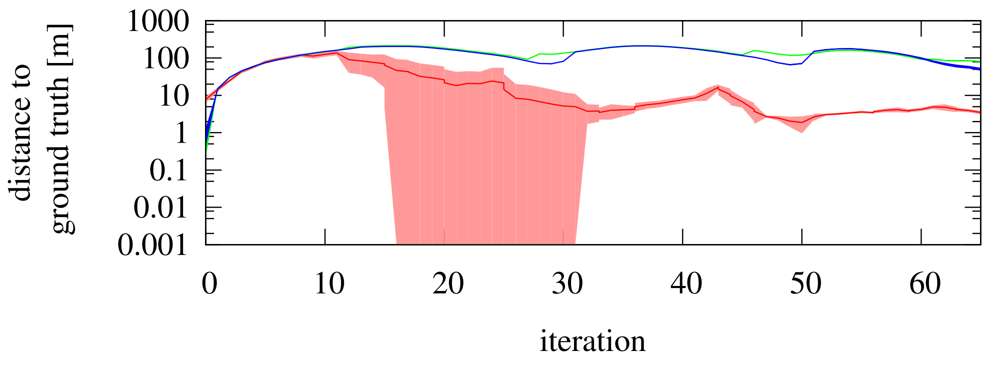


(b) Global localization paths

Figure 3.7: Resulting paths for the simulated offroad experiments. We evaluated each of the three methods using ten runs. Figure (a) shows the paths for the ten runs created for the tracking task while, Figure (b) shows the same for the global localization task.



(a) Tracking errors



(b) Global localization errors

Figure 3.8: Errors of the simulated offroad experiments. We evaluated each of the three methods using ten runs. Figure (a) shows the errors with their standard deviation over the ten runs created for the tracking task, Figure (b) shows the same for the task of global localization. The errors are the differences between the weighted mean of all particles and the ground truth. The scale of the y-axis is logarithmic.

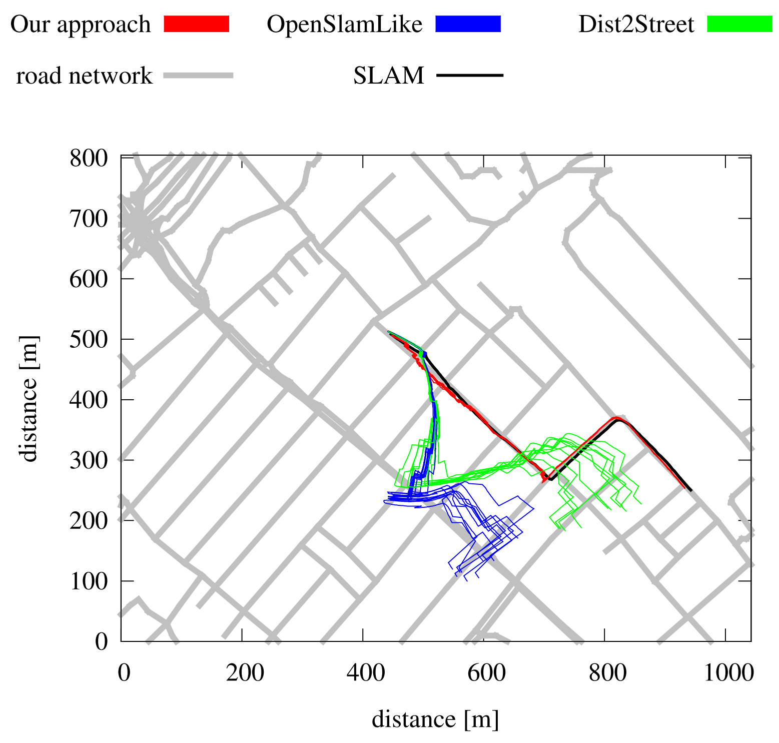
calibrated and suffers from deterministic rotational errors. Figures 3.9a and 3.10a show the result for all three methods for tracking. We see that our approach is able to keep the correct track. The other methods suffer from the rotational errors of the odometry and some of the particles incorrectly turn right early in the trajectory. Our method is able to recover from this situations.

Figures 3.9b and 3.10b show the results for global localization. In this dataset, the robot first drives on a small straight pedestrian path. On this path, the robot is able to perceive two junctions. These junctions allows our method to improve the filter belief with respect to the correct position. After the first turn, the filter is able to converge. The other methods just see the two turns, which is insufficient to localize the robot.

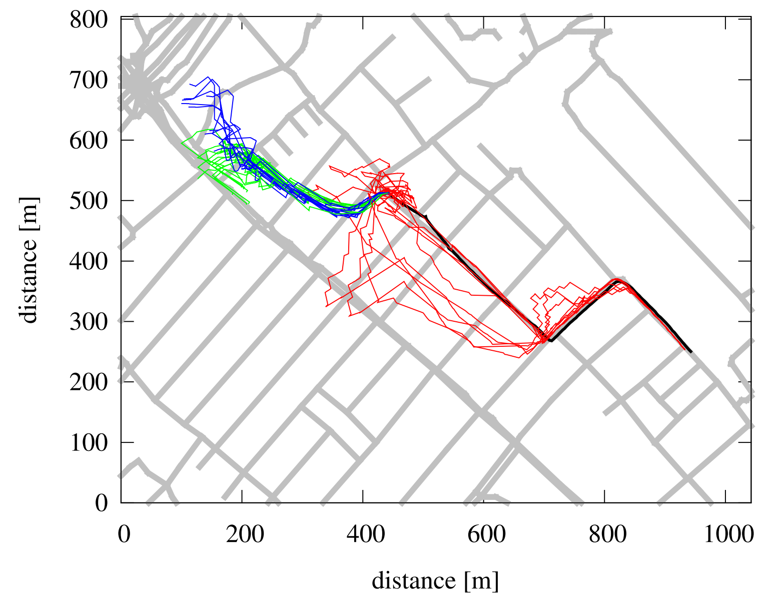
The runtime of our approach was 95 seconds on a standard i7 desktop machine. Note that the robot traveled for about 13 minutes to collect the dataset. Therefore, our system is applicable for online operation.

3.6 Conclusion

In this chapter, we presented an approach to localize a mobile robot using 3D range scans with respect to a road network such as the one provided by OpenStreetMap. In our approach, we employ a classifier to distinguish road from non-road in the scans and use the classified scans as the sensory input for a Monte Carlo Localization approach. In practical experiments, both in simulation and on real-world data, we showed that our method can reliably perform global localization as well as pose tracking even in challenging situations in which other state-of-the-art approaches fail. The presented method has the advantage that it does not require the robot to actually travel on a road.

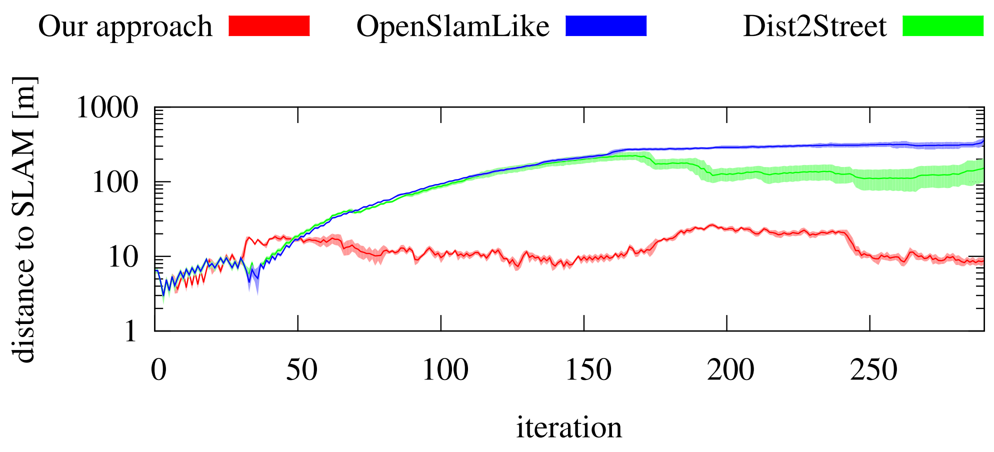


(a) Tracking paths

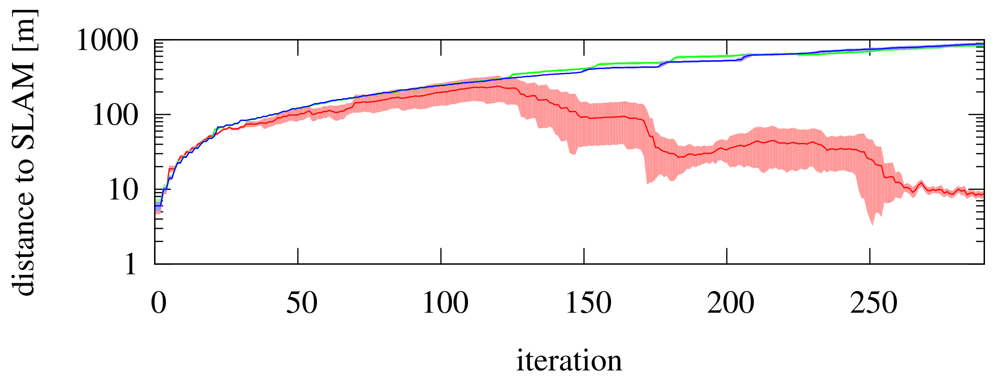


(b) Global localization paths

Figure 3.9: Resulting paths for the real robot experiments. We evaluated each of the three methods using ten runs. Figure (a) shows paths of ten runs created for the tracking task while, Figure (b) shows the same for the global localization task.



(a) Tracking errors



(b) Global localization errors

Figure 3.10: Errors of the real robot experiments. We evaluated each of the three methods using ten runs. Figure (a) shows errors with their standard deviation over ten runs created for the tracking task, Figure (b) shows the same for the task of global localization. The errors are the differences between the weighted mean of all particles and our SLAM result including GPS measurements. The scale of the y-axis is logarithmic.

CHAPTER 4

Feature-based Approach for the Detection of Dynamic Objects in 3D Range Data

Information about dynamic objects in sensor data is highly relevant for many perception and state estimation problems in the area of mobile robotics. Particularly when robots are employed in dynamic environments, accurate knowledge about which components of the robot's perception belong to dynamic and static aspects of the environment can greatly improve navigation functions. For example, localization and scan matching performance can be improved if only static parts are matched against each other. Furthermore, during path planning, dynamic objects should be treated differently than static ones. In this chapter, we estimate as to whether surfaces belong to dynamic objects in 3D scans. Our method is based on local surface features and is learned from real 3D laser data, either using manually labeled scans, an automatic classifier based on scan differencing or publicly available datasets such as KITTI. The advantage of the proposed method is that it can detect dynamic obstacles in single scans, even when they are currently not moving, like parked cars or standing pedestrians. There is no implicit need to model specific object types. We also show that our approach can be used in combination with typical differencing methods to improve recognition results.

4.1 Introduction

Besides localization one fundamental prerequisite for autonomous navigation in dynamic real-world environments is to model the changes that can occur. This is especially important to plan smooth and collision free trajectories for a mobile robot in the presence of high numbers of moving objects, like one expects in densely populated city centers or shopping malls. Since the appearance of dynamics in an environment is typically unknown beforehand, it has to be estimated online from the

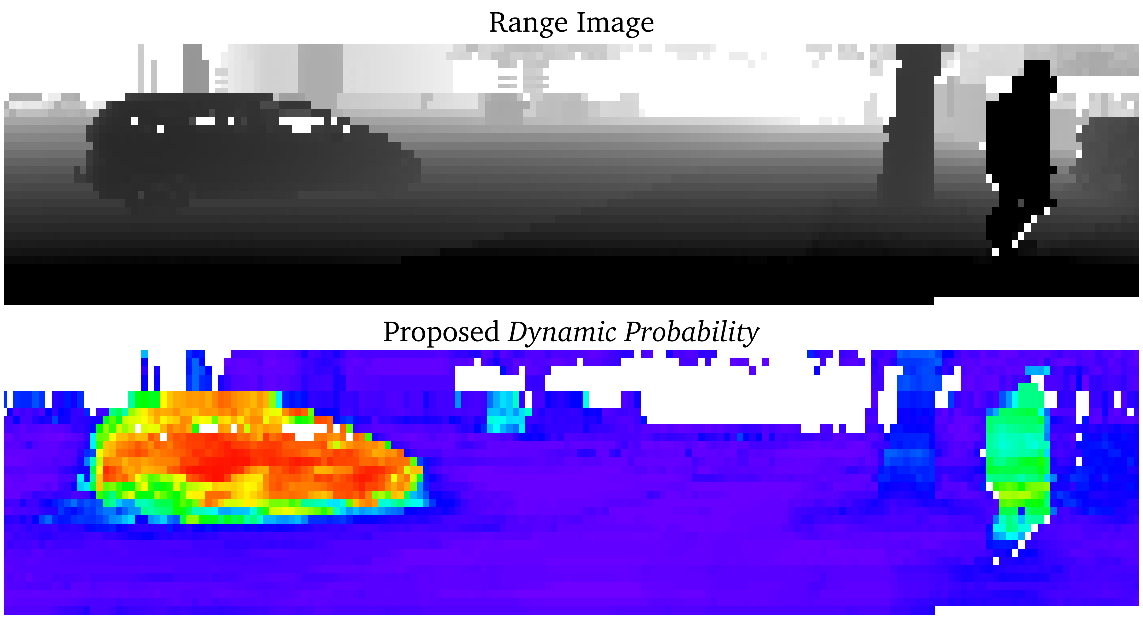


Figure 4.1: The top image shows part of a range image representation of a scan including a car (left) and a pedestrian (right). The range is encoded from black (close) to white (far). The bottom image shows the estimated *dynamic probability*, where red represents high *dynamic probabilities*.

sensor data perceived by the robot. The majority of related work using depth sensors phrases the problem of estimating dynamic objects as a segmentation task [53, 74, 91]. Vision-based approaches on the other hand tend to use feature-based recognition [3, 29, 85] and often rely on detecting object classes trained for different scales. In contrast, most laser-based methods first segment the highly accurate geometry for classifying dynamics. Object segmentation is either performed by removing the ground plane and clustering the remaining points or by computing the difference between two range scans in a small time window. Both methods might fail in cases in which dynamic objects are very close to each other or are not moving, e.g., people holding hands or standing still. Since they might start to move at any time, the autonomous robot should be aware of them being dynamic.

In this chapter, we present a novel probabilistic approach to estimate which surfaces belong to dynamic objects in 3D scans. This prediction, which is based on local surface features, provides a probability of being caused by a dynamic object for every point. It thus facilitates solutions to various problems. It can guide detection algorithms or make scan matching procedures more robust by weighting dynamic objects differently. Furthermore, in the context of life-long mapping, it allows the robot to update the map according to the probability that measurements are caused by dynamic objects. See Figure 4.1 for an example of a scan with the calculated probability. In the remainder of this thesis we refer to the probability that measurements were caused by dynamic objects by *dynamic probability*.

We learn the probabilities that surfaces belong to dynamic objects using a classifier based on a set of local features in a supervised fashion. Given sets of surface features and ground truth labels for each point in the training dataset, we train a random decision forest (see Section 2.5.2) to predict the *dynamic probability* for a given point. The main contribution of this work is that our method computes a point-based probability without requiring a segmentation or matching a template with a sliding window. This makes our method highly robust in cases in which segmentation becomes difficult or only small parts of objects are visible, for example due to occlusions.

This chapter is structured as follows. First we give an overview about related work on feature-based detection of dynamic objects in camera images or laser scans. After this we describe our method using local features for the detection of dynamic objects. We introduce our features as well as the learning methods used in the experiments. After this, we present the experimental evaluation including the datasets and a comparison of different learning methods to predict the probability that surfaces belong to dynamic objects.

4.2 Related Work

The development of self-driving cars led to an increasing interest in the detection and tracking of pedestrians and cars with 3D laser scanners. For example, Teichman *et al.* [74] classify cars, cyclists, pedestrians and background from 3D range scans with additional intensity information. They first segment the scan into objects, which they then track over time. Their method uses features for the segments which include velocity, acceleration, bounding-box, spin images, and Histogram of Oriented Gradients (HOG) descriptors. In a similar context Zhao *et al.* [91] first build a map from laser scans and segment the fused model into individual objects. They classify the segmented objects into object classes based on features that capture the geometrical appearance and smoothness. The map building process reduces the noise in the data but prevents a scan-wise classification. Moosmann *et al.* [53] propose a joined tracking and localization approach that segments a 3D range scan and uses the parts of the scan labeled as background to compute a displacement between consecutive scans to track the vehicle. This demonstrates that prior information about dynamic obstacles improves the pose estimates, especially in highly crowded scenes. Wang *et al.* [84] propose a method to segment a 3D laser scan into parts that could move and parts that cannot. They use a graph-based clustering approach to divide a scan into the classes car, pedestrian, bicyclists or background. In contrast to our approach, these methods highly depend on the quality of the segmentation and suffer from poor performance in cases with false segmentations, e.g., false merges of close-by objects.

In the context of social interaction of pedestrians Spinello *et al.* [67] propose to classify segmented object candidates based on boosted spatial features and track valid objects over time. In our work, we use a different set of features, do not require pre-segmentation and apply random decision forests instead of boosting.

We provide a comparison of boosting and random decision forests using our features in the experimental section (see Section 4.4.6).

Vision-based approaches often rely on a sliding window that matches an object template with the image. Since the scale of the objects in the scene is unknown, sliding windows of different scales are applied. Benenson *et al.* [3] present an approach that uses a variant of the HOG descriptor for template matching. In a similar context, Jafari *et al.* [29] present a combination of a sliding-window-based upper body detection in range images for close range detections fused with a HOG-based template matching approach for detecting pedestrians at further range. In contrast to our method, the presented method focuses on pedestrians and one would need to apply a separate detector for every additional object class. Furthermore, our method needs no sliding window and we do not match a specific object template. Instead, we compute 3D surface features and apply a random decision forest classifier.

González *et al.* [23] fuse data from a laser scanner with camera images to get a dense depth map with color information. Similar to the approach presented here, the authors train a random decision forest. However, in contrast to our work, they train each decision tree individually to predict the presence of a single object part. They use multiple views of pedestrians to train a random decision forest for classification. This method is only suitable in the presence of high resolution texture. Xu *et al.* [89] use a joint 2D/3D scheme to detect objects like cars or pedestrians in images and assign a segment from a 3D laser scan to the detection. An additional validation step on the 3D data reduces the number of false positives. Again, this requires a good segmentation of the 3D scene and heavily relies on the robustness of the detections in the 2D imagery.

Compared to existing work described in the literature our method has several advantages. First, it does not rely on a segmentation approach. Second, we estimate a probability for each individual point, which reduces the probability of missing an entire dynamic object because of occlusions or noise in the data. Third, our method can also identify dynamic objects that are not moving.

4.3 Feature-based Detection of Dynamic Objects

The goal of our method is to estimate the probability of being caused by a dynamic object for every point in a range image. To achieve this, we first compute a set of features for every point and then apply a learned random decision forest on every feature descriptor. The random decision forest provides a voting score between 0 and 1 which we interpret as the probability that the corresponding point was caused by a dynamic object or short the *dynamic probability*. In the remainder of this section, we will explain the feature extraction step in detail and describe how we train the decision forest for dynamic objects.

4.3.1 Feature Extraction

To learn a classifier from a labeled dataset, we compute local features for each point $P = \{x_P, I_P\}$ in the scans, where x_P is the 3D position and I_P its measured intensity. The features are based on the distribution of the neighboring points $n_j = \{x_j, I_j\}$. To get fast access to these neighbors we can either use a range image computed from the 3D scan or in case of organized scans we can omit the calculation of range images and just rely on the scan structure. First, we collect all neighboring points $N = \{n_1, \ldots, n_m\}$ within a defined 3D distance. For faster computation of the features we subsample the input cloud by only taking every i-th pixel of the range image, whereas we choose i according to the maximum allowed neighborhood size in the range image. We then compute the covariance matrix:

$$C = \frac{1}{m-1} \sum_{j=1}^{m} (x_P - x_j)(x_P - x_j)^\top \tag{4.1}$$

of all these neighboring points, from which we compute the eigenvalue decomposition $C\mathbf{v} = \lambda\mathbf{v}$. The feature vector $F = \{f_1, \ldots, f_{12}\}$ is built up as follows. We first use the eigenvalues normalized such that they sum up to one:

$$f_1 = \lambda_1 / c, \tag{4.2}$$

$$f_2 = \lambda_2 / c, \tag{4.3}$$

$$f_3 = \lambda_3 / c, \tag{4.4}$$

where $c = \lambda_1 + \lambda_2 + \lambda_3$. The eigenvalues describe the extend of the point neighborhood in the directions of the eigenvectors. These values describe the relative surface curvature, independent of the scale. The material of an object provides an additional cue whether a surface is dynamic or not. Therefor, we use the mean and variance of the intensity values averaged over the point neighborhood as feature values:

$$f_4 = \frac{1}{m} \sum_{i=1}^{m} I_i, \tag{4.5}$$

$$f_5 = \frac{1}{m-1} \sum_{i=1}^{m} (I_i - f_4)^2. \tag{4.6}$$

The mean distance of all neighboring points to the point P and the variance of these distances describe how the points are distributed within the point neighborhood:

$$f_6 = \frac{1}{m} \sum_{j=1}^{m} \|x_P - x_j\|, \tag{4.7}$$

$$f_7 = \frac{1}{m-1} \sum_{j=1}^{m} (\|x_P - x_j\| - f_6)^2. \tag{4.8}$$

We also use the number of points outside the neighborhood radius, but inside the search window in the range image as a feature (f_8). This value is low for planar surfaces and high for small objects located in open space. Furthermore, it provides a measure for how cluttered a neighborhood is. To allow the learning method to compensate for potential range dependencies, we use the range measurement of point P as an additional feature value:

$$f_9 = \|x_P\| . \tag{4.9}$$

The eigenvector v_1 corresponding to the smallest eigenvalue describes how the point neighborhood is oriented in the world. We therefore use the components of v_1 as feature values: $(f_{10}, f_{11}, f_{12})^\intercal$.

The dynamic objects we usually encounter, such as pedestrians, cars or bikes have very different dimensions. As we will demonstrate in Section 4.4.6, using a single scale for the feature extraction to cover these different types of dynamic objects is challenging. To improve robustness, we calculate the feature vector F hierarchically for different neighborhood sizes r_0, r_1, r_2. The hierarchical feature vector F' is then built as follows:

$$F' = \{f^{r_0}_{1-9}, f^{r_1}_{1-9}, f^{r_2}_{1-9}, f^{r_1}_{10-12}\}. \tag{4.10}$$

In addition to these features, which are calculated on the point neighborhood within a Euclidean distance, we also use the neighboring pixels in the range image $d_j = \{x_j, I_j\}$ to calculate a second part of our feature vector. We first compute the eigenvalue decomposition of the covariance C' of the points d_j as above: $C'\mathbf{v}' = \lambda'\mathbf{v}'$. From this we derive another feature vector as follows. We use the z-value of the smallest eigenvector v'_1 as g_1. This value describes how accurately the points can be described as a horizontal plane. The range normalized eigenvalues can be interpreted as the shape of the neighborhood independent of its size. We use

$$g_2 = \lambda'_1 / c', \tag{4.11}$$
$$g_3 = \lambda'_2 / c', \tag{4.12}$$
$$g_4 = \lambda'_3 / c', \tag{4.13}$$

as feature values, where $c' = \|x_P\| \cdot (\lambda'_1 + \lambda'_2 + \lambda'_3)$. The range dependent normalization is necessary as the point density decreases with the distance from the origin of the scanner.

By combining the point neighborhood features with the features computed on the pixel neighborhood we have a final feature vector with an absolute size of 34 elements.

4.3.2 Learning Method

After we computed the features for all points we use them together with the provided labels to train a random decision forest to estimate the *dynamic probability* for each point. For comparison, we additionally train an SVM to estimate the probability and a boosting method to predict the classification into dynamic and static. We used the implementations available in OpenCV [6]. A comparison of these methods is shown at the end of the experimental section (see Section 4.4.6). As the random decision forest performs best in runtime, generalization and reproduction of our training data we use this learning method throughout the rest of this work.

To predict the *dynamic probability* for a given feature we average the individual classifications of the different decision trees in the random decision forest to get a real-valued prediction which we interpret as the probability of the point being caused by a dynamic object.

4.4 Experimental Evaluation

In this section, we present experiments to evaluate the quality of the proposed method. We performed our experiments on a set of different real-world datasets collected with one of our robots, equipped with a Velodyne HDL-32E LiDAR (see Section 2.6) as well as on the KITTI dataset [22]. Our datasets include different dynamic objects such as cars, pedestrians and cyclists while the KITTI dataset additionally includes trucks, trams, trailers, etc.

As baseline for our comparison, we use a dynamics detection method that performs differencing between range scans over multiple time windows. In this way, dynamic objects that are moving are detected but movable objects like standing pedestrians or non moving cars are treated as static. We will refer to this approach as difference-based segmentation and denote it as *diffSeq* in our plots.

To learn a random decision forest based on the labels and the features extracted per point we choose a subset of the labeled ground truth scans as the training set. To test our method, we classify a point as dynamic if its probability of representing a dynamic object is higher than a threshold. We validate this classification against the ground truth for varying thresholds and provide the resulting precision-recall curves. Additionally, we show how the *dynamic probability* can be used to improve the result of the difference-based segmentation method. For this we compute the mean of both methods as a joint estimate.

4.4.1 Training Dataset

To capture a training dataset, we placed our robot in two different locations on a sidewalk close to a crosswalk and a bus stop and collected data for about 20 minutes. We encountered pedestrians, cyclists, cars and a bus. For both locations we selected a background scan without any dynamic objects and labeled the dataset by differencing the individual scans against this reference scan. To avoid a bias in the training data regarding observing specific objects only at a certain range, we extended our training dataset with additional data recorded while traversing the same urban environment close to our campus. To generate ground truth, we first applied the difference-based segmentation method and afterward manually corrected the labeling. We restrict the maximum range of the used measurements

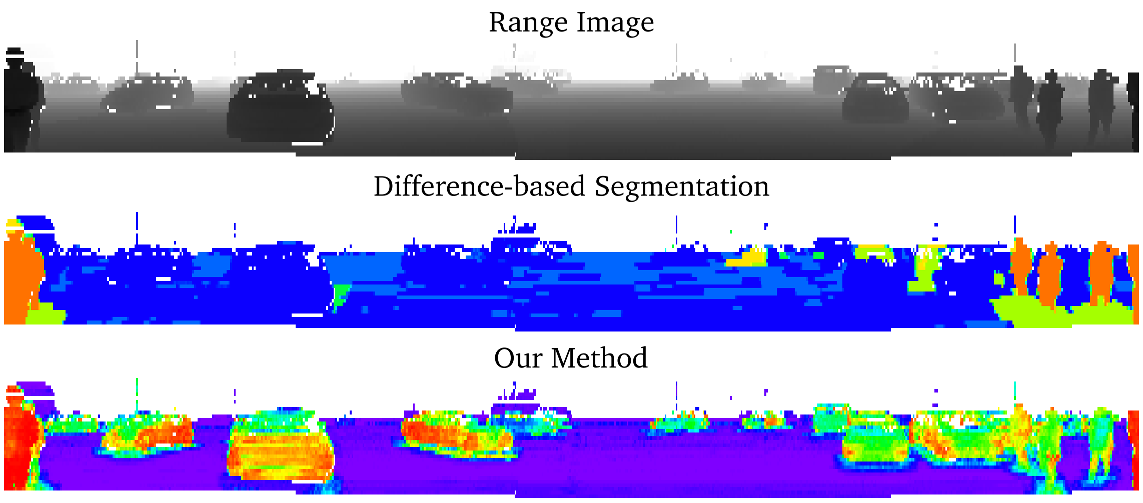


Figure 4.2: The top image shows a range image of a parking lot scene with cars and pedestrians. Black represents small, while white represents large ranges. The center image shows the result of the difference-based segmentation method. The bottom image shows the *dynamic probability* estimated with our method. Obviously, the difference-based segmentation method cannot infer that the parked cars are dynamic, while our method is able to classify them correctly.

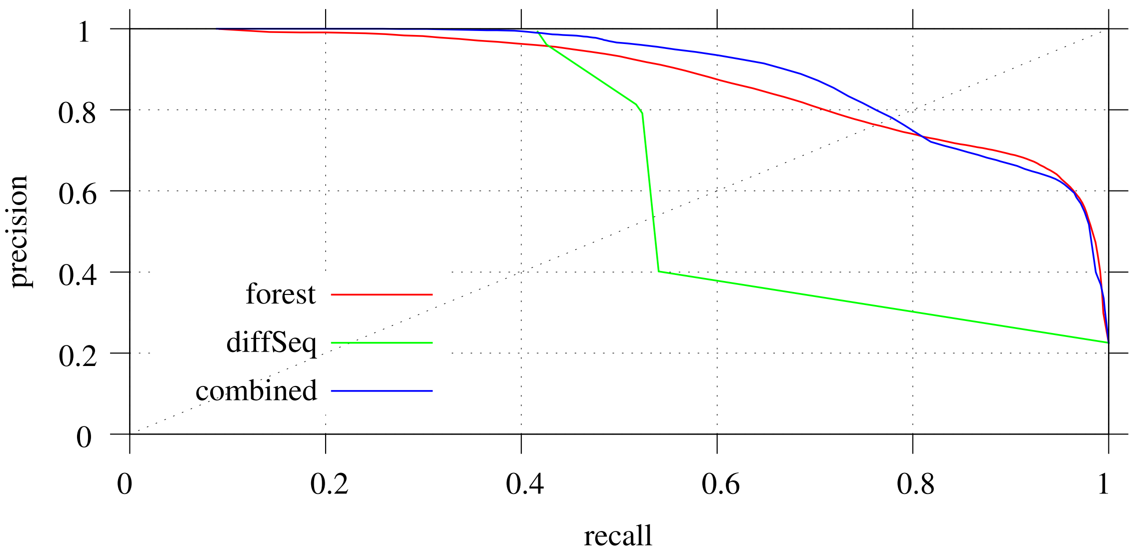
to $30\,\mathrm{m}$ because of the rapidly decreasing point density. In our training dataset which consists of 130 scans, we labeled $4{,}455{,}574$ points as static and $638{,}634$ as dynamic. We trained a random decision forest on this dataset and used it during our experiments.

For the hierarchical feature computation we choose $r_0 = 0.2\,\mathrm{m}$, $r_1 = 0.35\,\mathrm{m}$ and $r_2 = 0.5\,\mathrm{m}$ as radii through all our experiments.

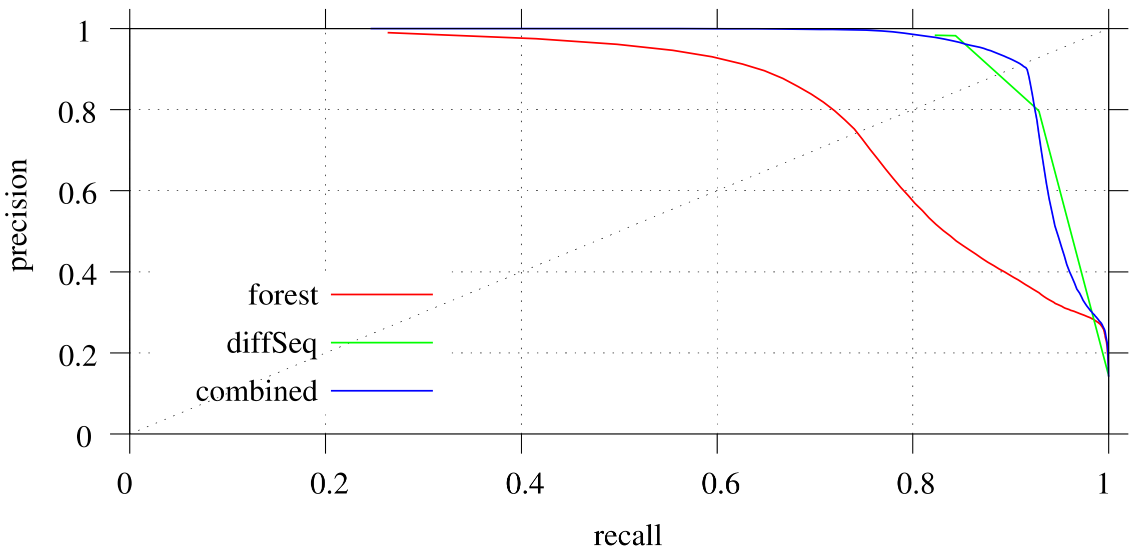
4.4.2 University Parking Lot

To test the learned random forest, we recorded a validation dataset on our university parking lot by steering the robot over the entire space. We encountered a large number of parked cars as well as some cyclists and pedestrians. For this dataset we manually annotated the scans.

Figure 4.2 visualizes the estimated probability in an exemplary scan. As we expect parked cars to be dynamic, the difference-based segmentation method is not able to produce reasonable results. Our method, on the other hand, estimates high probability of belonging to a dynamic object to the measurements on the parked cars. Figure 4.3a shows the precision-recall curves of this experiment. At the equal error rate point our method achieves more than 75%. By combining our estimated probabilities with the result of the difference-based segmentation method, which has an equal error rate of around 53%, we are able to further improve the estimate to more than 78%.



(a) Parking lot dataset



(b) Freiburg city center dataset

Figure 4.3: Precision-recall curves for two different datasets. Figure (a) shows that for the parking lot data our method outperforms the difference-based segmentation baseline, since parked cars are dynamic objects according to our definition. Figure (b) demonstrates that at the same time our method achieves a comparable result in a very dynamic pedestrian zone environment.

4.4.3 City Center of Freiburg

To highlight the fact that our method does not require any pre-segmentation, we recorded a dataset in the pedestrian area in the city center of Freiburg. In this challenging dataset, we captured dense crowds of pedestrians and a few cyclists. Due to the high amount of occlusions, we had to label these scans manually. While we trained our dynamic detection on data that mostly shows gardens and smaller houses, this dataset includes larger buildings with arcades. The corresponding pillars can easily be confused with pedestrians. Figure 4.3b shows the precision-recall curve of the estimated probabilities on this dataset.

4.4.4 KITTI

Furthermore, we evaluated our approach on the publicly available KITTI object dataset created by Geiger *et al.* [22]. This dataset contains camera images with labeled object bounding-boxes and 3D LiDAR scans. To apply this data to our framework, we projected the laser scans into the camera frame and transferred the labels that fell into a bounding-box to the 3D points. Unlabeled points were treated as static. Unfortunately, the provided ground truth labels are limited to the field-of-view of the camera. Therefore, we only use the parts of the 3D scans that overlap with the camera. Due to slightly different perspectives of the LiDAR and the camera (even after transforming the scans into the camera coordinate system), we encountered frequent false negatives, especially close to boundaries of the label boxes. For some objects the ground truth labels are even missing (see Figure 4.4 for some examples).

To reduce the impact of the projective inaccuracies of the bounding-boxes, we increased the box sizes automatically in both horizontal directions (not vertically) for the entire dataset. To evaluate the impact of this parameter, we varied it between $0\,\mathrm{m}$ and $2\,\mathrm{m}$ for both training and testing phase of our method. The resulting precision-recall curves are shown in Figure 4.5. For a small increase in bounding-box size the results improve (see values $0.1\,\mathrm{m}$ to $0.4\,\mathrm{m}$). Once the bounding-boxes get to large, the amount of false positives increases and the performance drops. Based on these results we chose a value of $0.4\,\mathrm{m}$ for our experiments. An exemplary result of our method is shown in Figure 4.6. Note that we do not use camera images and the ground truth labeling is only provided for the area covered by the camera view.

4.4.5 Model Generalization

One interesting question is, how well our learned random decision forest can be transferred to another dataset. This provides an insight on how well the learned dynamic detection handles unknown or novel data and also how a training set should be designed. Our datasets were recorded using a Velodyne HDL-32E LiDAR with a robot traveling in urban environments and on the sidewalk. The KITTI

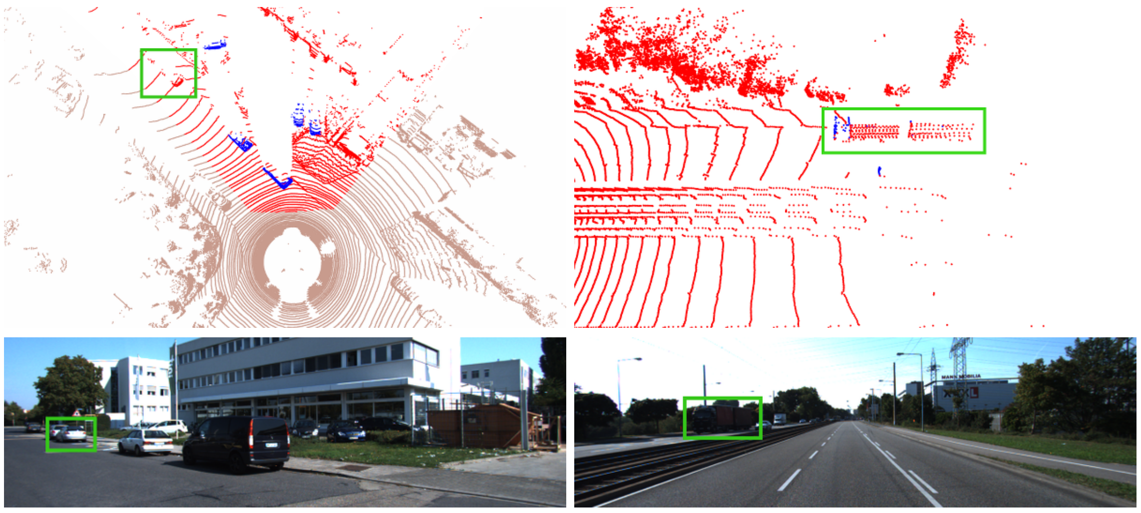


Figure 4.4: Two scans (top) and the corresponding color images (bottom) with incorrect labels. The blue dots in the scan are the labeled dynamic points. The marked car in the left is completely missing, while for the truck on the right only the points of the front side have been labeled. Brown dots correspond to points not present in the camera image.

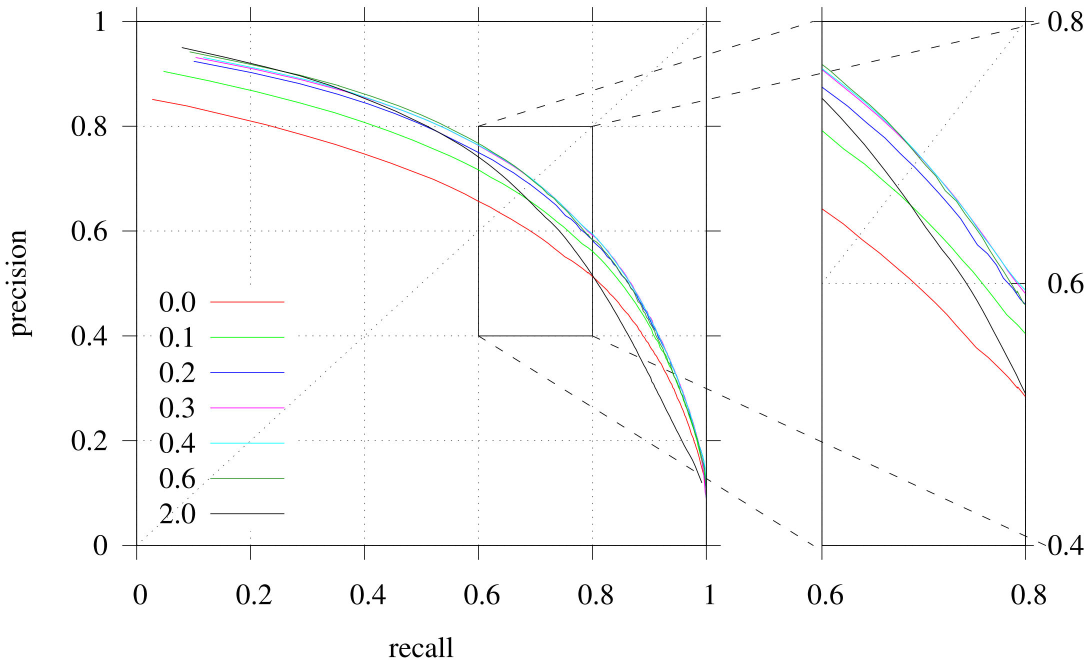


Figure 4.5: Precision-recall curves for the spatial label expanding on the KITTI dataset. We trained and tested our method with different increased region sizes. The legend shows the size of the increase in meters. On the left side a magnification of the equal error rate region is shown.

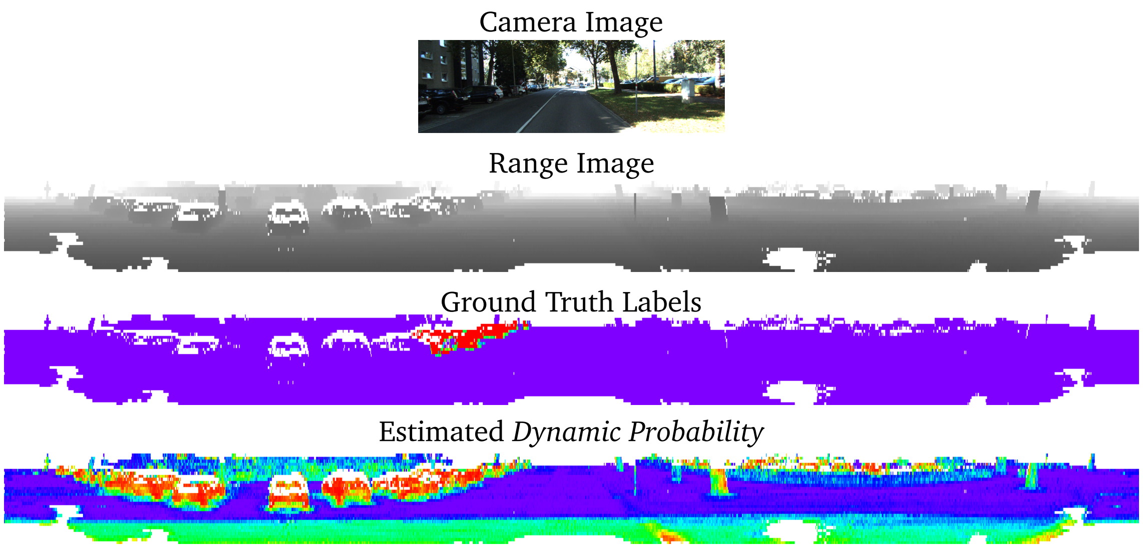


Figure 4.6: This figure shows a scene from the KITTI dataset. The top image shows the camera image, the second shows the range image where ranges are encoded from close to far by black to white respectively. The third image shows the provided labels, which are only available within the field of view of the camera. The bottom row shows the estimated probability. You can see that even the cars on the right behind the hedge are detected by the learned method.

dataset however was recorded with a Velodyne HDL-64E LiDAR mounted on a car, driving on streets. Besides the obvious difference in sensor resolution, a vehicle driving on a road observes pedestrians typically at a higher range and more cars in close proximity.

For this experiment we added the scans from the test data used in the first two experiments to our training set and learned a random decision forest. First we applied the learned model to the KITTI test set used in the previous experiment with a growth of the labels by $0.4\,\mathrm{m}$. The result is shown in Figure 4.7 (green). We then trained a model on the KITTI dataset and applied this model to our city center test set. The result is shown in Figure 4.7 (pink). We note that the model learned on the KITTI dataset applied to our data results in an equal error rate of around 75%. By applying our augmented model to the KITTI dataset we receive an equal error rate of around 38%. The reduced performance for this setting is to be expected, since the KITTI dataset is much larger and therefore better suited to learn a more general classifier. Our data only includes cars, pedestrians and bicyclists, while the KITTI dataset additionally includes classes like trucks, trams, trailers and a larger variety of background elements and viewpoints than our training data.

In summary, given a large enough dataset, our algorithm generalizes well to other datasets, even for different environments and laser scanners.

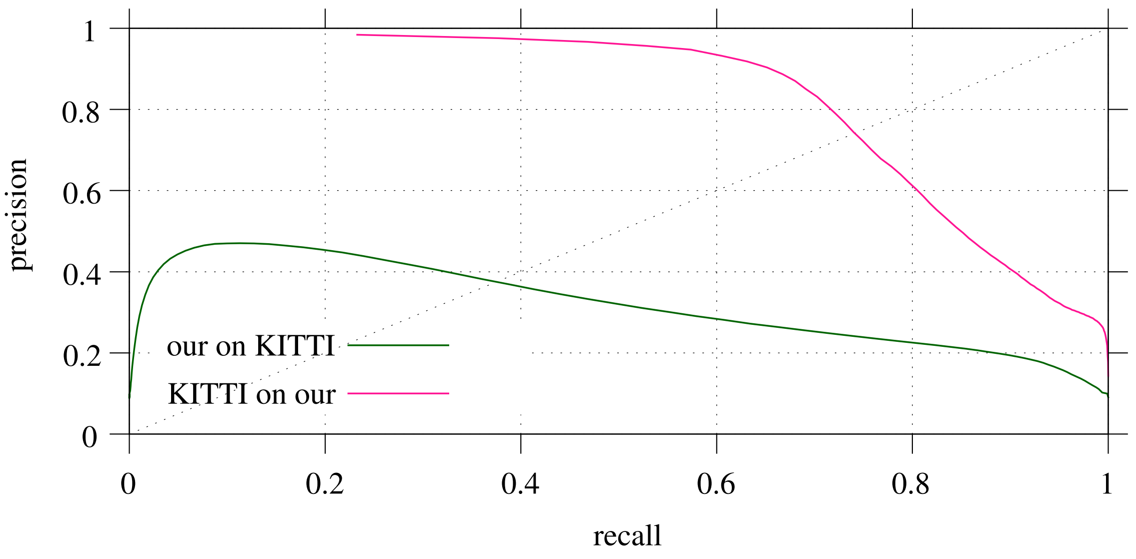


Figure 4.7: Test of the generalization of our proposed method. The pink line shows the performance of the model learned on the KITTI dataset applied to our test data. The green line shows the precision-recall curve for the model learned on our datasets applied to the KITTI dataset.

4.4.6 Feature Extraction and Learning Methods

We further investigate the performance gain in computing the feature statistics at different scales compared to one fixed scale. To do so, we compare the classifier learned on the KITTI dataset with a classifier learned on the same training set but using a feature vector that captures only one neighborhood size of $0.5\,\mathrm{m}$. The results are shown in Figure 4.8. As expected, the *dynamic probability* estimated using the hierarchical feature (solid line) represent the ground truth more precise than the fixed-size version (dashed line). This highlights the fact that dynamic objects can have very different extends and computing the features on different scales helps to deal with the varying sizes.

Furthermore, we compared the performance of the random decision forest with a SVM and a classification using AdaBoost trained on our training dataset. The random decision forest was one order of magnitude faster in training, faster in classification, and provided better classification results, as can be seen in Figure 4.9.

The mean time for computing our hierarchical features on a laser scan with around $55{,}600$ points is approximately 0.46 seconds. The mean classification time of the learned random decision forest is approximately 0.23 seconds per scan. These timings were measured on an Intel i7-4930K and on the city center experiment described in Section 4.4.3.

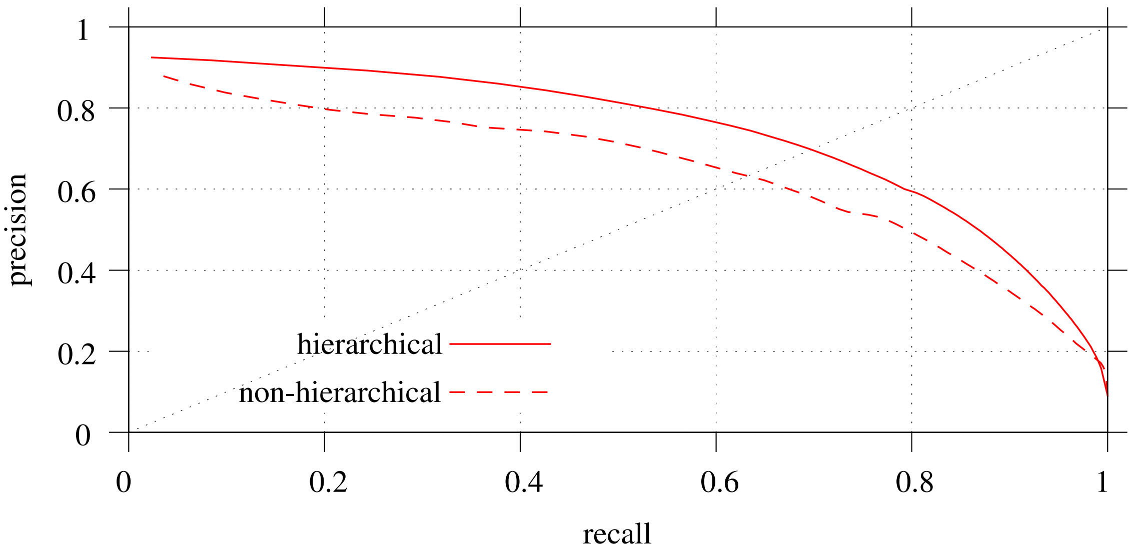


Figure 4.8: This plot highlights the usefulness of the hierarchical feature. In this experiment different sized dynamic objects occur, such as cars, cyclist, pedestrians, trucks, trams, etc. By use of the hierarchical feature (solid line) we are able to get a reasonable result. Without a hierarchical feature (dashed line) you can see that the performance drops.

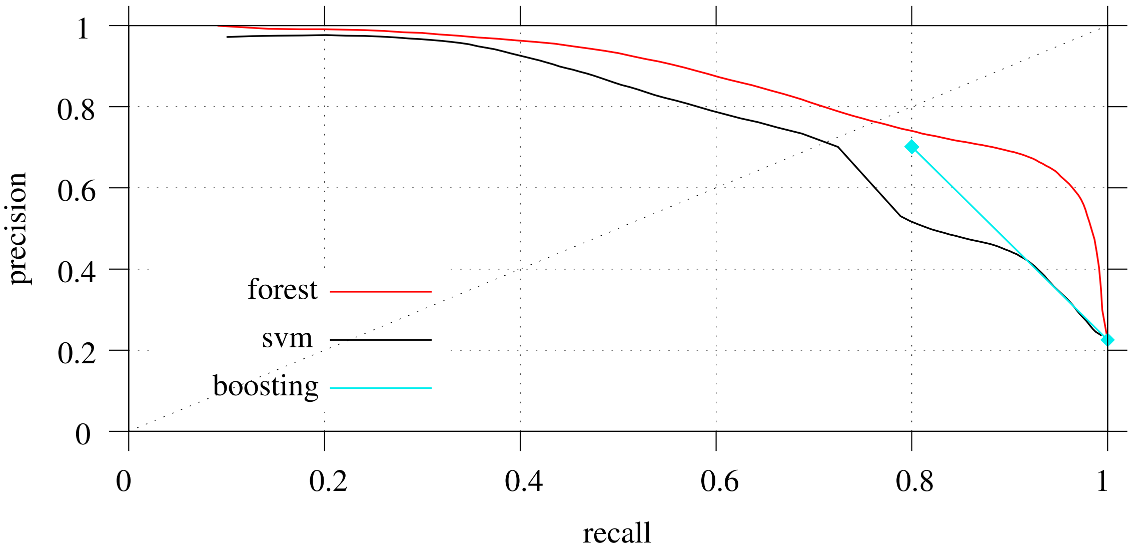


Figure 4.9: Comparing three different learning methods to predict the probability that surfaces belong to dynamic objects. While the boosting method computes a classification, the SVM and the random decision forest predict a probability. Applied to the parking lot dataset the random decision forest produces better and faster results than the other two methods.

4.5 Conclusion

In this chapter, we presented a novel approach to learn a method which allows to annotate each point in a 3D scan with a probability of being caused by a dynamic object. In contrast to other methods, which compare two scans to predict which parts of a scan belong to dynamic objects, the proposed probabilities can be estimated on single scans. The method does not take consecutive scans into account nor does it require a segmentation step. We train a random decision forest on local point features in a supervised fashion. The method allows us to even detect parts of dynamic objects that are stationary in a scan such as parked cars or pedestrians waiting for the bus. In extensive experiments we evaluated the classification performance of the presented approach and favorably compared it to two alternative learning methods. We also demonstrated that our estimated probabilities can be used to improve the recognition results of a difference-based segmentation method.

Chapter 5

Deep Learning Approach for the Detection of Dynamic Objects in 3D Range Data

The detection of dynamic parts of the environment is fundamental for many mobile robotic tasks. It makes scan matching and localization more robust, allows to avoid dynamic obstacles during navigation and helps to generate maps which are valid over an extended time period by including only the static parts of the environment. In recent years, a growing number of perception tasks are being solved by deep learning techniques. They are fast during prediction and do not need features, whose design often requires expert knowledge. In this chapter, we present a method based on a neural network to detect dynamic aspects in 3D LiDAR scans. The presented approach uses a convolutional neural network to estimate for each point the probability of belonging to a dynamic object in an end-to-end fashion. To train and evaluate the performance of our network, we use the KITTI object dataset. Finally, we compare the presented neural network approach against the feature-based method presented in the previous chapter.

5.1 Introduction

In this chapter, we propose a deep learning approach to detect dynamics in single 3D range scans similar to the method presented in the previous chapter. The method presented here uses a neural network to predict point-wise the probability of 3D laser points being reflected by dynamic objects. We denote this estimated probability, as in the previous chapter, by *dynamic probability*. In contrast to many other approaches and as the method presented in the previous chapter, this probability is determined using a single 3D laser scan and does not rely on previous scans or camera images.

Our approach has several features that improve the detection of dynamic objects for mobile robots in highly dynamic environment. First, as the *dynamic probability* is calculated from individual scans, it does not require a comparison of pairs or multiple scans to detect moving objects. Rather, it can identify also dynamic objects that are currently not moving like a standing pedestrian or a parked car. In the remainder of this chapter, we refer to moving and movable objects as dynamic objects. Second, as the prediction of the dynamic objects is based on single 3D scans, our approach can also be applied to robots with bad or no IMU or odometry. Finally, our method is highly efficient and can operate online at 20 Hz. Thus, there is a potential to utilize it to avoid dynamic objects while navigating in dynamic environments. Figure 5.1 shows an example of our proposed probability that measurements were caused by dynamic parts of the environment together with the corresponding range image. Please note, that we do not use camera images in our approach.

The remainder of this chapter is organized as follows. After presenting related work, we introduce our approach to predict the *dynamic probabilities* which is based on a modified ResNet proposed by Valada *et al.* [79]. The network was presented to be used with camera images. We thus show how the 3D scans can be transformed into 2D images to be suitable for the network. This section is followed by the experimental evaluation.

5.2 Related Work

There has been a tremendous amount of work regarding the detection of dynamic objects in either camera data [10, 17, 79] or laser scans [14, 16, 41]. To detect dynamic objects in camera data Fan *et al.* [17] as well as Reddy *et al.* [59] feed images into a neural network to segment the scene into different object classes while also estimating which segments move. In a similar context, Vertens *et al.* [80] apply a neural network to jointly detect cars and predict if these are moving. The network gets consecutive camera images as well as optical flow as input. Chabot *et al.* [10] propose a convolutional network to detect cars in color images. They employ a coarse to fine approach to predict bounding-boxes for cars and additionally fit 3D shape templates to the detection to even predict object parts that are occluded. Chen *et al.* [12] use camera images as well as different views of 3D scans to predict 3D bounding-boxes for different object classes.

Other than these methods, our algorithm does not use camera images. We convert the individual 3D laser scans to two 2D images, one for range and one for intensity. These images have a smaller resolution than a camera image and hold less information.

Instead of images previous, work also employed 3D range scans together with neural networks for object detection. Similar to our work, Li *et al.* [41] convert 3D scans into range images before applying a neural network for object detection. Engelcke *et al.* [16, 83] propose a fast network based on a sliding window to detect objects directly in 3D scans. In contrast to these works, which generate

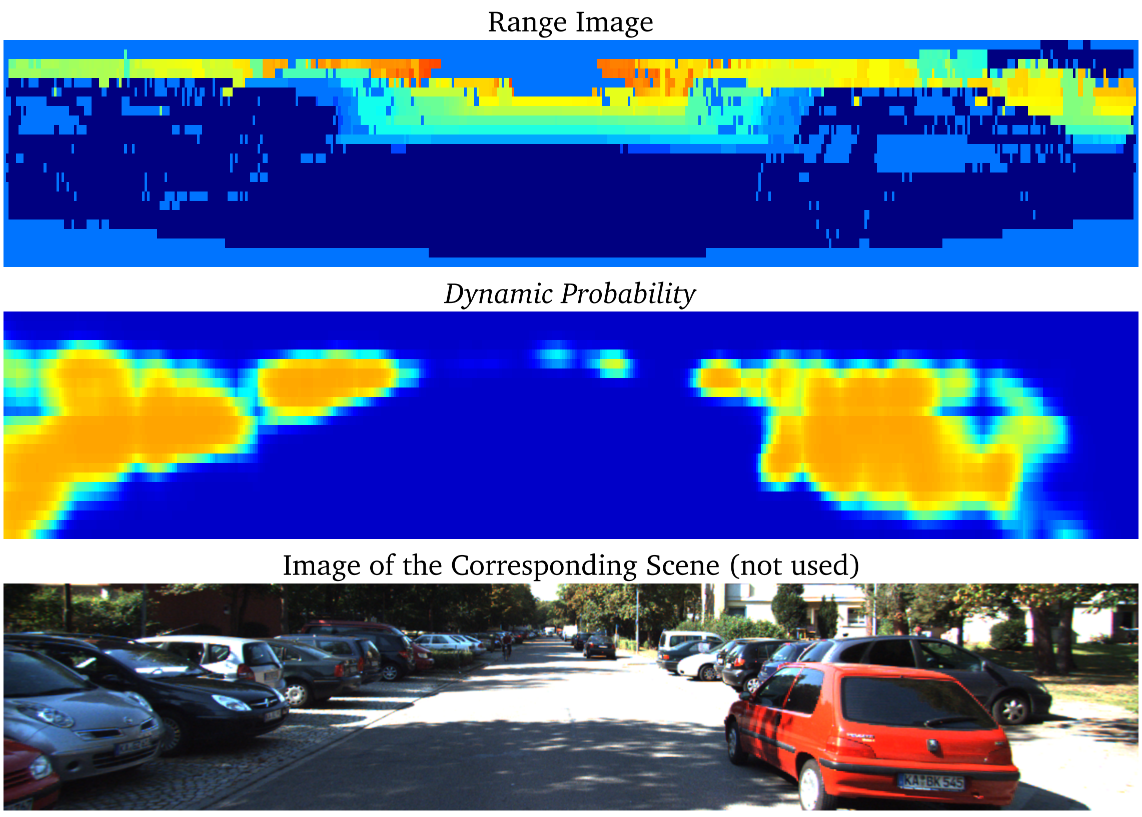


Figure 5.1: An example of a range image (blue to red depicts near to far) together with its computed probability of the point being caused by a dynamic object (orange shows a high *dynamic probability*). Note that the camera image is not used by our algorithm.

bounding-boxes around detected objects, we predict a point-wise probability of belonging to a dynamic object.

Dewan *et al.* [14] propose a method to detect and distinguish moving and movable points in 3D laser scans. While this approach first computes motion flow between two consecutive scans and seeks to identify entire objects, our method uses a single 3D scan as input.

Compared to other approaches about detecting dynamics, our method has several advantages. While the majority of the previously developed methods for laser range data take more than one scan to determine the measurements caused by dynamic objects, our method uses a single 3D scan to predict a per point *dynamic probability* and does not need to take previous measurements into account. Thus, it does not require scan matching or tracking methods. Second, our method does not use camera images and thus is not limited to proper lightning conditions. Third, our method can also identify movable objects that are not moving in the current scan.

5.3 Detection of Dynamic Objects

In this chapter, we propose a method using a neural network to predict the probability that points in single 3D range scans are caused by a dynamic object. Our approach does not only consider moving objects as dynamic objects but also movable objects as they might move in future.

In this work, we apply the *neural expert network* proposed by Valada *et al.* [79] to 3D laser scans. It is a network for semantic segmentation of images and builds upon a modified ResNet50 network. The network follows the general principle of an encoder-decoder network. In the first half, it aggregates the image features while in the second half it upscales the feature maps to the original image size to get the segmentation. Compared to ResNet50, the network uses *multi-scale blocks* to detect objects of different sizes. By applying dilation instead of down-sampling the network allows for a segmentation of higher resolution. For a more detailed network description please refer to Valada *et al.* [79].

The employed network was proposed to be used with RGB-color camera images. To apply the neural network to a 3D laser scan, we first have to transform the scan to a 2D image. We investigate different modalities to fill the image channels, including range and intensity generated from the 3D laser scan (see Section 5.3.1).

For a more robust learning process, we compute the mean for each channel over the whole training dataset and use this to generate zero mean training data. The original network predicts binary class labels only. In this work, however, we are interested in obtaining a probability that a point is caused by a dynamic object. To achieve this, we remove the final *argmax*-layer of the network and interpret the output of the *softmax*-layer as an approximation of the desired *dynamic probability*. After applying the trained network to our 2D representation of the 3D scans, we need to project the prediction back into the 3D scan. To do so, we project each 3D scan point into the range image and assign the corresponding estimated *dynamic* *probability* to it. The result of applying the trained network on a scan from the KITTI object dataset is shown in Figure 5.1.

5.3.1 Modalities

To apply the neural network, which works on 2D images, to 3D laser scans, we first transform the data into 2D modalities such as range or intensity obtained from the 3D scan. In our experiments, we test different modalities and evaluate combinations of them (see Section 5.4.4). In the following, we explain how to compute the modalities used in this chapter.

When recording 3D scans with a moving robot one has to compensate for the motion of the robot based on IMU or odometry data. Accordingly, the back-projection of the 3D points into the range image is approximate and multiple or no points might fall into a single pixel of the generated range image.

To generate the different modalities we first collect all 3D points that are projected into the pixel at position (x, y):

$$\{x_j, I_j\},$$

where $x_j \in \mathbb{R}^3$ is the 3D position and I_j is the measured intensity of the 3D point. Using these we then generate multiple modalities: First, we calculate the minimum distance of all 3D points falling into a cell

$$r_{(x,y)} = \min_j \|x_j\|, \tag{5.1}$$

which we denote as *range*. Furthermore, we calculate the mean *intensity* of all points falling into a pixel

$$I_{(x,y)} = \overline{I_j}. \tag{5.2}$$

For the *height* we compute the mean z-value (up) over all points x_j falling into the pixel (x, y). To get rid of the absolute distance value, we compute the *rangeDiff* modality given by the deviation from the range of the pixel (x, y). More precisely, using all eight neighboring 2D-pixels (x', y') of a pixel (x, y) we calculate

$$d_{(x,y)} = \frac{1}{8-1} \sum_{(x',y')} \left(r_{(x,y)} - r_{(x',y')}\right)^2. \tag{5.3}$$

A comparison of the performance of these modalities for the task of estimating *dynamic probabilities* can be found in the experimental evaluation.

5.4 Experimental Evaluation

In this section, we provide experiments carried out with the KITTI object dataset to test the performance of the dynamic detection. We also present how our approach is able to segment moving and movable objects using our *dynamic probability*. We furthermore compare the performance of the proposed method based on a neural network with the feature-based approach presented in the previous chapter.

5.4.1 Training Data from the KITTI Object Dataset

To train and evaluate our neural network to predict the probability that surfaces belong to dynamic objects, we use the publicly available KITTI object dataset recorded by Geiger *et al.* [22]. The dataset consists of camera images and 3D range scans with labeled bounding-boxes. Our method assigns to each point in a scan the probability that these were created by dynamic objects. For training, we need a 3D point-wise labeling of the training data into dynamic and static points. For this, we project the bounding-boxes into the frame of the 3D LiDAR scans and label all points inside a bounding-box. Each of the provided bounding-boxes encloses an object that can move such as cars, vans, trucks, pedestrians, cyclists, or trams. We treat points falling into the corresponding bounding-boxes as dynamic and all others as static. The bounding-boxes are provided in the camera image and are limited to the camera view. We therefore label only the parts of a 3D scans that overlap with the camera view and set all other labels to unseen.

As we found out during our experiments in the previous chapter, the dataset contains a substantial amount of errors. Several bounding-boxes are missing and others are either displaced or too small. Two examples are shown in Figure 4.4. To reduce the impact of the inaccuracies of the bounding-boxes in this dataset, we increase the box sizes in both horizontal directions (not up and down) for the entire dataset by $0.4\,\mathrm{m}$ (see Section 4.4.4 and Figure 4.5). For our experiments we split the labeled training data into a test and a validation dataset, each with roughly $3{,}700$ scans, as proposed by Chen *et al.* [11].

5.4.2 Data Augmentation

To increase the diversity of our training set we use data augmentation. When creating a range image from a given 3D laser scan one has to provide the sensor origin, which usually is $(0{,}0{,}0)$. We create augmented scans by moving the origin in a radius of $1\,\mathrm{m}$ in the horizontal plane while generating the range image. This produces range images from a slightly different perspective. We then further augment the resulting images by applying augmentations on image level. More precisely, for each image and augmentation, we select the corresponding augmentation with a given pre-defined probability. Thereby, we enforce that at least one augmentation is chosen. Thus, we apply between one and all six augmentations to each image. In our current system, we use the following augmentations with parameters sampled from the denoted intervals (probability of choosing that augmentation in brackets):

- Rotate the image by $[\text{-}2{,}2]$ degrees ($p = 0.4$).
- Scale the image by a factor of $[0.8, 1)$ ($p = 0.4$).
- Translate the image by $[(\text{-}50, \text{-}5), (50{,}5)]$ pixels ($p = 0.2$).
- Flip the image horizontally ($p = 0.3$).
- Crop the image by a factor of $[0.8{,}0.9]$ ($p = 0.4$).
- Skew the image by $[0.025, 0.05]$ ($p = 0.3$).

For each scan, we generate three range images by shifting the origin. We then four times augment each of these images plus the original range image by applying the augmentations described above. Together with the non-augmented images this yields 20 images per scan. For the training, we generate a multi-channel image where the different channels are filled with all tested modalities.

5.4.3 Training the Neural Network

To train the neural network, we use the labels 0 = *static*, 1 = *dynamic* and the *ignore* label 2 for unseen/unlabeled points. The scans in the KITTI dataset were recorded using a laser scanner with 64 individual laser beams. By back-projecting the scans, we receive images with a size of $2{,}000 \times 24$ pixels. We pad the 2D images with zeros such that width and height are a power of two. We also crop the images from a size of $2{,}048 \times 64$ pixels to the field of view of the camera (512×64 pixels). We then trained the network for $100{,}000$ iterations.

5.4.4 IoU Results

We train our network based on different modalities and compare the results with and without augmentation. We use the validation dataset to compute the per-class intersection over union score (IoU, see Section 2.5.4) for our learned neural networks. We use the KITTI object dataset to generate a binary labeling of the data into static and dynamic 3D points. To compute the IoU score, we transform our estimated *dynamic probabilities* into a classification by thresholding the probabilities.

In our first experiment, we demonstrate how well different modalities perform individually and how they can be combined to improve the prediction result. Table 5.1 shows the intersection over union score on the KITTI object dataset for different modalities trained using augmented data. As one can see, *range* and *rangeDiff* perform well as standalone modalities. By combining modalities, the result further improves. The combination of *rangeDiff*, *intensity* and *height* yields the best results. It performs better than the same combination using *range* instead of *rangeDiff*. This is due to the fact that *rangeDiff* shows more contrast as image which seems to help the network.

We also evaluate how much the augmentation boosts the performance of the network. Table 5.2 shows that the result improves for all modalities and their combinations. This result shows that adding more training data would further improve the learned prediction method.

5.4.4.1 Runtime

In this experiment, we demonstrate how much time is spent on the individual components of our approach. For this experiment, carried out on the KITTI object dataset, we used a computer equipped with an i7-2700K and a GeForce GTX 980

	intersection over union		
modalities	**static**	**dynamic**	**mean**
range	0.956	0.632	0.794
rangeDiff	0.958	0.633	**0.795**
intensity	0.945	0.526	**0.735**
height	0.943	0.477	**0.710**
range, intensity	0.961	0.664	0.813
rangeDiff, intensity	0.964	0.688	0.826
range, intensity, height	0.961	0.667	0.814
rangeDiff, intensity, height	0.965	0.695	**0.830**

Table 5.1: Prediction quality of our learned network trained on augmented data using different modalities. The highlighted values show how the combination of modalities increases the performance.

	intersection over union			**mean IoU**
modalities	**static**	**dynamic**	**mean**	**increase**
range	0.945	0.573	0.759	0.035
rangeDiff	0.949	0.592	0.771	0.024
intensity	0.939	0.509	0.724	0.011
range, intensity, height	0.949	0.599	0.774	0.040
rangeDiff, intensity, height	0.957	0.647	0.802	0.028

Table 5.2: Prediction quality of our learned network trained without augmented data using different modalities. The last column shows the increase of the mean IoU when using augmentation. The augmentation increases the performance of all modalities and their combinations.

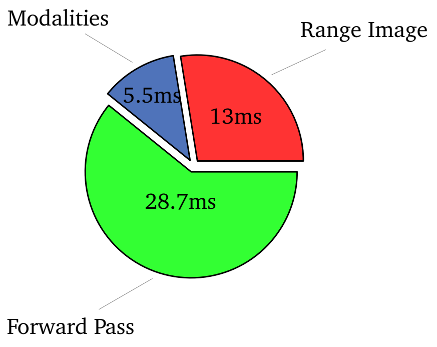


Figure 5.2: Visualization of the time spend on different parts of the estimation of *dynamic object probabilities*.

and ran the detection in a single thread. A graphical representation of the timing is shown in Figure 5.2. To transform the laser scans into a range image, we use the PCL implementation [64] which requires 13.4 ms per scan. Generating the different modalities takes between 0.3 ms (*height*) and 5.2 ms (*rangeDiff*). Finally, predicting the *dynamic probability* given the modalities takes 28.7 ms. Our approach allows to perform the detection of movable objectes at a rate of 20 Hz so that every scan can be processed. The majority of the time required to create the range image can be reduced to less than 1 ms by using ordered laser scans, which are normally provided by Velodyne laser scanners. Unfortunately this ordering is not preserved in the KITTI dataset.

5.4.5 Comparison with Feature-based Dynamic Detection

In the previous chapter, we proposed a method for feature-based estimation of the *dynamic probabilities*. The following experiment compares this method against the method based on a neural network proposed here. We compare the methods using the intersection over union score and discuss the training and prediction time of both approaches. For this comparison, we train the feature-based method using the same KITTI training data used to train the neural network. The training of the random decision forest is sensitive to large differences in the size of training examples for different classes. We counter this by sampling the same amount of training data for each class. We then estimate the *dynamic probabilities* for each scan in the validation dataset. For the computation of the IoU score, a classification into dynamic and static scan regions is needed which we receive by thresholding the estimated *dynamic probabilities*.

Table 5.3 shows the IoU score computed on the predictions of the the feature-based method as well as the IoU score for the network prediction using the modalities *rangeDiff*, *intensity* and *height* as denoted in Table 5.1 and 5.2.

As presented in the previous section, our proposed neural network method takes 47.3 ms per scan, therefore running at 20 Hz on the KITTI object dataset. The feature-based method takes 230 ms for feature computation and classification only

	intersection over union		
method	**static**	**dynamic**	**mean**
neural network (augmented)	0.965	0.695	0.830
neural network (not augmented)	0.957	0.647	0.802
feature-based method	0.945	0.445	0.695

Table 5.3: Comparison of the proposed neural network and the feature-based method presented in the previous chapter. The table shows the prediction quality of our learned network using the combination of *rangeDiff*, *intensity* and *height* trained on augmented and non-augmented data compared to the feature-based results.

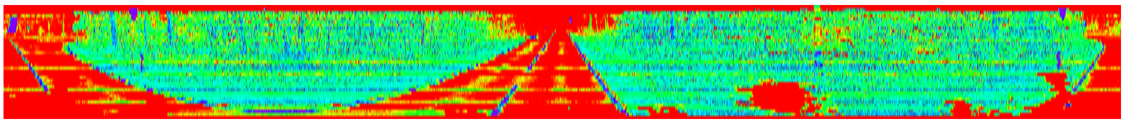
allowing $4\,\mathrm{Hz}$. Note that we are comparing a prediction running large parts on the GPU with a prediction solely implemented on the CPU. The training of the neural network for $100{,}000$ iterations takes 10 hours while the training of the random decision forest takes 4.7 hours on all scans provided in the training data.

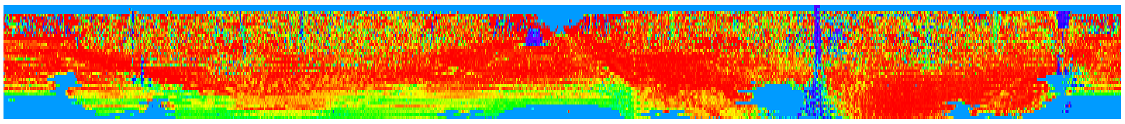
As a result of this experiment, we note that while the training time for the neural network method is larger the estimation of the *dynamic probabilities* benefits from the GPU implementation and is faster during prediction. The quality of the neural network prediction is also notably higher. Another advantage of the neural network method is that we do not need to design features which is a tedious work.

Figures 5.3 to 5.6 show a comparison of the feature-based and deep learning approach for *dynamic probability* estimation. The figures show an intensity image with the estimated *dynamic probabilities* of the feature-based and neural network approach on different scenarios from the KITTI object dataset. Cars and pedestrians are reliably detected by both methods. The feature-based method tends to confuse pillar-like objects as tree trunks or street lights with dynamic objects.

5.5 Conclusion

In this chapter, we presented a method to detect dynamic objects in 3D scans. We train a neural network to predict the probability that points are caused by dynamic objects for range images generated from single 3D laser scans. Despite only using single scans, we are capable of detecting moving objects as well as parked cars and other movable objects. We demonstrated the performance of our approach using the publicly available KITTI object dataset. We compared the proposed method against the feature-based dynamic detection presented in the previous chapter. Our proposed neural network method is able to run at $20\,\mathrm{Hz}$ and achieves more precise *dynamic probabilities* which results in higher IoU scores than those estimated by the feature-based method.


(a) Intensity image (red: low reflectance, blue: high reflectance)


(b) Feature-based predition (purple: dynamic, red: static)

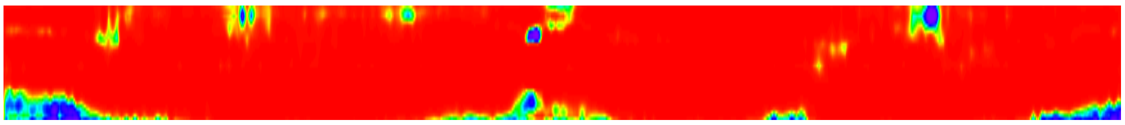
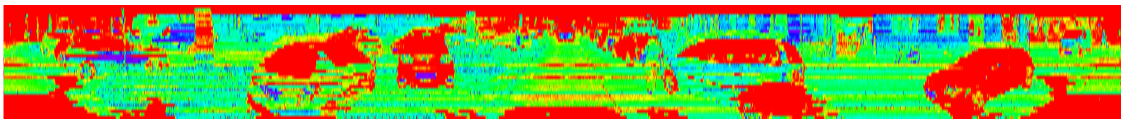
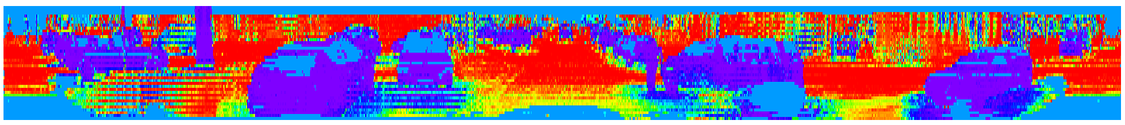

(c) Neural network prediction (purple: dynamic, red: static)

Figure 5.3: Comparison of the *dynamic probability* estimation methods on a road lined by trees. The car in the center, which is barley notable in the intensity image, is detected by both methods. The feature-based method confuses a tree on the right side for a dynamic object.


(a) Intensity image (red: low reflectance, blue: high reflectance)


(b) Feature-based predition (purple: dynamic, red: static)

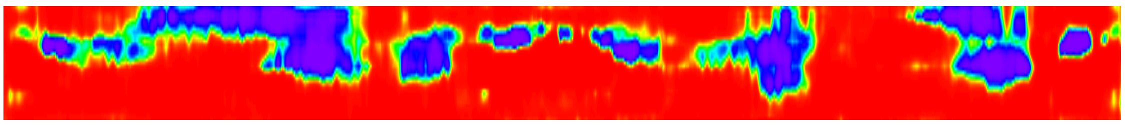
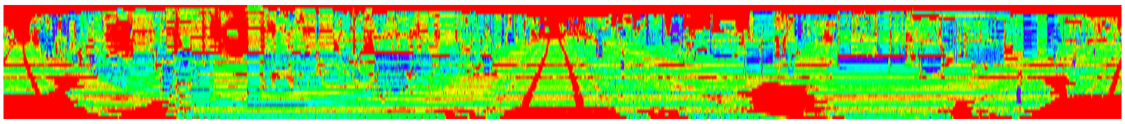
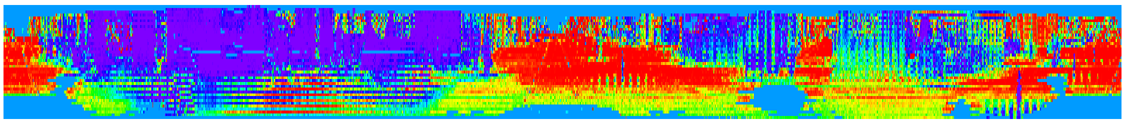

(c) Neural network prediction (purple: dynamic, red: static)

Figure 5.4: Comparison of the *dynamic probability* estimation methods on a road with parked cars. Both methods detect the cars, while the feature-based method additionally is able to detect the person on the right side of the road.


(a) Intensity image (red: low reflectance, blue: high reflectance)


(b) Feature-based predition (purple: dynamic, red: static)

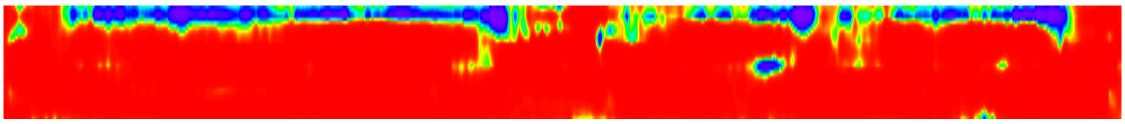
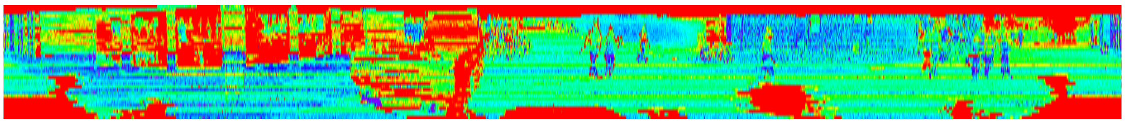
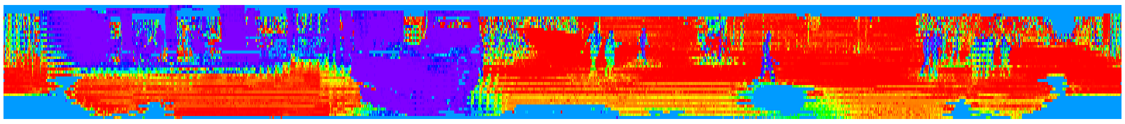

(c) Neural network prediction (purple: dynamic, red: static)

Figure 5.5: Comparison of the *dynamic probability* estimation methods on a inner city road without dynamic objects. While the neural network predicts dynamic objects mostly on image parts which are not covered by the scan, the feature-based method falsely reports large regions as dynamic.


(a) Intensity image (red: low reflectance, blue: high reflectance)


(b) Feature-based predition (purple: dynamic, red: static)

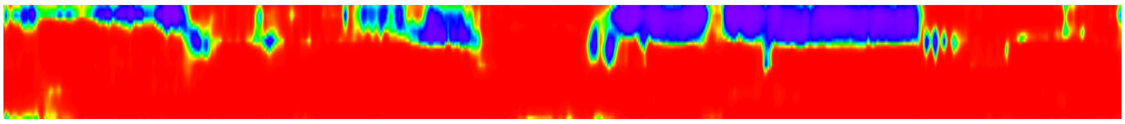

(c) Neural network prediction (purple: dynamic, red: static)

Figure 5.6: Comparison of the *dynamic probability* estimation methods on a road with multiple pedestrians. There are at least eight pedestrians on the right side of the image detected by both methods. The feature-based method predicts dynamics for the truck on the left side but confuses the building on the left with a dynamic object. The neural network detects the flat building surface as dynamic object.

CHAPTER 6

Mapping with *Dynamic Probabilities*

Localization on publicly available road networks is well suited for the navigation in previously unseen outdoor environments. However the precision provided by this method is limited by how accurate the road network represents the streets. It also does not represent information about the surrounding which is crucial for many robotic tasks. In this chapter, we propose a novel method for building 3D grid maps using laser range data in dynamic environments. Our approach feeds the estimated *dynamic probabilities* proposed in the previous chapters into the mapping module to build 3D grid maps containing only the static parts of the environment. We present experimental results evaluating our mapping process using a parking lot dataset created using one of our robots. In extensive experiments, we show that maps generated using the proposed method and our probability about dynamic objects increases the accuracy of the resulting maps.

6.1 Introduction

Building maps is a fundamental requirement in many robotic tasks. There are numerous types of maps, ranging from graph-based representation of road networks to maps representing the geometry of surfaces in the environment. In this chapter, we present a method to learn 3D grid maps. These volumetric maps subdivide the environment into discrete cells. For each cell the mapping algorithm estimates properties of the represented part of the environment such as occupancy probabilities, color or reflectance. These maps enable different robotic tasks including collision checking, path and task planning. More general maps are used to support all kinds of navigation and localization tasks. However, the presence of dynamic objects in the map increases the difficulty of such tasks. For this reason, localization is usually done using a map that only represents the static aspects of the environment. The generation of such maps, however, requires a robust detection of dynamic objects or measurements caused by such objects. Generating maps without dynamics is usually either done by filtering out measurements caused by dynamic objects before giving the data to the mapping process or by removing dynamic objects by matching scan against each other while building the map.

In this chapter, we propose a novel mapping approach to learn three-dimensional maps from 3D laser data. Our approach uses the probability of 3D laser points being reflected by dynamic objects (which we denote as before by *dynamic probability*) to build a map of the static components only. In our approach, we first apply the neural network proposed in the previous chapter to predict the probability that a measurement is reflected by a dynamic object. During mapping, our approach integrates data proportional to this estimated *dynamic probability* into the map. Generating maps without dynamic objects typically yields maps that remain valid for extended time periods. The maps we generate are 3D grid maps in which cells store the probability that a scan beam is reflected by a static object.

The remainder of this chapter is organized as follows. After presenting related work, we present the mapping process which is a modified version of the mapping approach presented by Hähnel *et al.* [25]. This section is followed by the experimental evaluation.

6.2 Related Work

Many mobile robot systems rely on volumetric maps for localization. Moravec *et al.* [54] introduced occupancy grid maps in the 1980s. Using sonar measurements the authors estimate 2D grid maps where each cell is either occupied or free. Stachniss *et al.* [69] introduced an alternative map representation called coverage maps. In contrast to the binary cell occupancy these maps estimate how much of a cell is occupied by an obstacle. Both mapping approaches assume a static environment. Building maps in dynamic environments requires to cope with measurements caused by moving, appearing or disappearing objects. Within a SLAM framework Wolf *et al.* [86] present a mapping algorithm which estimates separate maps for dynamic and static objects to improve the performance of the system in dynamic environments. Meyer-Delius *et al.* [51] introduced a variant of occupancy grids in which they utilize a Hidden Markov Model for every cell to keep track of the potential changes in the occupancy of each cell. Hähnel *et al.* [25] introduced a probabilistic approach based on the expectation maximization (EM) algorithm to estimate the beams reflected by moving objects from entire laser scans and to build a map of the static aspects of the scans. In this chapter we build upon this mapping approach but instead of the EM-based estimation of static objects, we employ a prior of movable objects and thus we are able to remove measurements caused by dynamic objects that are not moving during the data collection process such as parked cars or standing pedestrians.

6.3 Mapping with *Dynamic Probabilities*

The goal of our method is to create a 3D grid map which only contains those components of the environment that are static over an extended time period. To achieve this, we first use the neural network proposed in the previous chapter to compute a per-point probability of being caused by a dynamic object on all input LiDAR scans. We utilize this *dynamic probability* to build a grid map of the static environment parts. In the remainder of this section, we describe how we build a 3D grid map from the labeled scans.

To compute a 3D grid map from the set of scans we adapt the map building method proposed by Hähnel *et al.* [25]. This approach employs an expectation maximization (EM) framework to decide which beams of a range scan are reflected by static objects. These beams are then used to compute α- and β-values for each cell of the map. Here, α corresponds to the number of beams which end in this cell and β counts the beams that pass through a cell without ending in it. These values are then employed to compute the reflectance probability of a grid cell according to

$$m = \frac{\alpha}{\alpha + \beta}. \tag{6.1}$$

In contrast to their work we do not need EM as we predict a *dynamic probability* with our trained neural network. Instead we directly incorporate our continuous probability that measurements were caused by dynamic parts of the environment into the calculation of α and β. As we are only interested in the static aspects of the environment we treat measurements that hit a dynamic object as miss in the mapping process.

Let p be the probability that measurements were caused by dynamic parts of the environment, calculated by our network for a beam that is not a maximum range measurement. For the cell, in which that beam ends, we add the probability that the measurement was caused by a static object: $1 - p$ to the α-value. In addition, we add the *dynamic probability* p to the β-value of that cell. If the beam was a maximum-range measurement, we update neither the α- nor the β-value. Independent of maximum range measurements, we increment the β-value of every cell traversed by the beam by one. More formally, for a beam that has a estimated *dynamic probability* p and passes through the cells $j = 1, \dots, k - 1$ and ends in cell k we calculate

$$\beta_j \leftarrow \beta_j + 1. \tag{6.2}$$

If the beam is not a maximum range measurement, we calculate

$$\alpha_k \leftarrow \alpha_k + (1 - p) \tag{6.3}$$

$$\beta_k \leftarrow \beta_k + p. \tag{6.4}$$

If we would employ binary estimates for the *dynamic probabilities* p the mapping algorithm would be equal to the reflectance mapping approach by Hähnel *et al.*

where each dynamic measurement would be assumed a max-range reading. The resulting map estimates high reflectance probabilities for parts of the environment which are static. Volumes with dynamic objects are represented by low probabilities that these cells reflect a beam by a static object.

An overview of our mapping algorithm is depicted in Figure 6.1. First we transform the 3D scans into 2D modalities such as range or intensity images. These images are then fed into our learned neural network. The network predicts for each pixel in the 2D image a probability that the corresponding 3D point was caused by a dynamic object. This *dynamic probability* is then projected back into the 3D scan. In the mapping step we employ the estimated probabilities to incorporate the measurements into the 3D grid map. The first part of this pipeline is described in Chapter 5.

6.4 Experimental Evaluation

In this section, we show how *dynamic probabilities*, as proposed in the previous chapters, can be used to generate a 3D grid map that represents only the static parts of the environment. We also show that our proposed mapping algorithm is able to remove moving objects as well as movable objects. In this evaluation we use the neural network proposed in the previous chapter for the detection of *dynamic probabilities* but any method predicting the probability that a measurement was caused by a dynamic object can be employed. Our network was trained using the KITTI object dataset.

We used our robot Viona (see Section 2.8) equipped with a Velodyne HDL-64E LiDAR and an Applanix PosLV (IMU and GPS) to record datasets on our campus parking lot. We applied a SLAM system to correct the scan poses reported by the Applanix system. Following the results of the previous chapter we use our neural network together with the modality combination of *rangeDiff*, *intensity*, *height* to estimate the *dynamic probabilities* in the rest of this chapter as it performs best.

To apply the neural network trained on the KITTI object dataset to our data we had to correct the intensities that are different on both datasets. To generate data for training and testing we compute the per channel mean of the training dataset and subtract it channel-wise to get zero mean data. This mean value per channel stays the same for training and testing. To compensate for the different intensity values in both datasets we recomputed the mean for the *intensity* modality on one of our parking lot dataset and used this value during prediction. For this experiment we use one of our recorded campus parking lot datasets to build a map as proposed above. The mean of the estimated *dynamic probabilities* per cell on the parking lot dataset is shown in Figure 6.2.

We generate two different maps from the same dataset. The first map incorporates points given their *dynamic probability* as well as a second map were we assume all points are static. We choose a cell size of $0.25\,\mathrm{m}$ for the mapping.

To show that our mapping process successfully removes dynamic objects we

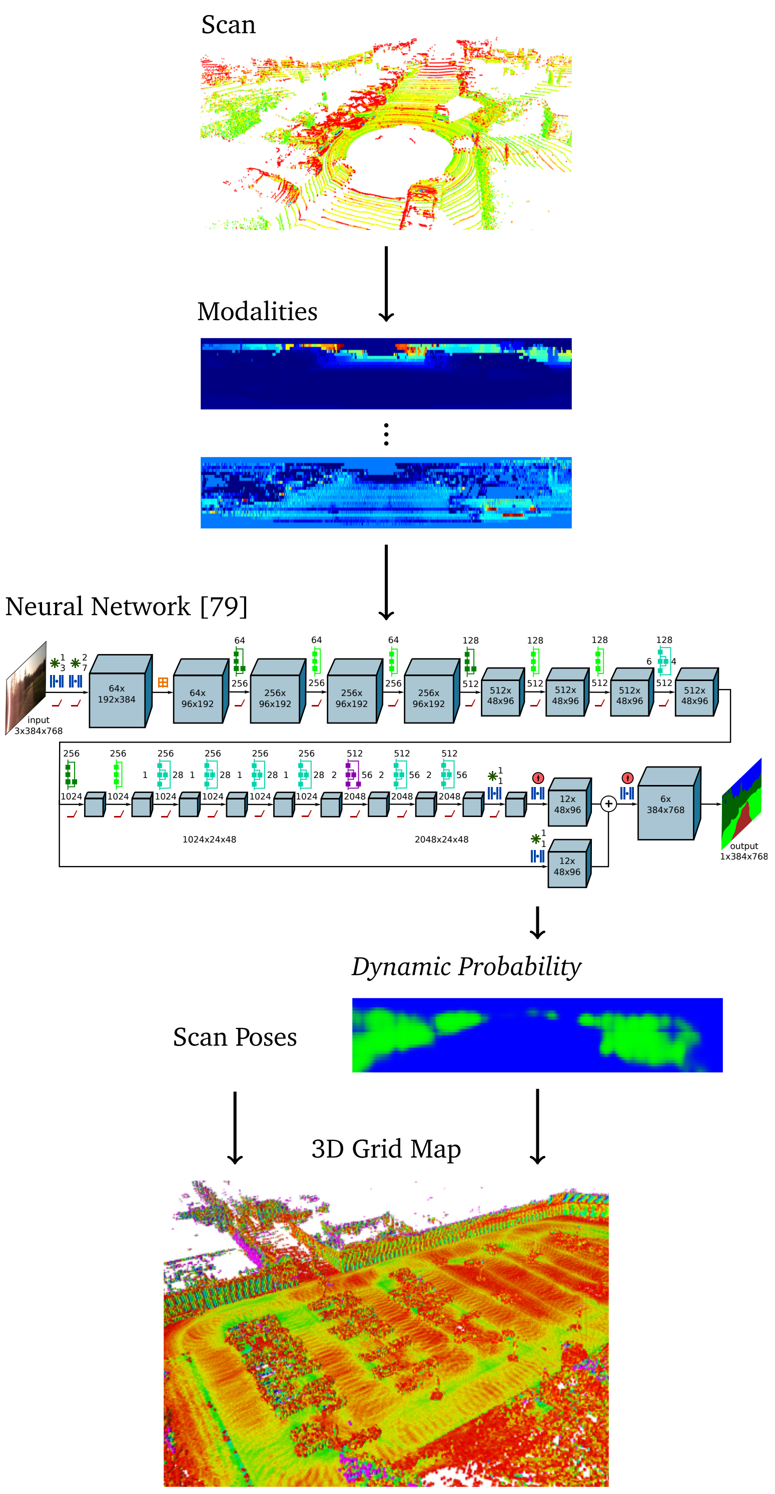


Figure 6.1: Overview of our proposed system. We first convert the 3D scan into 2D images, which we then feed into the network. We utilize the resulting *dynamic probability* with the scan poses to generate the 3D grid map of the static aspects of the environment.

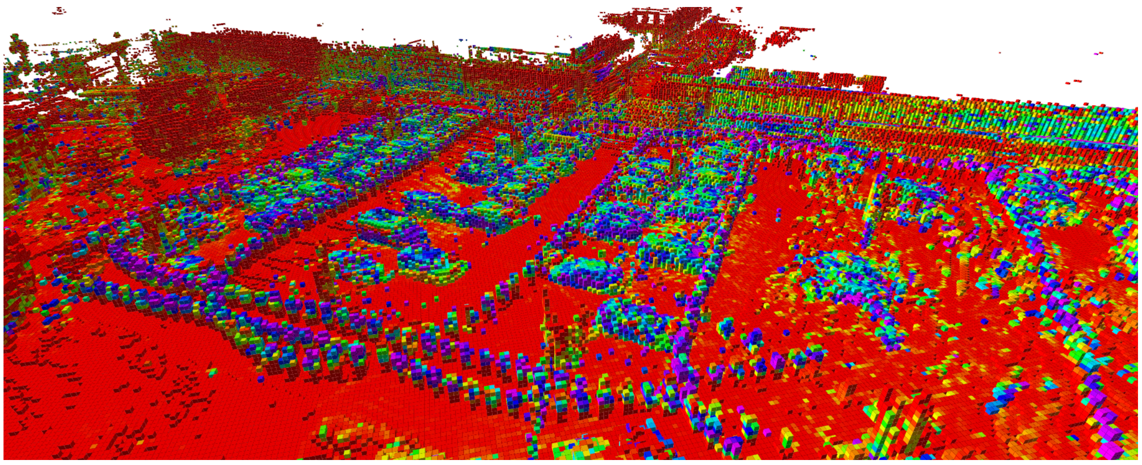
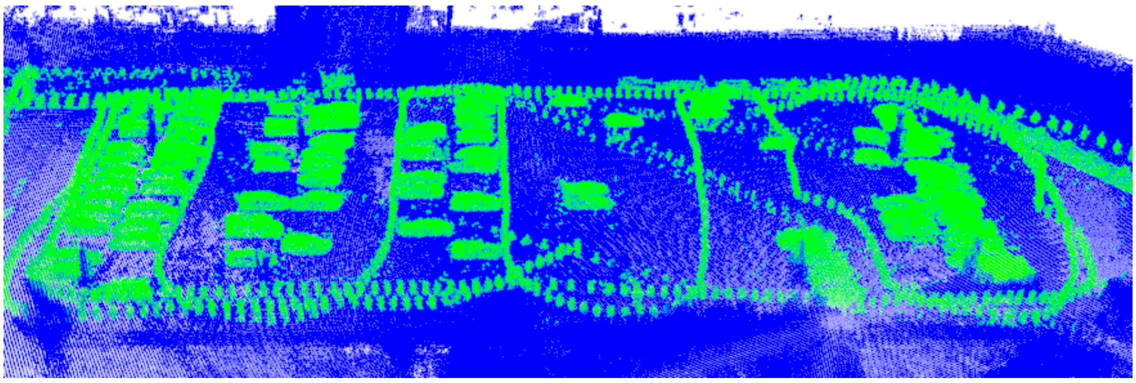


Figure 6.2: Mean estimated *dynamic probabilities* on the parking lot dataset. Dynamic objects are shown in blue (parked cars, walking pedestrians) while static objects are red.

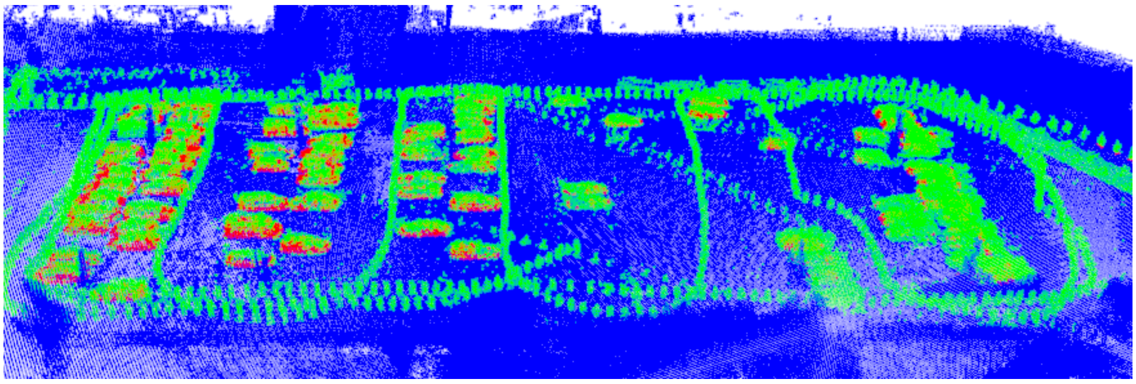
manually generated a ground truth labeling of static and dynamic objects for the dataset. Then, we determined if a dynamic object is represented in the map by a cell with a reflectance value of at least 0.5. The ground truth labeling as well as the two maps annotated with the not included (green) and included dynamic cells (red) are shown in Figure 6.3. We can see that objects like the moving person recording the dataset are not included in either map. On the other hand the parked cars are only removed by our mapping process using the *dynamic probabilities*. Our mapping method using the *dynamic probability* is able to remove 95.66% of all dynamic cells while the map generated assuming all points are static is able to remove the moving objects such as the pedestrians and cyclists but not the cars, thus removing 78.25% of all dynamic cells.

6.5 Conclusion

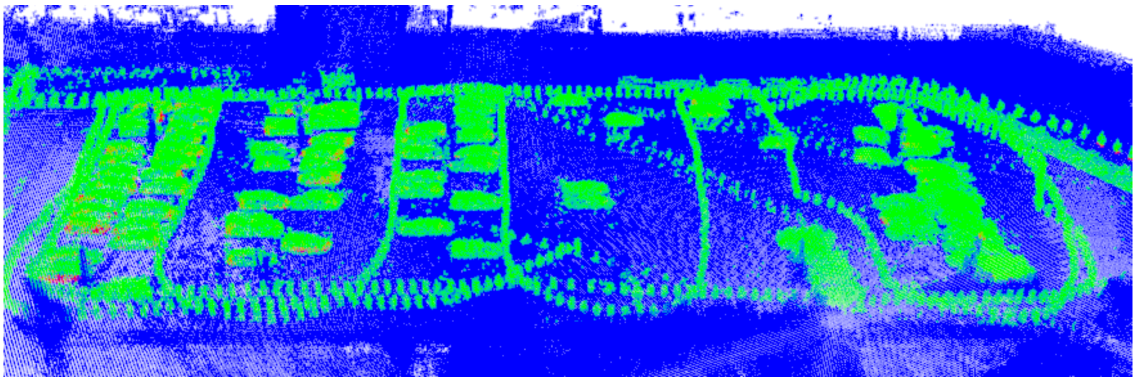
In this chapter, we presented a method to generate a 3D grid map of the static aspects of the environment of a mobile robot. We employ the neural network proposed in the previous chapter to predict a pixel-wise *dynamic probability* that a point was caused by a dynamic object for range images generated from single 3D laser scans. By incorporating measurements proportional to the predicted probability we generate maps containing the static aspects of the environment. In the experiments we demonstrated that our method generates accurate maps of the static parts of the environment while excluding the dynamic aspects.



(a) Ground truth



(b) Map assuming everything is static



(c) Map of our proposed algorithm

Figure 6.3: Our approach robustly removes dynamic objects from the generated maps. The top image shows the manually labeled dynamic objects in green. The second image shows the map built under the assumption that everything is static. While the moving pedestrians as well as the cyclists are not included in the map (green), the parking cars at least partially are (red). The bottom image demonstrates how effectively our approach also removes parked cars.

Chapter 7

Multi-Class Mapping

A map representing the static aspects of the environment is well suited for localization, as the map is not affected by changes caused by dynamic objects. For other tasks, such as safe navigation or local path planning, the information about appearance of dynamics in the environment is crucial. In this chapter, we present a method to learn 3D grid maps that represent the reflection probability of an object in the environment as well as an estimate about the dynamics of objects present in the map. We therefore extend our mapping approach presented in the previous chapter to include dynamic objects in the map and to keep track of the distribution of dynamic objects occupying cells. To learn a map representing the most recent consistent state of the environment, we detect changes in the environment during mapping. We evaluate our algorithm using the KITTI tracking dataset and show that the proposed method generates maps that represent the most recent state of the environment.

7.1 Introduction

A key requirement for mobile robots is their ability to localize themselves in the environment. For this task a map representing the static parts of the environment is sufficient and allows to be used over an extended time span, as it is not disturbed by dynamic objects. However, to treat dynamic objects as free space during path planning or navigation can lead to unforeseen encounters with dynamic objects. One solution to overcome this problem is to plan paths on a map representing the static environment and detect dynamic obstacles in the sensor data during navigation to avoid them. However, this approach can lead to suboptimal paths if a planned trajectory is blocked by obstacles not present in the static map. By planning paths on or navigating using maps representing also the temporary, dynamic objects the performance for these tasks can be increased.

In this chapter, we propose a method to learn maps that include static as well as dynamic parts of the environment. The maps proposed in the previous chapter estimate for each cell the probability that a beam hitting this cell is reflected by a static object. In this section, we extend our mapping algorithm to estimate the reflectance probability together with the distribution of dynamic objects which occupy the cells. To build maps of static environments, most mapping algorithms assume that the environment does not change during mapping. However, this assumption is violated in dynamic environments. A robot driving by a closed door multiple times, estimating a map of its environment using the original occupancy grid mapping algorithm [54], takes a long time to adjust its map to the new situation when this door is opened. By detecting such changes in the environment we can adjust the map right after we detected the change. To build a map including the dynamic parts of the environment we therefor need to keep track of the map parts which change due to moving objects or such that change their *dynamic object type*. An example of the later is a car that parks, as it changes its type from moving to movable. The method proposed in this chapter detects whenever a cell changes and only uses data after such a *break point*, as measurements before this change are not created by the same environment. The map does not only store the reflectance probability but also estimates a distribution of dynamic objects present in each cell. We use a neural network to estimate for each pixel the probability that measurements are caused by objects of one of multiple *dynamic object types*. For a stream of 3D LiDAR scans we estimate whenever the state of a cell changes. This information together with the probabilities that points were created by dynamic objects is then used to build a 3D grid map. To train and test our neural network and to evaluate our mapping approach we use the KITTI tracking dataset.

The remaining chapter is structured as follows. The next section outlines related work on the topics of mapping in dynamic environments and semantic segmentation. After this, we present our algorithm for multi-class mapping. The method consists of three parts. The first part is the estimation of *dynamic probabilities* on the scans. On this data we detect whenever the environment changes by computing *break points* in the stream of measurements. The last part of our algorithm is the generation of a 3D grid map representing the reflectance probability and the distribution of the objects represented in the cells. This section is followed by the evaluation of our dynamics and *break point* detection methods as well as the grid mapping algorithm.

7.2 Related Work

There are numerous approaches to map in dynamic environments. In an expectation maximization fashion Hähnel *et al.* [25] alternate the prediction which parts of the environment are dynamic with the creation of a map using only the static parts of the environment. In contrast to our work, they treat dynamic measurements as outliers and discard them. Arbuckle *et al.* [1] introduced *temporal occupancy grids*.

Instead of storing a single occupancy value per cell, the authors propose to estimate multiple values computed over data from different time intervals. This allows then to distinguish static from dynamic regions but does not provide the time when the change occurs. Luber *et al.* [44] build maps in which they estimate the probability of human occurrence using poison processes. Meyer-Delius *et al.* [51] model the dynamics of a scene by the state transition probabilities of a Hidden Markov Model that they fit to the observed data. In the previous chapter, we proposed an approach to learn maps that represent the static parts of the environment. We used a method to estimate the probability that individual points of a 3D laser scan are created by dynamic objects. We then incorporated beams proportional to this probability into a grid map to predict the probability that a beam hitting a cell is reflected by a static object. Other than these methods, which build a map of either the static environment or store where dynamics occur, we estimate in this chapter a map that represents the most recent environment state including static as well as dynamic parts in the map while accounting for map changes.

Luft *et al.* [46] propose the idea of *break point* detection on occupancy grid maps. These *break points* mark whenever the environment changes in a given cell. In their work, the authors estimate the *break points* in a stream of binary hits and misses. Using only data after the latest *break point* they build a map of only the most recent consistent measurements. In contrast to their work, we extend the *break point* detection to a stream of histograms representing real-valued probabilities.

The map generated by our proposed method is a semantic map in which labels correspond to the *dynamic object type* of the maximum of the estimated *dynamic probabilities*. Nüchter *et al.* [55] build semantic indoor maps. They classify planes into different semantic classes based on their orientation. These maps are then filled with objects labels from learned single class object detectors. Oliveira *et al.* [56] combine a dense SLAM system, a convolutional neural network and ElasticFusion to generate dense semantic 3D maps. Sünderhauf *et al.* [72] build 2D grid maps which they annotate per room with a room type. They then use this map to boost object classification results based on the room type the object is found in. Xiang *et al.* [88] jointly build a 3D scene together with a semantic labeling. The result of their method is a surface map in which learned objects are semantically annotated. While the above mentioned methods generate semantic maps of the environment, none of them accounts for change detection in the mapped area. In our approach, we detect for each cell whenever the state changed and use this information during mapping.

In this chapter, we extend our neural network approach for *dynamic probability* prediction to distinguish between measurements of moving and movable objects. In a similar context Dewan *et al.* [14] proposed a neural network to estimate an *objectness* score on consecutive 3D scans. Simultaneously, the authors estimate the motion of the scene using *Rigidflow* and use the result to filter the dynamic estimation over the whole sequence of scans by clustering objects. While Dewan *et al.* estimate motion for cars only, we do not distinguish between different types of objects and do not filter the resulting predictions.

7.3 Multi-Class Mapping

In this chapter, we present an extension to the 3D grid mapping method presented in the previous chapter. The method presented before estimates for each cell the probability that a beam hitting a cell is reflected by a static object. This section explains the extension of our mapping algorithm to estimate the reflectance probability as well as the distribution of *dynamic objects types* of the objects the cell is occupied by. We train a neural network to predict point-wise probabilities that 3D points were caused by one of multiple *dynamic object types*. In our implementation, we distinguish three types of dynamic objects: *static*, *movable* and *moving*. Parked cars or standing pedestrians are movable objects, while everything that changes its position is a moving object. Movable objects can start moving at any time and become movable and vice versa. In the mapping algorithm presented before, the reflectance probability is estimated using all measurements ending in or traversing a cell. This was possible as we treated all non-static measurements as a miss. If, however, a cell is occupied by an object and this object is moved or an object appears in a free cell, a standard grid mapping approach, which uses all data, needs multiple measurements to adjust the occupancy estimate to the new situation. Detecting such changing cells and resetting these allows for immediate change of the reflectance probability estimate, as all measurements seen until now are generated by a different state of the environment. For the detection of such *break points*, we propose two methods: an entropy-based method and a probabilistic approach. We then show how the estimated probabilities can be used, together with the detected *break points*, to build a 3D grid map in which each cell estimates the reflectance probability as well as a distribution of *dynamic object types*.

7.3.1 Three-Class *Dynamic Object Probability* Estimation

In the previous chapter we estimated *dynamic probabilities* that measurements were caused by dynamic objects. In this chapter we estimate similar probabilities for three *dynamic object types*: *static*, *movable* and *moving*. We denote these probabilities that surfaces belong to one of those *dynamic object types* by *dynamic object probabilities*. The first step of our algorithm is to assign to each point of a 3D laser scan the *dynamic object probabilities*. To predict these, we employ a neural network. As in the previous chapter, we use the AdapNet by Valada *et al.* [79]. To be suitable for the network our algorithm first generates range and intensity images from the 3D scans. To get rid of the absolute distance value in the range image it uses the *rangeDiff* modality (see Equation (5.3)). To differentiate between moving objects like cars on the road and movable objects such as parked cars, we additionally use an image with precomputed motion estimates. This image is joined with the *rangeDiff* and *intensity* (see Equation (5.2)) modalities into a three channel image and feed into the network. The network usually gives a single best label as a result. As we are interested in probabilities of each point belonging to

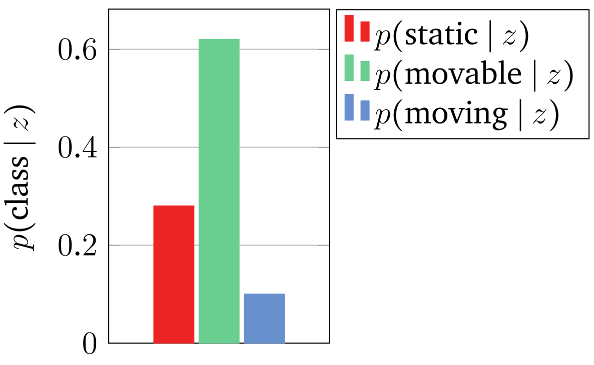


Figure 7.1: Histogram visualization of the probabilities that a measurement was created by an object of one of three *dynamic object types*. The red bar shows the probability for the point being *static*, while green and blue show *movable* and *moving*.

one of the three *dynamic object types*, we remove the last *argmax*-layer from the network and interpret the three resulting class responses as probabilities. To reduce overfitting and to train a more robust network, we augment our training data. This happens in two steps. During the generation of a range image from a 3D scan one needs to provide the sensor pose in the scan. By adding an offset to this pose, we generate range images with a small perspective change. We then apply additional augmentation on image basis to the resulting modality images.

By applying our trained network, we receive for each pixel in the input image a distribution that this measurement was caused by one of the three *dynamic object types*. Therefore we define our measurements as follows:

$$z = (p\,(\text{static} \mid z)\,,\, p\,(\text{movable} \mid z)\,,\, p\,(\text{moving} \mid z)) \tag{7.1}$$

In the remainder of this chapter, we represent these *dynamic object probabilities* as a histogram and denote it for a scan at time t as z^t. A visual representation is shown in Figure 7.1. The red bar shows the probability that the measurement was caused by a static object: $p\,(\text{static} \mid z)$, the green bar marks the probability for movable objects: $p\,(\text{movable} \mid z)$ and blue visualizes the moving object probability: $p\,(\text{moving} \mid z)$.

7.3.2 *Break Point* Detection

In our mapping algorithm proposed in the previous chapter the reflection probability of a cell is determined using all data falling into a cell. This is possible, as these maps represent only static parts of the environment. However, if the state of the cell changes during mapping, this is not taken into account. In a map representing also dynamic objects such cases might for example occur if a car drives by or a door is opened or closed. Luft *et al.* [46] propose a method to detect such changes for occupancy mapping and to estimate the time when this happened: the *break point*. To estimate the *break point* for a binary stream of hits and misses the authors propose methods based on entropy, the Bayesian Information Criterion and a probabilistic approach. A detailed explanation can be found in their paper [46]. Building on this work, we propose two methods to detect *break points* in a stream of histograms of multiple real-valued probabilities that measurements were caused by objects of different *dynamic object types*.

As the order in which points in a single scan are recorded result from the rotating sensor and does not represent a meaningful change in time, we only consider *break points* between scans. All measurements $z_1, \ldots, z_N$ a cell receives from a single scan at time t are therefore merged into one measurement: $\mathbf{Z}^t$. For the combination of measurements we compare two methods. The first method takes the mean of all measurements z_i:

$$\mathbf{Z}^t = \frac{1}{N} \sum_{1}^{N} z_i. \tag{7.2}$$

The second method combines the individual measurements by element-wise multiplication with an additional normalization, such that the result sums up to 1:

$$\hat{\mathbf{Z}}^t = z_1 \odot \ldots \odot z_N, \tag{7.3}$$

$$\mathbf{Z}^t = \frac{1}{\|\hat{\mathbf{Z}}^t\|} \hat{\mathbf{Z}}^t, \tag{7.4}$$

where $\odot$ denotes the element-wise multiplication. A comparison of both methods can be found in the experimental evaluation section (Section 7.4.3).

To compute the most likely *break point* b given a stream of N observations we implemented and tested an entropy-based and a probabilistic approach which are presented below.

7.3.2.1 Entropy Splitting

The first method to find the point in time where the underlying distribution of *dynamic object types* changes is based on the entropy of all data after the potential *break point* b. As long as adding a measurement to our stream does not increase the entropy, this measurement was caused by an object of the same *dynamic object type*. To find the *break point* b we minimize the entropy of the distribution of *dynamic object types* in our stream of histogram data:

$$\underset{b}{\operatorname{argmin}}\, H(b), \tag{7.5}$$

where we compute the entropy given a *break point* b as the mean of the individual histogram measurement entropies after time b:

$$H(b) = \frac{1}{N-b} \sum_{t=b}^{N} H(z^t). \tag{7.6}$$

The entropy of a single measurement of the distribution of discrete *dynamic object types* represented as a histogram of *dynamic object probabilities* is defined as follows:

$$H(z) = -\sum_{i=1} z_i \log z_i, \tag{7.7}$$

where z_i are the individual *dynamic object probabilities* (see Equation (7.1)). The method returns a break at the start of the sequence if there is no state change present.

7.3.2.2 Probabilistic Splitting

The second method, we propose to determine a *break point* in a stream of histogram measurements, is based on the probability that these are generated by an object of the same *dynamic object type*. The minimum of these probabilities over all potential *break points* is the most likely *break point*. We first assume that the distribution before and after our potential *break point* b is each generated by objects of the same *dynamic object type*. We therefore take the mean of all measurements before and after b and compute the probability that these come from the same distribution:

$$\operatorname*{argmin}_b P_b\left(\frac{1}{b}\sum_{t=1}^{b} z^t\ ,\ \frac{1}{N-b}\sum_{t=b+1}^{N} z^t\right). \tag{7.8}$$

The first argument of P_b is the mean of the histogram before and the second the mean after the *break point* candidate b. For two combined measurements $\mathbf{Z}$ and $\mathbf{Z}'$ the probability that these represent the same *dynamic object type* is given by:

$$P_b(\mathbf{Z}, \mathbf{Z}') = \|\mathbf{Z} \odot \mathbf{Z}'\|. \tag{7.9}$$

We defined a minimum probability P_0 that must be exceeded to be assumed a valid *break point*. This value is dependent on the data and needs to be learned (see Section 7.4.3).

7.3.3 Mapping

During mapping, we learn a 3D grid map which estimates for each cell the probability that a beam send into this cell is reflected. Additionally, the map estimate a histogram of probabilities that the cell is occupied by an object with the respective *dynamic object type*. For the mapping, we introduce a fourth *dynamic object type* representing that the measurement went through the cell without ending in it. We integrate such misses as a measurement to the seen measurements created by beam endpoints. During the mapping more than one measurement might fall into a cell. These measurements are combined either by taking the mean or by element-wise multiplication (see Equations (7.2) and (7.4)). Especially for the multiplication method no value should be zero. We ensure this by integrating hits as $z = \eta_1 (p(\text{static} \mid z), p(\text{movable} \mid z), p(\text{moving} \mid z), \epsilon_m)$ and misses as

$z = \eta_2 (\epsilon_h, \epsilon_h, \epsilon_h, 1)$, where η_1, η_2 are normalization factors such that the *dynamic object probabilities* sum up to one and $\epsilon_m, \epsilon_h > 0$.

The input to our mapping algorithm is a stream of 3D scans with known poses. We first predict the *dynamic object probabilities* for each scan. Then, the individual 3D points are integrate as measurements into our 3D grid by tracing rays from the sensor pose to the beam endpoint using our custom ray tracing algorithm (see Section 7.3.3.1). Each cell collects hits, misses, the distance rays travel through the cell, as well as the *dynamic object probability* histograms. The data is stored separated for each scan. After integrating N scans, we are interested in the map estimated using all observations of the same unchanged environment. We therefor split the stream of measurements in each cell as long as a break occurs. Each time our algorithm finds a *break point* b, it discards all data before this break and searches in the data $\mathbf{Z}^b, \ldots, \mathbf{Z}^N$ for the next *break point*. If no further *break point* can be found, we estimate the reflection probability as well as the histogram of *dynamic object probabilities* using all remaining data. To run the mapping algorithm online, each cell can be split whenever a new scan is integrated and data before a detected *break points* can be discarded. To estimate the reflectance probability, we employ reflectance mapping, as used in our mapping approach presented in the previous chapter, as well as decay rate mapping [45], which uses the distance a beams travel through a cell. We also present an estimate of the reflectance probability based on the histograms of *dynamic object probabilities*. In the remaining section we describe all components of the mapping approach in more detail.

7.3.3.1 Ray Tracing Algorithm

During the mapping process, we need to find all grid cells a beam traversed before it hits an object. Additionally, the decay rate mapping requires the length the ray is running through each cell. We therefor propose a new algorithm for tracing rays through a 3D grid map, which casts rays from the sensor pose to the measurement endpoint. Given a start and end position in 3D our algorithm first computes the grid cell the ray starts in as well as the direction the ray travels. We repetitively check the intersection of the line representing the ray with the next three planes in ray direction representing the cell borders. The calculation of the intersection between the line and the planes is carried out using barycentric parametrization of the line with a normalized direction vector. This allows, to use the barycentric parameter γ as distance between the line start point and the next border. We then take each plane with minimal distance on the line and in- or decrease the cell index in this dimension. Multiple planes with minimal distance are possible if the line runs through a cell edge or corner. Additionally to the indices of all traversed cells the algorithm stores the minimal distance as the distance the ray travels through a cell. The sum of this values per cell is employed as radius R in the decay rate mapping. The ray tracing algorithm is shown in Algorithm 1 in more detail.

Algorithm 1 Our Ray Tracing Algorithm

```
 1: procedure TRACERAY(Start ∈ $\mathbb{R}^3$, End ∈ $\mathbb{R}^3$, cellSize ∈ $\mathbb{R}$)
 2:     cells ← {}                                  // (cell ∈ $\mathbb{N}^3$, radius ∈ $\mathbb{R}$)
 3:     line ← End − Start
 4:     totalLength ← $norm$(line)
 5:     direction ← $normalized$(line)
 6:     (x, y, z) ← Start
 7:     cellIndex ← $getIndex$(Start)          // cell index from coordinates in meter
 8:     nextBorder.x ← cellSize · (cellIndex.x + (($sign$(direction.x) == 1)?1 : 0))
 9:     nextBorder.y ← cellSize · (cellIndex.y + (($sign$(direction.y) == 1)?1 : 0))
10:     nextBorder.z ← cellSize · (cellIndex.z + (($sign$(direction.z) == 1)?1 : 0))
11:     lengthRunning ← 0
12:     do
            // calculate the distance to the intersections with the cell borders
13:         $\gamma_x$ ← (nextBorder.x − x)/direction.x
14:         $\gamma_y$ ← (nextBorder.y − y)/direction.y
15:         $\gamma_z$ ← (nextBorder.z − z)/direction.z
16:         minima ← $min(\gamma_x, \gamma_y, \gamma_z)$                  // find minima
17:         cells.insert(cellIndex, minima.value)
18:         for $m \in \{x, y, z\}$ in minima do
19:             nextBorder.m ← nextBorder.m + cellSize · $sign$(direction.m)
20:             cellIndex.m ← cellIndex.m + 1 · $sign$(direction.m)
21:         (x, y, z) ← (x, y, z) + minima.value · direction
22:         lengthRunning ← lengthRunning + minima.value
23:     while (lengthRunning < totalLength)
24:     return cells
```

7.3.3.2 *Dynamic Object Type*

To estimate the distribution of *dynamic object types* in the part of the environment represented by a cell, we employ the histogram measurements. After splitting the stream of measurements, using the computed *break points*, we assume that all measurements are created by the same, unchanged environment. We compute the *dynamic object typ* histogram given the last *break point* b by taking the mean of all measurements:

$$\mathbf{Z} = \frac{1}{N - b + 1} \sum_{t=b}^{N} z^t \tag{7.10}$$

7.3.3.3 Reflectance Probability Estimation

We employ three different methods to estimate the reflectance probability of a cell. A comparison of the resulting estimations can be found in the experimental evaluation in Section 7.4.5.3. The three reflectance estimation methods are described in the following section.

Reflectance Mapping
The reflectance mapping counts the hits and misses per cell using two variables α and β. To estimate the reflectance probability the method uses the ratio:

$$m = \frac{\alpha}{\alpha + \beta}. \tag{7.11}$$

The reflectance mapping is described in more detail in Section 6.3.

Decay Rate Mapping
The decay rate model was proposed by Luft *et al.* [45]. For this model the number of hits H is used as in the reflectance mapping. Instead of the number of misses the sum of distances which beams traveled through a cell without ending in it is computed: R. The longer the distance a ray travels through a cell, the higher the probability that this cell is free. Using these values the decay rate is defined as:

$$\lambda = \frac{H}{R}. \tag{7.12}$$

To estimate an reflectance value for a cell the decay rate is transformed using the exponential function:

$$m = 1 - \mathrm{e}^{-\lambda/c}, \tag{7.13}$$

where c is the cell size in meter.

Histogram-based
We also employ the distribution of *dynamic object types* to estimate the reflectance probability. We estimate in the fourth histogram value the probability $p(\text{miss} \mid z)$ that the measurements collected in a cell represent a miss. The reflectance probability is then estimated using the combined measurements after the last *break point* as follows:

$$m = 1 - p(\text{miss} \mid z). \tag{7.14}$$

7.4 Experimental Evaluation

The proposed mapping approach consists of three steps. The first step in our algorithm is the estimation of *dynamic object probabilities* for each point in the input scans. Afterward, we detect *break points* in the data to account for changes of the environment due to dynamic objects. The last step is the generation of a 3D grid map using the detected *dynamic object probabilities* and *break points*. In our experiments, we evaluate all three parts of the mapping algorithm. The first experiment evaluates the estimation of the *dynamic object probabilities* using our neural network. After this we test the detection of *break points* in a stream of scans and compare our two *break point* detection methods. We then compare the two methods to combine measurements. The last experiment evaluates the resulting maps. Throughout these experiments we employ the KITTI tracking dataset.

7.4.1 Using the KITTI Tracking Dataset

The KITTI tracking dataset, created by Geiger *et al.* [22], provides Velodyne scans, recorded in urban areas, with a bounding-box-based labeling of dynamic objects. Other than the KITTI object dataset (see Section 5.4.1), were only single scans are provided, this dataset consists of sequences of scans and their poses which allows to distinguish moving and movable objects. In our work we learn a method to predict point-wise the probabilities that measurements were caused by objects of one of multiple *dynamic object types*. To use the KITTI dataset in our work, we first transfer the labels of the bounding-boxes onto the respective 3D scan points by projecting all point into the bounding-box frame. For each point inside a bounding-box, we transfer the label, while everything outside the bounding-boxes is assumed to be static. As only the view of the color camera is labeled, we mask everything outside this view as unknown. To evaluate our network we cannot use the test set provided by the KITTI tracking dataset, as there are no ground truth bounding-boxes provided, which we need for a point-wise labeling. Instead, we split the KITTI training dataset into a test and training set. The KITTI tracking training dataset consists of 21 sequences with 78 to 1059 scans each. We used the first 11 sequences for training and the rest for testing. To generate training data for our network we need ground truth labels which distinguish between movable and moving objects. Dewan *et al.* [14] manually divided the bounding-boxes provided for the KITTI tracking dataset into moving and movable boxes. We use this distinction to generate a point-wise labeling of the dataset into *static, movable* and *moving* 3D points.

7.4.2 Dynamic Detection

The first experiment evaluates the estimation of our *dynamic object probabilities*. We use a neural network to predict for each 3D point in a laser scan the probabilities that it belongs to our *dynamic object types*. In this work three *dynamic object types*

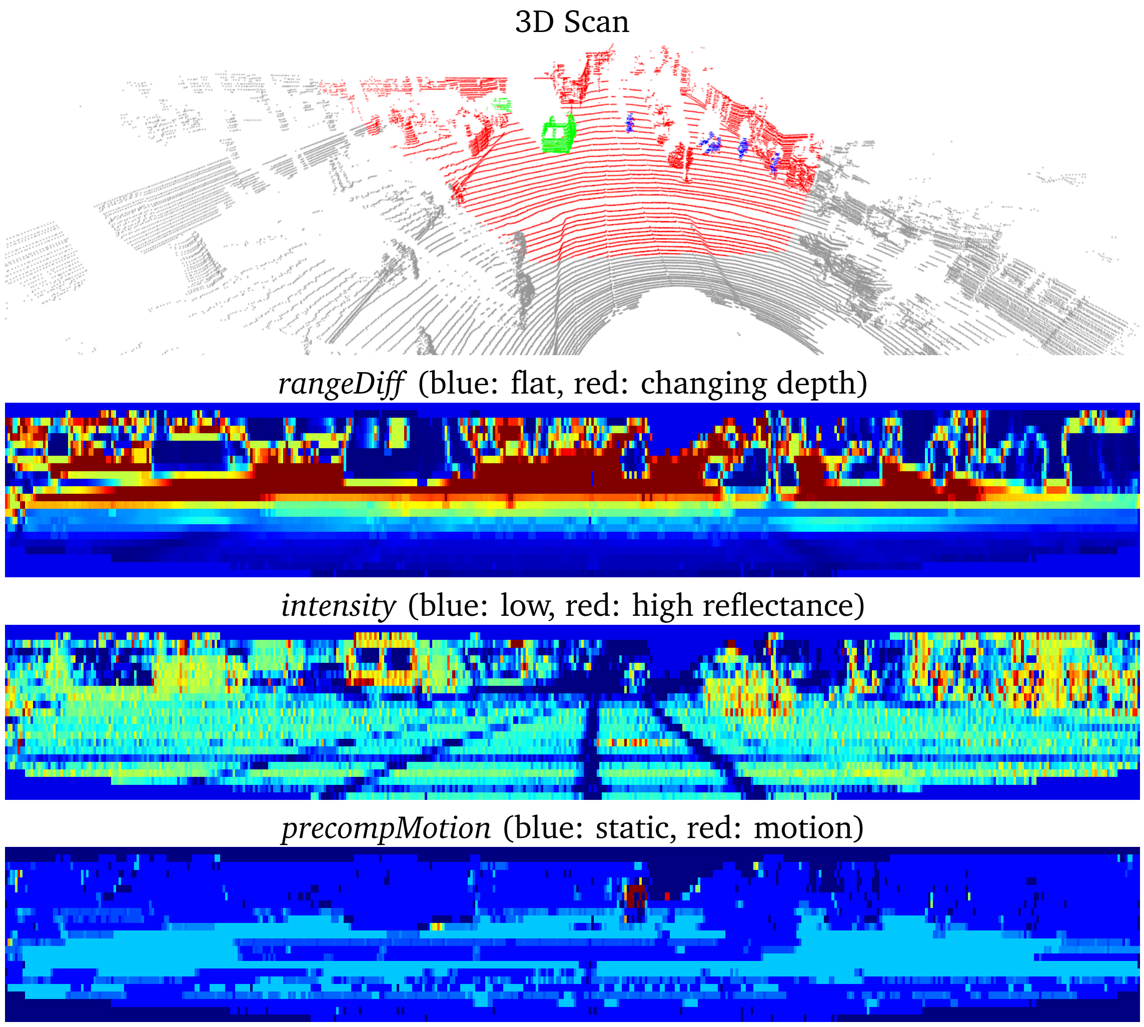


Figure 7.2: The first image shows a 3D scan where the colors visualize the ground truth label, green for two *movable* cars, blue shows *moving* pedestrians and a bike and red all static points. Grey points are outside the labeled camera view. The next three images show the computed modalities (scaled by three in height for better visualization). A van can be found left of the center in the images. The *precompMotion* shows high values for the cyclist on the road (center of the image).

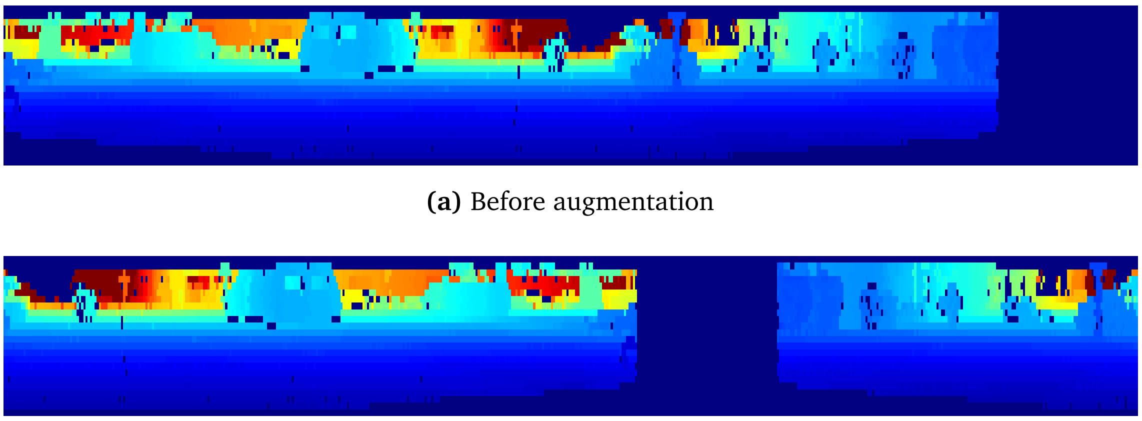
are used: *static*, *movable* and *moving*. To distinguish *movable* and *moving* points we use precomputed motion estimates. We apply the difference-based segmentation method, used as baseline in Chapter 4, to estimate a prior for moving scan parts. The method takes a stream of 3D scans with known poses as input and compares the most recent scan with the last scan and keyframes selected from previous scans in the sequence. By comparing these scans on a range image basis, it estimates motion priors depending on which points are present in the scans. The result is a motion estimate for each point in the most recent scan where moving objects are marked as dynamic. Additionally small objects often generate a perceived motion, due to the unsuccessful matching. Whenever objects move slow or to far away from the sensor the method is not able to detect motion. We project the motion prior from this method into an image and use it as *precompMotion* modality. For all scans in our training dataset, we generate training data by combining this modality with the modalities *rangeDiff* and *intensity* as described in Section 5.3.1. An example 3D scan together with the three computed modalities is shown in Figure 7.2.

To reduce overfitting and to increase our training set, we augment the data in two steps. The first step generates for each scan one image without and three with augmentation by adding an offset to the sensor pose during the range image generation. This offset generates small perspective changes. The second step augments each resulting image multiple times by sampling image augmentations and their parameters. We employ two image augmentations. The first augmentation samples whether or not the image gets flipped horizontally. The second augmentation translate the image horizontally by an offset between 1 and the image width thereby computing the new horizontal index for each pixel modulo the image width. This leads to the fact, that everything that falls out one side of the image is added on the other side. An example of our augmentation is shown in Figure 7.3. This augmentation simulates a rotation of the robot and eliminates potential place dependent priors during training such as that cars appear mostly in front of the robot.

We train our network as in Chapter 5 based on a neural network but instead of three, we use four classes: 0 = *static*, 1 = *movable*, 2 = *moving*, 3 = *ignore*. The resulting prediction of the network is a pixel-wise histogram of three *dynamic object probabilities* on the 2D representation of the scan. This prediction is then projected onto the 3D points of the laser scan. An example of the estimated probabilities is shown in Figure 7.4.

The KITTI scans predominantly consist of static areas. To counter this miss-balance in the training data, we evaluate the weighting of the loss of the network with the reciprocal class frequency during training.

To test the quality of our prediction, we classify all our test data by taking for each pixel the class of the maximum estimated probability as classification and compare it against the ground truth. We compute a class-wise IoU score (see Section 2.5.4) which is shown in Table 7.1. The estimated probabilities work best for the class *static* as the majority of all labeled points is generated by static objects such as roads and buildings. The augmentation therefore mainly boosts the performance of the



(a) Before augmentation

(b) After augmentation

Figure 7.3: An example of the used augmentations on a *rangeDiff* image. The image gets shifted to the right while each index is computed modulo the image width. The image is also horizontally flipped. For better visualization the height is scaled by three. The image is wider than the content as the image size is fixed to 2^k pixels.

	intersection over union			
training data	**static**	**movable**	**moving**	**mean**
without augmentation	0.962	0.532	0.413	0.636
with augmentation	0.967	0.623	0.479	0.690
with augmentation and weighting	0.967	0.636	0.492	0.698

Table 7.1: Prediction quality of our estimated *dynamic object probabilities* with and without augmentation of the training data. We also show the result for the weighting of the loss function.

two dynamic classes. The weighting of the loss function increases the result further. We also computed a confusion matrix of the classification on the KITTI sequence 0011 which is shown in Figure 7.5. While the prediction works well for *static* points *movable* and *moving* points are sometimes confused for static objects, mostly on borders of objects. The *moving* objects are additionally sometimes confused for *movable* objects especially when the motion on slow moving or distant objects is not properly detected. Throughout the rest of this chapter, we employ the network trained using augmented training data and weighting of the loss function, as this combination results in the highest IoU score.

7.4.3 *Break Point* Detection

The mapping algorithm presented in the previous chapter builds maps consisting of the static parts of the environment while treating dynamic measurements as a miss. In this chapter, we include dynamic objects into the map. We are interested

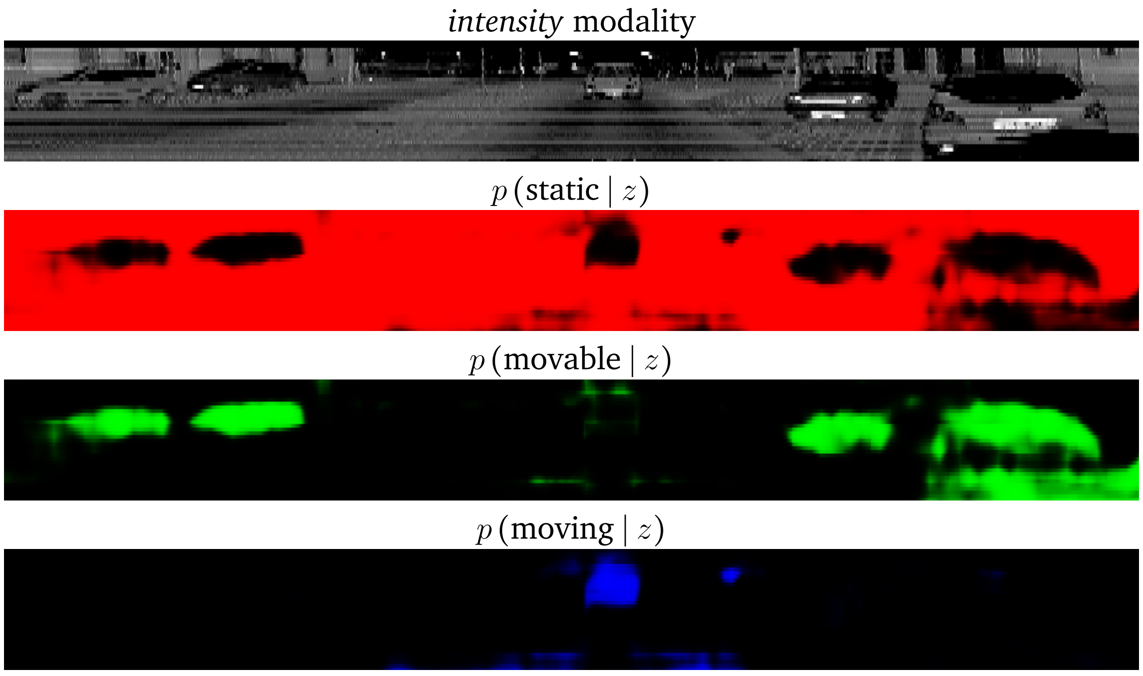


Figure 7.4: *Dynamic object probability* estimation result together with the corresponding intensity image. The cars as well as the pedestrian can be spotted in the images. Black values represent low while colors show high probabilities.

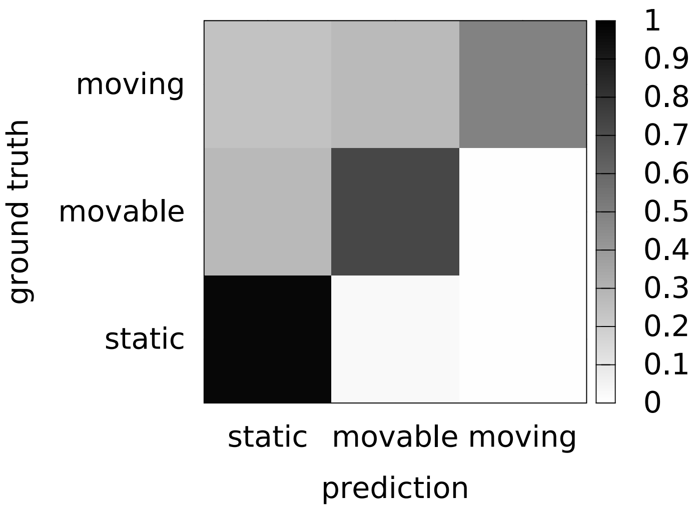


Figure 7.5: Confusion matrix of the *dynamic object probabilities* estimated on KITTI sequence 0011. The confusion happens mostly on borders and when the motion of slow moving objects is not correctly represented in the *precompMotion* modality.

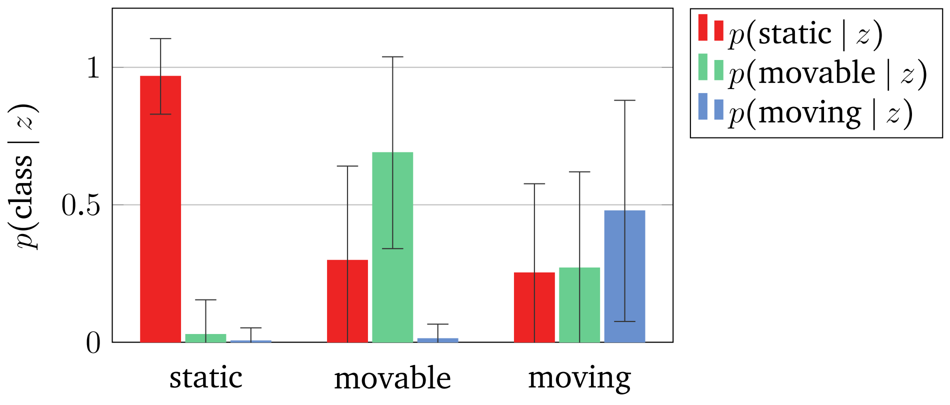


Figure 7.6: Mean and standard deviation of our three sets of measurements used for *break point* detection. The first three bars represent measurements with a *static* ground truth label, the second and third *movable* and *moving*. Each color represents the estimated *dynamic object probability* for one of the three *dynamic object types*.

in the most recent map created from consistent measurements. Therefor our approach needs to detect changes in the environment and take these into account while mapping. The following experiment evaluates our two proposed methods to detect such *break points* in a stream of measurements annotated with the estimated *dynamic object probabilities*. The second part of this section shows how the parameter P_0 of the probabilistic method can be learned.

To evaluate the *break point* detection, we first sample measurements from our annotated KITTI scans from sequence 0011 and divide them into three sets each with $10{,}000$ measurements depending on their ground truth label L. Each measurement consists of three estimated *dynamic object probabilities* which sum up to 1: $p\,(\text{static} \mid z)$, $p\,(\text{movable} \mid z)$ and $p\,(\text{moving} \mid z)$.

Figure 7.6 shows the mean and standard deviation of the three *dynamic object probabilities* for the three sets of measurements used in this experiment. The first three bars visualize the mean of all measurements with a ground truth label L of *static*. The colored bars show the probability that a beam was created by a static (red), movable (green) or moving (blue) object. We can see that for each label L the mean of the probability $p\,(L \mid z)$ is the highest. Using this data, we build an artificial measurement stream by sampling a random order of those measurements.

Both *break point* detection methods aim to generate a sequence of *break points* $b_1, \ldots, b_M$ such that for each $i = 1, \ldots, M$ all measurements $z_{b_{i-1}}, \ldots, z_{b_i}$ share the same ground truth label L. After we predicted all *break points* on the stream of data, we count how many of the true changes of L the methods detected, as well as the number of times they predicted a *break point* while the label L stayed the same.

Table 7.2a shows the result for the entropy-based method. The rows show the ground truth label L before the *break point* while the columns show the ground truth label L' after. The off-diagonal cells denote the ratio of correctly identified *break points* where the environment switches from L to L'. The main

		label after:		
		static	**movable**	**moving**
label before:	**static**	0.526	0.637	0.663
	movable	0.956	0.653	0.761
	moving	0.975	0.881	0.740

(a) All data

		label after:		
		static	**movable**	**moving**
label before:	**static**	0.490	0.885	0.847
	movable	0.994	0.474	0.935
	moving	0.999	0.990	0.505

(b) Filtered data

Table 7.2: *Break point* detection result for the entropy-based method. The columns represent the ground truth label L before, the rows after the *break point*. The off-diagonal cells represent the correctly detected *break point* ratio. The main diagonal shows the ratio of falsely detected *break points* where there are none present.

diagonal shows the ratio of falsely detected *break points* where no change in the environment occurred. A perfect result would have a diagonal of value 0 (white) while all off-diagonal elements have a value of 1 (black). To evaluate the *break point* detection methods independent of false classifications we repeated this experiment and only sampled correctly classified measurements where the maximum of all three *dynamic object probabilities* is equal to the ground truth label: $L = \operatorname{argmax}(p(\text{static} \mid z), p(\text{movable} \mid z), p(\text{moving} \mid z))$. The result is shown in Table 7.2b. We note that the result improves but the values on the main diagonal show that we still predict a number of false *break points*.

7.4.3.1 Learning the Minimum *Break Point* Probability P_0

The entropy-based method is a parameter-free approach, which returns a *break point* at the start of the stream if no *break point* is found. The probabilistic method on the other hand, uses the value P_0 to decide if a detected *break point* was valid. This value depends on the underlying data and must be estimated on the training data. To estimate P_0 we employ a gradient-based optimization technique. We first create an artificial measurement stream from a set of around $40{,}000$ measurements of each *dynamic object type* using the same method as above but scans sampled from the training set. The gradient-based method first samples a starting value for $P_0 \in [0,1]$. Using this value, we predict all *break points* in the stream and compute the result table $\mathbf{R}$ as shown in Table 7.2 for the entropy method. This table $\mathbf{R}$ is then used to evaluate a score s for this value of P_0:

$$\mathrm{s} = \frac{1}{9}\sum_{i,j}\mathbf{r}_{i,j}, \tag{7.15}$$

$$\mathbf{r}_{i,j} = \begin{cases} R_{i,j}, & i \neq j \\ 1 - R_{i,j}, & i = j, \end{cases} \tag{7.16}$$

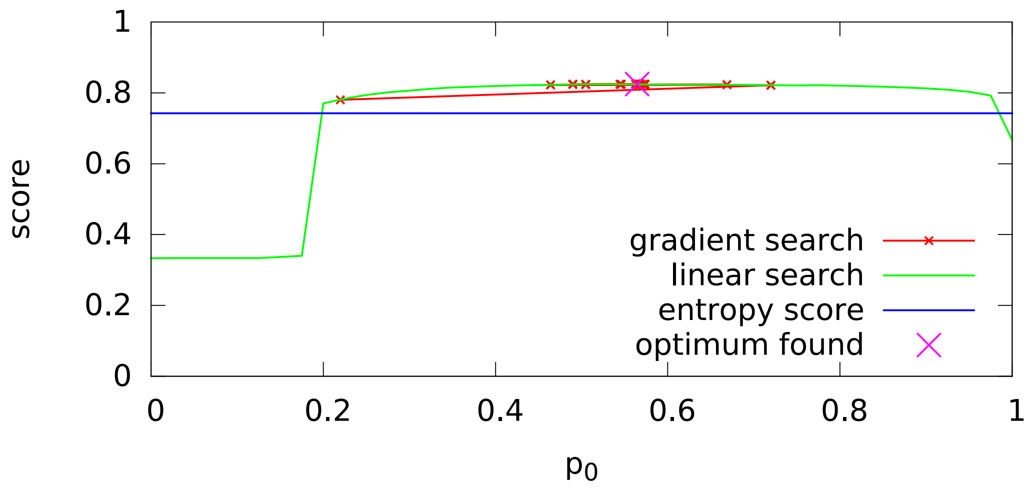
where $\mathbf{r}_{i,j}$ are the values in the result table and $\mathbf{r}_{i,i}$ are the diagonal elements representing the ratio of falsely detected *break points*. We then evaluate the gradient of the score function at P_0 by sampling a value below and above our current P_0 and compute the score values as above. Based on this gradient a new value of P_0 is selected. If both scores are lower than the score for P_0 we reduce the distance of the gradient evaluation points and repeat the process. If for 25 iterations the value of P_0 does not change this value is returned as the optimal value. The process was repeated 100 times with different starting values while keeping the value P_0 with the highest score. The search leading to the final value of P_0 as well as a linear evaluation of P_0 values is shown in Figure 7.7a. The score of the entropy-based method is visualized by a horizontal blue line. We found the best value $P_0 = 0.565$ with a score of 0.825. With increasing value of P_0 the score rises as the *break point* method rejects possible *break points* between measurements of the same class. Towards the end the score decreases as we reject almost all found *break points*. We additionally repeat this experiment filtering out falsely classified measurements. The resulting optimization is shown in Figure 7.7b. The best value found is $P_0 = 0.494$ with a score of 0.998. We note that the score for the filtered optimization is more peaked and results in higher values.

Using the learned values of P_0 we compute the result table using the same artificial measurement stream from sequence 0011 as for the gradient-based method: Table 7.3b. Applying the P_0 value we learned on the filtered training data to the filtered measurements from sequence 0011 we receive an almost optimal result. In the unfiltered case the method struggles to detect *break points* between movable measurements (see lower right cell of Table 7.3a). This is due to fact, that misclassified movable measurements lead to the situation where the measurement before and after the detected *break point* look different even though they share the same ground truth label.

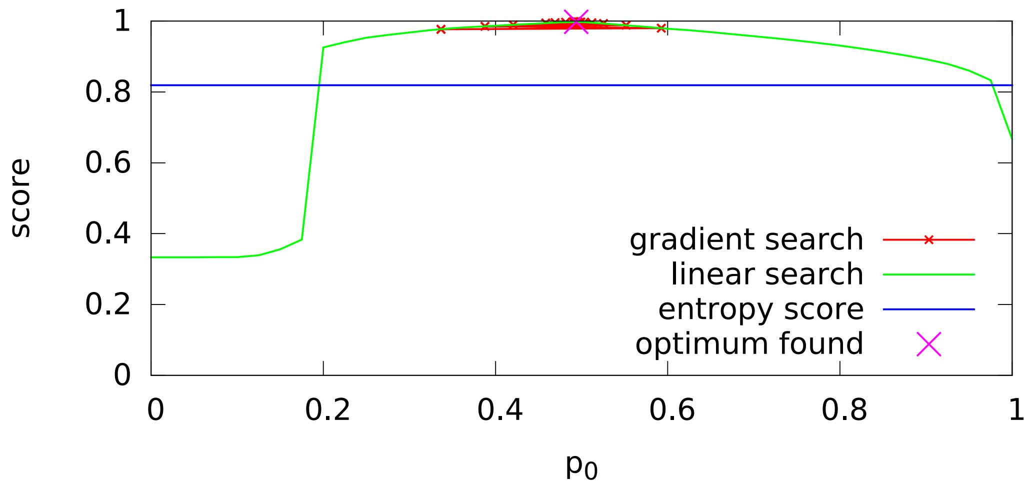
The probabilistic *break point* detection method employs a parameter which needs to be estimated on the training data. By employing a learned value, the resulting *break point* detection is superior to the result of the entropy-based method.

7.4.4 Combining Measurements

During the mapping process, we assume that all measurements coming from one scan occur at the same time. Therefore the *break point* methods only search for *break points* between scans. While building the map more than one measurement coming from one scan might fall into a cell. Due to the spacial extent of cells, these



(a) All data



(b) Filtered data

Figure 7.7: Gradient-based search for the optimal P_0 value. The red line visualizes the gradient-based search leading to the final value of P_0. The best found value and score is marked with a magenta colored cross. The green line shows a linear evaluation of P_0 scores and the horizontal blue line depicts the score for the entropy-based method. The upper plot shows the search sampling from all measurements while the lower shows the same with filtering of falsely classified measurements.

		label after:		
		static	**movable**	**moving**
label before:	**static**	0.058	0.740	0.793
	movable	0.735	0.485	0.807
	moving	0.792	0.802	0.725

(a) All data, $P_0 = 0.565$

		label after:		
		static	**movable**	**moving**
label before:	**static**	0.001	0.997	0.999
	movable	0.997	0.012	1.000
	moving	1.000	1.000	0.094

(b) Filtered data, $P_0 = 0.494$

Table 7.3: *Break point* detection result for the probabilistic method using the learned values for P_0. The columns represent the ground truth label L before, the rows after the *break point*. The off-diagonal cells represent the correctly detected *break point* ratio. The main diagonal shows the ratio of falsely detected *break points* where there are none present.

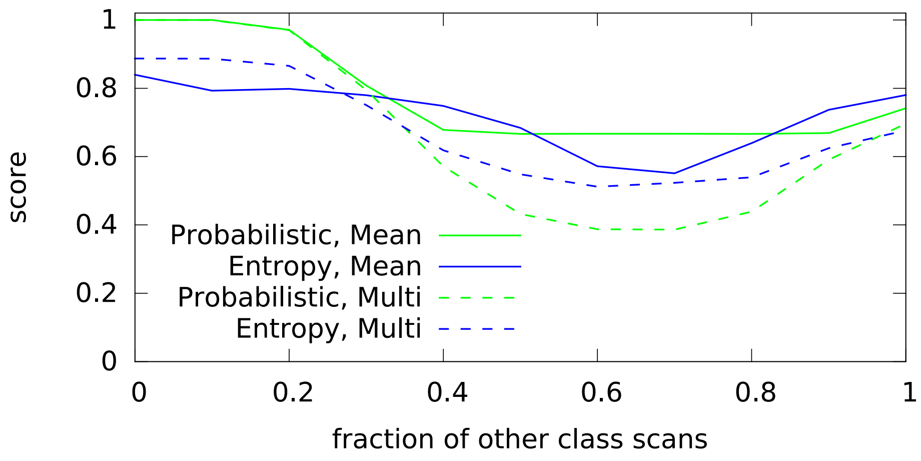
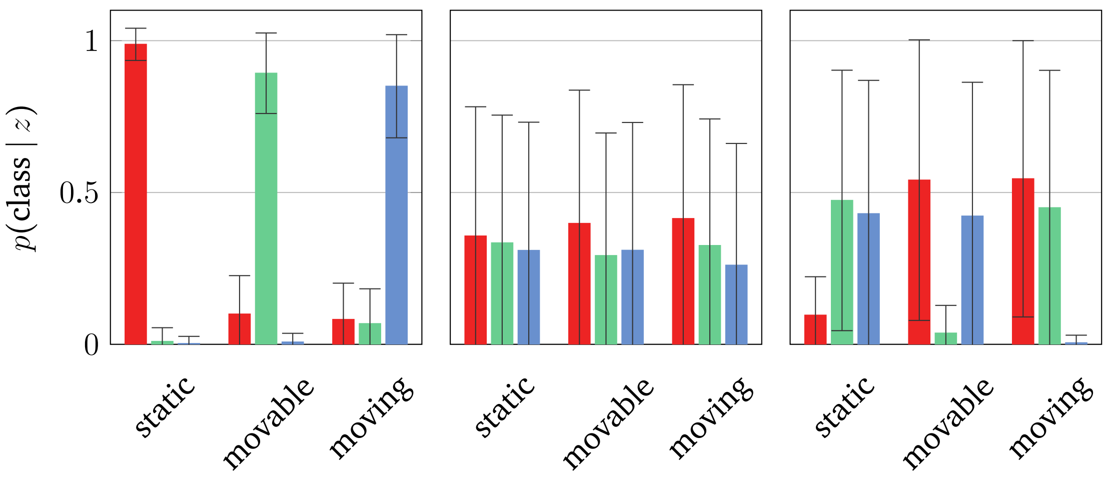


Figure 7.8: Evaluation of two methods to combine measurements and two *break point* detection methods. By adding an increasing number of measurements from different *dynamic object types* we can test the robustness of the combination methods.

measurements do not need to be created by an object of the same *dynamic object type*. In the following, we evaluate two methods to combine multiple measurements. To test how much measurements coming from an object of a different *dynamic object type* can be added to still get a stable *break point* detection, we use the score explained above (see Equation (7.15)). Instead of single measurements the measurements used here are a combination of multiple individual measurements. We compare two methods for measurement combination: the mean over the individual measurements as well as the element-wise product normalized to sum up to one (see Equation (7.4)). To test the robustness of the measurement combination methods we create a stream of combined measurements. Each of these consists of 10 individual measurements combined together. We start by sampling all measurements from the same ground truth class and replace more and more measurements with measurements from a different *dynamic object type*. The resulting scores are shown in Figure 7.8. As expected, the score for all methods drops the more measurements from different classes are added. This is due to the fact that the combined measurements look almost identical for all three classes for 70% of scans from other *dynamic object types* (see Figure 7.9b). In this case we sample almost uniformly from each *dynamic object type*. By adding more samples from other *dynamic object type* the measurements become discriminable again (see Figure 7.9c). As a result of this experiment we note that the mean over the measurement suffers less from noisy data than the normalized element-wise multiplication.



(a) 0% other scans **(b)** 70% other scans **(c)** 100% other scans

Figure 7.9: Mean of the class probabilities for 0%, 70% and 100% measurements sampled from the two other *dynamic object types*. While the histograms in (a) and (c) look different for each class and allow the detection of *break points*, the histograms in (b) look almost identical and do not allow a reliable *break point* detection.

7.4.5 Mapping

In this chapter, we propose a mapping algorithm which learns maps in dynamic environments representing static as well as dynamic parts of the environment. Each cell of the map estimates the reflectance probability as well as a histogram of probabilities that the objects represented by the cells are of one of three *dynamic object types*. The goal of our proposed method is to generate 3D grid maps which represent the most recent environment. To evaluate this, we compare in our first mapping experiment the most recent observation with the maps generated by our methods. As a simple solution to this task is to discard all data whenever a new scan arises we also evaluate how well our maps represents maps of all static observations.

We generate maps using a grid resolution of $20\,\mathrm{cm}$ and a max-range of $80\,\mathrm{m}$. We use a value of $P_0 = 0.477$. This value was learned by our gradient-based method proposed in Section 7.4.3.1 but including the *miss* class (see Section 7.3.3) and considering the values of ϵ_h, ϵ_m. For mapping we use the scan poses as provided by the KITTI dataset. In our experiments, we show how our algorithm deals with changing environments. We therefore removed the ground floor as it is not relevant for dynamics estimation.

An example of a map generated from sequence 0014 of the KITTI tracking dataset including ground measurements is shown in Figure 7.10. Each cell is colored by its most likely *dynamic object type*. The map represents a street with cars parked at both sides (green). On the opposite lane a moving car (blue) is driving by. The

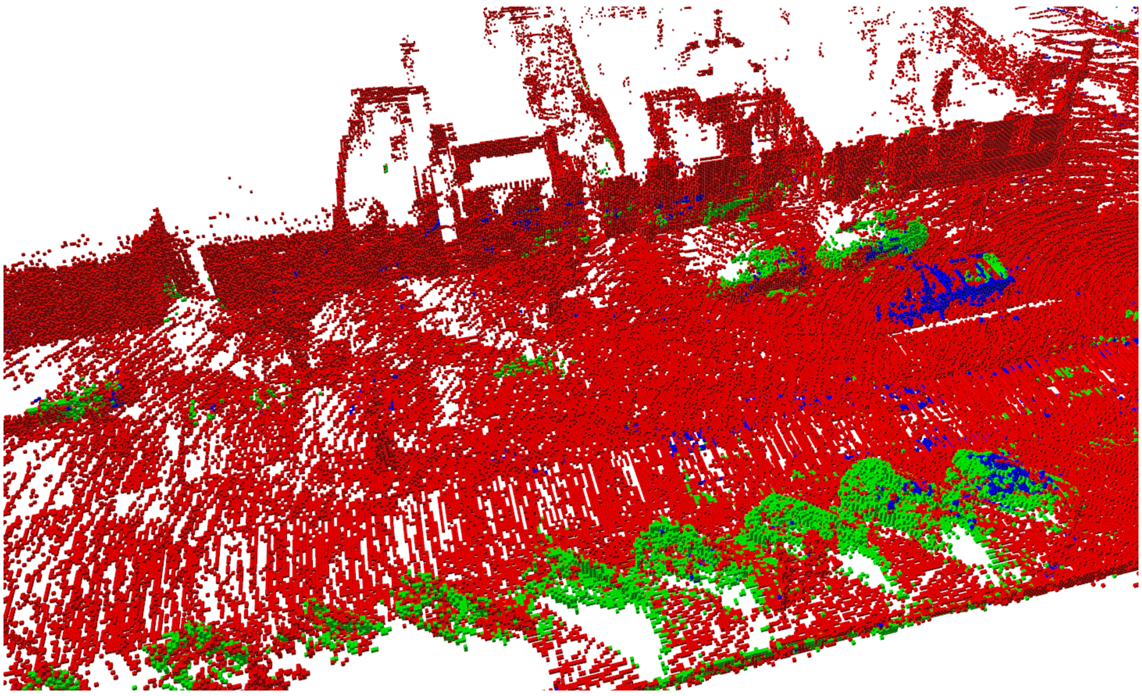


Figure 7.10: An example of a map generated by our proposed mapping algorithm on the KITTI tracking dataset sequence 0014. The map shows a road with cars parked on the sides which are detected as movable objects (green) and a moving car (blue).

static environment is shown in red. We note that due to the *break point* detection the space behind the car is as intended not occupied.

Figure 7.11 visualizes the *break points* detected by our probabilistic *break point* detection method. The image shows a top down view of a map where a car is standing still on the right side while another moves from the left towards the sensor. The color shows the last detected *break point*, where blue refers to the last scan added to the map. Whenever the car moves for the cells behind it our algorithm detects a change from occupied to free and reports a *break point*. There are relatively few *break points* between scan index 3 and 8 (orange and yellow points) on the car as while the car moves through a cell no changes of the object type in the cell occur, thus our algorithm correctly detects no *break points*. Note that the image shows all cells with a reflectance value above 0 meaning that also cells with a low reflectance probability are shown to visualize the *break points*.

The method proposed in this chapter computes in each cell *break points* to decide which data consistently represents the same environment. In this experiment we compare our method against three other methods to decide which data to include during map estimation. We describe the methods we compare against in the following section.

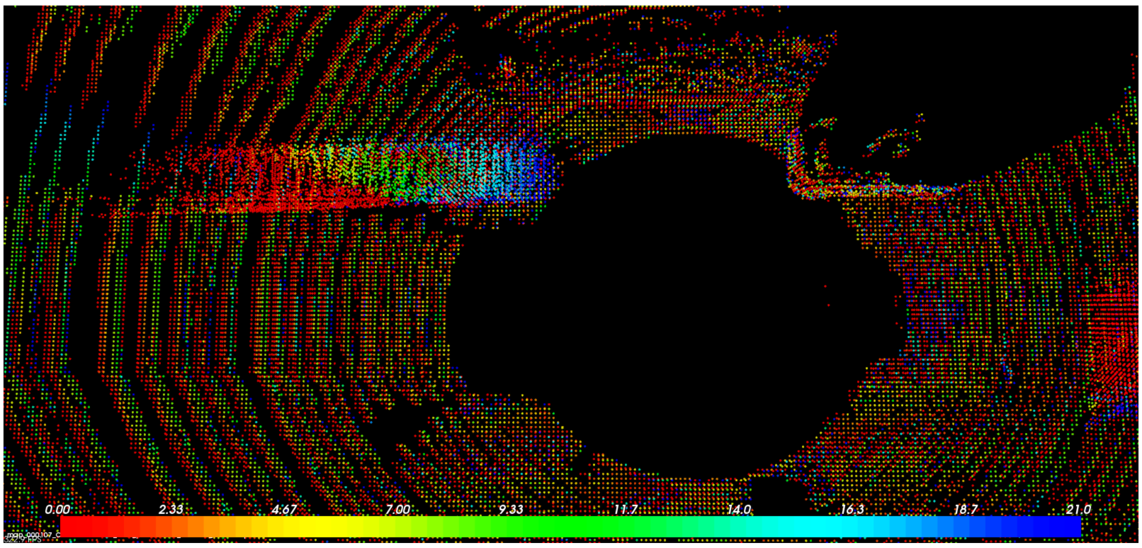


Figure 7.11: Top down view on a map showing the last detected *break point* by scan index where blue is the last added scan. The color gradient is the result of a moving car. The map shows all cells with a reflectance probability above zero.

All Data
The first method uses all data seen in a cell to estimate the reflectance and *dynamic object probabilities* independent of the consistency. The method thereby ignores that the environment changes during mapping due to dynamic objects.

Last Scan
We also compare against a method which only uses data which was created by the last integrated scan. This method is well suited to estimate the most recent observation but is limited to the sensor range.

Last Three Scans
By using the last three scans we might estimate the map values from inconsistent observations as dynamic objects are able to move in between these scans. On the other hand, more scans allow for a more robust estimate and a map larger than the sensor range.

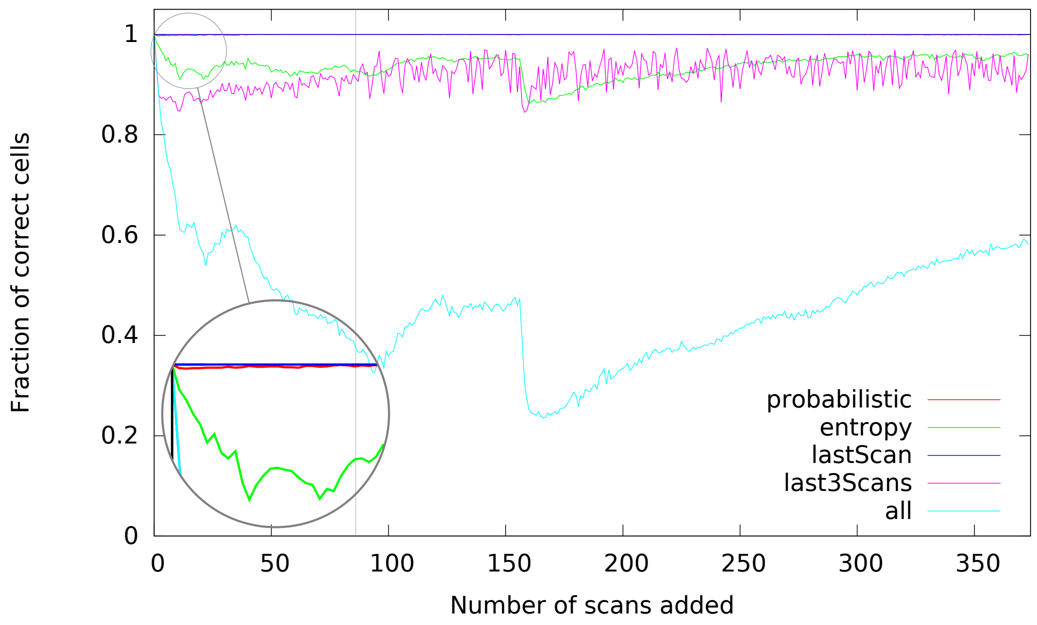
7.4.5.1 Actuality of the Map

In this experiment, we evaluate to what extend the maps learned by our mapping approach agree with the most recent observations. To be able to compare not only occupied cells but also free space, we generate maps from the most recent observation. First we use our proposed mapping algorithm to generate maps M_N using the first $N = 1, \ldots, 374$ scans of the KITTI tracking sequence 0015 and store the resulting 374 maps. We then define ground truth maps $\overline{M}_N$ using only the Nth scan of the sequence and a standard occupancy grid mapping approach. We then project the single scan maps $\overline{M}_N$ into the corresponding maps M_N which used all data up to scan N. For each cell of $\overline{M}_N$ we check if it is represented in M_N by a cell of the same binary occupancy by thresholding our map M_N at an reflectance probability of 0.5. Figure 7.12a shows the fraction of correctly represented occupied cells, while Figure 7.12b shows the same for the free cells. These figures show results using the mean to combine multiple observations from one scan as well as the histogram-based method for reflectance probability estimation as those perform the best.

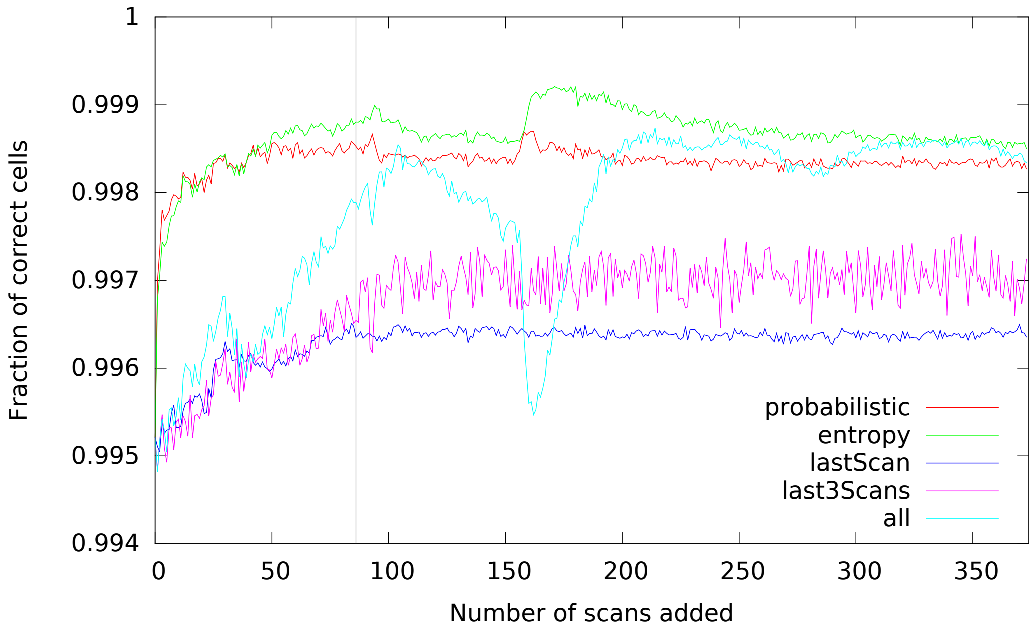
Thresholding a reflectance grid map at 0.5 leads to an occupancy grid map. Therefore our ground truth map is equal to the *last scan* method resulting in a perfect match. Our proposed methods *probabilistic* and *entropy* split whenever the measurements coming from a new scan disagree with the current estimate. This leads to a high agreement between the ground truth occupancy map and our maps. Especially the probabilistic splitting method with its parameter trained on the KITTI dataset reaches an almost perfect match. We further note that splitting the data at detected *break points* greatly reduces the amount of falsely classified cells in contrast to using all data seen during mapping. The sensor moves from the beginning of the recording up to scan 86, which is marked by a gray vertical line. After this stop the sensor stays static while cars, cyclists and pedestrians move by or cross the street. At around scan 160 the poses provided by the KITTI dataset report a movement of about half a meter while the static parts in the scans do not move. This leads to the fact that the map does not agree with old data. While our methods detect *break points* and reset inconsistent cells the map using all data needs multiple measurements to adjust. This highlights the usefulness of the *break point* detection. A comparison of the scans with and without the reported motion is shown in Figure 7.13. We note that the scans overlay by using the same pose, not the reported movement. This indicates that the reported motion was a mistake.

7.4.5.2 Representing the Static Environment

In the first mapping experiment, we evaluate to what extend the maps represent the most recent observation of the environment. In this experiment, we compare our maps against maps of the static measurements to test how well the maps estimate the environment over an extended time period. For this experiment, we employ the same sequence as in the above experiment, but only use the labeled measurements in the camera view. First, we generate 374 maps M_N for this sequence using our mapping algorithm as described above. Then, we compute occupancy grid maps using the same $N = 1, \ldots, 374$ scans of the sequence but neglecting all measurements with dynamic ground truth labels. By projecting these maps of the static environment into our estimated maps M_N, we count the correctly represented occupied cells. Comparing the free cells is does not provide any insight as correctly represented dynamic objects would count as falsely estimated free cells. The result is shown for the histogram-based reflectance probability estimation and the mean measurement combination method in Figure 7.14. As expected the *lastScan* and *last3Scans* methods can only estimate the map of the area seen by

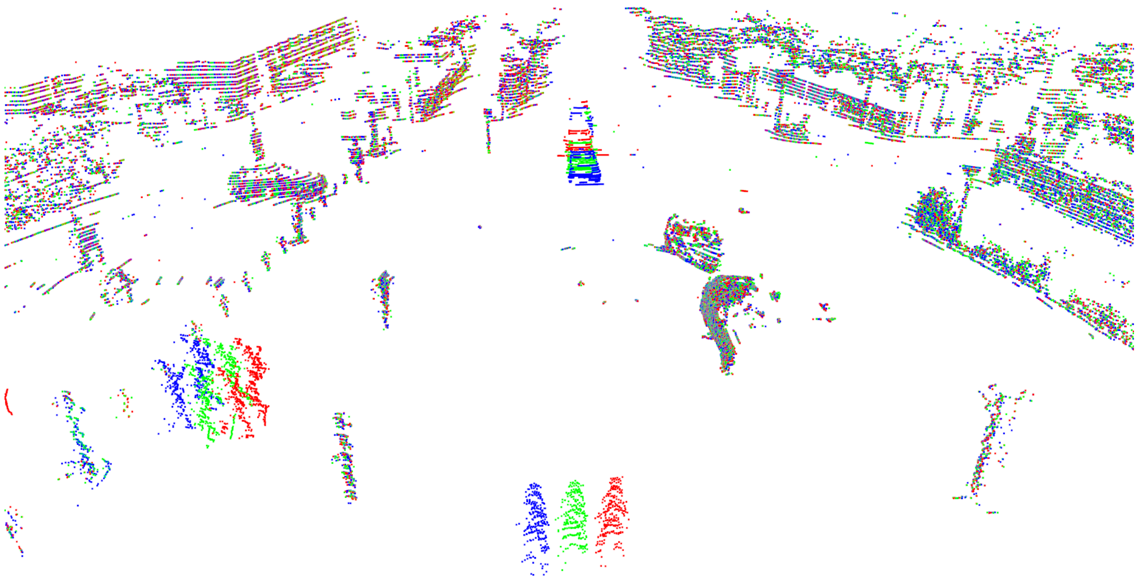


(a) Fraction of correctly represented occupied cells

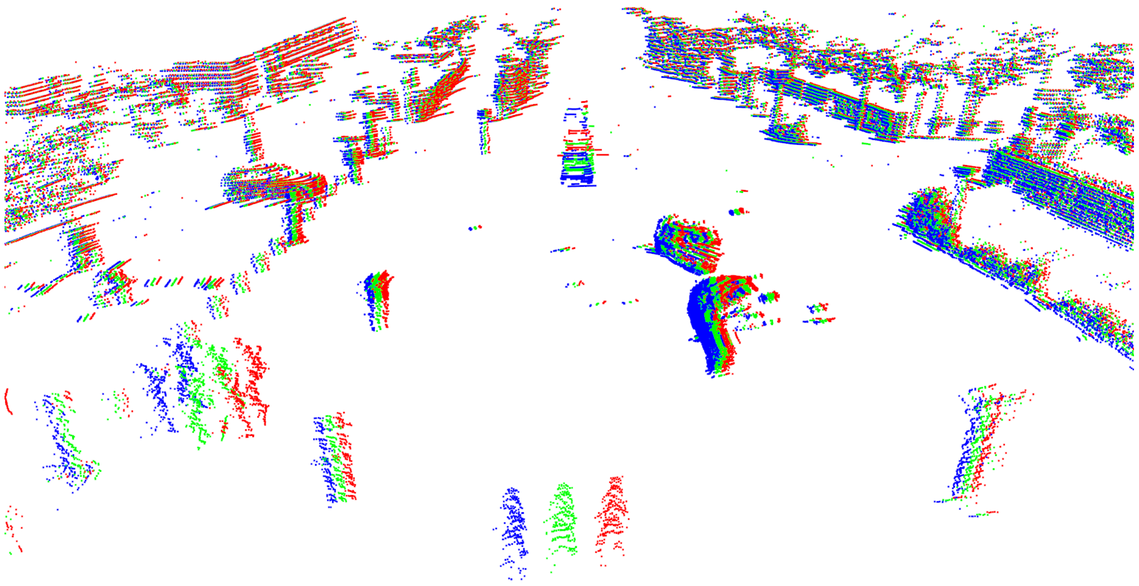


(b) Fraction of correctly represented free cells

Figure 7.12: Fraction of correctly classified cells for different *break point* detection methods. The sensor moves from the start of the sequence and stops after scan 86 while cars and pedestrians move by. The gray vertical line marks the time where the car with the sensor stops. In (a) the start is enlarged at the bottom left to show the probabilistic performance curve. Not the range of the y-axis in Figure (b).



(a) Scans overlayed with the same pose



(b) Scans overlayed with the reported poses

Figure 7.13: Overlay of scans before (red), during (green) and after (blue) the falsely reported motion of about half a meter. Figure (a) shows three scans overlaid using the same pose for all three scans. The scans are each shown in one color. We can clearly recognize pedestrians moving in the front and a moving car in the back. Figure (b) displays the scans with the pose reported by the KITTI dataset. We note that now additionally static and movable parts are displaced.

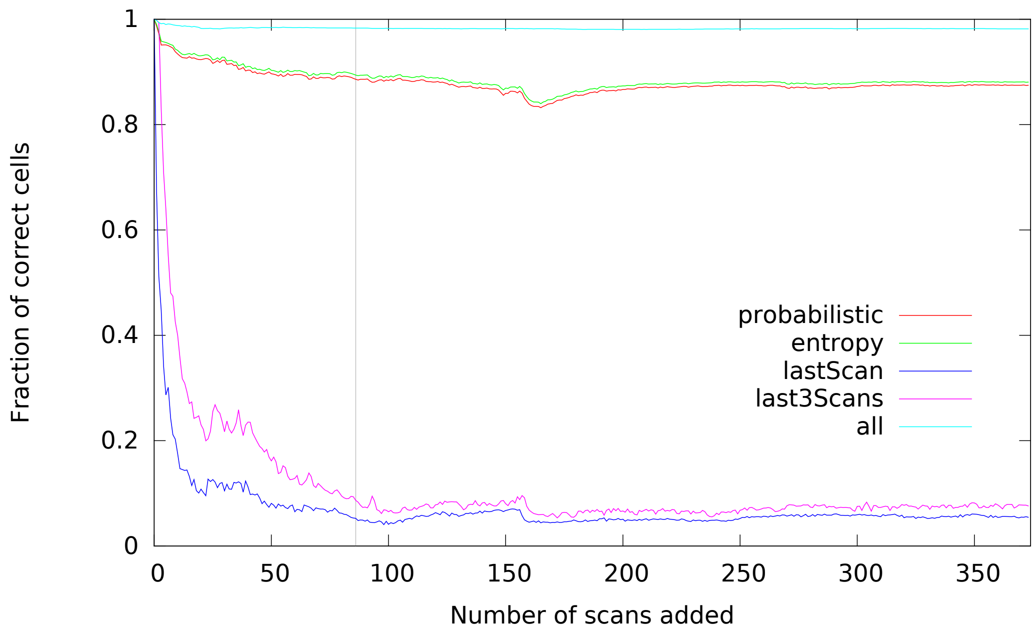


Figure 7.14: The fraction of correctly classified occupied cells for different *break point* detection methods. We compare our maps against maps representing only the static observations. The sensor moves from the start of the sequence and stops at scan 86 while cars and pedestrians move by. The methods using only a fixed number of scans cannot represent the whole environment.

the last one or three scans thus resulting in low scores. The method using all data produces almost no difference as it uses the same data as the occupancy grid map, but with the histogram-based reflectance probability estimation. Our methods perform well as they use all consistent measurements after the last *break point*.

The two presented mapping experiments demonstrate that our proposed algorithm is well suited to build maps in dynamic environments. By detecting *break points* in the measurement stream of each cell, the map is estimated from all consistent observations after the last *break point*. Whenever measurements do not agree with the current estimate, either through changing environment our due to an erroneous scan poses, our method resets the corresponding cells. The resulting maps represent the most recent environment by estimating the map from all consistent measurements after the last *break point*.

7.4.5.3 Reflectance Probability Methods

In the experiments above we only showed results for the histogram-based reflectance estimation method. The following experiment compares the three different methods proposed to estimate the reflectance probability of a cell. We therefor show Figure 7.12a again but only for the probabilistic *break point* detection method now showing the result of all three reflectance estimation methods: Figure 7.15.

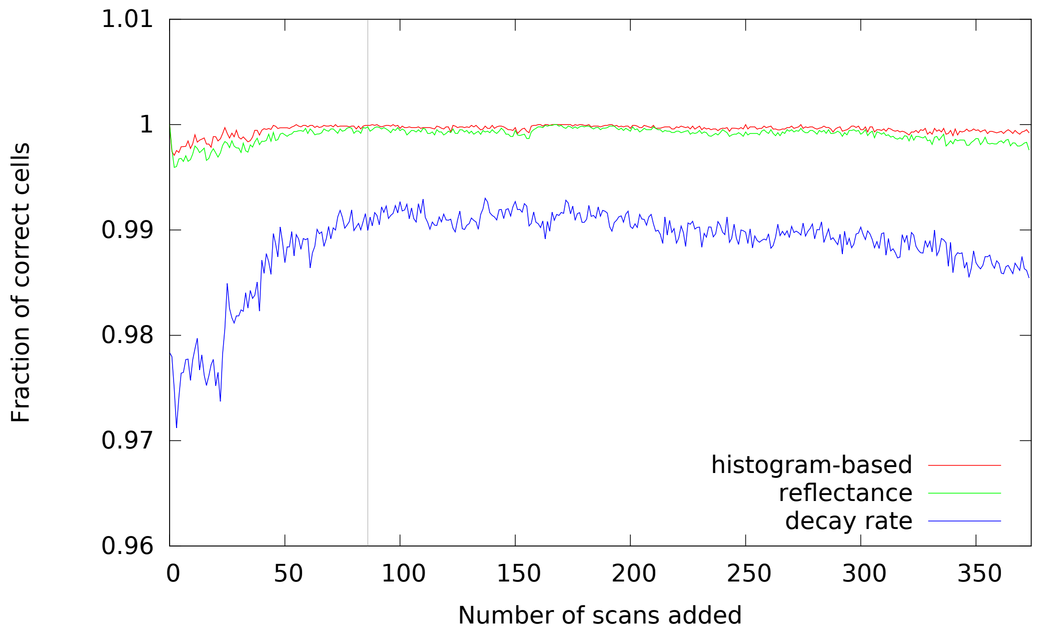
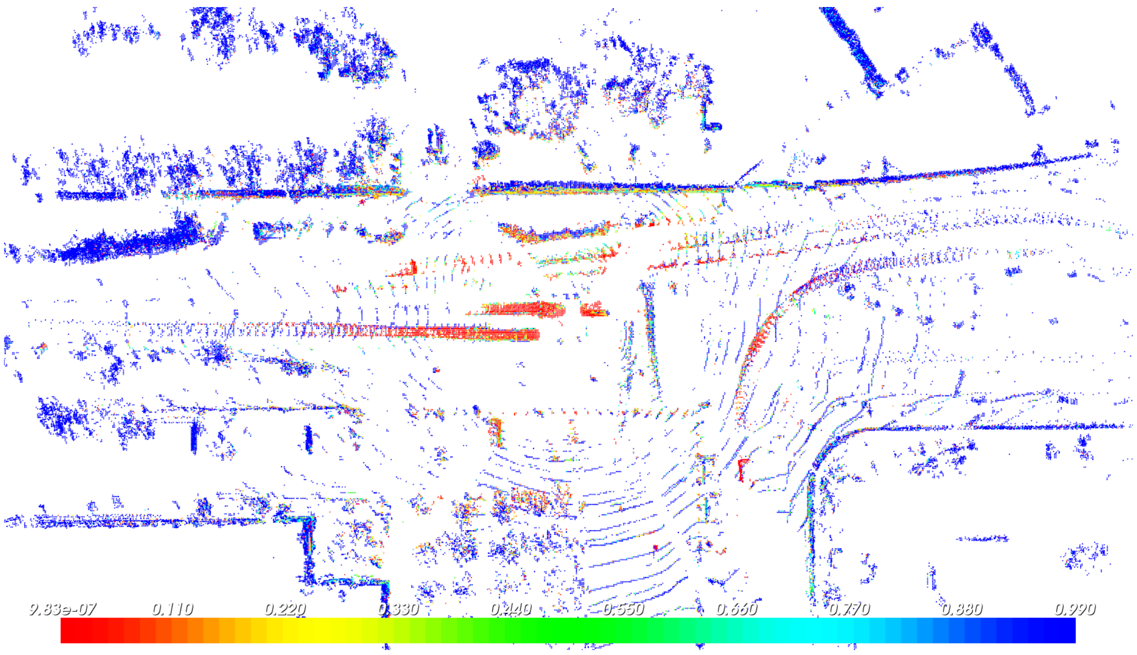


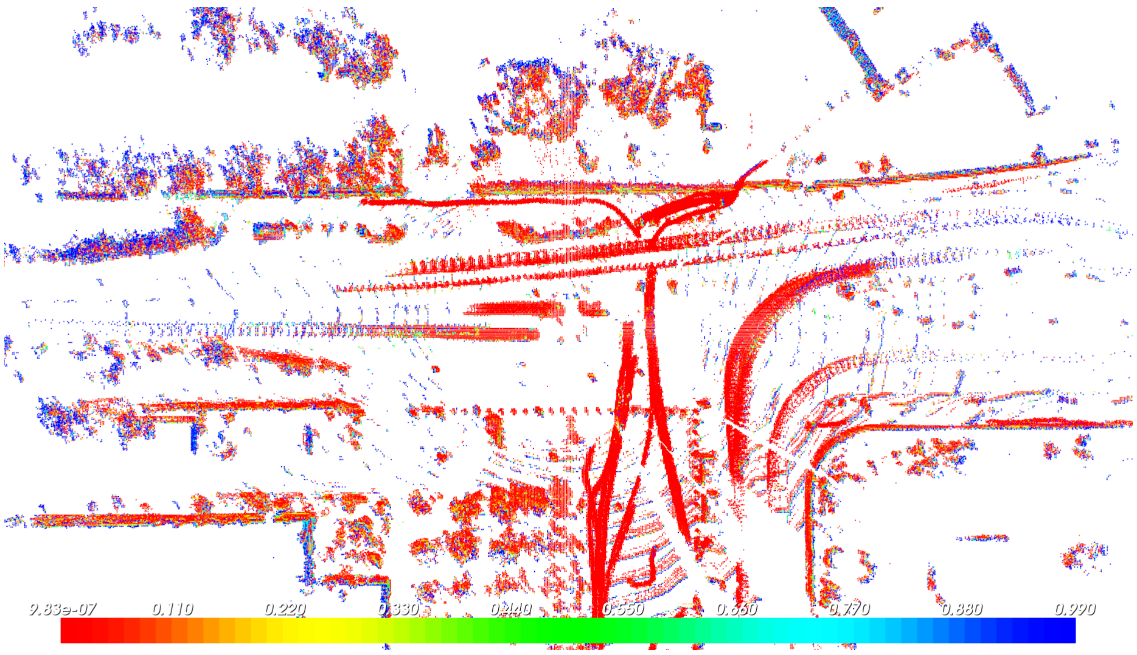
Figure 7.15: The fraction of correctly classified occupied cells for different reflection probability estimation methods. The plot shows the result of the probabilistic *break point* detection and the mean combination method. Not the range of the y-axis.

We note, that all three methods show a similar curve but with slightly different values. The *reflection* and *probabilistic* method are closer to the occupancy grid mapping as they likewise use hits and misses for the reflectance probability estimation. The decay rate model employs the length a beam travels through a cell to weight how important an observation is for the estimation. Therefore the reflectance value estimated by this method does not coincide this close with the occupancy grid mapping method. The decay rate model is not suited to reproduce an binary occupancy grid but performs well on other mapping tasks (see Schaefer *et al.* [65]).

The result of our mapping approach on the KITTI tracking sequence 0015 is shown in Figure 7.16. The map cells are colored by the histogram-based estimation that these cells represent static observations. The resulting map of the probabilistic splitting shows in red (not static) the paths of the moving objects as we show all estimated cells with an occupancy probability above zero. While the entropy splitting method map only estimates the static probability for occupied cells, the probabilistic splitting method map also keeps this information for free cells.



(a) Entropy-based splitting



(b) Probablistic splitting

Figure 7.16: Top down view on the resulting maps using the entropy-based (a) and the probabilistic (b) *break point* detection method. The color encodes the estimated probability that this cell represents a static object. The objects which moved trough the scans can still be seen as red cells, especially in (b).

7.5 Conclusion

In this chapter, we presented a method to build 3D grid maps in dynamic environments. The map represents static as well as dynamic objects. Each map cell estimates the reflectance probability as well as a distribution of *dynamic object types* of objects which occupy the cell. In our implementation, we distinguish three *dynamic object types*: *static*, *movable* and *moving*, but these can be easily and arbitrarily extended. To learn our maps, we first feed range and intensity images as well as precomputed motion estimates into a neural network to predict for each pixel a histogram of *dynamic object probabilities*. These are then used to compute in each cell the reflectance probability and the distribution of *dynamic objects types*. During the mapping process, we keep track of changes of the dynamic environment and reset cells whenever the occupancy or the *dynamic object type* of a cell changes. In extensive experiments, we evaluated all parts of our mapping pipeline and showed that the resulting maps reliably represents the most recent state of the dynamic environment.

CHAPTER 8

Conclusion

In this thesis, we presented several methods for localization and mapping in dynamic environments. Furthermore, we introduced approaches for the detection of dynamic observations in 3D laser scans.

Many tasks carried out by mobile robots require that the robot knows its pose in the environment. The ability to plan paths towards a goal becomes useless if the robot's pose is not known. Also manipulation can not carried out if the robot is not able to find the object it should handle. In Chapter 3, we presented a method to localize a robot with respect to a graph-based road network as provided by OpenStreetMap. We first classified the 3D LiDAR scans into road and non-road cells. By comparing this classification with the road network, we guided a Monte Carlo Localization. The proposed approach allowed the robot to localize itself in previously unseen outdoor environments. By detecting roads in the LiDAR scans we allowed the robot to leave the road network and to drive even off road. In extensive experiments, we demonstrated that we are able to localize the robot in urban and suburban environments using simulated as well as real-world data. Additionally, we showed that, by localizing the robot relative to a graph-based road network, we were able to plan paths on the graph which can be used by the robot's local planer for navigation.

Mobile robots in dynamic environments, such as urban streets or pedestrian areas, need to be aware of which parts of their observations are generated by dynamic objects like pedestrians or cars. The detection of dynamics allows the robot to avoid obstacles during navigation and enables to build reliable maps of their environment. To detect dynamics in a single 3D LiDAR scan, we proposed two different methods. The first method (Chapter 4) trains a random decision forest using handcrafted features computed on the local point neighborhood. The second method (Chapter 5) is an end-to-end deep learning approach which takes range image-based representations of the scans as input. Both methods estimate the probability that 3D scan points are caused by dynamic objects. We demonstrated that both methods were able to detect moving objects such as moving cars or walking pedestrians as well as movable objects like parked cars. During comparison of the dynamic detection methods, we found that the deep learning approach is superior in detection quality and runtime during prediction.

Localization on a publicly available graph-based road network allows the robot to localize itself in outdoor environments. However the map only represent the road network. Volumetric maps on the other hand allow reasoning about free and occupied parts of the environment. Using the probability that measurements were caused by dynamic objects we proposed in Chapter 6 a method to learn a map of the static parts of a dynamic environment. As these maps only represent static observations the map is not disturbed by dynamic objects and is valid for extended time periods. In our experiments, we showed that our mapping algorithm is able to build maps excluding dynamic objects, moving as well as movable.

Using a map which represents dynamic objects as misses is well suited for localization but leads to suboptimal paths during path planning due to unforeseen dynamic obstacles. By representing dynamic objects in the map these can be taken into account. In Chapter 7, we presented an extension of the mapping algorithm above to include dynamic observations in our map. We extended our dynamic detection to distinguish static, movable and moving objects. By including dynamic objects into the map, we need to keep track which parts of the environment change. We detected these changes and integrated this knowledge into the mapping algorithm. During the evaluation, we showed that the maps learned by our mapping approach are well suited to represent the most recent environment.

The key contributions of this thesis are:

- a localization algorithm on OpenStreetMap data
- methods to predict the probability that 3D LiDAR measurements are caused by dynamic objects
- a gird mapping approach to learn maps of the static parts of a dynamic environment
- a method to estimate probabilities that 3D LiDAR points are caused by objects of one of multiple *dynamic object types* (e.g. static, movable, moving)
- a mapping approach which learns the distribution of *dynamic object types*.

In summary the work presented in this thesis enables autonomous cars or other mobile robots to localize themselves in dynamic environments. The detection of dynamics in the surrounding allows to build maps of either the static parts of the environment or to include the information about dynamics into the map accounting for changes.

8.1 Future Work

There are several parts of this thesis that could be extended or exchanged.

To predict the probability that parts of a 3D LiDAR scan are generated by dynamic objects we used in Chapter 5 a network which was designed to be used with color images. We therefore transformed our 3D scans into 2D modalities. However recently people try to solve segmentation or classification of 3D data without a projection to 2D (see for example VoxNet [48], PointNet [58] or OctNet [60]). Theses approaches could possibly applied to our dynamic detection.

In Chapter 7, we used precomputed motion as input to our estimation of the *dynamic object probabilities*. The motion was thereby estimated by comparing the most recent scan with previous scans. By using these motion as precomputed input the network is able to distinguish movable and moving objects. The estimated motion is however inaccurate if objects move slow or are far away. For the task of depth estimation from color images so called *siamese networks* can be applied (see for example [47, 90]). These networks take two images as input to estimate the depth of the scene. A siamese approach could possibly applied to estimate the motion within our network using two or more range images as input.

In our last chapter, we learned maps which represent not only the reflectance probability but also a distribution of *dynamic object types*. This estimation could be used during localization to detect localization errors. Static objects for example should not appear unexpected in free space while for movable or moving objects this behavior is not as unexpected.

Bibliography

[1] Daniel Arbuckle, Andrew Howard, and Maja Mataric. „Temporal occupancy grids: a method for classifying the spatio-temporal properties of the environment.“ In: *Intelligent Robots and Systems, 2002. IEEE/RSJ International Conference on*. Vol. 1. IEEE. 2002, pp. 409–414.

[2] Augusto Luis Ballardini, Daniele Cattaneo, Simone Fontana, and Domenico Giorgio Sorrenti. „Leveraging the OSM building data to enhance the localization of an urban vehicle.“ In: *Intelligent Transportation Systems (ITSC), 2016 IEEE 19th International Conference on*. IEEE. 2016, pp. 622–628.

[3] Rodrigo Benenson, Markus Mathias, Radu Timofte, and Luc Van Gool. „Pedestrian detection at 100 frames per second.“ In: *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*. IEEE. 2012.

[4] Christopher M Bishop et al. *Pattern Recognition and Machine Learning*. Vol. 1. springer New York, 2006.

[5] Alexander Bochman. „The Markov Assumption: Formalization and Impact.“ In: *IJCAI*. 2013, pp. 782–788.

[6] G. Bradski. „OpenCV.“ In: *Dr. Dobb's Journal of Software Tools* (2000).

[7] Leo Breiman. „Random forests.“ In: *UC Berkeley TR567* (1999).

[8] Leo Breiman. „Random forests.“ In: *Machine learning* 45.1 (2001), pp. 5–32.

[9] Marcus A Brubaker, Andreas Geiger, and Raquel Urtasun. „Lost! leveraging the crowd for probabilistic visual self-localization.“ In: *Proc. of the IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*. 2013.

[10] Florian Chabot, Mohamed Chaouch, Jaonary Rabarisoa, Céline Teulière, and Thierry Chateau. „Deep MANTA: A Coarse-to-fine Many-Task Network for joint 2D and 3D vehicle analysis from monocular image.“ In: *arXiv preprint arXiv:1703.07570* (2017).

[11] Xiaozhi Chen, Kaustav Kundu, Yukun Zhu, Andrew G Berneshawi, Huimin Ma, Sanja Fidler, and Raquel Urtasun. „3d object proposals for accurate object class detection.“ In: *Advances in Neural Information Processing Systems*. 2015, pp. 424–432.

[12] Xiaozhi Chen, Huimin Ma, Ji Wan, Bo Li, and Tian Xia. „Multi-view 3D Object Detection Network for Autonomous Driving.“ In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE. 2017, pp. 6526–6534.

[13] Frank Dellaert, Dieter Fox, Wolfram Burgard, and Sebastian Thrun. „Monte Carlo Localization for Mobile Robots.“ In: *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*. 1999.

[14] Ayush Dewan, Gabriel L Oliveira, and Wolfram Burgard. „Deep Semantic Classification for 3D LiDAR Data.“ In: *arXiv preprint arXiv:1706.08355* (2017).

[15] Luke Dormehl. *Self-Driving Forklift Takes The Human Factor Out Of Warehouse Work | Digital Trends*. `https://www.digitaltrends.com/cool-tech/seegrid-gp8-series-6-forklift/`. Accessed: 2018-12-04. 2017.

[16] Martin Engelcke, Dushyant Rao, Dominic Zeng Wang, Chi Hay Tong, and Ingmar Posner. „Vote3Deep: Fast object detection in 3D point clouds using efficient convolutional neural networks.“ In: *Robotics and Automation (ICRA), 2017 IEEE International Conference on*. IEEE. 2017, pp. 1355–1361.

[17] Qiu Fan, Yang Yi, Li Hao, Fu Mengyin, and Wang Shunting. „Semantic motion segmentation for urban dynamic scene understanding.“ In: *Automation Science and Engineering (CASE), 2016 IEEE International Conference on*. IEEE. 2016, pp. 497–502.

[18] Manuel Fernández-Delgado, Eva Cernadas, Senén Barro, and Dinani Amorim. „Do we need hundreds of classifiers to solve real world classification problems?“ In: *The Journal of Machine Learning Research* 15.1 (2014), pp. 3133–3181.

[19] Georgios Floros, Benito van der Zander, and Bastian Leibe. „OpenStreetSLAM: Global vehicle localization using OpenStreetMaps.“ In: *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*. 2013.

[20] Jodi Forlizzi and Carl DiSalvo. „Service robots in the domestic environment: a study of the roomba vacuum in the home.“ In: *Proceedings of the 1st ACM SIGCHI/SIGART conference on Human-robot interaction*. ACM. 2006, pp. 258–265.

[21] Dieter Fox, Jeffrey Hightower, Lin Liao, Dirk Schulz, and Gaetano Borriello. „Bayesian filtering for location estimation.“ In: *IEEE pervasive computing* 3 (2003), pp. 24–33.

[22] Andreas Geiger, Philip Lenz, and Raquel Urtasun. „Are we ready for Autonomous Driving? The KITTI Vision Benchmark Suite.“ In: *Conference on Computer Vision and Pattern Recognition (CVPR)*. 2012.

[23] Alejandro González, Gabriel Villalonga, Jiaolong Xu, David Vázquez, Jaume Amores, and Antonio M López. „Multiview random forest of local experts combining RGB and LIDAR data for pedestrian detection.“ In: *Intelligent Vehicles Symposium (IV), 2015 IEEE*. IEEE. 2015, pp. 356–361.

[24] Giorgio Grisetti, Cyrill Stachniss, and Wolfram Burgard. „Improving grid-based SLAM with rao-blackwellized particle filters by adaptive proposals and selective resampling.“ In: *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*. 2005.

[25] Dirk Hähnel, Rudolph Triebel, Wolfram Burgard, and Sebastian Thrun. „Map building with mobile robots in dynamic environments.“ In: *Robotics and Automation, 2003. Proceedings. ICRA'03. IEEE International Conference on*. Vol. 2. IEEE. 2003, pp. 1557–1563.

[26] Peter E Hart, Nils J Nilsson, and Bertram Raphael. „A formal basis for the heuristic determination of minimum cost paths.“ In: *IEEE transactions on Systems Science and Cybernetics* 4.2 (1968), pp. 100–107.

[27] Matthias Hentschel and Bernardo Wagner. „Autonomous robot navigation based on OpenStreetMap geodata.“ In: *Proc. of the IEEE Int. Conf. on Intelligent Transportation Systems (ITSC)*. 2010.

[28] Tin Kam Ho. „Random decision forests.“ In: *Document Analysis and Recognition, 1995., Proceedings of the Third International Conference on*. Vol. 1. IEEE. 1995, pp. 278–282.

[29] Omid Hosseini Jafari, Dennis Mitzel, and Bastian Leibe. „Real-time RGB-D based people detection and tracking for mobile robots and head-worn cameras.“ In: *Robotics and Automation (ICRA), 2014 IEEE International Conference on*. IEEE. 2014.

[30] Patric Jensfelt, David J Austin, Olle Wijk, and Magnus Andersson. „Feature based condensation for mobile robot localization.“ In: *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*. 2000.

[31] Rudolph Emil Kalman. „A new approach to linear filtering and prediction problems.“ In: *Journal of basic Engineering* 82.1 (1960), pp. 35–45.

[32] Chris Kaminski, Tristan Crees, James Ferguson, Alexander Forrest, Jeff Williams, David Hopkin, and Garry Heard. „12 days under ice–an historic AUV deployment in the Canadian High Arctic.“ In: *Autonomous Underwater Vehicles (AUV), 2010 IEEE/OES*. IEEE. 2010, pp. 1–11.

[33] Sören Kammel, Julius Ziegler, Benjamin Pitzer, Moritz Werling, Tobias Gindele, Daniel Jagzent, Joachim Schröder, Michael Thuy, Matthias Goebl, Felix von Hundelshausen, et al. „Team AnnieWAY's autonomous system for the 2007 DARPA Urban Challenge.“ In: *Journal of Field Robotics* 25.9 (2008), pp. 615–639.

[34] Kärcher. *KIRA B 50 - Professioneller Scheuersaugroboter | Kärcher*. `https://www.kaercher.com/at/kira-professioneller-scheuersaugroboter.html`. Accessed: 2018-12-04. 2018.

[35] Andreas Kuhner, Tobias Schubert, Christoph Maurer, and Wolfram Burgard. „An online system for tracking the performance of Parkinson's patients." In: *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2017.

[36] Rainer Kümmerle, Michael Ruhnke, Bastian Steder, Cyrill Stachniss, and Wolfram Burgard. „A navigation system for robots operating in crowded urban environments." In: *Robotics and Automation (ICRA), 2013 IEEE International Conference on*. IEEE. 2013, pp. 3225–3232.

[37] Rainer Kümmerle, Bastian Steder, Christian Dornhege, Alexander Kleiner, Giorgio Grisetti, and Wolfram Burgard. „Large Scale Graph-based SLAM using Aerial Images as Prior Information." In: *Journal of Autonomous Robots* 30.1 (2011), pp. 25–39.

[38] Christian Landsiedel and Dirk Wollherr. „Augmenting and reasoning in semantically enriched maps using open data." In: *Proceedings of the Workshop on Spatial Reasoning and Interaction for Real-World Robotics at IROS 2015*. 2015.

[39] Christian Landsiedel and Dirk Wollherr. „Global localization of 3D point clouds in building outline maps of urban outdoor environments." In: *International journal of intelligent robotics and applications* 1.4 (2017), pp. 429–441.

[40] P Chris Leger, Ashitey Trebi-Ollennu, John R Wright, Scott A Maxwell, Robert G Bonitz, Jeffrey J Biesiadecki, Frank R Hartman, Brian K Cooper, Eric T Baumgartner, and Mark W Maimone. „Mars exploration rover surface operations: Driving spirit at gusev crater." In: *Systems, Man and Cybernetics, 2005 IEEE International Conference on*. Vol. 2. IEEE. 2005, pp. 1815–1822.

[41] Bo Li, Tianlei Zhang, and Tian Xia. „Vehicle detection from 3d lidar using fully convolutional network." In: *arXiv preprint arXiv:1608.07916* (2016).

[42] Linde. *automated truck K-MATIC*. `http://www.linde-mh.com/en/Products/Automated-Trucks/K-Matic/`. Accessed: 2018-12-04. 2018.

[43] Jun S Liu. „Metropolized independent sampling with comparisons to rejection sampling and importance sampling." In: *Statistics and Computing* 6.2 (1996), pp. 113–119.

[44] Matthias Luber, Gian Diego Tipaldi, and Kai O Arras. „Place-dependent people tracking." In: *The International Journal of Robotics Research* 30.3 (2011), pp. 280–293.

[45] Lukas Luft, Alexander Schaefer, Tobias Schubert, and Wolfram Burgard. „Closed-Form Full Map Posteriors for Robot Localization with Lidar Sensors.“ In: *Proc. of the IEEE Int. Conf. on Intelligent Robots and Systems (IROS)*. Vancouver, BC, Canada, 2017.

[46] Lukas Luft, Alexander Schaefer, Tobias Schubert, and Wolfram Burgard. „Detecting Changes in the Environment Based on Full Posterior Distributions Over Real-Valued Grid Maps.“ In: *IEEE Robotics and Automation Letters* 3.2 (2018), pp. 1299–1305.

[47] Wenjie Luo, Alexander G Schwing, and Raquel Urtasun. „Efficient deep learning for stereo matching.“ In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2016, pp. 5695–5703.

[48] Daniel Maturana and Sebastian Scherer. „Voxnet: A 3d convolutional neural network for real-time object recognition.“ In: *Intelligent Robots and Systems (IROS), 2015 IEEE/RSJ International Conference on*. IEEE. 2015, pp. 922–928.

[49] Stephen McPhail. „Autosub6000: A deep diving long range AUV.“ In: *Journal of Bionic Engineering* 6.1 (2009), pp. 55–62.

[50] Donald Meagher. „Geometric modeling using octree encoding.“ In: *Computer graphics and image processing* 19.2 (1982), pp. 129–147.

[51] Daniel Meyer-Delius, Maximilian Beinhofer, and Wolfram Burgard. „Occupancy Grid Models for Robot Mapping in Changing Environments.“ In: *AAAI*. 2012.

[52] Michael Montemerlo, Jan Becker, Suhrid Bhat, Hendrik Dahlkamp, Dmitri Dolgov, Scott Ettinger, Dirk Haehnel, Tim Hilden, Gabe Hoffmann, Burkhard Huhnke, et al. „Junior: The stanford entry in the urban challenge.“ In: *Journal of field Robotics* 25.9 (2008), pp. 569–597.

[53] Frank Moosmann and Christoph Stiller. „Joint Self-Localization and Tracking of Generic Objects in 3D Range Data.“ In: *Proceedings of the IEEE International Conference on Robotics and Automation*. Karlsruhe, Germany, May 2013.

[54] Hans P Moravec and Alberto Elfes. „High resolution maps from wide angle sonar.“ In: *Proceedings of the IEEE Conference on Robotics and Automation*. 1985, pp. 19–24.

[55] Andreas Nüchter and Joachim Hertzberg. „Towards semantic maps for mobile robots.“ In: *Robotics and Autonomous Systems* 56.11 (2008), pp. 915–926.

[56] Luciano Oliveira, Urbano Nunes, Paulo Peixoto, Marco Silva, and Fernando Moita. „Semantic fusion of laser and vision in pedestrian detection.“ In: *Pattern Recognition* 43.10 (2010), pp. 3648–3659.

[57] OpenStreetMap contributors. *OpenStreetMap*. `http://www.openstreetmap.org`. Accessed: 2014-09-28. 2014.

[58] Charles R Qi, Hao Su, Kaichun Mo, and Leonidas J Guibas. „Pointnet: Deep learning on point sets for 3d classification and segmentation.“ In: *Proc. Computer Vision and Pattern Recognition (CVPR), IEEE* 1.2 (2017), p. 4.

[59] N Dinesh Reddy, Prateek Singhal, and K Madhava Krishna. „Semantic motion segmentation using dense crf formulation.“ In: *Proceedings of the 2014 Indian Conference on Computer Vision Graphics and Image Processing*. ACM. 2014, p. 56.

[60] Gernot Riegler, Ali Osman Ulusoy, and Andreas Geiger. „Octnet: Learning deep 3d representations at high resolutions.“ In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. Vol. 3. 2017.

[61] Jorg Rowekamper, Christoph Sprunk, Gian Diego Tipaldi, Cyrill Stachniss, Patrick Pfaff, and Wolfram Burgard. „On the position accuracy of mobile robot localization based on particle filters combined with scan matching.“ In: *Proc. of the Int. Conf. on Intelligent Robots and Systems (IROS)*. 2012.

[62] Philipp Ruchti and Wolfram Burgard. „Mapping with Dynamic-Object Probabilities Calculated from Single 3D Range Scans.“ In: *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*. Brisbane, Australia, May 2018.

[63] Philipp Ruchti, Bastian Steder, Michael Ruhnke, and Wolfram Burgard. „Localization on OpenStreetMap Data using a 3D Laser Scanner.“ In: *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*. Seattle, Washington, USA, May 2015.

[64] Radu Bogdan Rusu and Steve Cousins. „3D is here: Point Cloud Library (PCL).“ In: *IEEE International Conference on Robotics and Automation (ICRA)*. Shanghai, China, May 2011.

[65] Alexander Schaefer, Lukas Luft, and Wolfram Burgard. „An Analytical Lidar Sensor Model Based on Ray Path Information.“ In: *IEEE Robotics and Automation Letters* 2.3 (2017), pp. 1405–1412.

[66] Jürgen Schmidhuber. „Deep learning in neural networks: An overview.“ In: *Neural networks* 61 (2015), pp. 85–117.

[67] Luciano Spinello, Matthias Luber, and Kai O Arras. „Tracking people in 3D using a bottom-up top-down detector.“ In: *Robotics and Automation (ICRA), 2011 IEEE International Conference on*. IEEE. 2011.

[68] Cyrill Stachniss. *EUROPA - The European robotic pedestrian assistant*. `http://europa.informatik.uni-freiburg.de/`. Accessed: 2018-10-08. 2018.

[69] Cyrill Stachniss and Wolfram Burgard. „Using coverage maps to represent the environment of mobile robots.“ In: *Proc. of the European Conference on Mobile Robots (ECMR)*. 2003, pp. 59–64.

[70] Cyrill Stachniss and Gian Diego Tipaldi. *EUROPA2 - The European robotic pedestrian assistant*. `http://europa2.informatik.uni-freiburg.de/`. Accessed: 2018-10-08. 2018.

[71] Benjamin Suger and Wolfram Burgard. „Global outer-urban navigation with OpenStreetMap.“ In: *Robotics and Automation (ICRA), 2017 IEEE International Conference on*. IEEE. 2017, pp. 1417–1422.

[72] Niko Sünderhauf, Feras Dayoub, Sean McMahon, Ben Talbot, Ruth Schulz, Peter Corke, Gordon Wyeth, Ben Upcroft, and Michael Milford. „Place categorization and semantic mapping on a mobile robot.“ In: *Robotics and Automation (ICRA), 2016 IEEE International Conference on*. IEEE. 2016, pp. 5729–5736.

[73] Peter Swerling. *A proposed stagewise differential correction procedure for satellite tracking and prediciton*. Rand Corporation, 1958.

[74] Alex Teichman, Jesse Levinson, and Sebastian Thrun. „Towards 3D object recognition via classification of arbitrary object tracks.“ In: *International Conference on Robotics and Automation*. 2011.

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

[76] Gian Diego Tipaldi and Cyrill Stachniss. *LifeNav - Reliable Lifelong Navigation for Mobile Robots*. `http://lifenav.informatik.uni-freiburg.de/`. Accessed: 2018-10-09. 2018.

[77] Chris Urmson. *The latest chapter for the self-driving car: mastering city street driving*. `http://googleblog.blogspot.de/2014/04/the-latest-chapter-for-self-driving-car.html`. Accessed: 2014-09-28. 2014.

[78] Chris Urmson, Joshua Anhalt, Drew Bagnell, Christopher Baker, Robert Bittner, MN Clark, John Dolan, Dave Duggins, Tugrul Galatali, Chris Geyer, et al. „Autonomous driving in urban environments: Boss and the urban challenge.“ In: *Journal of Field Robotics* 25.8 (2008), pp. 425–466.

[79] Abhinav Valada, Johan Vertens, Ankit Dhall, and Wolfram Burgard. „AdapNet: Adaptive semantic segmentation in adverse environmental conditions.“ In: *Robotics and Automation (ICRA), 2017 IEEE International Conference on*. IEEE. 2017, pp. 4644–4651.

[80] Johan Vertens, Abhinav Valada, and Wolfram Burgard. „SMSnet: Semantic Motion Segmentation using Deep Convolutional Neural Networks.“ In: *IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS*. 2017.

[81] Olga Vysotska and Cyrill Stachniss. „Exploiting building information from publicly available maps in graph-based SLAM.“ In: *Intelligent Robots and Systems (IROS), 2016 IEEE/RSJ International Conference on*. IEEE. 2016, pp. 4511–4516.

[82] Olga Vysotska and Cyrill Stachniss. „Improving slam by exploiting building information from publicly available maps and localization priors.“ In: *PFG–Journal of Photogrammetry, Remote Sensing and Geoinformation Science* 85.1 (2017), pp. 53–65.

[83] Dominic Zeng Wang and Ingmar Posner. „Voting for Voting in Online Point Cloud Object Detection.“ In: *Robotics: Science and Systems*. 2015.

[84] Dominic Zeng Wang, Ingmar Posner, and Paul Newman. „What could move? Finding cars, pedestrians and bicyclists in 3D laser data.“ In: *Robotics and Automation (ICRA), 2012 IEEE International Conference on*. 2012.

[85] Xiaolong Wang, Liang Lin, Lichao Huang, and Shuicheng Yan. „Incorporating structural alternatives and sharing into hierarchy for multiclass object recognition and detection.“ In: *Computer Vision and Pattern Recognition (CVPR), 2013 IEEE Conference on*. IEEE. 2013.

[86] Denis F Wolf and Gaurav S Sukhatme. „Mobile robot simultaneous localization and mapping in dynamic environments.“ In: *Autonomous Robots* 19.1 (2005), pp. 53–65.

[87] Jürgen Wolf, Wolfram Burgard, and Hans Burkhardt. „Robust vision-based localization by combining an image-retrieval system with Monte Carlo localization.“ In: *Robotics, IEEE Transactions on* 21.2 (2005), pp. 208–216.

[88] Yu Xiang and Dieter Fox. „DA-RNN: Semantic mapping with data associated recurrent neural networks.“ In: *arXiv preprint arXiv:1703.03098* (2017).

[89] Jiejun Xu, Kyungnam Kim, Zhiqi Zhang, Hai-wen Chen, and Yuri Owechko. „2D/3D Sensor Exploitation and Fusion for Enhanced Object Detection.“ In: *Computer Vision and Pattern Recognition Workshops (CVPRW), 2014 IEEE Conference on*. IEEE. 2014, pp. 778–784.

[90] Jure Zbontar and Yann LeCun. „Stereo matching by training a convolutional neural network to compare image patches.“ In: *Journal of Machine Learning Research* 17.1-32 (2016), p. 2.

[91] Huijing Zhao, Yiming Liu, Xiaolong Zhu, Yipu Zhao, and Hongbin Zha. „Scene understanding in a large dynamic environment through a laser-based sensing.“ In: *Robotics and Automation (ICRA), 2010 IEEE International Conference on*. IEEE. 2010.

[92] Vittorio A Ziparo, Marco Zaratti, Giorgio Grisetti, Taigo Maria Bonanni, Jacopo Serafin, Maurilio Di Cicco, Marc Proesmans, L Van Gool, Olga Vysotska, Igor Bogoslavskyi, et al. „Exploration and mapping of catacombs with mobile robots.“ In: *Safety, Security, and Rescue Robotics (SSRR), 2013 IEEE International Symposium on*. IEEE. 2013, pp. 1–2.

List of Figures

List of Tables