universität ulm



Deep Learning on Point Clouds with Applications in Vehicle Self-Localization

Nico Engel

Band 44

Schriftenreihe des Instituts für Mess-, Regel- und Mikrotechnik

Nico Engel

Deep Learning on Point Clouds with Applications in Vehicle Self-Localization

Schriftenreihe des
Instituts für Mess-, Regel- und Mikrotechnik
Universität Ulm

Herausgeber:
Prof. Dr.-Ing. Klaus Dietmayer

Band 44

Nico Engel

Deep Learning on Point Clouds with Applications in Vehicle Self-Localization

Bibliografische Information der Deutschen Nationalbibliothek
Die Deutsche Nationalbibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über http://dnb.dnb.de abrufbar.

Dissertation, Universität Ulm,
Fakultät für Ingenieurwissenschaften, Informatik und Psychologie, 2022

Impressum

Universität Ulm
Institut für Mess-, Regel- und Mikrotechnik
Prof. Dr.-Ing. Klaus Dietmayer
Albert-Einstein-Allee 41
89081 Ulm
http://www.uni-ulm.de/mrm

Eine Übersicht über alle Bände der Schriftenreihe finden Sie unter
http://www.uni-ulm.de/mrmschriften.

Diese Veröffentlichung ist im Internet auf dem Open-Access-Repositorium der Universität Ulm (https://oparu.uni-ulm.de) verfügbar und dort unter der “Lizenz B (ohne Print-On-Demand)” publiziert. Details zur Lizenz sind unter http://www.uni-ulm.de/index.php?id=50392 zu finden.

Institut für Mess-, Regel- und Mikrotechnik der Universität Ulm 2023
Print on Demand

ISBN 978-3-941543-69-0 ISBN 978-3-941543-70-6 (E-Book)



universität ulm

Deep Learning on Point Clouds with Applications in Vehicle Self-Localization

DISSERTATION

zur Erlangung des akademischen Grades eines

DOKTOR-INGENIEURS

(Dr.-Ing.)

der Fakultät für Ingenieurwissenschaften, Informatik und Psychologie der Universität Ulm

von

Nico Engel
aus Lübeck

Gutachter:	Prof. Dr.-Ing. Klaus Dietmayer
	Prof. Dr.-Ing. Christoph Stiller
Amtierende Dekanin:	Prof. Dr. Anke Huckauf

Ulm, 03.11.2022

Acknowledgements

I would like to express my sincere gratitude to my supervisor Prof. Dr.-Ing. Klaus Dietmayer, who has supported me throughout my Ph.D. journey and has always been available to offer valuable guidance and feedback. I am also grateful to Prof. Dr.-Ing. Christoph Stiller for taking on the role of co-examiner.

Furthermore, I am thankful to Prof. Dr. Vasileios Belagiannis for the invaluable support, belief in my abilities, and encouragement to pursue my research interests, which have been essential to my success. I am also deeply grateful for the engaging and thought-provoking discussions we have had and the insights that enhanced my research skills. Additionally, I would like to thank Prof. Dr.-Ing Thomas Meurer and Dr.-Ing. Gerhard Freudenthaler for paving the way for my career in academia and the support throughout my master's studies that encouraged me to start my Ph.D. journey.

I would like to extend my thanks to all of my colleagues at the Institute of Measurement, Control and Microtechnology for their support and encouragement. The fun times and interesting discussions we shared have made my time as a Ph.D. student enjoyable and rewarding. Your support has been invaluable, and I am deeply grateful to have had the opportunity to work alongside such talented and supportive individuals. In particular, I would like to thank Dr.-Ing. Maximilian Graf, Oliver Speidel, Carsten, and Thomas Griebel for their friendship, support, and all the fun and interesting coffee and lunch break discussions we had over the years. Our conference trips will be remembered for a long time. Moreover, I would like to thank Dr.-Ing. Stefan Hörmann for the support and guidance during my early days at the institute. Finally, I would like to thank the entire administrative team at the institute, especially Claudia and Diemut, for always being there to help me with any issues and for their enormous support in all administrative matters.

I am grateful to my friends and family for their unwavering support throughout my studies. The support and advice from each of you is greatly appreciated and will always be remembered. This thesis would not have been possible without you. Finally, and most importantly, I am eternally grateful and thankful to my girlfriend, Michelle Cyron, for her unwavering and never-ending support and guidance, without which I would not have been able to complete this journey.

Abstract

The precise identification of the vehicle pose, which is comprised of the position and orientation within a local or global coordinate system, is a critical backbone module and an essential prerequisite for the safe and responsible operation of automated vehicles. In well-structured environments with low complexity, for example, on highways, series-produced vehicles already operate in an automated manner in specific scenarios using environment perception only. However, the deployment in highly dynamic and urban areas requires sophisticated perception and action planning algorithms to calculate and perform a driving maneuver appropriate to the current situation. For this purpose, the pose is estimated using self-localization approaches, which, combined with accurate digital map information acts as an additional sensor. For example, an intersection that the sensors cannot correctly perceive can be loaded from the digital map, and valuable and required information is extracted accordingly. Then, appropriate trajectories can be planned, which would not be possible without knowing the vehicle's position on the map.

Subsequently, localization methods can be divided into several categories, often characterized by the type of map and measurement representation. This thesis focuses on processing pure point cloud-based measurements and landmarks, which means that the environment is represented by two or three-dimensional unordered and unstructured point sets. Traditional filter-based localization methods are well-explored in the literature, matching multi-modal sensor measurements, such as radar, camera, or lidar measurements, with information from a previously acquired high-accuracy digital map to infer the pose. This process requires a so-called association and matching algorithm, which is usually implemented by heuristic or hand-tuned algorithms. The application of deep neural networks has already revolutionized many areas of computer vision, especially the processing of camera-based image information. Based on these advances, this work aims to present state-of-the-art methods in the area of data-based point cloud processing, as well as novel techniques that are applied to landmark-based self-localization.

First, this thesis introduces the Point Transformer network architecture, a general method that processes geometric information from three-dimensional point clouds and learns shape and local dependencies. A novel sorting module is presented, which learns a latent feature representation of the input that induces permutation-

invariance. Furthermore, the widespread attention mechanism is adapted to deal with the difficult task of processing arbitrary point sets. The network can be applied to solve common computer vision tasks, such as object classification or part segmentation.

The second part of this thesis focuses on a deep neural network that adapts well-known and established point cloud processing architectures for vehicle self-localization. Here, two unstructured point sets represent the input, i.e., the landmarks and multi-modal measurements. The goal is to approximate the traditional self-localization task comprised of multiple subtasks such as measurement-to-landmark association and point cloud registration, using a permutation-invariant neural network that processes the landmarks and measurement sets to infer the vehicle's pose. Moreover, different improvements, a novel training strategy, and two system architectures that allow the deployment in real-world scenarios are presented. The methods are evaluated using a dataset creation pipeline that is able to generate automatically labeled data on the fly. Finally, the introduced Point Transformer approach is adapted for self-localization, where high-density lidar point clouds that contain semantic features in addition to geometric information are processed. Again, an automatically generated and annotated dataset is presented for this method that uses state-of-the-art lidar segmentation networks.

The evaluation of all methods presented in this thesis is first performed in synthetically generated scenarios, allowing systematic analysis of different influences and interfering factors of the localization task. Subsequently, the methods are evaluated in experiments using real-world data acquired with an automatically operated vehicle. Compared to traditional filter-based localization algorithms, significant improvements in localization accuracy can be achieved, meeting the requirements for safe and reliable automated operation in urban environments.

Kurzfassung

In den vergangenen Jahren haben neue Technologien und Algorithmen im Bereich der Fahrerassistenzsysteme einen starken Anstieg der Funktionalität und Einsatzbereiche ermöglicht, sodass bereits heute in bestimmten Szenarien und in gut strukturierten Umgebungen von geringer Komplexität wie zum Beispiel auf Autobahnen, ein weitestgehend automatisiertes Fahren von Serienfahrzeugen gewährleistet werden kann. Jedoch sind für den sicheren und robusten, voll automatisierten Betrieb in dynamischen Szenarien, beispielsweise in urbanen Bereichen, eine Vielzahl von aufeinander aufbauenden Softwaremodulen und komplexe Prozessketten erforderlich. Dabei ist die hochgenaue Bestimmung der Fahrzeugpose eine grundlegende Voraussetzung und ein essenzieller Bestandteil für die verlässliche und detaillierte Umgebungswahrnehmung, das Situationsverstehen, sowie die Handlungs- und Trajektorienplanung. Im Allgemeinen setzt sich die Fahrzeugpose aus einer zweidimensionalen Position und der Orientierung zusammen, die in einem globalen oder lokalen Koordinatensystem definiert sind. Um ein der aktuellen Situation angemessenes Fahrmanöver zu planen und durchzuführen, wird die Fahrzeupose mit einer Genauigkeit im Zentimeterbereich erwartet, welche durch handelsübliche GPS-Sensorik nicht gewährleistet werden kann. Aus diesem Grund wird die Fahrzeugpose mithilfe von Lokalisierungsalgorithmen geschätzt, indem Sensormessungen und statische Objekte aus einer digitalen Karte, auch Landmarken genannt, miteinander in Beziehung gesetzt werden. So lassen sich beispielsweise relevante Informationen für eine Kreuzung, die von den Sensoren nicht umfassend eingesehen und wahrgenommen werden kann, aus der digitalen Karte laden und für nachgelagerte Algorithmen zur Verfügung stellen. In der Literatur werden Lokalisierungsmethoden üblicherweise in verschiedene Kategorien untergliedert, wobei die Unterscheidung anhand der Darstellung der Karten- und Messinformationen durchgeführt wird. Diese Arbeit konzentriert sich dabei auf die Verarbeitung punktwolkenbasierter Eingangsdaten, was bedeutet, dass die Umgebung durch zwei- oder dreidimensionale, ungeordnete und unstrukturierte Punktmengen dargestellt wird. Weit verbreitet sind traditionelle filterbasierte Lokalisierungsmethoden, die multimodale Sensormessungen wie zum Beispiel Radar-, Kamera- oder Lidarmessungen, mit Informationen aus einer zuvor erstellten hochgenauen digitalen Karte abgleichen, um daraus die Pose abzuleiten. Dieser Prozess erfordert unter anderem einen Assoziationsalgorithmus für die Zuordnung von Messungen und Landmarken, der in der Regel durch heuristische oder manuell eingestellte Algorithmen implementiert wird.

Tiefe neuronale Netze haben bereits viele Bereiche des maschinellen Sehens revolutioniert, insbesondere die Verarbeitung von Kamerabildern. Aufbauend auf diesen Fortschritten zielt diese Arbeit darauf ab, neuartige Verfahren im Gebiet der datenbasierten Punkwolkenverarbeitung vorzustellen und diese Methoden für die Selbslokalisierung von automatisierten Fahrzeugen anzupassen. Dafür wird zunächst die Point Transformer Netzwerkarchitektur vorgestellt, wobei es sich um eine allgemeine Methode zur Verarbeitung geometrischer und punktbasierter Eingangsdaten handelt, die lokale Abhängigkeiten zwischen individuellen Punkten berücksichtigt. Das Verfahren adaptiert den weitverbreiteten *Attention* Mechanismus, um die Verarbeitung von unstrukturierten Punkwolken zu ermöglichen und stellt ein neues Modul vor, welches eine permutationsinvariante Repräsentation der Eingangspunktwolke anhand von gelernten Kernpunkten generiert. Die Netzwerkarchitektur wird auf bekannten *3D Computer Vision Benchmarks* evaluiert, wobei vielversprechende Ergebnisse erzielt werden.

Der zweite Teil dieser Arbeit beschäftigt sich mit einem *Framework* aus tiefen neuronalen Netzen, welches etablierte Netzwerkarchitekturen zur Verarbeitung von Punktwolken für die Selbstlokalisierung von automatisierten Fahrzeugen adaptiert. Bei den Verfahren stellen zwei unstrukturierte Punktmengen den Eingang dar, die im Folgenden als Landmarken und Messungen bezeichnet werden. Das Ziel dabei ist es, die lernbasierten Ansätze an traditionelle Formulierungen des Lokalisierungsproblems anzupassen, welche die Lokalisierung in mehrere Teilaufgaben untergliedern. Hierzu gehört zum Beispiel die Assoziation von Landmarken und Messungen und die Registrierung von Punktwolken unter Verwendung eines permutationsinvarianten neuronalen Netzes, um die Pose des Fahrzeugs zu ermitteln. Darüber hinaus werden verschiedene Verbesserungen, neuartige Trainingsstrategien sowie zwei Systemarchitekturen vorgestellt, die den Einsatz der Methoden in realen Szenarien ermöglichen. Für die Evaluation wird eine Prozessierungspipeline vorgestellt, die automatisiert annotierte Daten generiert. Schließlich wird der vorgestellte Point Transformer Ansatz für die Selbstlokalisierung angepasst, wobei dichte Lidar-Punktwolken verarbeitet werden, die zusätzlich zu den geometrischen Informationen auch semantische Merkmale enthalten.

Die Evaluierung aller in dieser Arbeit vorgestellten Methoden erfolgt zunächst in einer synthetischen Simulationsumgebung, wodurch eine systematische Analyse von verschiedenen Einflüssen und Störfaktoren auf die Lokalisierungsgenauigkeit ermöglicht wird. Anschließend werden die Verfahren mit realen Daten evaluiert. Im Vergleich zu traditionellen filterbasierten Lokalisierungsalgorithmen können signifikante Verbesserungen der Lokalisierungsgenauigkeit erzielt werden, die den Anforderungen an einen sicheren und zuverlässigen automatisierten Betrieb in urbanen Umgebungen gerecht wird.

Contents

1 Introduction **1**

2 Fundamentals **5**
2.1 Point Clouds and Set Theory 5
2.1.1 Operations on Sets 7
2.1.2 Set Functions 7
2.1.3 Transformations 9
2.2 Probability Theory 12
2.2.1 Probability Distributions 12
2.2.2 Information Theory 16
2.3 Vehicle Self-Localization 18
2.3.1 Basics 19
2.3.2 Landmark-based Localization 21
2.4 Deep Learning 23
2.4.1 Basics 23
2.4.2 Multilayer Perceptron 25
2.4.3 Convolutional Neural Network 26
2.4.4 Row-wise Feedforward Network 28
2.4.5 Additional Deep Learning Concepts 31

3 Related Work **41**
3.1 Deep Learning on Point Clouds 41
3.2 Vehicle Self-Localization 46
3.3 Deep Learning for Vehicle Self-Localization 49

4 Point Transformer **53**
4.1 Introduction 53
4.2 Fundamentals 57
4.2.1 Attention Mechanism 57
4.2.2 Transformer 58
4.3 Methodology and Network Architecture 61
4.3.1 Local Feature Generation and SortNet 63
4.3.2 Global Feature Generation 65
4.3.3 Local-Global Attention 65

4.3.4 Complete Network Architecture . . . 66
4.4 Experiments and Evaluation . . . 67
4.4.1 Shape Classification . . . 68
4.4.2 Part Segmentation . . . 69
4.4.3 Network Complexity Study . . . 70
4.4.4 Network Design Analysis . . . 71

5 Localization Datasets 75
5.1 Experimental Setup . . . 75
5.2 DeepLocalization Dataset . . . 77
5.2.1 Measurements . . . 77
5.2.2 Mapping . . . 80
5.3 Point Transformer Localization Dataset . . . 82
5.3.1 Measurements . . . 83
5.3.2 Mapping . . . 84

6 Deep Learning for Vehicle Self-Localization 87
6.1 Motivation . . . 87
6.2 Problem Formulation . . . 91
6.3 DeepLocalization . . . 92
6.3.1 Methodology . . . 92
6.3.2 DeepLocalization2 . . . 103
6.3.3 Attention-based DeepLocalization . . . 107
6.4 Point Transformer Localization . . . 111
6.4.1 Introduction . . . 112
6.4.2 Methodology . . . 114

7 Experiments and Evaluation 121
7.1 DeepLocalization . . . 121
7.1.1 Synthetic Experiments . . . 122
7.1.2 Real-world Experiments . . . 127
7.1.3 Network Design and Ablation Studies . . . 131
7.2 Point Transformer Localization . . . 133

8 Summary and Outlook 137
8.1 Summary . . . 137
8.2 Outlook . . . 138

A Point Transformer 141
A.1 Implementation Details . . . 141
A.2 Additional Evaluation Results . . . 143
A.2.1 SortNet . . . 143
A.2.2 Part Segmentation . . . 143

B DeepLocalization and Point Transformer Localization **149**
B.1 Datasets 149
B.2 Filter-based Inference 151

List of Symbols **157**

Bibliography **159**

Own Publications **177**

Supervised Theses **179**

Curriculum Vitae **181**

Chapter 1

Introduction

The development of automated vehicles has seen an enormous increase in research interest and progress in recent years, as established car manufacturers and fast-rising startups race to bring autonomous technologies into series production. However, besides the economic incentive, another reason has fueled technological and theoretical advances: Increasing road safety and reducing severe traffic accidents. A technical report conducted from 2005 to 2007 by the National Highway Traffic Safety Administration (NHTSA), a subdivision of the U.S. Department of Transportation, investigated about 5500 crashes to deduce the critical reason for each crash. Here, the critical reason is defined to be the last event in the crash causal chain [Sin15]. The report found that in over 94 % of all cases, the critical reason was assigned to the driver, i.e., almost all crashes were caused by human error. Moreover, 41 % of these human-caused crashes were due to recognition errors and about 33 % due to decision errors [Sin15]. Therefore, the adoption of advanced driver-assistance systems (ADAS) and even comprehensive automated driving capabilities, where accurate algorithms perform the decision making and perception, promises a safer and more reliable operation of road vehicles. For that, a mature and validated processing chain is the most important requirement. The goal is to obtain a robust and thorough representation of the environment through a multi-modal sensor setup that enables the vehicle to navigate and operate in dynamic and diverse scenarios. Since urban environments can include various other road participants, possible occlusions, and interferences, the consensus is to employ digital maps. The maps usually contain high-precision environment information, such as lane geometries, road logic, traffic signs, and static objects [BAYG17; KFK+18]. For that, a key component is the availability of an accurate pose estimate relative to the digital map frame, i.e., the position and orientation of the vehicle. Then, relevant information from the map can be fused with the environment perception [GDD18]. Moreover, subsequent modules can utilize this information to enhance the robustness of algorithms, such as object tracking [EHHD18] or trajectory planning [GSZD18]. The task of vehicle self-localization deals with providing the own pose with high accuracy. This is done

either by utilizing a-priori digital maps or creating an online map while localizing. Nevertheless, both approaches process environmental information obtained through sensor measurements and landmarks from the digital map.

Consequently, this thesis introduces a vehicle self-localization framework that provides a precise pose by aligning multi-modal sensor measurements with a digital map. Moreover, different types of environment representation are possible, whereas this work focuses on landmark-based maps. In this case, landmarks can be described by being static and easily recognizable objects [ED20; EHH+19] that are represented as 2D or 3D points with the possibility of additional feature dimensions. In the past decades, robust approaches have been proposed that employ filter-based localization methods for mobile robotic applications [MTKW03; Stü18; Thr02]. However, recent advances in the field of deep learning, which are fueled by rapid increases in computational capabilities and a wide range of publicly available datasets [GBC16, Chapter 1.2], have shown impressive results in general computer vision tasks and 3D point cloud processing, such as object detection [FHR+21], semantic segmentation [KND+21] or shape classification [EBD21b]. Naturally, adapting neural network-based approaches for vehicle self-localization promises to achieve state-of-the-art results in real-world driving applications. Thus, this thesis aims to reformulate the localization task into a learning-based problem using current network architectures and a novel training framework.

However, there are multiple challenges associated with processing point clouds using deep neural networks. Mathematically speaking, point clouds are represented as unordered sets, which means that generally, no ordering of the individual points exists. Traditional neural networks, such as multilayer perceptrons (MLP) or convolutional neural networks (CNN) require the input to be structured; thus, directly processing irregular point data is not possible. Multiple solutions to this problem have been proposed over the years, that include the transformation into a structured 3D representation, e.g., voxels, or a 2D representation, e.g., rendered images. However, due to the sparse nature of point clouds, this drastically increases memory requirements and can lead to a loss of geometric information. The pioneering work from Qi et al. [QSMG17; QYSG17] introduced a permutation-invariant framework that directly operates on point sets, which is the backbone of the proposed methods in this thesis. Moreover, due to the multi-modal sensor setup used in this work, different environmental representations exist that require an adaption of the network architecture to incorporate different point cloud densities. For example, radar point clouds are usually sparse in comparison with dense lidar point clouds. Furthermore, a varying number of input dimensions is possible, e.g., 2D versus 3D. For the task of vehicle self-localization, challenges arise from requirements of ADAS and perception modules that demand a position accuracy below 50 cm for safe operation in dynamic scenarios [WEK05]. Commercially available global navigation satellite systems (GNSS), e.g., GPS, fail to meet these requirements, as the accuracy is

subject to atmosphere disturbances, multi-path effects, and a clear line of sight to the satellites [LMT07; ZSC+12]. Therefore, the pose information obtained from regular GNSS, which can have an accuracy of about 2 m to 10 m, is not suitable for automated driving tasks. A widely known solution to this problem is the integration of a differential GPS that uses correction data from ground-based reference stations to improve the localization result. However, the operation and maintenance costs render them unsustainable for mass production [EBD21a]. Furthermore, the system is also prone to occlusion of the line of sight to the satellites that occur in tunnels or so-called street canyons. As mentioned before, urban scenarios are comprised of dynamic influences that change the perceived environment, such as other road participants that occlude the field of view, construction works that modify the road geometry, or seasonal effects that alter the environmental appearance. In addition, the sensors that are employed on the automated vehicle are subject to noise and erroneous measurements, e.g., clutter or missed detections, that increase the number of interferences. In summary, all the mentioned effects harm the localization task and complicate the measurement-to-landmark association process by causing ambiguous or improbable associations.

This thesis proposes a deep learning framework that copes with the aforementioned challenges and difficulties of the localization task in dynamic scenarios. In total, two main contributions are presented that focus on different input point cloud cardinalities. First, a method is introduced that processes sparse preprocessed measurements from a multi-modal sensor setup with cameras, radar and lidar sensors. In contrast to this method, an advanced deep neural network that focuses on geometric features of dense lidar point clouds is presented, where semantic information is used to differentiate between the static and dynamic environment. The challenge of the irregular input structure is solved by proposing permutation-invariant network architectures that are based on 1) the popular PointNet [QYSG17] architecture and 2) the attention mechanism [VSP+17], which is adapted from natural language processing (NLP) tasks to work on unstructured point sets. A novelty of the presented approaches in this work is the ability to process two independent point sets, i.e., the landmarks from the digital map and the measurements. Furthermore, particular emphasis is put on learning local relationships between both point clouds which resembles the association task of traditional localization algorithms. Then, to deal with the challenges of the localization task, a novel training strategy is proposed that, theoretically, allows to automatically generate an infinite amount of labeled training samples. Additionally, the network is designed to cope with noisy and ambiguous measurement-to-landmark associations and still infer an accurate pose that meets the requirements of urban driving scenarios. For that, two inference modes are introduced. First, an initial and inaccurate GPS signal is improved by matching measurements to the landmarks, and second, a filter-based approach that can recursively estimate the vehicle's pose over time using the network's prediction as input. Moreover, the methods rely on generic feature-based representations of the environment, i.e., 2D or 3D points, where

additional features, such as semantic class labels, can easily be added. Finally, the generalization capabilities of the entire system are essential: By being able to train the network on a small dataset recorded on a test track and deploying the algorithm in various cities, the economic footprint of the method is massively decreased. The ability to generate input samples while training the network makes the approach well-suited for broad deployment.

The main contributions of this thesis can be summarized as follows:

- Reformulation of the traditional localization task to be applicable in current state-of-the-art deep neural network architectures. For that, a preprocessing pipeline, that provides generic measurements and point-based maps, is introduced. Moreover, the network architectures are adapted to handle two independent point clouds as input to solve the measurement-to-landmark association in conjunction with point cloud registration and the subsequent pose estimation tasks.

- Point Transformer, a 3D point cloud processing network that incorporates recent advances in the field of NLP, and is able to achieve competitive results on common computer vision benchmarks. The ideas are also adapted for the task of lidar localization to cope with high-density input point clouds.

- Introduction of two localization datasets in addition to a training framework that allows for unsupervised label generation.

- Evaluation of the proposed methods on real-world datasets that show promising results and meet the localization requirements in urban environments. Furthermore, extensive experiments are carried out that investigate the generalization capabilities and influence of different interference effects in real-world scenarios.

The remainder of this thesis is structured as follows: First, Chapter 2 introduces the necessary mathematical and architectural fundamentals. Then, Chapter 3 discusses state-of-the-art methods for 3D-based neural networks and traditional vehicle self-localization approaches. One of the main contributions of this work, Point Transformer, is presented in Chapter 4. Afterward, the localization datasets are explained in Chapter 5. The second major contribution includes novel network architectures that work directly on unstructured point data to infer the vehicle's pose. The methodology, specific system architectures, and a novel learning strategy are introduced in Chapter 6. The proposed methods are evaluated on multiple datasets in Chapter 7. Finally, a summary and outlook for further research are given in Chapter 8.

Chapter 2

Fundamentals

This chapter serves as an overview of the fundamentals required for the proposed algorithms in this thesis. First, a short introduction to the general properties of 3D point clouds is given. Then, the fundamentals of vehicle self-localization and the required notations are laid out that subsequent sections rely on. Finally, this chapter introduces the mathematical theory and notation of deep learning algorithms used for the proposed methods.

2.1 Point Clouds and Set Theory

A point cloud is mathematically defined as a set of points. Without going into further detail, a set is a collection of individual elements [JAA95]. If an element y belongs to a set $\mathcal{Y}$, it is denoted by $y \in \mathcal{Y}$, otherwise $y \notin \mathcal{Y}$. If a set contains no element, it is called the empty set $\mathcal{Y} = \emptyset$. Furthermore, a set can be constructed by utilizing the set-builder notation, e.g., a set $\mathcal{Y}$ that contains all real numbers that satisfy the formula Ξ, also called the predicate, is given by

$$\mathcal{Y} = \{\ y \mid y \in \mathbb{R} \text{ and } \Xi(y)\ \}. \tag{2.1}$$

As mentioned before, the elements of a point cloud are points in metric space $\mathbb{R}^D$. A metric space is comprised of the nonempty set $\mathcal{X} \neq \emptyset$ and a distance function $d(x, y)$, that assigns a real number $d : \mathcal{X} \times \mathcal{X} \to \mathbb{R}^1$ to every pair $x, y \in \mathcal{X}$, such that it satisfies the following axioms [SV05, pp. 27–28]:

1. $d(x, y) \geq 0$,

2. $d(x, y) = 0$, if and only if $x = y$,

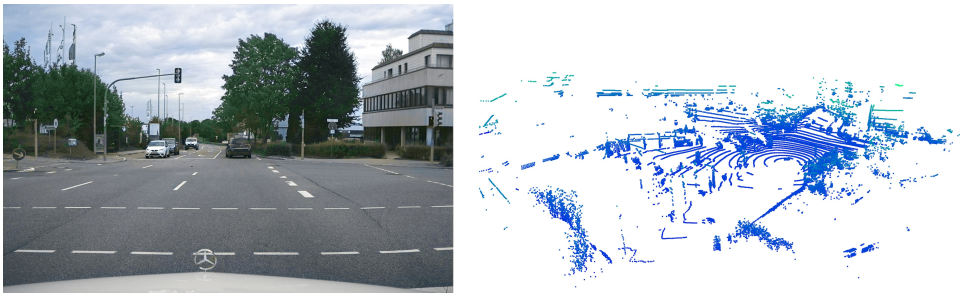


Figure 2.1: An exemplary point cloud is shown that is obtained from a lidar sensor (right) and the corresponding scene captured with a camera (left).

3. $d(x, y) = d(y, x)$,

4. $d(x, y) \leq d(x, z) + d(z, y)$, with $x, y, z \in \mathcal{X}$.

Subsequently, a point cloud $\mathcal{P}$ can be formally defined similar to Equation (2.1) using the set-builder notation

$$\mathcal{P} = \{\boldsymbol{p}_1, \ldots, \boldsymbol{p}_\nu \mid \boldsymbol{p}_i \in \mathbb{R}^D\} \quad \text{or} \quad \mathcal{P} = \{\boldsymbol{p}_1, \ldots, \boldsymbol{p}_\nu\}, \boldsymbol{p}_i \in \mathbb{R}^D, \tag{2.2}$$

where the dimension D depends on the application. In this thesis, only points in Euclidean space are considered; therefore, each point is comprised of 2D or 3D coordinates describing the points' spatial characteristics and, in some cases, additional feature or attribute dimensions. This notation denotes the number of elements in $\mathcal{P}$ by ν, also called the cardinality of the set. Point clouds are usually obtained from 3D scanners such as lidar and radar sensors or photogrammetry, i.e., the process of extracting shape and spatial information of objects from images, for example by using the bundle adjustment method [Kra11, Chapter 5.3] or the collinearity principle [Kra11, pp. 21–24]. Furthermore, point clouds can be retrieved from 3D CAD (computer-aided design) models by sampling points from the surface of the object. An exemplary point cloud from a lidar sensor with the corresponding scene from a camera is shown in Figure 2.1. In the following, properties of point clouds that are important for subsequent chapters as well as the proposed methods are outlined:

1. The elements of a point cloud are unordered, thus $\{\boldsymbol{p}_1, \boldsymbol{p}_2\} = \{\boldsymbol{p}_2, \boldsymbol{p}_1\}$.

2. A point cloud has arbitrary cardinality ν; in other words, the number of points in the set can be variable and is not known in advance.

3. The elements of point clouds considered in this thesis have spatial relations, meaning the points are not independent of each other as neighboring points are usually obtained from the same object.

4. The semantics of a point cloud should be invariant to arbitrary translation and rotation in 3D space. This implies that the object of origin has the same local spatial features, regardless of how the point cloud is rotated or translated.

Considering these properties, especially 1. and 2., it is necessary to define operations and functions that can handle unordered sets as input.

2.1.1 Operations on Sets

In this section, common operations on sets are introduced. These operations form new sets from one or more existing sets. In the following, two arbitrary sets $\mathcal{X}$ and $\mathcal{Y}$ are considered. The *union* of $\mathcal{X}$ and $\mathcal{Y}$ contains all elements of $\mathcal{X}$ together with all elements of $\mathcal{Y}$. It is denoted by

$$\mathcal{X} \cup \mathcal{Y}. \tag{2.3}$$

The *intersection* of $\mathcal{X}$ and $\mathcal{Y}$ contains all elements that are both in $\mathcal{X}$ and $\mathcal{Y}$ and is written as

$$\mathcal{X} \cap \mathcal{Y}. \tag{2.4}$$

Additionally, it is possible to define the *difference* $\mathcal{X} \setminus \mathcal{Y}$, i.e., all elements of $\mathcal{X}$ that are not in $\mathcal{Y}$, and the *symmetric difference* $\mathcal{X} \triangle \mathcal{Y}$, i.e., all elements that are in one, but not in both sets. If all elements of $\mathcal{X}$ also belong to $\mathcal{Y}$, $\mathcal{X}$ is considered a *subset* of $\mathcal{Y}$, which is denoted by $\mathcal{X} \subseteq \mathcal{Y}$. These operations are also visualized in Figure 2.2. Finally, a set $\mathcal{X}$ is said to be *ordered* $(\mathcal{X}, \leq)$, if there exists a sequence of all elements in $\mathcal{X}$, such that $x_i \leq x_j \; \forall \; x_i, x_j \in \mathcal{X}$ [Dev12].

2.1.2 Set Functions

Since sets can have arbitrary cardinality and are usually unordered, it is not apparent how to apply vector-valued functions $f : \mathbb{R}^n \to \mathbb{R}^m$ to sets. Therefore, functions that perform a mapping from one set to another are introduced in the following. A set function $\mathcal{F}$ defines a relation or mapping from one set $\mathcal{X}$ to another set $\mathcal{Y}$, such that

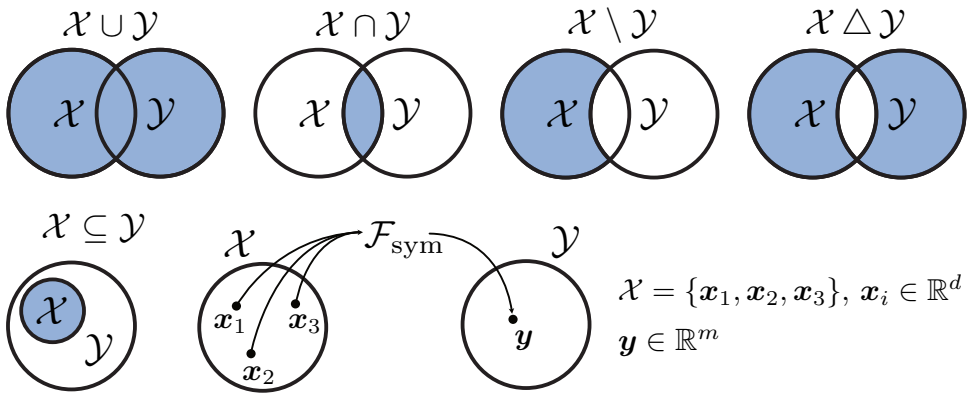


Figure 2.2: Visualization of different set operations, where each set is illustrated as a circle. The resulting set from the corresponding operation is marked in blue.

each element of $\mathcal{X}$ is assigned to at least one element of $\mathcal{Y}$

$$\mathcal{F}\colon \mathcal{X} \to \mathcal{Y}, \tag{2.5}$$

where $\mathcal{X}$ is called the *domain* of $\mathcal{F}$ and $\mathcal{Y}$ the *range* or *co-domain* of $\mathcal{F}$ [Dev12]. Depending on the type of mapping from the domain to the co-domain, different classes of set functions exist. For this thesis, however, it is only necessary to define a special set function that takes one set $\mathcal{X}$ or the the union of two or more sets $\mathcal{X}_1 \cup \cdots \cup \mathcal{X}_p$ as input, i.e., the domain, and jointly maps all elements of the input to one element of the output space, i.e., the co-domain,

$$\mathcal{F}\colon \{\mathcal{X}_1 \ \cup \ \ldots \ \cup \ \mathcal{X}_p\} \rightarrowtail \mathcal{Y}, \tag{2.6}$$

where the many-to-one mapping is denoted by $\rightarrowtail$. For simplicity, Equation (2.6) can also be written as

$$\mathcal{F}(\mathcal{X}_1, \ldots, \mathcal{X}_n) \rightarrowtail \mathcal{Y}. \tag{2.7}$$

For the general case, a multivariate function $f(x, y)$ is said to be *symmetric* on x and y, if $f(x, y) = f(y, x)$. In other words, the function is unchanged by any permutations of the input variables. Similarly, symmetric functions can also be defined on sets, where $\mathcal{F}_{\text{sym}}(\mathcal{X})$ does not depend on the order of elements in set $\mathcal{X}$. Thus, to highlight the symmetric characteristic of Equation (2.7), for the remainder of this thesis it is written as

$$\mathcal{F}_{\text{sym}}(\mathcal{X}_1, \ldots, \mathcal{X}_p) \rightarrowtail y, \quad y \in \mathcal{Y}, \tag{2.8}$$

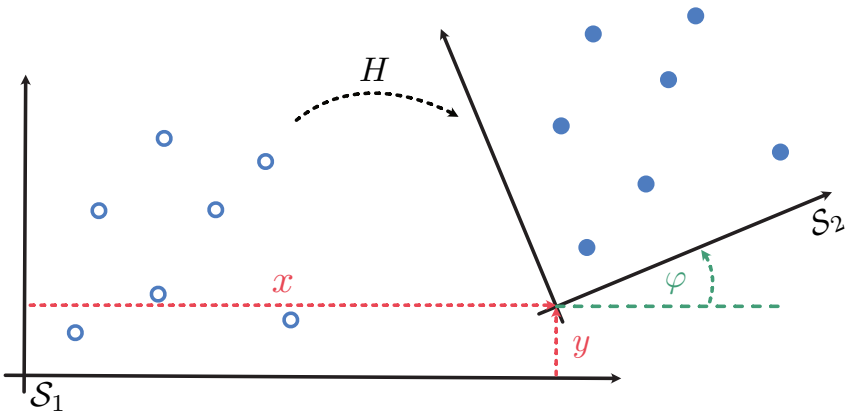


Figure 2.3: Points from the coordinate system $\mathcal{S}_1$ (○) are transformed to the coordinate system $\mathcal{S}_2$ (●) using the transformation matrix H.

with $\mathcal{F}_{\text{sym}} : \mathbb{R}^{n \times d} \to \mathbb{R}^m$. Here, d denotes the dimension of the elements in $\mathcal{X}$ and n the cardinality of $\mathcal{X}$. Later on, specific symmetric set functions will be defined for different use cases.

2.1.3 Transformations

The elements of a point cloud are points in Euclidean space where the spatial properties of each point are described by 2D or 3D coordinates. Typically, these coordinates are given in a local coordinate system, also called the frame. For example, the point cloud of a lidar sensor is usually given in the sensor frame, which has its origin in the physical center of the sensor. In some cases, it is desirable to provide the points of another sensor in the same coordinate system as the first. This process of transforming points from one coordinate system $\mathcal{S}_1$ to another $\mathcal{S}_2$ is shown in Figure 2.3 for the case of a two dimensional space, where $\mathcal{S}_2$ is translated by (x, y) and rotated by φ. In the more general case of three dimensions, arbitrary rotations between two coordinate systems can be expressed by the rotation matrix

$$R = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix}. \tag{2.9}$$

The 3×3 matrix R belongs to the special orthogonal group $SO(3)$, also called the group of rotation matrices, and satisfies the following conditions

i) $R^T R = I$,

ii) $\det R = 1$,

where I is the identity matrix and $\det R$ denotes the determinant of R [LP17]. The special matrices

$$R_x(\phi) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\phi & -\sin\phi \\ 0 & \sin\phi & \cos\phi \end{bmatrix}, \tag{2.10a}$$

$$R_y(\theta) = \begin{bmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{bmatrix}, \tag{2.10b}$$

$$R_z(\psi) = \begin{bmatrix} \cos\psi & -\sin\psi & 0 \\ \sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix}, \tag{2.10c}$$

represent a rotation around the x, y and z axis by angles ϕ, θ and ψ, respectively. It is possible to get arbitrary rotations around all axes using matrix multiplication

$$R = R_x(\phi) R_y(\theta) R_z(\psi). \tag{2.11}$$

However, it is important to note that matrix multiplications, in general, are not commutative; therefore, the order of the rotations matters, e.g., $R_x R_y R_z \neq R_y R_x R_z$.

Finally, a translation between the coordinate systems $\mathcal{S}_1$ and $\mathcal{S}_2$ is given by

$$\boldsymbol{t} = \begin{bmatrix} t_x \\ t_y \\ t_z \end{bmatrix}. \tag{2.12}$$

The transformation or mapping from a point $\boldsymbol{p}^{\mathcal{S}_1}$ defined in coordinate system $\mathcal{S}_1$ to coordinate system $\mathcal{S}_2$ is given by

$$\boldsymbol{p}^{\mathcal{S}_2} = R\boldsymbol{p}^{\mathcal{S}_1} + \boldsymbol{t}, \tag{2.13}$$

with $\boldsymbol{p}^{\mathcal{S}_1}, \boldsymbol{p}^{\mathcal{S}_2} \in \mathbb{R}^3$. Here, R and $\boldsymbol{t}$ denote the rotation and translation from the origin of the coordinate system $\mathcal{S}_1$ to $\mathcal{S}_2$. This transformation is also called an affine transformation, in other words a composition of a multiplication with an invertible matrix R and an addition with vector $\boldsymbol{t}$. For the case of two dimensional space $\mathbb{R}^2$, the special orthogonal group $SO(2)$ exists, which is a subgroup of $SO(3)$ and contains all 2×2 rotation matrices that also satisfy i) $R^T R = I$ and ii) $\det R = 1$.

The standard planar rotation matrix in $SO(2)$ has the following form

$$R(\varphi) = \begin{bmatrix} \cos\varphi & -\sin\varphi \\ \sin\varphi & \cos\varphi \end{bmatrix}, \tag{2.14}$$

and rotates the axes counterclockwise by φ as shown in Figure 2.3. Transformations of points in $\mathbb{R}^2$ are still subject to Equation (2.13) with R from Equation (2.14) and $\boldsymbol{t} \in \mathbb{R}^2$.

In order to represent arbitrary 3D transformations from Equation (2.13) using a single matrix multiplication, the 4×4 homogenous transformation matrix

$$H = \begin{bmatrix} R & \boldsymbol{t} \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.15}$$

is introduced, where $R \in SO(3)$ and $\boldsymbol{t} \in \mathbb{R}^3$ [LP17, pp. 90–91]. Equally, the homogenous transformation (2.15) is also defined for the two dimensional case with $R \in SO(2)$ from Equation (2.14) and $\boldsymbol{t} \in \mathbb{R}^2$, respectively. Points from Euclidean space are converted to homogenous coordinates by simply appending an additional dimension w. The conversion is given as follows:

$$\underbrace{\begin{bmatrix} \frac{x}{w} \\ \frac{y}{w} \\ \frac{z}{w} \end{bmatrix}}_{\text{Euclidean}} \Leftrightarrow \underbrace{\begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix}}_{\text{Homogenous}}. \tag{2.16}$$

Consequently, the point $\boldsymbol{p} = [x, y, z]^T$ is given in homogenous coordinates as $\boldsymbol{p}_H = [x, y, z, 1]^T$. Rewriting Equation (2.13) using homogenous coordinates yields

$$\boldsymbol{p}_H^{\mathcal{S}_2} = H\boldsymbol{p}_H^{\mathcal{S}_1}. \tag{2.17}$$

Finally, the inverse homogenous transformation matrix H^{-1} is given by

$$H^{-1} = \begin{bmatrix} R & \boldsymbol{t} \\ 0 & 1 \end{bmatrix}^{-1} = \begin{bmatrix} R^{-1} & -R^{-1}\boldsymbol{t} \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} R^T & -R^T\boldsymbol{t} \\ 0 & 1 \end{bmatrix}. \tag{2.18}$$

2.2 Probability Theory

This section serves as a basic introduction to probability theory to cover all necessary probabilistic methods for vehicle self-localization and current deep learning approaches described in this chapter.

The problem of vehicle self-localization is most commonly described using probabilistic formulations [Stü18; Thr02], for example, by describing the vehicle's pose as a probability distribution while considering the localization uncertainty. Moreover, many natural events, such as measurement noise and the occurrence of sensor failures, can be defined by random processes. Therefore, the introduction of proper probabilistic fundamentals is essential for this thesis.

In general, there exist two different interpretations of probability: the Frequentist view and the Bayesian view [Mur22, p. 31]. Without going into more detail, the Frequentist view explains the frequency of specific outcomes of random experiments that are performed multiple times. On the other hand, the Bayesian interpretation is used to describe the uncertainty of these outcomes. The Bayesian interpretation of probability is better suited for this thesis, considering the task of vehicle self-localization since it is desirable to model the uncertainty of the position estimate rather than describing the frequency of position occurrences.

2.2.1 Probability Distributions

In the field of probability theory and statistics, the random variable X describes a random event or experiment whose outcome, i.e., the value or realization of X, is unknown. The set of possible outcomes is denoted by $\mathcal{X}$ and is commonly known as the sample space, whereas one specific outcome is given by $x \in \mathcal{X}$.

If the number of possible outcomes, e.g., the sample space $\mathcal{X}$ is finite, the random variable X is discrete. Consequently, the random variable is continuous if the number of possible values is infinite. In summary, random variables quantify the outcomes of random experiments or events.

For the discrete case, the probability that a specific outcome occurs, i.e., $X = x$, is given by $Pr(X = x)$. Following this notation, the probability mass function (PMF), which assigns a probability to each possible value $\mathcal{X}$ is given as

$$p(x) \triangleq Pr(X = x) \tag{2.19}$$

and satisfies the following conditions

1. $0 \leq p(x) \leq 1 \; \forall \, x \in \mathcal{X}$,
2. $\sum_{x \in \mathcal{X}} p(x) = 1$.

Contrary, for a continuous random variable, where $X \in \mathbb{R}$, it is not feasible to find a finite set of possible outcomes. However, it is possible to define bounded intervals, such that the probability that the random variable lies within $[-\infty, a\,]$ is given by $Pr(X \leq a)$, which is also called the cumulative distribution function (CDF)

$$P(x) \triangleq Pr(X \leq x). \tag{2.20}$$

Cumulative distribution functions are monotonically non-decreasing functions [Mur22, pp. 33–35]. Given Equation (2.20), the probability that the random variable X lies within in the bounded interval $(a, b\,]$ follows as $Pr(a < X \leq b) = P(b) - P(a)$. The probability density function (PDF) is consequently defined as the derivative of the cumulative distribution function

$$p(x) \triangleq \frac{\mathrm{d}}{\mathrm{d}x} P(x), \tag{2.21}$$

which again satisfies the following properties

1. $p(x) \geq 0 \; \forall \, x$,
2. $\int_{-\infty}^{\infty} p(x) = 1$,

and is comparable to the PMF for the discrete case [Mur22, p. 35]. In the following, statistical moments of probability distributions are introduced. The first-order raw moment μ, which is also called the mean or expected value of the distribution, is defined for the discrete case as follows:

$$\mu \triangleq \sum_{x \in \mathcal{X}} x \cdot p(x), \tag{2.22}$$

and for continuous random variables as

$$\mu \triangleq \int_{-\infty}^{\infty} x \cdot p(x). \tag{2.23}$$

The mean μ is also known as the expected value or expectation of X, thus $\mu = \mathrm{E}\,[X]$. The second-order central moment, i.e., the variance σ^2, which describes the deviation

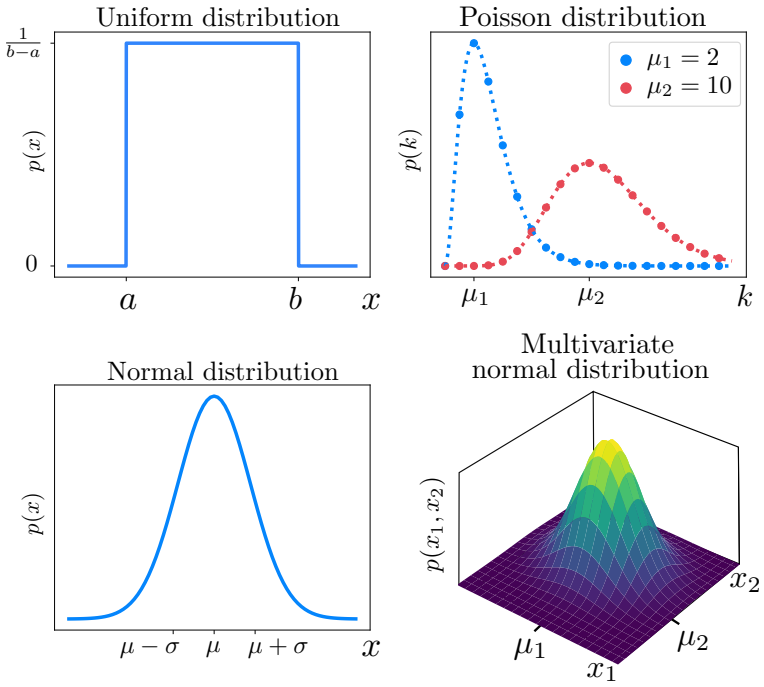


Figure 2.4: Overview of the probability distributions that are used in this thesis. For the uniform, normal and multivariate normal distribution the PDF is depicted. For the discrete poisson distribution the PMF is shown.

of the random variable from the mean, is given by

$$\sigma^2 = \mathrm{Var}\left[X\right] \triangleq \mathrm{E}\left[(X-\mu)^2\right]. \tag{2.24}$$

Furthermore, the square root of the variance $\sqrt{\sigma^2}$ is defined as the standard deviation σ [Mur22, pp. 38–39]. The remainder of this sections introduces common probability distributions that are relevant for the proposed methods in this thesis, whereas the PDF's and the PMF's are visualized in Figure 2.4.

Uniform Distribution

The uniform distribution is a symmetrical probability distribution defined on the interval $x \in [a, b]$, where each outcome has the same probability. It exists for

both discrete and continuous random variables. For the discrete case, it is often abbreviated as $\mathcal{U}\{a, b\}$ and the PMF is given by

$$\mathcal{U}\{a, b\} \triangleq p(x) = \frac{1}{n}, \tag{2.25}$$

where n denotes the amount of all possible outcomes. The mean and variance of $\mathcal{U}\{a, b\}$ follows as $\mu = (a + b)/2$ and $\sigma^2 = (n^2 - 1)/12$. The PDF for the continuous uniform distribution $\mathcal{U}_{[a,b]}$ is defined as

$$\mathcal{U}_{[a,b]} \triangleq p(x) = \begin{cases} \frac{1}{b-a} & \text{for } x \in [a, b], \\ 0 & \text{otherwise.} \end{cases} \tag{2.26}$$

Similarly, the mean and variance for $\mathcal{U}_{[a,b]}$ are $\mu = (a + b)/2$ and $\sigma^2 = (b - a)^2/12$, respectively.

Poisson Distribution

The poisson distribution is a discrete probability distribution and is usually applied to model the probability that a certain number of independent events occur in a fixed interval of time or space. The PMF is given as follows:

$$\text{Pois}(k; \lambda) \triangleq p(k) = \frac{\lambda^k e^{-\lambda}}{k!}, \tag{2.27}$$

where $k \in \mathbb{N}$ is the number of occurrences of the event and the average number of occurrences is given by $\lambda > 0$. Here, it is important to note that the mean of the poisson distribution is equal to the variance, i.e., $\lambda = \mathrm{E}[X] = \mathrm{Var}[X]$.

Normal Distribution

The normal or Gaussian distribution is one of the most commonly used probability distributions, as the normal distribution describes many natural occurrences. It is only defined for the continuous case and is completely described by its first-order raw moment, i.e., the mean μ, and the second-order central moment, i.e., the variance σ^2. Therefore, the normal distribution is denoted by $\mathcal{N}(x; \mu, \sigma)$ and the PDF is given as follows:

$$\mathcal{N}(x; \mu, \sigma) \triangleq p(x) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(x - \mu)^2}{2\sigma^2}\right). \tag{2.28}$$

For readability purposes, the variable x is often omitted from Equation (2.28) for the remainder of this thesis, i.e., $\mathcal{N}(\mu, \sigma)$.

Multivariate Random Variables and Distributions

Previously, only univariate probability distributions were considered, that is, distributions that depend on one random variable X. However, it is also possible to define multivariate distributions that consider l random variables that are combined as a l-dimensional vector

$$\boldsymbol{X} = [X_1, \ldots, X_l]^T, \tag{2.29}$$

where each $X_i, i \in \{1, \ldots, l\}$ denotes one univariate random variable as previously introduced. Consequently, the expected value or mean $\boldsymbol{\mu} \in \mathbb{R}^l$ is given by

$$\mathrm{E}[\boldsymbol{X}] = \boldsymbol{\mu} = [\mathrm{E}[X_1], \ldots, \mathrm{E}[X_l]]^T. \tag{2.30}$$

Following the definition of the variance from Equation (2.24), the covariance between two random variables X_i and X_j, which denotes the joint variability, is defined as follows:

$$\mathrm{Cov}(X_i, X_j) = \mathrm{E}[(X_i - \mathrm{E}[X_i])(X_j - \mathrm{E}[X_j]], \tag{2.31}$$

where the covariance between one random variable and itself is given by its variance. Finally, following Equation (2.31), the covariance matrix $\Sigma \in \mathbb{R}^{l \times l}$ can be constructed, where each element of Σ is given by $\Sigma_{ij} = \mathrm{Cov}(X_i, X_j)$.

For multivariate random variables, the normal distribution from Equation (2.28) extents to

$$\mathcal{N}(\boldsymbol{x}; \boldsymbol{\mu}, \Sigma) \triangleq p(\boldsymbol{x}) = \frac{1}{\sqrt{(2\pi)^l |\Sigma|}} \exp\left(-\frac{1}{2}(\boldsymbol{x} - \boldsymbol{\mu})^T \Sigma^{-1} (\boldsymbol{x} - \boldsymbol{\mu})\right). \tag{2.32}$$

2.2.2 Information Theory

This section briefly introduces core concepts of information theory necessary for the remainder of this thesis. Information theory deals with compressing data such that it is both compact and robust with respect to errors by employing error correction capabilities. A more in-depth introduction to information theory can be found in [Cov99].

Entropy

The entropy of a probability distribution is a measure of the uncertainty of a random variable [Mur12, Chapter 2.8.1]. However, entropy can also be defined as the information content of data [Mur22, Chapter 6.1]. Specifically, given a discrete random variable X that has K possible outcomes with probability distribution p, the entropy of p is defined as follows:

$$\mathbb{H}(X) = -\sum_{k=1}^{K} p(k) \log p(k). \tag{2.33}$$

The extension to the continuous case is called *limiting density of discrete points* [Ros12], but is not considered further for this thesis.

Similarity Measures

Suppose two discrete probability distributions p and q with random variables X and Y and K possible outcomes are given. In that case, it is possible to define the similarity or dissimilarity between the two distributions using the so-called Kullback-Leibler divergence (KL divergence) [Mur12, pp. 57–58], which is also known as relative entropy

$$\begin{aligned} \mathbb{KL}(p \,||\, q) &= \sum_{k=1}^{K} p(k) \log \frac{p(k)}{q(k)} \\ &= \sum_{k=1}^{K} p(k) \log p(k) - \sum_{k=1}^{K} p(k) \log q(k) \\ &= \mathbb{H}(X) - \mathbb{H}(X, Y), \end{aligned} \tag{2.34}$$

where $\mathbb{H}(X, Y)$ is called the cross entropy between X and Y. Note that the KL divergence is also defined for the continuous case. Evidently, the KL divergence is zero, if $p = q$ and $\mathbb{KL}(\cdot \,||\, \cdot) \geq 0$. Here, it is important to note that the KL divergence is not a distance measure, since it is asymmetric, i.e., $\mathbb{KL}(p \,||\, q) \neq \mathbb{KL}(q \,||\, p)$. However, it is possible to formulate a symmetric version of the KL divergence

$$JS(p, q) = 0.5 \cdot \mathbb{KL}(p \,||\, m) + 0.5 \cdot \mathbb{KL}(q \,||\, m), \tag{2.35}$$

which is called the Jensen-Shannon divergence with $m = 0.5 \cdot p + 0.5 \cdot q$.

2.3 Vehicle Self-Localization

In the field of automated driving and driver assistance systems, obtaining semantic information from high-definition digital maps increases the reliability and safety of important software components, for example, environment perception, situational understanding, and path planning [GDD18]. This is usually done by enriching the perceived environment with data from the digital map. Usually, the map contains information about the road geometry, such as lanes and curbstones. However, it is also common to include road logic, such as speed limits, traffic signs, right of way rules, and other traffic regulations. High-definition, in this case, means that the geometric information for each object (the position, orientation, and the spatial extent) are provided with high accuracy, usually in the range of a few centimeters. In highly dynamic scenarios that are difficult to understand, this information can reinforce scene and environment understanding. For example, other road participants may obscure traffic signs or road markings at busy intersections in urban scenarios. In these situations, it is difficult to infer the correct traffic rules. However, if the location of the ego vehicle inside the digital map is known, the missing information can be extracted and integrated into existing algorithms [GPBD20]. Therefore, the goal of self-localization is to infer the vehicle's position and orientation, which is also called the pose [Thr02, p. 17], within a given map.

In the literature, different approaches exist that try to solve the localization task, as it is one of the most prominent problems in robotics. A prevalent distinction is classifying localization methods by the availability of digital maps. Since the localization result is usually obtained relative to the map, most approaches assume an a-priori map created independently, which is available to the localization algorithm at run time. However, it is possible to perform the map creation concurrently while localizing in an unknown environment, which is known as simultaneous localization and mapping (SLAM) [Thr02, Chapter 10]. Furthermore, maps can be represented in various ways. Two general approaches include grid-based maps and landmark-based or feature-based maps. As the name suggests, grid map-based maps discretize the environment into equally-sized cells. Then, a cell is marked as occupied if a static object is perceived at that spatial location and empty otherwise. It is apparent that the accuracy of this approach is subject to the cell dimensions, which in turn relates to the memory and computational requirements. Moreover, due to the natural sparsity of static objects, this also creates a significant overhead since empty cells need to be considered as well [ED20]. Consequently, methods were proposed to tackle the aforementioned problems by employing efficient implementations that take the sparsity into account. Unlike grid-based maps, landmarks describe the position of easily recognizable and static objects with additional features, such as the spatial extent or the type. Then, each landmark can be stored efficiently inside the digital map. Therefore, this thesis deals with localization approaches that require

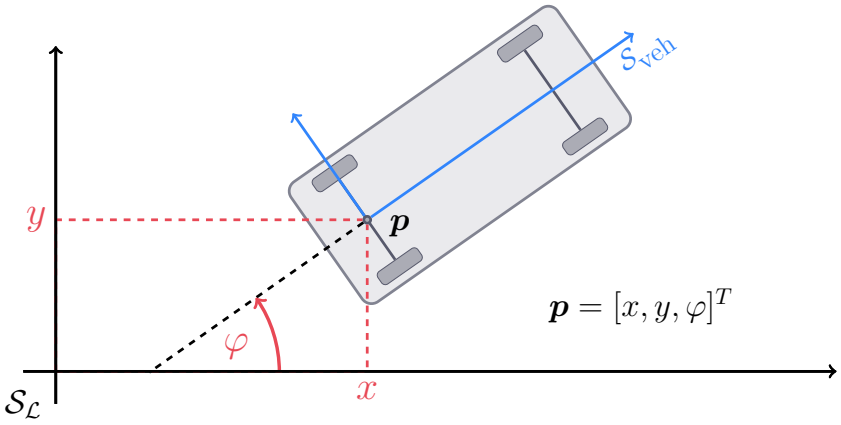


Figure 2.5: Definition of the vehicle pose in the map frame $\mathcal{S}_{\mathcal{L}}$. The pose is defined with respect to the center of the vehicle's rear axle, which acts as the origin of the vehicle coordinate system $\mathcal{S}_{\text{veh}}$.

an a-priori digital map, more precisely, landmark-based maps, which will also be introduced in the subsequent sections.

2.3.1 Basics

In this thesis, the problem of vehicle self-localization is stated in two-dimensional space; thus, the pose is given by

$$\boldsymbol{p} = \begin{bmatrix} x \\ y \\ \varphi \end{bmatrix}, \tag{2.36}$$

where x and y denote the vehicle's position inside a given coordinate system $\mathcal{S}_{\mathcal{L}}$ and φ denotes the orientation, which is also known as the heading or yaw. Usually, the position is specified with respect to the center of the vehicle's rear axle, and the orientation denotes the counterclockwise angle between the x-axis of the coordinate systems and the moving direction, as shown in Figure 2.5. Additionally, Equation (2.36) can easily be extended to three dimensional space. In that case, there are six degrees of freedom, i.e., x, y, z, and a rotation around each axis φ_x, φ_y, and φ_z, respectively. In the following, general assumptions that are required for the localization task are outlined:

- The digital map $\mathcal{L}$ with coordinate system $\mathcal{S}_\mathcal{L}$ already exists and is provided to the localization algorithm and contains static and easily recognizable objects (landmarks).

- The initial pose $\boldsymbol{p}_\text{init}$ within the map $\mathcal{L}$ is known; however, this pose may be faulty or noisy.

- The true pose, or ground-truth, which is denoted by $\boldsymbol{p}_\text{GT}$, is usually not available for inference but can be obtained for evaluation purposes.

- The local vehicle coordinate system, which has its origin at the rear axle, is denoted by $\mathcal{S}_\text{veh}$, as shown in Figure 2.5.

- The algorithms are required to meet real-time conditions, where the total processing time lies in the range of about 100 ms.

- Localization is subject to environmental effects that interfere with the localization accuracy, such as clutter, missed detections, and measurement noise. These effects will be explained later on in detail.

Considering these aspects, self-localization can be defined as follows: Given an initial pose $\boldsymbol{p}_\text{init}$, the goal is to find a pose estimate $\hat{\boldsymbol{p}}$ within the digital map $\mathcal{L}$, that is as close to the ground-truth $\boldsymbol{p}_\text{GT}$ as possible. Following the terminology introduced in [Thr02], this is also called a *local localization problem*, since the localization error is confined to a local region around the ground-truth. Consequently, the initial localization error is assumed as

$$\Delta\boldsymbol{p}_\text{init} = \boldsymbol{p}_\text{GT} - \boldsymbol{p}_\text{init} = [\Delta x,\, \Delta y,\, \Delta\varphi]^T \tag{2.37a}$$

and the localization error of the pose estimate $\hat{\boldsymbol{p}}$ is given by

$$\Delta\hat{\boldsymbol{p}}^* = \boldsymbol{p}_\text{GT} - \hat{\boldsymbol{p}} = [\Delta\hat{x}^*,\, \Delta\hat{y}^*,\, \Delta\hat{\varphi}^*]^T, \tag{2.37b}$$

which is visualized in Figure 2.6. It is apparent, that there are two coordinate systems involved in the localization task, the map coordinate system or map frame $\mathcal{S}_\mathcal{L}$, and the vehicle coordinate system $\mathcal{S}_\text{veh}$. Therefore, the localization task can also be formulated as a coordinate transformation [Thr02, p. 157] by estimating the correspondence between $\mathcal{S}_\text{veh}$ and $\mathcal{S}_\mathcal{L}$ using the homogenous transformation

$$H = \begin{bmatrix} \cos(\varphi) & -\sin(\varphi) & x \\ \sin(\varphi) & \cos(\varphi) & y \\ 0 & 0 & 1 \end{bmatrix}. \tag{2.38}$$

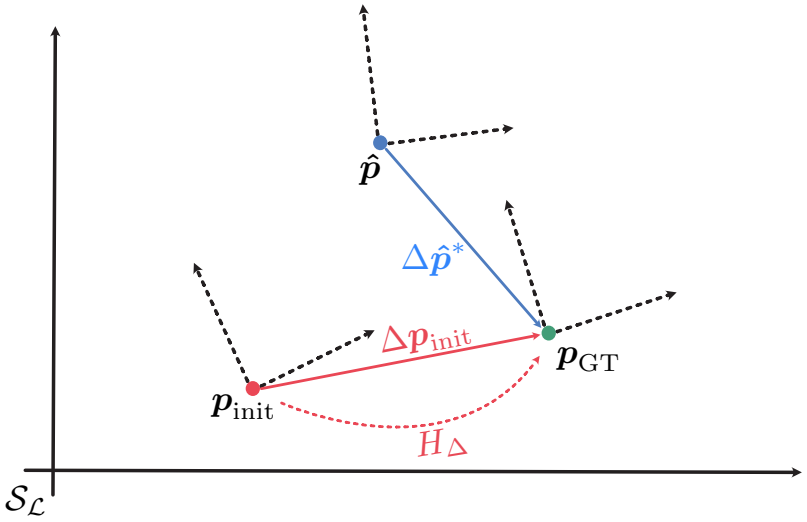


Figure 2.6: Visualization of the dependencies between the initial pose estimate $\boldsymbol{p}_{\text{init}}$, the ground-truth $\boldsymbol{p}_{\text{GT}}$, and the estimated pose $\hat{\boldsymbol{p}}$. The initial pose error is denoted by $\Delta\boldsymbol{p}_{\text{init}}$ and the estimated pose error is denoted by $\Delta\hat{\boldsymbol{p}}^*$. Here, the goal is to find the transformation H_Δ, that projects the pose from the initial guess to the ground-truth. The vehicle frames at the denoted points are pictured as dashed lines.

When an initial pose is available, this simplifies to

$$H_\Delta = \begin{bmatrix} \cos(\Delta\varphi) & -\sin(\Delta\varphi) & \Delta x \\ \sin(\Delta\varphi) & \cos(\Delta\varphi) & \Delta y \\ 0 & 0 & 1 \end{bmatrix}. \tag{2.39}$$

2.3.2 Landmark-based Localization

Until now, the digital map was considered available for the localization task. This section serves as a short introduction to the general structure of digital maps and how maps can be used to estimate the vehicle's pose. As previously mentioned, this thesis only considers landmark-based maps, whereas a landmark is defined as a generic low or high-level representation of an easily recognizable, static, and persistent object in the environment, such as traffic lights, traffic signs, road markings,

or vegetation. Furthermore, the landmarks are assumed to be recorded using high-definition reference sensors, such as real-time kinematic (RTK) systems with differential GPS (dGPS), that improve the accuracy of GPS measurements. This is usually done by relying on reference ground stations that wirelessly provide correction data for standard GPS signals.

The landmarks are recorded during the map creation process using one or more sensors, for example by using images from monocular or stereo cameras or range measurements from radar or lidar sensors. Optionally, the raw measurements are further processed, e.g., by using clustering algorithms, which is explained in subsequent chapters. In the following, the landmark-based localization is framed for the two dimensional case; however, it can easily be extended to the 3D case by appending a third spatial dimension. Consequently, a landmark $\boldsymbol{l}$ is associated with at least a 2D position and P additional features f^p

$$\boldsymbol{l} = [x,\, y,\, f_1^p,\, \ldots,\, f_P^p]^T, \quad \boldsymbol{l} \in \mathbb{R}^{2+P}, \tag{2.40}$$

where the features could be, for example, an object class or the orientation or spatial extent of the landmark. Thus, a landmark-based map can be described as an unordered set of arbitrary landmarks

$$\mathcal{L} = \{\boldsymbol{l}_1, \ldots, \boldsymbol{l}_\nu\}, \tag{2.41}$$

where ν denotes the cardinality of the map, i.e., the amount of landmarks, and every $\boldsymbol{l}_i$ belongs to one static object in the environment. In most cases, the position of each landmark $\boldsymbol{l}_i$, $i \in \{1, \ldots, \nu\}$ is given in global coordinates, for example in cartesian coordinates, e.g., the Universal Transverse Mercator (UTM) coordinate system, or geodetic coordinates using longitude, latitude and elevation information.

During localization, the goal is to infer the vehicle's pose inside the map frame $\mathcal{S}_\mathcal{L}$ by comparing the currently observed environment (from multi-modal sensors) with the landmarks from the digital map. The environment is described by preprocessed sensor measurements, that again have at least positional information with Q additional features f^q

$$\boldsymbol{m} = [x,\, y,\, f_1^q,\, \ldots,\, f_Q^q]^T, \quad \boldsymbol{m} \in \mathbb{R}^{2+Q}. \tag{2.42}$$

The entirety of the measurements is then given by the set

$$\mathcal{M} = \{\boldsymbol{m}_1, \ldots, \boldsymbol{m}_\pi\}. \tag{2.43}$$

Again, measurements can also be described in three-dimensional space, as explained above. Since landmarks are by definition recognizable and persistent, the goal is to associate and match the measurements to the landmarks, such that the vehicle's

pose can be derived. Therefore, the general landmark-based self-localization task f_{loc} is defined as follows:

$$f_{\text{loc}}(\mathcal{M}, \mathcal{L}) \rightarrowtail \hat{\boldsymbol{p}}, \tag{2.44}$$

where $f_{\text{loc}} : (\mathbb{R}^{\pi \times 2+Q}, \mathbb{R}^{\nu \times 2+P}) \rightarrow \mathbb{R}^3$ is required to be a symmetric set function if $\mathcal{M}$ and $\mathcal{L}$ are unordered sets. Then, the overall objective is to minimize the localization error according to some error metric, e.g., the L2-Norm

$$\min_{\hat{\boldsymbol{p}}} \|\boldsymbol{p}_{\text{GT}} - \hat{\boldsymbol{p}}\|^2, \tag{2.45}$$

with $\hat{\boldsymbol{p}}$ from Equation (2.44). This task can also be described as a point-to-point registration problem whereas Equation (2.45) is solved by minimizing the distances between the landmarks and measurements [Thr02].

The map creation approach, i.e., generating landmarks using high-precision reference systems, is explained in the following chapters. Additionally, obtaining processed and clustered multi-modal sensor measurements is further outlined. The remainder of this thesis deals with finding methods to approximate the localization function f_{loc} (2.44) and obtain an accurate pose estimate $\hat{\boldsymbol{p}}$.

2.4 Deep Learning

This section serves as an overview of the fundamentals of deep learning required for the remainder of this thesis. For that, the basics of neural networks and their respective algorithms are introduced and explained.

2.4.1 Basics

Deep Learning, in general, is a data-driven subtask of machine learning, in which a computer is taught to perform one or more tasks based on a training data distribution to generalize to unseen data samples. For this, artificial neural networks (ANN's) are employed to mimic the human brain using multiple computational units, i.e., artificial neurons. The neurons are interconnected and can pass weighted data signals. Similar to real biological neurons, artificial neurons have an activation threshold (or activation function) and fire depending on the weighted sum of all input connections. The multilayer perceptron (MLP) is comprised of various artificial neurons that are arranged in layers. In 1989, Hornik et al. [HSW89] proved that the multilayer

network architecture with non-linear activations can approximate arbitrary functions

$$F : \mathbb{R}^N \rightarrow \mathbb{R}^M, \tag{2.46}$$

which is known as the universal approximation theorem. Thus, a deep neural network H with the set of learnable parameters $\boldsymbol{\theta}$ that approximates F is given by

$$H_{\boldsymbol{\theta}} \approx F. \tag{2.47}$$

This discovery, in conjunction with the improvement of computational processing power in recent years and the availability of training data, led to a rapid increase of deep learning applications that exceeded previous traditional achievements, such as playing chess [SHS+18] or Go [SHM+16] and solving the protein folding challenge [SEJ+20]. In this thesis, the advances in deep learning methods are applied to point cloud processing and the task of vehicle self-localization for automated driving applications. Thus, only the fundamentals required to explain these methods are provided in this chapter. A comprehensive and in-depth study of neural networks can be found in, e.g., [Agg18; GBC16; Nie15].

In general, the process of applying deep learning applications can be described as follows: Using the aforementioned training dataset $\{(\boldsymbol{x}_1, \boldsymbol{y}_1), \ldots, (\boldsymbol{x}_n, \boldsymbol{y}_n)\}$, which is comprised of the input samples $\boldsymbol{x}_i$ and the corresponding true labels $\boldsymbol{y}_i$, a set of parameters $\boldsymbol{\theta}$ is adapted according to an optimization algorithm with regard to a loss l, that describes the difference between the network's prediction $H_{\boldsymbol{\theta}}(\boldsymbol{x}_i) = \hat{\boldsymbol{y}}_i$ and the ground-truth $\boldsymbol{y}_i$. The loss function $L : (H_{\boldsymbol{\theta}}(\boldsymbol{x}_i), \boldsymbol{y}_i) \rightarrow \mathbb{R}^1$ returns the penalty or loss l, that is, the model error for the current parameter set $\boldsymbol{\theta}$. Obviously, the goal is to minimize l, which is usually done by adapting the parameter set $\boldsymbol{\theta}$ according to an update rule, e.g., the stochastic gradient descent (SGD) [Ros58]

$$\boldsymbol{\theta}_t \leftarrow \boldsymbol{\theta}_{t-1} - \eta \cdot \nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}, H_{\boldsymbol{\theta}}(\boldsymbol{x}_i), \boldsymbol{y}_i), \tag{2.48}$$

where η denotes the learning rate and $\nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}, \cdot)$ the gradient of the cost or loss function L with respect to the parameters $\boldsymbol{\theta}$. Without going into more detail, this is done using backpropagation [RHW86], which can be divided into two parts: 1) A forward phase where the input is passed through the network and the predicted output is compared to the ground-truth using the loss or objective function L. 2) The backward phase computes the derivative of the loss with respect to the predicted output. Then, the gradients are propagated backward to update the network's weights using the chain rule of differential calculus [Agg18, Chapter 1.3]. These two processes are part of the training process, where the entire training dataset is used to optimize the network, which is also called one epoch. Afterward, the generalization capabilities of the current parameter set are usually evaluated by processing an unseen test dataset. One possibility to adapt the performance of a deep neural network besides fundamentally changing the network architecture is to adapt

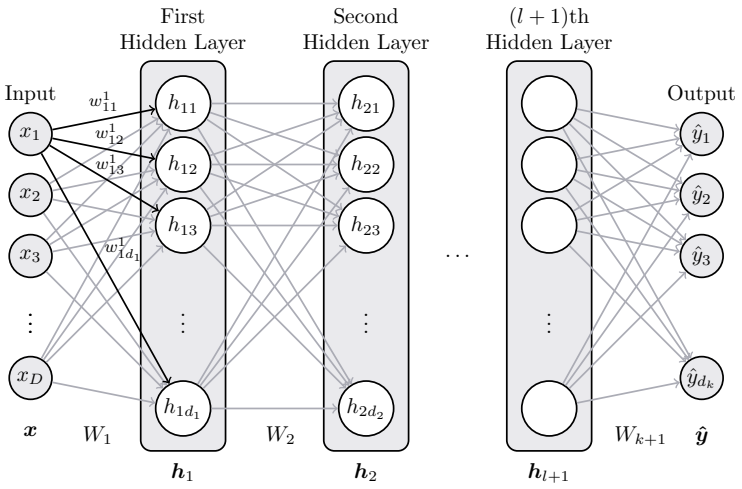


Figure 2.7: Multilayer perceptron network architecture. Each grey circle denotes one input and output neuron, respectively. The white circles, arranged in layers, represent a perceptron where the output is given by the sum of all input connections weighted by the learnable parameter matrix W_i. Additionally, an activation function $\sigma(\cdot)$ is applied to induce non-linearities.

hyperparameters. These model parameters are chosen prior to the training process to control the learning process. Finally, a popular training method simultaneously processes multiple samples from the training dataset and aggregates the gradients during the forward and backward pass. Then, the optimization step is performed using averaged gradients to obtain a precise estimate of the real gradients. This process is also known as the mini-batch principle, where B usually denotes the batch size [GBC16, pp. 275–279].

2.4.2 Multilayer Perceptron

The multilayer perceptron (MLP) shown in Figure 2.7 is the most fundamental network architecture. As previously explained, it is comprised of multiple computational units, also called perceptrons, with D input connections x_i, that are each multiplied with the corresponding edge weight w^k_{ij}, where k denotes the current layer. The

predicted output of one perceptron $\hat{y}_p$ is then given by

$$\hat{y}_p = \sigma\left(\sum_{j=1}^{d} w_j x_j + b\right) = \sigma(\boldsymbol{w} \cdot \boldsymbol{x} + b), \tag{2.49}$$

with the non-linear activation function $\sigma(\cdot)$ and the bias term b. To increase readability, the bias term b is omitted for the remainder of this thesis. Following this notation, the multilayer perceptron with k hidden layers is introduced as follows

$$\mathrm{MLP}(D \,|\, d_1, \ldots, d_k) : \begin{cases} \boldsymbol{h_1} = \sigma(W_1^T \boldsymbol{x}), \\ \boldsymbol{h_{l+1}} = \sigma(W_{l+1}^T \boldsymbol{h_l}) \quad \forall\, l \in \{1, \ldots, k-1\}, \\ \hat{\boldsymbol{y}} = \sigma(W_{k+1}^T \boldsymbol{h_k}), \end{cases} \tag{2.50}$$

where the output of the first hidden layer is given by $\boldsymbol{h_1}$, the hidden to hidden layers by $\boldsymbol{h_{l+1}}$ and the hidden to output layer by $\hat{\boldsymbol{y}}$ [Agg18, pp. 5–8]. The weight matrix W_1 between the input $\boldsymbol{x} \in \mathbb{R}^D$ and the first hidden layer has dimensions $D \times d_1$, the weight matrix W_l between the lth hidden layer and the $(l+1)$th hidden layer has dimensions $d_l \times d_{l+1}$. The weight matrix for the final output layer W_{k+1} is of size $d_{k-1} \times d_k$. Thus, D denotes the input dimension and d_k the output dimension.

2.4.3 Convolutional Neural Network

Similar to how perceptrons mimic the brain by modeling the interconnections between neurons, convolutional neural networks (CNN) were inspired by how the visual cortex processes input signals [HW68]. The neurons respond to different overlapping areas of the visual field, also known as the receptive field. As the name suggests, convolutional neural networks model these receptive fields for image processing using the convolution operation instead of simple matrix multiplication

$$\boldsymbol{y}(t) = \int \boldsymbol{x}(a)\boldsymbol{k}(t-a)da = (\boldsymbol{x} * \boldsymbol{k})(t), \tag{2.51}$$

where again $\boldsymbol{x}$ is the input and $\boldsymbol{k}$ is called the kernel or feature map [GBC16, Chapter 9]. Most of the time, digital input data is processed; thus, the discrete convolution is defined as follows:

$$\boldsymbol{y}(l) = (\boldsymbol{x} * \boldsymbol{k})(l) = \sum_{a=-\infty}^{\infty} \boldsymbol{x}(a)\boldsymbol{k}(l-a). \tag{2.52}$$

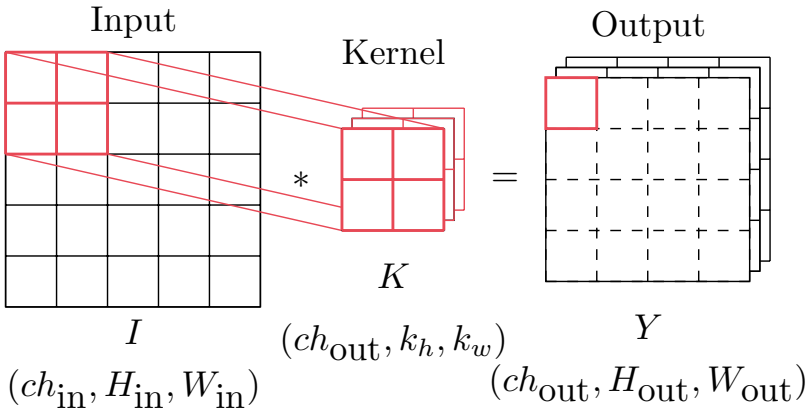


Figure 2.8: Illustrated example of a single convolutional neural network layer, where the kernel is applied over the input image with stride $s = 1$. Note, that the activation function is not depicted in this overview.

Moreover, since an image is usually provided as a 2D matrix (with one channel for grayscale images, and three channels for RGB images), the convolution (2.52) for a two-dimensional discrete input I and kernel K, which is shown in Figure 2.8, is given by

$$Y(i,j) = (I * K)(i,j) = \sum_m \sum_n I(m,n)K(i-m, j-n). \tag{2.53}$$

Usually, the convolution operation (2.53) is implemented using the related cross-correlation

$$Y(i,j) = (I * K)(i,j) = \sum_m \sum_n I(i+m, j+n)K(m,n), \tag{2.54}$$

where the neural network can optimize the parameters of the discrete kernel K. In other words, a learnable kernel is shifted over the input image where the kernel values $K(m,n)$ are multiplied with the corresponding intensity values $I(i+m, j+n)$. In terms of neural network design, the size of the 2D kernel can be chosen $k = (k_h, k_w)$. Here, k_h denotes the kernel height and k_w the kernel width. Additionally, it is possible to change the amount the kernel is shifted each time step, also known as the stride s. Moreover, multiple independent kernels ch_{out} can be deployed to increase the network's capacity and allow the network to learn from different subspaces. The same set of hyperparameters extends to the case of 3D convolutions. A convolutional

layer can be described by these three hyperparameters and is formally defined by

$$\mathrm{CNN}(ch_{\mathrm{in}}, ch_{\mathrm{out}}, (k_h, k_w), s) : Y(ch^j_{\mathrm{out}}) = \sigma\left(b(ch^j_{\mathrm{out}}) + \sum_{i=1}^{ch_{\mathrm{in}}} w^j(i) * I(i)\right), \quad (2.55)$$

where $\sigma(\cdot)$ is again a non-linear activation function, b is a bias term, $i \in \{1, \ldots, ch_{\mathrm{in}}\}$, $j \in \{1, \ldots, ch_{\mathrm{out}}\}$, $w^j(i) \in \mathbb{R}^{k_h \times k_w}$ is the learnable kernel, $I(i) \in \mathbb{R}^{H_{\mathrm{in}} \times W_{\mathrm{in}}}$ is the input image where H_{in} and W_{in} represent the image height and width respectively, and ch_{in} denotes the number of input channels, e.g., $ch_{\mathrm{in}} = 1$ for grayscale images and $ch_{\mathrm{in}} = 3$ for RGB images. If $ch_{\mathrm{in}} > 1$, it is possible to either use a one dimensional kernel for every input dimension or a volumetric kernel that is applied over multiple input channels. However, for simplicity, these cases are treated equally in this work. Following this notation, given an input image of size $(ch_{\mathrm{in}}, H_{\mathrm{in}}, W_{\mathrm{in}})$, the output after applying the convolution layer (2.55) is of size $(ch_{\mathrm{out}}, H_{\mathrm{out}}, W_{\mathrm{out}})$, with

$$H_{\mathrm{out}} = \left\lfloor \frac{H_{\mathrm{in}} - k_h + 2 \cdot p_h}{s} + 1 \right\rfloor, \quad (2.56a)$$

$$W_{\mathrm{out}} = \left\lfloor \frac{W_{\mathrm{in}} - k_w + 2 \cdot p_w}{s} + 1 \right\rfloor, \quad (2.56b)$$

where $\lfloor \cdot \rfloor$ denotes the floor function and p_h, p_w are additional padding parameters, that are set to zero for the remainder of this thesis. The advantage of convolutional neural networks compared to simply applying a multilayer perceptron to an image is the sparse number of learnable parameters, i.e., interactions between the input unit and the output unit, which is achieved by making the kernel smaller than the input image. This helps the network concentrate on meaningful features like edges or color spots. Additionally, the kernel parameters, i.e., the learnable weights, are shared across the entire image, which increases statistical efficiency and drastically reduces memory requirements [GBC16, pp. 324–328]. Furthermore, an extensive introduction to convolutional neural networks can be found in [Agg18; GBC16].

2.4.4 Row-wise Feedforward Network

In the following, the row-wise feedforward network (rFF) is introduced [QSMG17], which tries to adapt the strengths of convolutional neural networks to input data of arbitrary length. It is primarily used for unordered two dimensional input that has a fixed dimension, for example a point cloud $\mathcal{P}$ with N individual points $\mathcal{P} = \{\boldsymbol{p}_1, \ldots, \boldsymbol{p}_N\}, \boldsymbol{p}_i \in \mathbb{R}^D$. Since point clouds are unstructured, a multilayer perceptron with $N \times D$ input neurons can not directly be applied since it is not permutation-invariant. However, if the input point dimension is known, e.g., $D = 3$

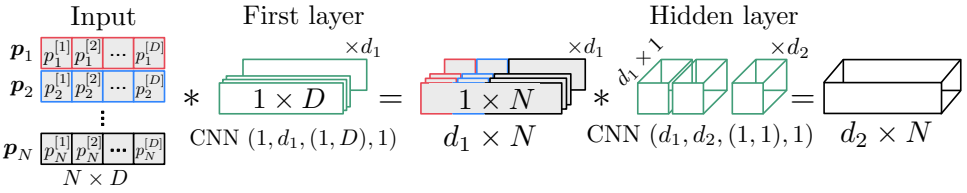


Figure 2.9: Simplified illustration of a row-wise feedforward network $\mathrm{rFF}(D \,|\, d_1, d_2)$ using CNN modules, since convolutions are efficiently implemented in common deep learning frameworks. By adapting the hyperparameters of each layer according to the input point cloud, this imitates a multilayer perceptron that processes each point individually but shares its weights for all input points. The resulting network is thus invariant to the number of input points and not dependent on the ordering of the input.

for a three-dimensional point cloud, a multilayer perceptron with D input neurons can be employed that processes each point individually with the goal of projecting all points to latent dimension d_k. Consequently, the learnable weights are shared across all points $\boldsymbol{p}_i$, $i \in \{1, \ldots, N\}$. When the points are arbitrarily arranged row-wise

$$\mathcal{P} = \begin{bmatrix} \boldsymbol{p}_1 \\ \boldsymbol{p}_2 \\ \vdots \\ \boldsymbol{p}_N \end{bmatrix} = \begin{bmatrix} p_1^{[1]} & p_1^{[2]} & \ldots & p_1^{[D]} \\ p_2^{[1]} & p_2^{[2]} & \ldots & p_2^{[D]} \\ & \vdots & & \\ p_N^{[1]} & p_N^{[2]} & \ldots & p_N^{[D]} \end{bmatrix}, \mathcal{P} \in \mathbb{R}^{N \times D}, \tag{2.57}$$

a more effective implementation is to use a 2D convolutional neural network with a kernel of size $k = (1, D)$ and stride $s = 1$, since most deep learning frameworks implement the convolution operation more efficiently. Subsequently, a row-wise feedforward network with k layers is defined as follows:

$$\mathrm{rFF}(D \,|\, d_1, \ldots, d_k) : \begin{cases} H_1 = \mathrm{CNN}(1, d_1, (1, D), 1), \\ H_{l+1} = \mathrm{CNN}(d_l, d_{l+1}, (1, 1), 1) \quad \forall\, l \in \{1, \ldots,\ k-2\}, \\ Y = \mathrm{CNN}(d_{k-1}, d_k, (1, 1), 1). \end{cases} \tag{2.58}$$

It becomes apparent, that the row-wise feedforward network independently projects each point to latent space $\mathrm{rFF} : \mathbb{R}^{N \times D} \to \mathbb{R}^{N \times d_k}$ with shared weights. Moreover, the output dimension can be controlled by the number of kernels for each layer, i.e., ch_{out}, which is shown in Figure 2.9. This implementation can handle arbitrary point set cardinality and is not dependent of the arrangement of points in the matrix (2.57).

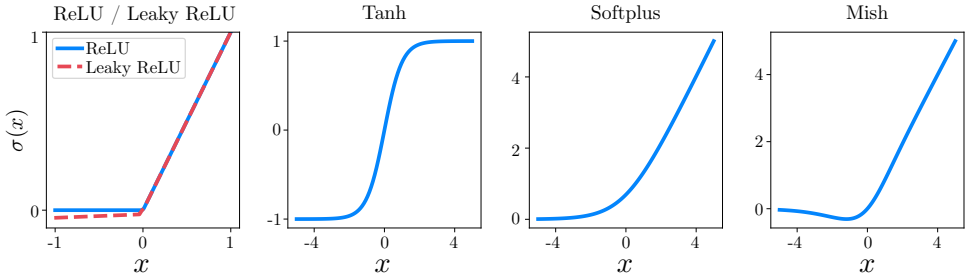


Figure 2.10: Overview of common non-linear activation functions.

Activation Functions

This section serves as a short overview of a selection of activation functions used in this thesis. Activation functions are used to introduce non-linearities into the deep learning model. It can be shown that h hidden layers, where every layer directly returns the result of the input multiplied by the layer weights, e.g., $\hat{\boldsymbol{y}}_i = \boldsymbol{w}_i \cdot \boldsymbol{x} + b_i, i \in \{1, \ldots, h\}$, is just a composition of affine transformations of all layers. This transformation can then be represented by a single affine transformation [GBC16, Chapter 6]. Therefore, non-linear activation functions are employed after each layer. Most of the presented activation functions are additionally visualized in Figure 2.10. Following the notation introduced for the definition of the multilayer perceptron (2.50) and the CNN (2.55) the most popular activation function, the rectified linear unit (ReLU), is given by

$$\sigma_{\text{ReLU}}(\boldsymbol{x}) = \max(0, \boldsymbol{x}), \tag{2.59}$$

which is applied to the input $\boldsymbol{x}$ element-wise. It is apparent, that the ReLU activation function only transmits positive values, or zero if $\boldsymbol{x} \leq 0$. If negative values are supposed to be considered as well, Equation (2.59) can be extended to allow negative inputs as follows:

$$\sigma_{\text{LeakyReLU}}(\boldsymbol{x}) = \max(0, \boldsymbol{x}) + \alpha \cdot \min(0, \boldsymbol{x}), \tag{2.60}$$

which is called the LeakyReLU activation function. The slope α is usually set to a small number, e.g., $\alpha = 1e{-}2$. Another widely used activation function is the hyperbolic tangent function, which is given by

$$\sigma_{\text{Tanh}}(\boldsymbol{x}) = \tanh(\boldsymbol{x}) = \frac{\exp(\boldsymbol{x} - \exp(-\boldsymbol{x})}{\exp(\boldsymbol{x} + \exp(-\boldsymbol{x})}, \tag{2.61}$$

that maps the input to the interval $[-1, 1]$. Next, the Mish activation function is introduced [Mis19], which is a self-regularized non-monotonic activation function

$$\sigma_{\text{Mish}}(\boldsymbol{x}) = \boldsymbol{x} \cdot \sigma_{\text{Tanh}}\left(\sigma_{\text{Softplus}}(\boldsymbol{x})\right), \tag{2.62}$$

where the softplus activation function is a smooth approximation of the ReLU function (2.59)

$$\sigma_{\text{Softplus}}(\boldsymbol{x}) = \frac{1}{\beta} \cdot \log(1 + \exp(\beta \cdot \boldsymbol{x})) \tag{2.63}$$

and β is a hyperparameter for the softplus formulation.

Finally, when the output has to lie in the range $[0, 1]$ and the sum of all outputs is supposed to be $\sum_j o_j = 1$, which is usually the case for a classification task to predict a valid probability distribution, the softmax activation function is deployed

$$\sigma_{\text{Softmax}}(x_i) = \frac{\exp(x_i)}{\sum_j \exp(x_j)}. \tag{2.64}$$

Similarly, the sigmoid activation function

$$\sigma_{\text{Sigmoid}}(\boldsymbol{x}) = \frac{1}{1 + \exp(-\boldsymbol{x})} \tag{2.65}$$

is used, if only one output in the range of $[0, 1]$ is required.

2.4.5 Additional Deep Learning Concepts

Pooling Operations

In the following, pooling operations are introduced, that were originally designed to reduce the dimension of an input signal, e.g., images, while retaining features through small rotational and translational invariance [GBC16, Chapter 9.3]. Similar to convolutional layers, pooling layers are comprised of a window of fixed size (w_h, w_w), where w_h and w_w denote the window height and width, respectively. The window is moved over the input according to the stride s_P. However, in contrast to the convolution operation, pooling layers have no kernel, i.e., no learnable parameters. Instead, a deterministic function is applied to the part of input that currently overlaps the sliding window. In the following, the two most popular pooling operations are

presented: maximum pooling (max-pooling) [ZC88]

$$\text{Maxpool}((w_h, w_w), s_P) : Y(C_j, h, w) = \max_{i=0,\dots,w_h-1} \max_{j=0,\dots,w_w-1} X(C_j, s_P \cdot h + i, s_P \cdot w + j) \quad (2.66)$$

and average pooling

$$\text{Avgpool}((w_h, w_w), s_P) : Y(C_j, h, w) = \frac{1}{w_h \cdot w_w} \sum_{i=0}^{w_h-1} \sum_{j=0}^{w_w-1} X(C_j, s_P \cdot h + i, s_P \cdot w + j), \quad (2.67)$$

where $C_j, j \in \{1, \dots, ch_{\text{in}}\}$ denotes the current channel and h and w the current output positions. Considering an input signal of size $(C, H_{\text{in}}, W_{\text{in}})$ with a window size of (w_h, w_w) and stride s_P, the output size $(C, H_{\text{out}}, W_{\text{out}})$ is again given by

$$H_{\text{out}} = \left\lfloor \frac{H_{\text{in}} - w_h}{s_P} + 1 \right\rfloor, \quad (2.68a)$$

$$W_{\text{out}} = \left\lfloor \frac{W_{\text{in}} - w_w}{s_P} + 1 \right\rfloor. \quad (2.68b)$$

Here, it is important to note that Equations (2.66) and (2.67) are shown for the case of a two-dimensional input but easily extend to the case of one-dimensional or three-dimensional input signals as well.

Following the ideas of [ZKR+17] and [QSMG17; QYSG17], the set-pooling function is introduced, which is a symmetric set function as defined in Equation (2.8). The symmetric characteristic is achieved by choosing the hyperparameters of the corresponding pooling method according to the properties of the input data. Therefore, set-pooling neither depends on the ordering nor on the number of input signals, making it a preferable choice for point cloud processing. Considering an arbitrary input set with N elements $\mathcal{X} = \{\boldsymbol{x}_1, \dots, \boldsymbol{x}_N\}, \boldsymbol{x} \in \mathbb{R}^d$, set-pooling is defined as follows:

$$\text{Setpool}(\mathcal{X}) : \text{pool}((N, 1), 1), \quad (2.69)$$

where pool is either maximum pooling (2.66) or average pooling (2.67). Following the definition of the pooling operation, it becomes apparent, that set-pooling (2.69) in general, is a symmetric function: $\mathbb{R}^{N \times d} \to \mathbb{R}^d$, that maps arbitrary elements from a set with fixed dimension d to a vector valued output of constant size $\mathbb{R}^d$. In summary, the pooling operation is performed over the dynamic dimension, which in the case for point clouds is the number of points, by adapting the kernel size on-the-fly. This is possible since pooling has no learnable parameters. Consequently, the resulting vector only depends on the latent dimension, which is a fixed hyperparameter.

Loss Functions and Performance Measures

Traditionally, deep learning tasks can be categorized into multiple groups, whereas the classification and regression tasks are considered in this thesis. The difference is that classification tries to predict a discrete output variable in the form of a class label, while for the regression task, the output is continuous. However, these two tasks can overlap, e.g., the output for a classification task can be continuous, such that the network predicts a probability for each label, and the regression task may also infer a discrete variable. Different loss functions and performance measures have to be deployed depending on the task. When the goal is to assign the input to one of C categories or classes, i.e., multi-class classification, it is often desirable to predict a probability for each class $p_k, k \in \{1, \ldots, C\}$, such that the underlying probability distribution is valid according to Section 2.2.1, i.e., $\sum_i p_i = 1$ and $0 \leq p_i \leq 1$. For that, the network is designed to have C output features with a softmax activation function (2.64) as the last layer, which by design provides a valid probability distribution. Likewise, for the binary classification task, where the output is mapped to one of two classes, the sigmoid activation layer (2.65) is employed.

Usually, the loss function used for classification tasks is the cross-entropy loss, which calculates the total entropy between two probability distributions, in this case, the true and the predicted probability distribution. Without going into more detail and omitting the derivation, the cross-entropy loss for C classes is defined as follows:

$$L^{\mathrm{CE}}(\hat{\boldsymbol{y}}, \boldsymbol{y}) = -\sum_{j=1}^{C} y_j \cdot \log(\hat{y}_j), \tag{2.70}$$

where y_j denotes the ground-truth and $\hat{y}_j$ the estimate after the softmax layer. To evaluate the classification performance, one of many possible performance measures is the classification accuracy, which specifies the ratio between correct predictions and all predictions, i.e., the percentage of correct predictions

$$\text{Accuracy} = \frac{\mathrm{TP} + \mathrm{TN}}{\mathrm{TP} + \mathrm{TN} + \mathrm{FP} + \mathrm{FN}}, \tag{2.71}$$

where TP denotes true positives, TN true negatives, FP false positives and FN false negatives.

The network typically has one or more continuous output units for the regression task. One of the most fundamental loss functions for regression is the mean squared

error loss (MSE-Loss), which is also called the squared L2 norm,

$$L^{\mathrm{MSE}}(\hat{\boldsymbol{y}}, \boldsymbol{y}) = \frac{1}{N} \sum_{j=1}^{N} (y_i - \hat{y}_i)^2 = \mathrm{E}\left[(\boldsymbol{y} - \hat{\boldsymbol{y}})^2\right], \tag{2.72}$$

where N denotes the number of output variables, and again, y_i is the ground-truth and $\hat{y}_i$ the estimate. Here, a problem arises in the case of a multi-task regression problem, i.e., the network estimates multiple continuous variables. When the output variables have different units or are given in different orders of magnitude, which can be the case, for example, when one output is given in meters and the other in radians, e.g., pose estimation. In that case, the optimization is skewed towards minimizing the part of the MSE-Loss that has a bigger impact on the total loss. To counteract this issue, it is possible to weigh the loss parts

$$L^{\mathrm{MSE}}(\hat{\boldsymbol{y}}, \boldsymbol{y}) = \frac{1}{N} \left(\lambda_1 (y_1 - \hat{y}_1)^2 + \cdots + \lambda_N (y_N - \hat{y}_N)^2\right), \tag{2.73}$$

where $\lambda_i, i \in \{1, \ldots, N\}$ are hyperparameters. However, tuning λ_i is usually a difficult and time-consuming task that depends on the training data distribution. Therefore, the geometric loss function according to [KC17; KGC18] is introduced, which was proposed for the task of pose estimation. The goal of the geometric loss function is to allow the network to learn an optimal weighting for the position and the orientation by itself. The geometric loss is formulated using probabilistic deep learning, in which Kendall et al. [KG17] introduce homoscedastic uncertainty and heteroscedastic uncertainty. To summarize, Kendall et al. found that deep learning models are associated with two different kinds of uncertainties: First, there exists an uncertainty that stems from the limited training data distribution, i.e., heteroscedastic uncertainty, and another uncertainty that is bound to the task itself and does not depend on the input data, i.e., homoscedastic uncertainty. Following this, the geometric loss function is derived from the homoscedastic uncertainty and is given by

$$L^{\mathrm{geom}}(\hat{\boldsymbol{y}}, \boldsymbol{y}) = L^{\mathrm{MSE}}_{\mathrm{pos}} \cdot \exp(-s_{\mathrm{pos}}) + s_{\mathrm{pos}} + L^{\mathrm{MSE}}_{\mathrm{ori}} \cdot \exp(-s_{\mathrm{ori}}) + s_{\mathrm{ori}}, \tag{2.74}$$

with $s_{\mathrm{pos}} = \log(\sigma^2_{\mathrm{pos}})$ and $s_{\mathrm{ori}} = \log(\sigma^2_{\mathrm{ori}})$. Here, σ^2_{pos} and σ^2_{ori} are learnable parameters for the position and the orientation, respectively, that denote the homoscedastic uncertainties, in other words the variance or noise of the specific task. The total geometric loss L^{geom} (2.74) is comprised of two parts: First, the regression loss, which in this case is the mean squared error loss function, is weighted by the homoscedastic uncertainty in a way that a higher variance decreases the overall regression loss and vice versa. Thus, pose regression, where the orientation noise is usually orders of magnitude smaller than the position noise, causes the orientation regression to be weighted higher. Furthermore, the geometric loss function contains uncertainty

regularization terms in order to prevent the network from predicting infinite uncertainties σ^2_{pos} and σ^2_{ori} that would cause the total loss to be zero. A more in-depth explanation can be found in [KG17].

Finally, to evaluate the regression performance, the absolute error (AE)

$$\text{AE}(\hat{\boldsymbol{y}}, \boldsymbol{y}) = \frac{1}{N} \sum_{j=1}^{N} |y_i - \hat{y}_i| \tag{2.75}$$

or the root mean squared error (RMSE)

$$\text{RMSE}(\hat{\boldsymbol{y}}, \boldsymbol{y}) = \sqrt{\frac{1}{N} \sum_{j=1}^{N} (y_i - \hat{y}_i)^2} \tag{2.76}$$

can be used.

Optimizer

Previously, the stochastic gradient descent (2.48) has been introduced as one update rule for the learnable weights of the network. Additionally, more comprehensive algorithms have been proposed in recent years. In the following, two widely used optimizers are explained that are used for the deep learning approaches for vehicle self-localization in this thesis. The first one is the popular Adam optimizer [KB15]. The name is derived from adaptive moment estimation, which means that the learning rate is adapted according to first and second-order moment estimates of the gradients. Contrary to the SGD algorithm (2.48), which has one global learning rate η for the entire parameter set, the Adam optimizer uses different learning rates for the individual weights, which combines the ideas of the Adagrad [DHS11] and the RMSProp optimizers. The update rule for the weights is given as follows:

$$\boldsymbol{\theta}_t \leftarrow \boldsymbol{\theta}_{t-1} - \frac{\eta}{\sqrt{\hat{\boldsymbol{v}}_t} + \epsilon} \hat{\boldsymbol{m}}_t, \tag{2.77}$$

where t is the current time step, η is the learning rate, and ϵ is a regularization term. The bias-corrected moment estimates are given by

$$\hat{\boldsymbol{m}}_t = \frac{\boldsymbol{m}_t}{1 - \beta_1^t}, \tag{2.78a}$$

$$\hat{\boldsymbol{v}}_t = \frac{\boldsymbol{v}_t}{1 - \beta_2^t}, \tag{2.78b}$$

where m_t denotes the the moving average of the first moment, i.e., the mean, and v_t denotes the moving average of the the second moment, i.e., the uncentered variance of the gradients

$$\boldsymbol{m}_t = \beta_1 \boldsymbol{m}_{t-1} + (1-\beta_1)\boldsymbol{g}_t, \tag{2.79a}$$

$$\boldsymbol{v}_t = \beta_2 \boldsymbol{v}_{t-1} + (1-\beta_2)\boldsymbol{g}_t^2. \tag{2.79b}$$

In this case $\boldsymbol{g}_t$ denotes the gradients of the objective function L with respect to the learnable parameters $\boldsymbol{\theta}$. The decay rates β_1, β_2 and the regularization term ϵ are set to $\beta_1 = 0.9, \beta_2 = 0.999$ and $\epsilon = 1e{-}8$ as proposed by the authors. A more efficient reformulation of the update rule is given by $\eta_t = \eta\sqrt{1-\beta_2^t}/(1-\beta_1^t)$ and $\boldsymbol{\theta}_t \leftarrow \boldsymbol{\theta}_{t-1} - \eta_t \boldsymbol{m}_t/(\sqrt{\boldsymbol{v}_t} + \epsilon)$, where the dynamic adaption of the learning rate is more clear. An improved variant of the Adam optimizer was proposed by Liu et al. [LJH+20] in 2019, where the authors introduced the rectified Adam optimizer (RAdam). In the early stages of the training process, when the amount of training samples is limited, the Adam optimizer (2.77) can cause slow convergence, which may happen due to the high variance of the adaptive learning rate. One solution is to use lower learning rates initially; however, the authors propose another approach by adding a term that rectifies the variance of the adaptive learning rate. The weight update depends on whether the variance is tractable. For this, the length ρ_t and the maximum length ρ_∞ of the simple moving average (SME) is calculated as follows:

$$\rho_\infty = \frac{2}{1-\beta_2} - 1, \tag{2.80a}$$

$$\rho_t = \rho_\infty - \frac{2t\beta_2^t}{(1-\beta_2^t)}. \tag{2.80b}$$

The authors argue that the variance is tractable if $\rho_t > 4$, thus the parameters are updated with adaptive momentum; otherwise, the unadapted momentum is used

$$\boldsymbol{\theta}_t \leftarrow \begin{cases} \boldsymbol{\theta}_{t-1} - \eta_t r_t \hat{\boldsymbol{m}}_t \boldsymbol{l}_t & \rho_t > 4, \\ \boldsymbol{\theta}_{t-1} - \eta_t \hat{\boldsymbol{m}}_t & \text{otherwise}, \end{cases} \tag{2.81}$$

with the adaptive learning rate

$$\boldsymbol{l}_t = \sqrt{\frac{(1-\beta_2^t)}{\boldsymbol{v}_t}}, \tag{2.82}$$

and the variance rectification term

$$r_t = \sqrt{\frac{(\rho_t - 4)(\rho_t - 2)\rho_\infty}{(\rho_\infty - 4)(\rho_\infty - 2)\rho_t}}. \tag{2.83}$$

Further information about the RAdam optimizer, including the derivation, can be found in [LJH+20].

Regularization

As mentioned before, the central goal of deep learning applications is to make the algorithm perform well on unseen data. The training process is always confined to a limited data distribution; however, when the model is deployed to real-world scenarios, it is desirable to achieve a similar performance as observed during the training process. i.e., the difference between the training and test errors should be as low as possible. Furthermore, since the weights are optimized according to the training data distribution, the objective is to reduce the training error. There are usually two effects associated with the overall performance of a deep neural network: 1) underfitting, which occurs when the training error is not reduced to a satisfying degree, and 2) overfitting, where the training error is low, but the gap between the training and the test error is large.

Generally speaking, during the design process of the network architecture, these problems can be addressed by changing the capacity of the network, i.e., the number of learnable parameters. Models with a low capacity may have problems reducing the training error; however, when the model has a high capacity, it is likely to memorize features of the specific training samples, leading to poor generalization capabilities. Consequently, the learning algorithm must be designed to fit the specific task and be adapted according to training and test data availability. For the first aspect, multiple approaches are available to counteract the overfitting problem, e.g., regularization or parameter norm penalties [GBC16, Chapter 7]. For the availability of training data, it is possible to augment the training data samples to obtain slightly different examples [GBC16, Chapter 7.4].

The main idea of regularization is to introduce a parameter norm penalty $\Omega(\boldsymbol{\theta})$ to the loss function

$$L_{\text{total}}(\hat{\boldsymbol{y}}, \boldsymbol{y}, \boldsymbol{\theta}) = L(\hat{\boldsymbol{y}}, \boldsymbol{y}) + \alpha\Omega(\boldsymbol{\theta}), \tag{2.84}$$

that acts as a weight and bias decay. In case the bias term is omitted, i.e., $\boldsymbol{\theta} = \boldsymbol{w}$, Equation (2.84) becomes $L_{\text{total}}(\hat{\boldsymbol{y}}, \boldsymbol{y}, \boldsymbol{w}) = L(\hat{\boldsymbol{y}}, \boldsymbol{y}) + \alpha\Omega(\boldsymbol{w})$. Specifically, for the parameter norm penalty $\Omega(\boldsymbol{w})$, it is common practice to employ the L^2 regularization

$$\Omega^{L^2}(\boldsymbol{w}) = \frac{1}{2}\boldsymbol{w}^T\boldsymbol{w}, \tag{2.85}$$

where the squared model weights are penalized. This forces the weights to be small,

such that it is more difficult for the model to overfit to the training data. Here, L denotes an arbitrary loss function from the previous section and the parameter α is equivalent to the learning rate η, in a way that it controls the weighting of the parameter norm penalty.

In addition to penalizing the weights, dropout is a popular choice to reduce overfitting [SHK+14]. The idea of dropout is to imitate a deployment strategy called bagging [GBC16, Chapter 7.11], in which multiple models are trained and evaluated together. However, this quickly becomes intractable for large networks since the memory and computational requirements scale linearly with the number of different models. Dropout solves this problem by randomly deactivating single computational units (neurons) during the training process, which is usually done by multiplying the output of the deactivated neuron with zero. This corresponds to an exponential number of different trained models. Each unit is assigned a probability p_d that indicates its dropout probability during training. Here, it is important to note that during the test process, the output of each neuron has to be weighted with p_d to retain appropriate scaling. Dropout has shown to be one of the most effective ways to regularize a neural network and reduce overfitting.

Normalization Methods

Another dominant problem that occurs while training deep neural networks is the so called covariate shift, which occurs when the input distribution changes compared to previously seen data samples. This can happen, for example, when the test data distribution is different from the training data distribution, which is also known as dataset shift. This can lead to poor generalization capabilities and an increase of the test error. Moreover, when the covariate shift occurs in hidden layers during the training process, i.e., the internal covariate shift [IS15], the training process is slowed down noticeably, because the optimization algorithm forces the layer to adapt to the new data distribution. Ioffe et al. [IS15] propose the batch normalization (BatchNorm) algorithm to counteract the internal covariate shift by normalizing the input to the next layer over one mini-batch

$$\mathrm{BN} : \boldsymbol{y}_i(\boldsymbol{x}_i) = \gamma \hat{\boldsymbol{x}}_i + \beta, \tag{2.86}$$

where γ and β are learnable parameters for scaling and shifting the normalized input $\hat{\boldsymbol{x}}_i$, which is given by

$$\hat{\boldsymbol{x}}_i = \frac{\boldsymbol{x}_i - \boldsymbol{\mu}_\mathcal{B}}{\sqrt{\boldsymbol{\sigma}_\mathcal{B}^2 + \epsilon}}, \tag{2.87a}$$

with

$$\boldsymbol{\mu}_{\mathcal{B}} = \frac{1}{B}\sum_{i=1}^{B}\boldsymbol{x}_i, \tag{2.87b}$$

$$\boldsymbol{\sigma}^2_{\mathcal{B}} = \frac{1}{B}\sum_{i=1}^{B}(\boldsymbol{x}_i - \boldsymbol{\mu}_{\mathcal{B}})^2, \tag{2.87c}$$

where ϵ is a small constant that is added for numerical stability and $\boldsymbol{\mu}_{\mathcal{B}}$ and $\boldsymbol{\sigma}^2_{\mathcal{B}}$ denote the mean and the variance of the mini-batch $\mathcal{B} = \{\boldsymbol{x}_i, \ldots, \boldsymbol{x}_B\}$, respectively.

Recently, Santurkar et al. [STIM18] found that batch normalization increases the speed of the training process not by reducing the internal covariate shift but by smoothing the error landscape, which helps to obtain more stable gradients. However, batch normalization proved to be an effective method to increase training speed and stability.

Finally, layer normalization (LayerNorm) is another normalization method [BKH16] that is widely used in natural language processing tasks. Again, considering the mini-batch $\mathcal{B} = \{\boldsymbol{x}_i, \ldots, \boldsymbol{x}_B\}$ with $\boldsymbol{x}_i \in \mathbb{R}^D$, layer normalization is defined as follows:

$$\text{LN} : \boldsymbol{y}_i(\boldsymbol{x}_i) = \gamma\hat{\boldsymbol{x}}_i + \beta, \tag{2.88}$$

where the flattened representation of one input sample is given as $\boldsymbol{x}_i = [x_{i,1}, \ldots, x_{i,D}]$. Again, γ and β are learnable parameters. Subsequently, the normalized input is given by $\hat{\boldsymbol{x}}_i = [\hat{x}_{i,1}, \ldots, \hat{x}_{i,D}]$. Contrary to batch normalization, cf. Equation (2.86), the mean and variances are not computed over one mini-batch but over the feature dimensions of each input sample

$$\hat{x}_{i,d} = \frac{x_{i,d} - \mu_i}{\sqrt{\sigma_i^2 + \epsilon}}, \tag{2.89}$$

where $\epsilon = \textit{const.}$ ensures numerical stability and

$$\mu_i = \frac{1}{D}\sum_{d=1}^{D}x_{i,d}, \tag{2.90a}$$

$$\sigma_i^2 = \frac{1}{D}\sum_{d=1}^{D}(x_{i,d} - \mu_i)^2. \tag{2.90b}$$

Chapter 3

Related Work

This chapter introduces methods that are closely related to this thesis. As mentioned before, the contributions of this work can be divided into two parts: advancing the state-of-the-art in 3D point cloud processing by adopting the popular attention mechanism for unordered point sets and applying the insights and solutions to the task of vehicle-self localization in the field of automated vehicles. Therefore, the related work is presented similarly: First, pioneering deep learning-based network architectures that operate on unstructured point data are introduced. Some of the related approaches are adapted for the proposed methods in this thesis. Thus, a particular focus is set on specific architectures and principles that introduce the fundamentals needed for the remainder of this thesis. Then, common traditional localization frameworks are presented that use a point-based representation of the environment, i.e., landmark-based maps. Finally, deep learning-based vehicle self-localization approaches are introduced to combine both topics covered in this thesis.

3.1 Deep Learning on Point Clouds

One of the most influential contributions to the 3D computer vision community in recent years, the PointNet architecture, has been proposed by Qi et al. in 2017 [QSMG17]. The approach introduces a methodology to handle unordered point clouds of arbitrary cardinality. Before PointNet, directly processing point sets was impossible since neural networks required the input to be structured and organized. The problem with point clouds stems from the fact, that the network architecture has to be permutation-invariant, since sets and point clouds in particular have no ordering associated, as explained in Section 2.1. Consequently, the network is required to produce the same output regardless of how the points are arranged, e.g., in an arbitrary vector or matrix-based representation of the

input set. The common solution to this challenge was transforming the points into a structured form. For that, two main approaches exist that have shown great performance on popular 3D computer vision benchmarks, e.g., point cloud classification on the ModelNet40 dataset [WSK+15] or part segmentation on the ShapeNet dataset [CFG+15]. The first approach involves the voxelization of the input space. Here, the input space is homogenously divided into a regular three-dimensional grid. Each individual subdivision is called a *voxel*, similar to a *pixel* in 2D computer graphics. Then, each point is assigned to the voxel it is encapsulated by. Consequently, this procedure discretizes the entire input space and produces a structured and permutation-invariant representation of the point cloud. Additionally, it is also possible to encode the number of points that lie within one voxel. The transformed points can then be used as input to standard neural network architectures, such as 3D convolution neural networks, similar to 2D convolutional networks for images [MS15].

It is important to note that voxels themselves do not have a specific coordinate associated, but the network infers the position based on the relative location to other voxels [BLRW16; LD18; SZAB17]. Two notable contributions for voxel-based processing of point clouds include the work from Liu et al. [LTLH19] and PointPillars from Lang et al. [LVC+19]. Liu proposes a Point-Voxel CNN that combines the ideas of using point-based neural networks and a voxel-based representation to circumvent the irregular form of point clouds. Lang et al. introduce a novel volumetric format, i.e., vertical columns called point pillars that can be processed by standard 2D convolutional layers. However, the apparent downside of this approach is the increased memory consumption, e.g., the number of voxels increases cubically with respect to the input resolution. Therefore, the network complexity and the computational footprint impede easy scale-up when the resolution is too high. On the other hand, a low voxel resolution causes irrecoverable loss of geometric information. Additionally, due to the sparse nature of point clouds, it is possible to obtain a considerable amount of empty voxels, i.e., voxels that no point is associated with, which unnecessarily inflates the input size. Besides voxel rasterization, similar approaches have been proposed that utilize an octree-based representation [RUG17; WLG+17].

The second type of transformation are the so-called *view-based* methods that render a 2D image of the input point cloud. Thus, the 3D shape is projected into a 2D image, where again, common convolutional neural networks can be applied to extract latent features for further processing. This transformation produces a permutation-invariant and structured image of the point cloud [PLH+19; SBZB15; SMKL15]. Since neural networks for image processing have intensively been studied in recent years, the application to rendered projections of 3D point clouds has shown impressive results. Based on this idea, many successful approaches improved the performance by incorporating *multi-view* methods, where rendered images are generated from multiple viewpoints or different input representations [KMN18; QSN+16; SGWM18]. The obvious disadvantage is the additional preprocessing step

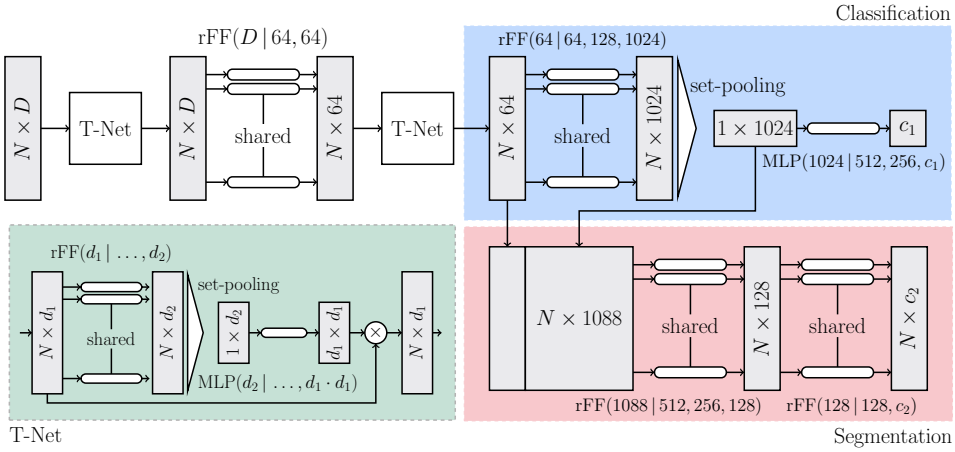


Figure 3.1: Overview of the PointNet network architecture, that uses set-pooling to induce permutation-invariance for arbitrary input set cardinality. Based on [QSMG17].

that requires a computational intensive rendering engine. Moreover, reducing the 3D input to a 2D image may cause a loss of important geometric relations and information, e.g., when one side is not visible from the camera perspective or relevant shapes are occluded. Finally, for lidar processing neural networks, it is also common practice to render a birdseye image, also called birdseye view (BEV), of the lidar point cloud, similar to popular grid map approaches, e.g., [NRT+18]. Then, the BEV can easily be processed by common 2D computer vision approaches [BGM+18; ZZD+20]. A thorough and in-depth survey of existing deep learning approaches for various tasks in 3D computer vision, including 3D point cloud processing, is presented in [GWH+21]. Consequently, the introduction of the PointNet and PointNet++ architectures helped advance the state-of-the-art in point cloud processing.

Although the basic ideas of PointNet were first proposed by Zaheer et al. [ZKR+17], who applied the pooling operation as a symmetric function on unordered input data to induce permutation-invariance, cf. Section 2.4.5, Qi et al. [QSMG17] introduced a generic framework to specifically process and work with 3D point clouds. In addition, the pioneering work allowed for easy adaption to different tasks in 3D computer vision. In order to keep this section within the scope, the methodology will be briefly described in the following. PointNet , which is shown in Figure 3.1, first processes the input set using a row-wise feedforward network (rFF), cf. Section 2.4.4, that projects N points from the input space $\mathbb{R}^{N\times 3}$ to latent space. Since the rFF transforms each point individually, this operation does not depend on the ordering of the input. Afterward, a max-pooling operation is applied over the feature dimensions to produce

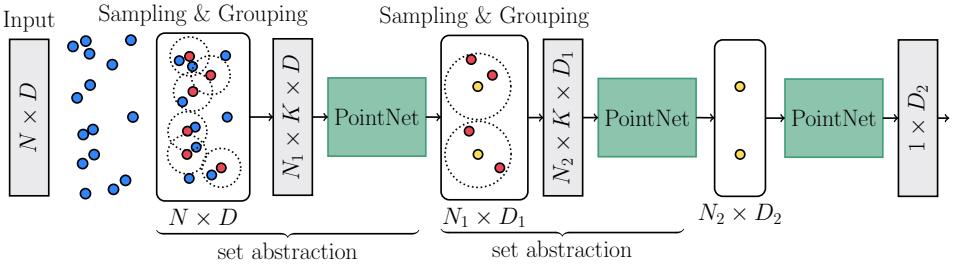


Figure 3.2: Simplified overview of the PointNet++ architecture. The aggregation of local features is achieved by employing a hierarchical grouping of neighboring points (set abstraction). The combination of multiple set abstraction layers is also known as multi-scale grouping (MSG) since the sampling and grouping operations are performed with different radii. Here, only the base network is shown without the output heads for readability purposes. Based on [QYSG17].

a feature vector of fixed size $\in \mathbb{R}^{1 \times d_m}$, i.e., set-pooling from Equation (2.69). Then, multiple processing steps follow, for example, applying a rotation matrix to the input point cloud, which is supposed to induce robustness against spatial transformations, e.g., translation and rotation. These steps are repeated with different latent feature dimensions, as shown in Figure 3.1, to allow the network to learn from different subspaces. Depending on the current computer vision task, different output heads can be deployed, which are depicted in Figure 3.1. More importantly, since the resulting latent feature representation is permutation-invariant, the output heads can employ common neural network architectures. However, the authors themselves mention that PointNet lacks the ability to attend to local geometric information since the result of the max-pooling operation is only a global signature of the original point cloud. Therefore, Qi. et al. propose an improved version, called PointNet++, that deals with aggregating localities and spatial neighborhoods [QYSG17]. The resulting network architecture is visualized in Figure 3.2. The main improvement over the original PointNet architecture is explained in the following. Instead of applying the set-pooling operation on the entire point cloud, the input is first downsampled. In the case of PointNet++, the authors use the furthest point sampling approach (FPS). The algorithm reduces the input from N to N' points, starting from a random initial point, and, subsequently, finds the other $N' - 1$ points that have the largest distance from each other. Here, it is important to note that the FPS is usually a bottleneck in the processing chain, as the computational time increases drastically with respect to the size of the input point set. Then, for each of the N' points, a ball query search algorithm is employed that finds all points from the original input point cloud $\in \mathbb{R}^{N \times 3}$ that are within a predefined radius r. A mini PointNet is

applied to the subset from the ball query search to aggregate latent features from the local neighborhood that are also permutation-invariant. This procedure can be repeated, whereas the radius is usually increased compared to the previous step to combine more geometric features. This process is also known as multi-scale grouping (MSG). Therefore, more and more local information is aggregated with each step, which significantly improves the network's overall performance in learning important shape information. A comprehensive introduction to PointNet and PointNet++ is also available in [Qi18].

Finally, Liu et al. [LQG19] extend the network by introducing the concept of scene flow for 3D point cloud processing, which can be interpreted as the translation between two consecutive 3D scenes, e.g., consecutive time steps from lidar scans. The idea is to apply the PointNet++ network structure, whereas the subsampling is performed on one point cloud and the ball query search within the second point cloud. That way, it is possible to relate two independent sets of points with the same network architecture, e.g., for finding the motion between two consecutive time frames to infer the odometry. This application is related to the task of self-localization, where instead of two sequential scans, landmarks and measurement sets are related with the goal of finding probable associations. Besides set-pooling-based approaches, another method has emerged in the field of 3D point cloud processing in recent years, where the concept of convolutions has been applied to point-based neural networks. For that, a special kernel function is usually proposed that operates on the metric space of the input points, where additionally, a permutation-invariance is required as well [LBS+18; TQD+19; XFX+18]. However, exploring the concept of point convolutions is out of scope for this thesis but is mentioned here for the sake of completeness.

Finally, a popular method that is able to efficiently process large-scale point clouds mainly for the task of semantic segmentation is introduced. Semantic segmentation involves assigning a unique class label to each point. In other words, it is a per-point classification task that is often employed to process lidar scans, which are characterized by being dense. Moreover, the application of semantic segmentation for automated driving requires real-time capability. Thus, fast inference is necessary to use the segmented point cloud for further processing. For that, Hu et al. propose the RandLA-Net network architecture [HYX+20], that is depicted in Figure 3.3. Without going into detail, the network uses random sampling to downsample the large input point cloud efficiently. Then, a local feature aggregation module that preserves the local shape and geometric information, similar to the PointNet++ architecture, is introduced. This is achieved by progressively increasing the receptive field for each point. Furthermore, only multilayer perceptrons with shared weights, i.e., rFFs from Equation (2.58), are employed, reducing the overall memory footprint. The network architecture is composed of three blocks as shown in Figure 3.3: Local Spatial Encodings (LocSE), Attentive Pooling, and Dilated Residual Blocks. The

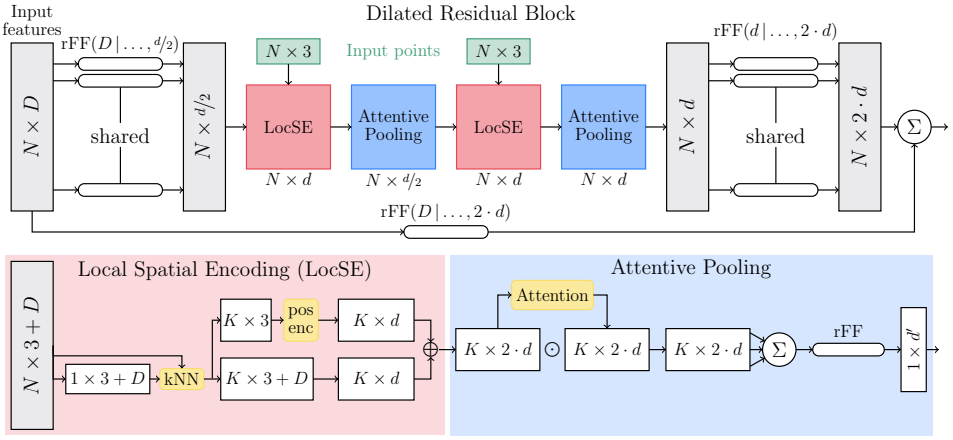


Figure 3.3: Overview of the dilated residual block from the RandLA-Net architecture. The goal is to aggregate local features using consecutive local spatial encoding blocks (LocSE) and attentive pooling modules. The LocSE and attentive operations are employed for each point separately. In practice, multiple dilated residual layers and downsampling operations are applied together to decrease inference time while retaining spatial information. Based on [HYX+20].

LocSE block aims to encode local features of the spatial neighborhood to each point of the input point cloud by aggregating local features using a k-Nearest Neighbor algorithm (kNN). Furthermore, the relative point position is encoded for each point (pos enc). Then, the result is attended using the Attentive Pooling block, which in contrast to traditional pooling operations, learns spatial information from the local neighborhood. Following the ideas of popular vision-based neural networks, e.g., ResNet [HZRS16], the authors propose a similar architecture that employs multiple LocSE, Attentive Pooling, and residual connections, which additionally are designed to increase the receptive field while downsampling the size of the input point cloud. Again, an in-depth overview of the RandLA-Net is given in [HYX+20].

3.2 Vehicle Self-Localization

In this section, prevalent vehicle self-localization approaches are presented. As explained in Section 2.3, the task of localizing an autonomous agent in a known environment using a landmark-based representation of the environment can be

considered a point-to-point registration problem. The goal is to match the current perceived environment with a digital map to infer the vehicle pose inside the map frame. Therefore, one of the most fundamental approaches is to find the transformation between the measurement point cloud and the landmarks from the digital map using the iterative closest point algorithm (ICP), which was proposed by Besl et al. [BM92]. The algorithm iteratively transforms one point cloud while the second point cloud is fixed with the goal of minimizing an error metric. For example, a widely used approach is to calculate the distance between both point clouds using the euclidean distance between all point pairs. Building upon the ideas of the ICP, Censi et al. introduce PLICP [Cen08], which improves the iterative closest point algorithm. This is done by implementing a point-to-lane metric instead of the point-to-point metric, which additionally improves the convergence properties and converges in a finite number of iteration steps. Additional point cloud registration algorithms for mobile robotics can be found in [PCS15]. Similar iterative algorithms can also be deployed for the task of self-localization, e.g., Fontanelli et al. [FRS07] introduce a RANSAC-based algorithm that accurately registers lidar measurements and is able to cope with noisy measurements.

Next, localization methods that employ filter-based approaches to estimate the vehicle's pose over a time interval are introduced. One of the most fundamental probabilistic filter-based localization approaches is the so-called Markov localization, which is derived from the Bayes filter [Thr02, Chapter 7], which is a general framework for state estimation. Here, a probabilistic belief about the pose at time $t-1$ is transformed into a belief at time t by updating the pose with Bayes rule using the measurements and the digital map. A special case of the Markov localization is the Extended Kalman-Filter localization, where the pose estimate is denoted by its first and second-order statistical moments, i.e., the mean and the covariance [Thr02, Chapter 7.5]. The idea of the EKF is to linearize the non-linear motion and measurement model using the concept of Taylor series expansion [TBF98]. Since the concept of the EKF has extensively been studied in the field of robotics and achieved satisfying results in a variety of tasks, many Extended Kalman-Filter based localization approaches have been proposed in recent years, e.g., [DLMP15; IKP10; MD95; TSK11]. Besides the application to landmark-based localization, these methods can be adapted to work for the case of grid-based localization as well [Thr02, Chapter 8].

Naturally, similar filter-based approaches were applied, e.g., the non-parametric Monte-Carlo localization (MCL), where the vehicle's state space is modeled by a probabilistic particle-based density. MCL is also known as particle filter-based localization that implements a recursive Bayes filter and can efficiently represent arbitrary distributions using particles as possible state space hypotheses, e.g., the robot's pose [DFBT99]. An iteration of the MCL was proposed by Thrun et al. [TFBD01]. where the particle generation process in the estimation phase is

improved. Furthermore, a kernel density tree is learned that enables faster sampling. Additionally, variants of the sampling approach have been introduced that outperform classic MCL methods and generally achieve faster convergence [Fox01]. In recent years, building on top of the advances of the EKF and MCL approaches, more sophisticated filter-based methods have been proposed. In the following, a particle-filter localization approach is presented that incorporates the random finite set theory RFS [MVAV11]. The approach was developed by Stübler at the Institute of Measurement, Control and Microtechnology at Ulm University [Stü18] and achieved impressive results in practical localization scenarios. The RFS localization approach models both the landmarks and measurements as random finite sets [Mah07] and employs a particle-filter framework to approximate the state distribution, similar to previous localization methods. Moreover, the associations between measurements and landmarks are subject to uncertainty; thus, the multi-object likelihood is incorporated, which was adapted from previous RFS-based tracking applications to consider clutter measurements and missed detections. Furthermore, the method introduces a preprocessing pipeline that extracts static and reliable landmarks and measurements from a multi-modal sensor setup, which is also applied in this thesis. In summary, the RFS theory is used to probabilistically infer the weight of each particle, where a higher weight is assigned if the corresponding particle better explains the current observations. Thus, a robust and mathematically sound implementation is achieved.

At this point, well-known localization approaches are introduced; however, another area of localization has seen an increase in research activity over the past decades. Until now, the methods assumed an a-priori map available at run time. This requires a vehicle equipped with a similar sensor setup to record and extract landmarks in the area of operation, which can be highly cost-intensive if a global deployment of the automated vehicle is desired. The task of simultaneous localization and mapping (SLAM) tries to solve this disadvantage by constructing a digital map while localizing the robot in an unknown environment. A notable contribution in the field of SLAM is the FastSLAM algorithm [MTKW02; MTKW03], which employs a particle filter to estimate the vehicle's state and the current map. Here, every particle denotes one pose hypothesis and additionally includes an individual map hypothesis, where every landmark $\boldsymbol{m} \in \mathbb{R}^2$ is tracked using a single Extended Kalman-Filter. Thus, for ν landmarks, every particle deploys ν two-dimensional EKF's, where the complexity is given by $\mathcal{O}(\nu)$. This approach drastically reduces the complexity compared to previous methods that use one EKF for all landmarks. This case leads to a complexity that scales quadratically with respect to the landmarks $\mathcal{O}(\nu^2)$. Similar to the advances in vehicle self-localization, improvements have been proposed for the SLAM task as well. Mullane et al. [MVA10] adapt the RFS theory and introduce the Rao-blackwellised PHD SLAM that incorporates uncertainty of measurement and landmark associations as well as an unknown number of features per time step. The basic idea is to model landmarks and measurements as random finite sets, and both the joint posterior distribution of the digital map and the vehicle

pose is propagated with each new measurement [MVA10]. As the name suggests, a Rao-blackwellised implementation of the PHD filter is introduced for the map, and a particle filter is employed for estimating the pose. Recently, Deusch et al. proposed a new algorithm based on the work of Mullane, i.e., adapting the RFS theory for vehicle localization, that uses the labeled multi Bernoulli filter LMB [RVVD14] for the SLAM task. Consequently, the presented approach outperforms prior work [DRD15].

Finally, another method that is closely related to the task of vehicle self-localization is mentioned here, that uses consecutive camera images to estimate the relative motion between two frames, i.e., visual odometry (VO) [MHW+19]. However, an in-depth study is out of scope for this thesis since the main contributions focus on point cloud-based localization only. Further information about traditional localization approaches for a wide range of mobile robotics applications is presented in [EH20; KFK+18].

In summary, the presented filter-based localization approaches have been well-established in the field of robotics. However, recent advances in deep learning have outperformed traditional computer vision algorithms in various areas; thus, this thesis explores the deployment of state-of-the-art neural network architectures for the task of vehicle self-localization. The goal is to employ data-driven approaches to better approximate the pose estimation that is also more robust to external interferences and noise.

3.3 Deep Learning for Vehicle Self-Localization

The development of deep learning-based vehicle self-localization approaches has not received wide attention compared to other areas in automated driving applications, such as object detection or semantic segmentation. Nevertheless, the following section provides an overview of current state-of-the-art methods that apply neural networks for robot localization. Following the ideas of traditional localization methods, the ICP algorithm has been adapted to operate in an end-to-end trainable neural network that can be applied to feature or landmark-based localization [FTZ+18; LWZ+19]. In that case, the localization task is proposed as a point cloud registration problem. Similarly, RANSAC neural networks, e.g., [BR19], reformulate traditional algorithms as deep learning approaches. Here, the matching and association part can be achieved by employing data-based methods. Then, the algorithm is performed iteratively, improving the registration accuracy with each step.

Since impressive advances in computer vision have been achieved for 2D image processing, various localization approaches that operate on camera images have been

proposed, where the goal is to either infer the camera pose explicitly or implicitly. Explicit localization usually describes the task of estimating the camera pose relative to known poses from a dataset, i.e., the shift and rotation. On the other hand, implicit localization directly regresses the camera pose from the input, which is usually achieved by encoding the entire scene in the neural network. Only implicit localization is considered in the following since it is more popular and closely related to how the localization task is formulated in this thesis. PoseNet [KGC15] was the first end-to-end approach to propose a solution to the camera relocalization problem, where conventional CNN layers are employed to infer the camera pose from a single RGB image. The network architecture is based on popular image-based network backbones that were adapted to output a position and orientation vector. Since then, many iterations have been introduced that tackle challenges of the original PoseNet architecture, such as [KC16; KC17].

A similar network architecture called VLocNet was proposed by Valada et al. [VRB18]. The method introduces a novel end-to-end convolutional neural network architecture that uses sequential monocular images to regress a global six degrees of freedom pose of the camera in conjunction with the odometry of the robot. In order to reduce the model complexity, parameters are shared among the different tasks. Additionally, a geometric consistency loss is introduced that employs auxiliary learning to incorporate relative pose information to increase localization accuracy. The proposed multi-task network outperforms traditional handcrafted feature-based approaches like SIFT-based localization. Then, to incorporate and exploit the relationship of the individual tasks, e.g., learning semantics, pose regression, and learning the odometry, Valada et al. improve the existing network architecture by introducing VLocNet++ [RVB18], which notably increases the networks performance and robustness.

Alternatively, Yang et al. [YBT+19] propose SANet, a camera localization framework that separates the scenes and model parameters and learns a hierarchical scene representation. Furthermore, similar to one of the methods presented in this thesis, AtLoc [WCL+20] is proposed to associate features based on the attention mechanism that helps the network focus on important spatial regions that are consistent and robust. However, AtLoc is restricted to camera images and is not able to generalize to unseen environments. Another widely used method is the transformation of irregular point measurements into a structured representation. This includes the transformation of range or depth data into grids or voxels and lidar scans into birdseye view (BEV) images [BPB+18].

Finally, L3-Net is a deep learning-based vehicle self-localization method that operates on dense lidar point clouds and achieves centimeter-level accuracy [LZW+19]. The authors try to circumvent the tedious work of handcrafting traditional frameworks by employing a neural network-based approach. The goal is to teach the network

local descriptors of the lidar point cloud that are used to infer the vehicle's pose inside a digital map that is created offline. Using the lidar scan, the well-known PointNet architecture is employed to extract latent point features, cf. Section 3.1. Then, the features are fed to a 3D convolutional neural network that acts as a regularizer over the pose dimensions. Finally, a subsequent recurrent neural network (RNN) is employed to implicitly incorporate the temporal filtering concept of traditional localization methods, i.e., the particle filter. However, due to the computational intensive network modules that are employed, such as RNNs and 3D convolutional neural networks, the model complexity as well as the inference time make the approach not suitable for localization. The run-time is required to be below 100 ms, which cannot be guaranteed for this method. A comprehensive survey on the topic of deep learning for mapping and localization and an evaluation of various network architectures can be found in [CLF+21; CWL+20].

In summary, a severe downside of most of these presented methods is that they are not scene-agnostic, i.e., they are not capable of generalizing to new and unseen environments and scenarios since the map is implicitly encoded in the weights of the neural network. Therefore, providing the network with data from a different input distribution would render the network useless. However, this thesis aims to provide a deep learning-based localization framework that can easily generalize to unseen environments by focusing on matching landmarks from a digital map to current sensor measurements without relying on a specific scene. Furthermore, most methods incorporate only one modality for regressing the pose. However, combining the strengths of multi-modal sensor measurements that supplement each other has shown to achieve a more robust localization result, e.g., [Stü18]. Therefore, the goal of this thesis is to formulate a generic localization task that can adapt to different modalities and scenarios.

Chapter 4

Point Transformer

This chapter introduces a novel deep neural network for point cloud processing that leverages the strengths and advantages of the Transformer architecture [VSP+17]. The network is called Point Transformer [EBD21b] and tackles the prominent problem of input permutation-invariance for irregular input data using a module that learns to sort points based on a specific task. In the following, the motivation and fundamentals are outlined. Then, the proposed method is introduced, which includes the network architecture and task-specific design choices. Finally, the network is evaluated on popular 3D computer vision benchmarks, and an extensive network design analysis is given, where the advantages of applying the Transformer architecture to point cloud processing are highlighted.

4.1 Introduction

The task of processing point clouds has become increasingly popular in computer vision in recent years, as many applications handle 2D or 3D shape data in the form of point clouds, i.e., sets of individual points. Significant advances in the field of robotics and autonomous agents are powered by representing the environment using 3D information and point clouds, for example, from lidar or radar sensors and stereo cameras. However, processing point clouds using traditional machine learning and deep learning approaches is not trivial because point clouds or point sets are characterized by being unordered and unstructured, as explained in Chapter 2 and Chapter 3. This poses a major problem for feedforward networks or CNN and RNN architectures that require the input data to be ordered and structured, like RGB images or time series-data. Additionally, the number of points per object, shape, or scan may vary across data samples. Therefore, the network also needs to adapt to different data shapes, i.e., an arbitrary number of rows of the input matrix. Besides the unstructured nature of point clouds, it is essential for the network to

be robust against rotation and translation and capture spatial relations between individual points.

The prevalent approach to deal with the difficult characteristics of point clouds was to transform the shape into a structured representation prior to processing the data with the neural network, i.e., a mapping from the continuous domain into discrete space. Possible approaches include the rendering of the 3D shape into images or transforming the point cloud into voxel-based data structures, where both representations can directly be processed by a 2D or 3D convolutional neural network, respectively. This transformation induces a loss of geometric features and shape information, as occluded patches of the object are not rendered to the image. However, to counteract this disadvantage, multi-view approaches exist that create images from multiple perspectives. This captures more spatial information but also increases storage and computational resource requirements. Similarly, for the voxel-based representation, the possible loss of information stems from the rasterization process, where the resolution of the resulting object depends on the dimension of each voxel.

As explained in Chapter 3, the introduction of set-pooling methods [QSMG17; QYSG17; ZKR$^+$17] enabled traditional neural network architectures to directly process unordered point sets while inducing input point permutation-invariance through a max-pooling operation (2.66) over the latent feature dimension, i.e., Equation (2.69). The idea of set-pooling networks, such as PointNet and PointNet++, is to process each point individually using a MLP with shared weights and reduce the latent features to a vector of fixed size that is additionally invariant to input point permutations. Thus, the set-pooling operation is a symmetric set function, whereas, under certain conditions, it also acts as a universal function approximator [ZKR$^+$17]. Although PointNet-based network architectures achieved state-of-the-art results on multiple prominent computer vision benchmarks, concerns have been raised recently by Wagstaff et al. [WFE$^+$19; WFE$^+$21] about the limitations of set-pooling methods. In particular, the authors argue that the dimension of the latent space limits the capacity of the latent feature representation. This directly affects the previously mentioned universal approximation property, which, according to Wagstaff et al., is only valid if the latent dimension is at least the size of the maximum number of elements in the input set. However, given the rapid development progress of ranging sensors, e.g., lidar, where more and more layers are employed to generate a detailed point cloud of the environment, this limitation negatively impacts the performance of set-pooling neural networks. Consequently, this has practical implications in the case of a high number of input points, where the reduction of the latent feature space to a vector of fixed size may fail to capture important spatial relations and geometric dependencies, thus becoming a bottleneck for the entire network [EBD21b]. For a more thorough discussion on the limitations of set-pooling methods for deep neural networks, the reader is referred to [WFE$^+$21].

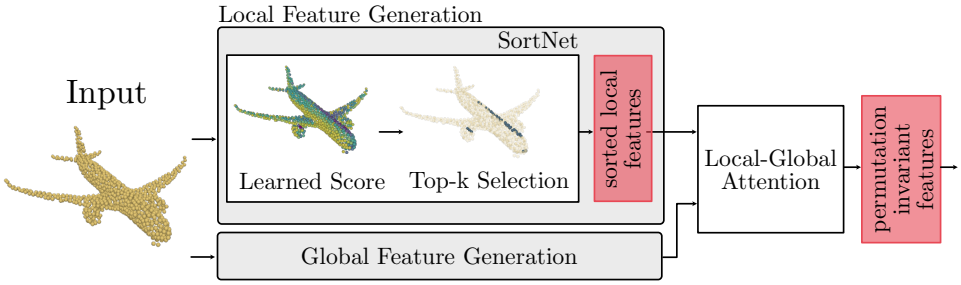


Figure 4.1: Simplified overview of the Point Transformer system architecture. A point cloud serves as input to the network from which local and global features are extracted. Local features are sorted using SortNet, a module that focuses learns to focus on important key points. Then, the local-global attention operation is employed to relate latent features in order to capture geometric relations and shape information. The resulting feature representation is permutation-invariant and can be used for common computer vision tasks. Based on [EBD21b].

The motivation for the Point Transformer architecture, which is illustrated in Figure 4.1, stems from removing the bottleneck of the set-pooling method while presenting a novel permutation-invariant network module that is capable of capturing shape information and spatial relations of neighboring points. For that, the popular self-attention mechanism is employed [VSP+17], which was originally introduced for the task of natural language processing (NLP). Although self-attention itself does not depend on the ordering of the input sequence, the output is not permutation-invariant. Therefore, the SortNet module is proposed to learn a sorted feature set in latent space. In combination with modified self-attention operations that are adapted to handle the characteristic of point clouds, SortNet creates permutation-invariant features without employing set-pooling. These features can directly be used for common computer vision tasks, such as object classification and part segmentation.

As mentioned before, attention has its origin in the field of natural language processing [BCB15; LPM15], where it superseded traditional encoder-decoder recurrent neural networks (RNN). Without going into detail, the common approach was to use the last hidden state of the processed sequence as the context vector for the decoder, where it is used to create the translated output sequentially. However, it is challenging to relate distant samples in the sequence when the input is made up of a long sentence. Bahdanau et al. [BCB15] introduced the attention mechanism, which considers every possible combination of input words. Vaswani et al. [VSP+17] improved the idea with the multi-head attention approach, where the encoder and

decoder solely rely on attention instead of additional convolutional or recurrent layers [VSP+17]. Additionally, the computational complexity was drastically reduced by introducing the multi-head architecture. Nowadays, attention-based networks achieve state-of-the-art results on major NLP benchmarks. Naturally, the ideas were adapted for different areas as well, including point cloud processing.

Vinyals et al. proposed a method that processes unordered sets [VBK16]. They first showed that attention could be applied to sets by sorting numbers of arbitrary sets using sequence to sequence modeling. Lee et al. [LLK+19] improved the concept by introducing Set Transformer, where point sets are processed by attending the input to induced points, i.e., trainable parameters of the network. Moreover, ShapeContextNet, introduced by Xie et al. [XLCT18] in 2018, hierarchically applies a convolutional building block to the input points, which is called the shape context layer. Here, self-attention is employed to learn the shape context parameters and combine the feature aggregation and selection process. However, the shape context kernels have to be selected manually, whereas the selection process is prone to irregularities of the input points, similar to point cloud convolutions, e.g., [TQD+19]. Liu et al. [LHLZ19] propose the Point2Sequence architecture, which employs a sequence to sequence model that is based on the attention mechanism. A long term-short memory (LSTM) is used to produce local features from local regions of the shape. Then, a set-pooling method, applied to the local features, induces permutation-invariance, which has the same disadvantages as traditional set-pooling networks. Moreover, LSTM-based architectures require higher computational resources than multi-head attention. Usually, heuristic downsampling methods are applied to the original point cloud to retain as much of the original shape as possible while decreasing the number of input points. This is done to reduce the overall size and complexity of the network. Prevalent heuristic downsampling approaches include random downsampling and the furthest point sampling approach (FPS). Yang et al. [YZN+19] introduce a network architecture that replaces traditional downsampling methods with an attention-based subsampling process using the Gumbel-softmax operation. Finally, Toa et al. [TZWL21] employ the multi-head attention mechanism together with a rotation-invariant representation of the input point cloud. The authors propose a multi-head attentional convolutional layer (MACL) to aggregate point features. Again, a global max-pooling operation is used to induce permutation-invariance, similar to the Point2Sequence model.

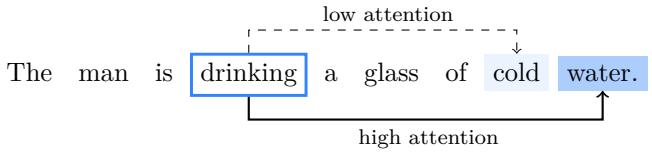
The following sections introduce the essential fundamentals required for the proposed algorithms. Furthermore, the methodology and network architecture are explained in detail, and multiple experiments are performed on popular 3D computer vision benchmarks to demonstrate competitive results compared to related approaches. Finally, a thorough network design study is given to analyze the characteristics and advantages of the Point Transformer architecture.

4.2 Fundamentals

This section introduces the general concepts of the attention mechanism and the Transformer architecture, which was proposed by Vaswani et al. [VSP+17]. Since attention was initially introduced for the task of natural language processing [BCB15], the following methods are framed in the context of NLP.

4.2.1 Attention Mechanism

The ideas of the attention mechanism stem from the way humans correlate words in a sentence. For the sake of this example, the following sentence is considered:



When the word *drinking* is encountered, naturally, a human reader expects the kind of drink somewhere in the sentence. Therefore, the word *water* has a higher likelihood of being related to the word *drinking* than, for example, to the word *cold*. On the other hand, the word *cold* primarily relates to the word *water*. Therefore, the goal is to put more weight on the connection of highly related words when, for example, translating the sentence into another language. The weight of the arrows also visualizes this. Previously, when processing sentences of arbitrary lengths, it was common to employ sequence-to-sequence models [SVL14], where an encoder-decoder structure is used. Both the encoder and the decoder are usually RNNs, such as LSTMs. The encoder compresses the entire input sequence into a vector of fixed size, i.e., the context vector, which is then fed to the decoder to produce an output sequence word by word. The obvious disadvantage of this approach is the incapability to model relations between distant words in the sentence [BCB15]. The attention mechanism was introduced to solve this problem by relating every word of the input sequence by employing a vector of importance weights and a shortcut between the input sequence and the context vector. This means that the context vector has access to every input element.

Although multiple different attention approaches exits in the literature, e.g., [BGLL17; LPM15; XBK+15], the definition of attention in this thesis follows the notation introduced by [VSP+17]. Without loss of generality, the attention function $\mathcal{A}(\cdot)$ is defined as a mapping of N queries $Q \in \mathbb{R}^{N \times d_k}$ and N_k key-value pairs

$K \in \mathbb{R}^{N_k \times d_k}$, $V \in \mathbb{R}^{N_k \times d_v}$ of dimension d_k and d_v respectively, to the output space $\mathbb{R}^{N \times d_k}$ [EBD21b; VSP+17]. As mentioned earlier, the idea is to generate a vector of importance weights that is used to scale the input. The importance weights are given by applying the pairwise dot-product $QK^T \in \mathbb{R}^{N \times N_k}$ to generate a score as follows:

$$\text{score}(Q, K) = \sigma(QK^T), \tag{4.1}$$

where score$(\cdot)$ creates an attention map, in which every query $q \in Q$ is scored against key $k \in K$, thus score$(\cdot) : \mathbb{R}^{N \times d_k}, \mathbb{R}^{N_k \times d_k} \to \mathbb{R}^{N \times N_k}$. The activation function $\sigma(\cdot)$ is usually set to the softmax activation function $\sigma(\cdot) = \sigma_{\text{Softmax}}(\cdot)$ from Equation (2.64) to obtain amplified scores in the range of $[0, 1]$. Note that different activation functions are possible, however, the softmax function is the most popular choice for the attention mechanism. In order to weight the input with the importance scores from Equation (4.1), the score is multiplied with the values $v \in V$

$$\mathcal{A}(Q, K, V) = \text{score}(Q, K)V, \tag{4.2}$$

where the attention operation now maps the three input sequences to the output $\mathcal{A}(\cdot) : \mathbb{R}^{N \times d_k}, \mathbb{R}^{N_k \times d_k}, \mathbb{R}^{N_k \times d_v} \to \mathbb{R}^{N \times d_k}$, as mentioned above. Moreover, the output now is a weighted sum according to the importance scores obtained from Equation (4.1), that is, values are more important if the corresponding dot-product of the queries and keys results in a higher score. In practice, the pairwise dot-product is additionally normalized by $1/\sqrt{d_k}$ to increase numerical stability for large values of d_k [VSP+17]. Thus, the complete attention operation is given by

$$\mathcal{A}(Q, K, V) = \sigma_{\text{Softmax}}\left(\frac{QK^T}{\sqrt{d_k}}\right)V, \tag{4.3}$$

which is also known as scaled dot-product attention [VSP+17]. In the following section, important improvements to the attention mechanism are introduced, i.e., the multi-head attention approach and the Transformer architecture. Moreover, the remaining question of how the queries, keys and values are obtained is answered in the context of natural language processing.

4.2.2 Transformer

The Transformer architecture extends the traditional attention mechanism and introduces an encoder-decoder architecture that solely relies on attention without additional recurrent or convolutional layers. The multi-head attention mechanism is explained in the following, as subsequent approaches in this thesis strongly rely on this method.

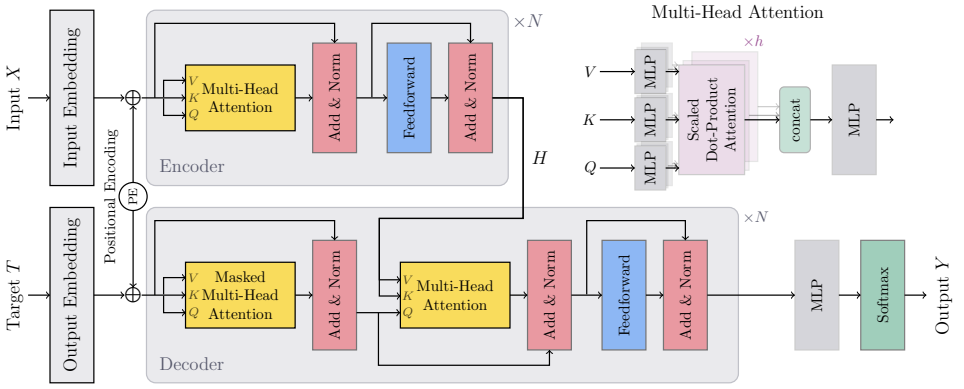


Figure 4.2: Overview of the Transformer architecture with N encoder and decoder blocks. The add & norm modules sum both inputs and apply layer normalization. The positional encoding ensures that the relative position of each embedding is known. Based on [VSP+17].

The general concept of multi-head attention is to employ not one but several attention operations in parallel to the same input, each processing the input with reduced dimension, as shown in Figure 4.2. Considering that the dimensions d_k and d_v are set to $d_k = d_v = d_m$, h attention heads are employed, where the queries, keys and values are first linearly projected to d'_k, d'_k and d'_v dimensions, respectively, using learnable weight matrices, with $d'_k = d'_v = d_m/h$. This ensures that each attention head receives a reduced set of features which drastically increases the computational performance. The linear projection is obtained using simple feedforward networks to allow the network to learn from different subspaces. Following these ideas, the multi-head attention block is defined as follows:

$$\mathcal{A}^{\text{multi-head}}(Q, K, V) = (\text{head}_1 \oplus \cdots \oplus \text{head}_h)W^O, \tag{4.4}$$

where each attention head is given by $\text{head}_i = \mathcal{A}(QW_i^Q, KW_i^K, VW_i^V)$ and $\oplus$ denotes matrix concatenation. Here, $W_i^{(\cdot)}$ denote the learnable parameter matrices, i.e., the feedforward network, with $W_i^Q \in \mathbb{R}^{d_m \times d'_k}$, $W_i^K \in \mathbb{R}^{d_m \times d'_k}$, $W_i^V \in \mathbb{R}^{d_m \times d'_v}$ and $W^O \in \mathbb{R}^{hd'_v \times d_m}$.

Now, considering the goal of mapping (or translating in the case of NLP) an input sequence $X = [\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_N]$ of length N to an output sequence $Y = [\boldsymbol{y}_1, \boldsymbol{y}_2, \ldots, \boldsymbol{y}_M]$ of length M, X is fed to the encoder to produce the latent representation $H = [\boldsymbol{h}_1, \boldsymbol{h}_2, \ldots, \boldsymbol{h}_N]$. Then, the hidden state H is used together with a target sequence T to obtain the desired output Y. The target is usually set to a start token de-

noting the beginning of a sentence. After translating the first word, the result $\boldsymbol{y}_1$ is appended to the target sequence T, such that the next iteration has access to the previous translation. Here, sophisticated learning strategies are involved that are out of scope for this thesis, however, further implementation details are provided in [VSP+17]. For obtaining the output Y, the Transformer architecture employs multiple encoder and decoder stacks of identical layers, which are illustrated in Figure 4.2. In particular, each stack consists of a multi-head attention layer from Equation (4.4) followed by a feedforward network together with a residual connection and layer normalization (2.88) [BKH16; VSP+17]. Consequently, one complete multi-head attention block (MAB) is defined as follows:

$$\mathcal{A}^{\text{MAB}}(X, Y) = \text{LayerNorm}(S + \text{FF}(S)), \tag{4.5}$$

where $\mathcal{A}^{\text{MAB}}(\cdot) : \mathbb{R}^{N \times d_m}, \mathbb{R}^{N_k \times d_m} \to \mathbb{R}^{N \times d_m}$, LayerNorm is given in Equation (2.88) and FF is a simple feedforward network as defined in Equation (2.50). The sublayer S is given by $S = \text{LayerNorm}(X + \mathcal{A}^{\text{multi-head}}(X, Y, Y))$. Here, it becomes apparent, that for the multi-head attention mechanism, two input sequences $X \in \mathbb{R}^{N \times d_m}$ and $Y \in \mathbb{R}^{N_k \times d_m}$ are required. The queries Q are then obtained from the linear projection of X and the keys K and values V from two separate linear projections of Y. As visualized in Figure 4.2, for the encoder blocks it follows that $X = Y$, i.e., the same input is used for the queries, keys and values, which is defined as **self-attention**

$$\mathcal{A}^{\text{self}}(X) \triangleq \mathcal{A}^{\text{MAB}}(X, X). \tag{4.6}$$

Similarly, for the decoder stacks $X \neq Y$; therefore, the queries are generated from a different sequence than the keys and values. In this thesis, this operation is called **cross-attention** and is defined as follows:

$$\mathcal{A}^{\text{cross}}(X, Y) \triangleq \mathcal{A}^{\text{MAB}}(X, Y), \tag{4.7}$$

in order to differentiate between the two possible input variations. Here, it is important to note that the ordering of the output of both the self-attention mechanism and the cross-attention mechanism depends on the ordering of the input sequence X. However, the actual output values of each element are the same, regardless of the ordering, because only matrix multiplications and summation operations are employed. Thus, attention is not strictly permutation-invariant but can process unordered sets, like point clouds. This is an important characteristic of the attention mechanism that subsequent sections make use of. Revisiting the exemplary sentence from Section 4.2.1, it is common in the field of NLP to use word embeddings, i.e., numerical feature encodings for each word. The input sequence X is now given as $X = [\boldsymbol{x}_1, \ldots, \boldsymbol{x}_9]$, where $\boldsymbol{x}_1 = \text{embed}(\text{The}), \boldsymbol{x}_2 = \text{embed}(\text{man}), \ldots, \boldsymbol{x}_9 = \text{embed}(\text{water})$ and $\text{embed}(\cdot)$ denotes an arbitrary embedding function, e.g., bag of words [Agg18, Chapter 2.6], with $\boldsymbol{x}_i \in \mathbb{R}^{d_m}$. The hidden state $H \in \mathbb{R}^{9 \times d_m}$, which is

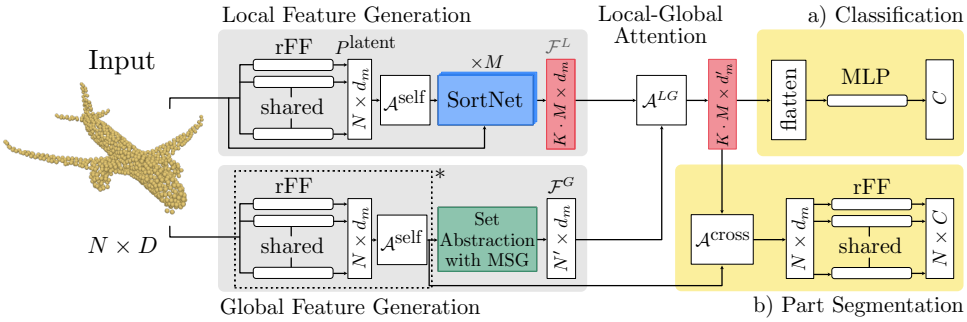


Figure 4.3: Overview of the Point Transformer architecture which consists of two branches to generate local and global features. SortNet produces an ordered set of local features that are attended against the global structure of the input point cloud. Depending on the task, classification or part segmentation heads are employed. Red Boxes denote sorted sets. * only for part segmentation [EBD21b]. Licensed under CC BY 4.0 - `https://creativecommons.org/licenses/by/4.0/`.

also called the context, is generated by applying the encoder stack, in other words, multiple multi-head attention blocks. From Equation (4.6), it is apparent, that the context has the same dimensions as the input sequence. The hidden state is generated by weighting each word with the importance score (4.1) of every other word in the sequence. Thus, attention considers all permutations regardless of position and distance. Since the approaches in this work deal with point cloud processing, explaining every concept of the Transformer architecture for NLP is out of scope for this thesis. However, the reader is referred to [BCB15; HM15; VSP+17], [GBC16, Chapter 12.4], [Agg18, Chapter 7.2.1] and [DL18] for further details of common natural language processing methods. The following section deals with framing the attention mechanism and Transformer model in the context of point cloud processing and introduces the Point Transformer architecture shown in Figure 4.3.

4.3 Methodology and Network Architecture

In general, the unordered point set

$$\mathcal{P} = \{\boldsymbol{p}_1, \ldots, \boldsymbol{p}_N\},\ \boldsymbol{p} \in \mathbb{R}^D \tag{4.8}$$

is considered, where D denotes the dimension a point and N specifies the total number of points, i.e., the cardinality of the point set. Usually the dimension is either set to $D = 2$ for 2D point clouds or $D = 3$ for 3D point clouds, however, if additional features are required, for example, lidar intensities or semantic labels, the input dimension D can easily be extended. The task is to map the point cloud $\mathcal{P}$ to the output space $\mathbb{R}^o$, where o denotes the output dimension, using the set function

$$\mathcal{F} : \mathcal{P} \to \mathbb{R}^o, \tag{4.9}$$

with $\mathcal{F}$ from Equation (2.5) [EBD21b]. Here, it is apparent that $\mathcal{F}$ is required to be a symmetric set function (2.8), i.e., $\mathcal{F} = \mathcal{F}_{\text{sym}}$ and $\mathcal{F}_{\text{sym}} : \mathcal{P} \rightarrowtail \mathbb{R}^o$. As explained in Section 2.1, processing point clouds requires a permutation-invariant function that is able to deal with arbitrary cardinality of the input set. Moreover, since real objects are processed in the proposed methods, for example lidar scans or points sampled from Computer-Aided Design (CAD) models, the relations between individual points as well as the objects shape should be considered. For that, the Point Transformer architecture is introduced, which consists of three main building blocks:

1. Local feature generation using **SortNet**. The SortNet module produces a permutation-invariant local feature set. For that, a task-dependent scoring of the input set is learned that is used to sort and select points with the highest score. The selected points act as key points for the object and define a spatial neighborhood that is used to aggregate local features. Since the key points are ordered, the result of the local feature generation module is ordered and permutation-invariant as well.

2. **Global feature generation** of the entire shape using feature aggregation networks to obtain a latent signature of the object.

3. **Local-global attention** to relate the learned local and global features and produce permutation-invariant latent descriptors that can be utilized for common computer vision tasks.

First, an overview of the complete architecture is given, which is visualized in Figure 4.3. Then, the individual modules are explained in detail. Here, an arbitrary point cloud with N points and D dimensions serves as input to the network, where two branches generate local and global features. The results are related using a cross-attention operation to obtain a permutation-invariant representation that can be used for common computer vision tasks. Additionally, the input point cloud $\mathcal{P}$ from Equation (4.8) is fed to the network as a matrix representation P, where the elements $\boldsymbol{p}_i \in \mathcal{P}$, $i \in \{1, \ldots, N\}$ are arbitrarily arranged row-wise, similar to Equation (2.57). Thus, the point cloud can directly be used as input, as the network produces a permutation-invariant and ordered latent feature vector.

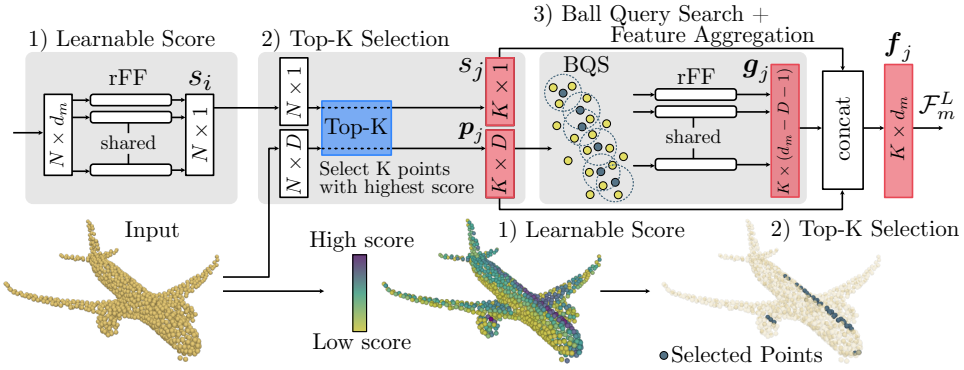


Figure 4.4: Overview of the SortNet architecture. A score s_i is learned from a latent feature representation to extract important points p_j from the input. Local features are then aggregated from local neighborhoods using the ball query search algorithm (BQS) and a subsequent feature aggregation module [QYSG17]. Finally, SortNet outputs a permutation-invariant and sorted feature set. Red boxes denote sorted sets. The bottom row illustrates the scoring and selection process on a sample point cloud. Based on [EBD21b].

4.3.1 Local Feature Generation and SortNet

The local feature generation branch first projects the matrix representation P of the point cloud $\mathcal{P}$ to latent space with dimension d_m using a row-wise feedforward network (rFF) from Equation (2.58), such that $P^{\text{latent}} = [\boldsymbol{p}_1^{\text{latent}}, \ldots, \boldsymbol{p}_N^{\text{latent}}]$, $\boldsymbol{p}_i^{\text{latent}} \in \mathbb{R}^{d_m}$, as shown in Figure 4.3. This transformation is performed to learn higher order dependencies. Then, the self-attention mechanism $\mathcal{A}^{\text{self}}(P^{\text{latent}})$ is directly applied to the latent features to find and learn relations between the input points $\boldsymbol{p}_i \in \mathcal{P}$. Afterward, SortNet produces a collection of permutation-invariant sorted sets of fixed size. In practice, multiple SortNet modules are employed to learn from different subspaces. Besides the projected features, SortNet also receives the original point cloud $\mathbb{R}^{N \times D}$ as input, which is shown in Figure 4.4. First a rFF reduces the feature dimension d_m to one in order to obtain the point-wise learned score $s_i \in \mathbb{R}, i \in \{1, \ldots, N\}$ for each input point. Since the multi-head attention layer precedes the score generation, spatial relations and shape information are encoded.

Next, the pair that assigns the learned score to the corresponding input point is defined as follows:

$$\langle \boldsymbol{p}_i, s_i \rangle_{i=1}^{N}. \tag{4.10}$$

Furthermore, let $(\mathcal{Q}, \geq)$ be a totally ordered set. Then, K points are selected from the original input point cloud $\mathcal{P}$, with $K \leq N$, and are assigned to the ordered set $(\mathcal{Q}, \geq)$, such that

$$\mathcal{Q} = \{\boldsymbol{q}_1 \ldots, \boldsymbol{q}_K\}, \tag{4.11}$$

where the pair of the learned score and the corresponding input point $\boldsymbol{q}_j = \langle \boldsymbol{p}_j, s_j \rangle_{j=1}^{K}, \boldsymbol{p}_j \in \mathcal{P}, s_j \in \mathbb{R}$ is defined such that $s_1 \geq \cdots \geq s_K \geq \cdots \geq s_N$. In other words, a scoring network learns an importance score for each point to sort the points in descending order. Then, a top-k selection operation is employed, that selects K input points with the highest score s_i. Using the K selected points, the ball query search algorithm introduced by Qi [QYSG17] is performed, that aggregates local features from neighboring points using a reduced PointNet++-based network topology. Thus, the K selected key points each define a learned local neighborhood. The resulting latent encodings are denoted by $\boldsymbol{g}_j \in \mathbb{R}^{d_m - 1 - D}, j \in \{1, \ldots, K\}$. Finally, the top-k selections $\boldsymbol{p}_j \in \mathbb{R}^{K \times D}$ with scores $s_j \in \mathbb{R}^{K \times 1}$, and the aggregated features from the ball query algorithm $\boldsymbol{g}_j \in \mathbb{R}^{K \times (d_m - 1 - D)}$ are concatenated

$$\boldsymbol{f}_j = \boldsymbol{p}_j \oplus s_j \oplus \boldsymbol{g}_j, \quad \boldsymbol{f}_j \in \mathbb{R}^{d_m}, \tag{4.12}$$

to form one sorted local feature set

$$\mathcal{F}_m^L = \{\boldsymbol{f}_1, \ldots, \boldsymbol{f}_K\}, \quad \mathcal{F}_m^L \in \mathbb{R}^{K \times d_m}, \tag{4.13}$$

with $m \in \{1, \ldots, M\}$ and M denotes the total number of SortNet modules that are employed. The learned score is used as an additional feature to include the top-k selection process in the optimization process. Since $\mathcal{Q}$ is an ordered set, it directly follows that $\mathcal{F}_m^L$ is ordered as well. Here, it becomes apparent that the dimension of the aggregated features $\boldsymbol{g}_j$, i.e., $d_m - 1 - D$, is chosen such that the resulting local feature dimension after the concatenation operation in Equation (4.12) is equal to d_m, which allows for easier processing of the local features. Moreover, M independent SortNet modules are used to learn dependencies and geometry information from different subspaces. Finally, the features from all SortNet modules are concatenated to form the sorted output set of the local feature branch

$$\mathcal{F}^L = \mathcal{F}_1^L \cup \ldots \cup \mathcal{F}_M^L, \quad \mathcal{F}^L \in \mathbb{R}^{K \cdot M \times d_m}. \tag{4.14}$$

Consequently, SortNet is comparable to the concept of kernels in convolutional neural networks, where the activation depends on different regions of the input space, i.e.,

the receptive field. Here, the proposed SortNet module works similarly: points of interest are selected and focused on depending on a learnable score, which is derived from the latent feature representation of the input points [EBD21b]. These points define a spatial neighborhood that is used to generate one local feature set. Since multiple SortNet modules are employed, local signatures from different regions of the input shape are considered.

4.3.2 Global Feature Generation

The global feature generation branch extracts latent features from the entire input point cloud, in contrast to the local feature branch, where only small neighborhoods of points are considered. The concept of producing global features is simple: a rFF projects the points to latent feature dimension d_m. Since the computational complexity of the attention mechanism scales quadratically with respect to the sequence length, i.e., the number of points, a multi-scale grouping layer (MSG) [QYSG17] from Section 3.1 is employed, which acts as a set abstraction mechanism in order to reduce the number of points while retaining the original shape. For that, a subsampling mechanism, such as the furthest point sampling method (FPS) [QSMG17], is used to reduce the total number of points to N' with $N' < N$. Additionally, a k-Nearest Neighbor (kNN) approach aggregates features with d_m dimension to counteract the downsampling process. The output is a global representation $\mathcal{F}^G \in \mathbb{R}^{N' \times d_m}$, that is still unordered since no sorting or set-pooling operations were employed. Here, it is important to note that the downsampling ratio, i.e., the number of points for the global feature generation N', is a hyperparameter that needs to be adjusted to consider computational complexity and performance requirements.

4.3.3 Local-Global Attention

After processing the input points using the local and global feature generation branches, the goal is to relate both feature sets, $\mathcal{F}^L$ and $\mathcal{F}^G$, such that shape information is captured and permutation-invariance is induced. The idea is that the local features contain information about small regions of the object, like distinct edges or shapes. Combined with the overall signature, i.e., the global features, this improves the network's ability to understand the object's geometry. For that, self-attention from Equation (4.6) is first applied to the local features $\mathcal{F}^L$ and global features $\mathcal{F}^G$, followed by a cross-attention layer from Equation (4.7). Cross-attention is deployed, such that each local region is scored against the entire global feature representation; in other words, the local context is related to the underlying shape.

This operation is called local-global attention, and it is defined as follows:

$$\mathcal{A}^{LG}(\mathcal{F}^L, \mathcal{F}^G) \triangleq \mathcal{A}^{\text{cross}}(\mathcal{A}^{\text{self}}(F^L), \mathcal{A}^{\text{self}}(F^G)), \tag{4.15}$$

where F^L and F^G denote the matrix representation of the local and global feature sets $\mathcal{F}^L$ and $\mathcal{F}^G$, respectively. Additionally, the last feedforward layers in the cross-attention layer, see Equation (4.5), reduces the feature dimension to $d'_m < d_m$ to decrease computational complexity, therefore $\mathcal{A}^{LG} : \mathbb{R}^{K \cdot M \times d_m}, \mathbb{R}^{N' \times d_m} \to \mathbb{R}^{K \cdot M \times d'_m}$. Again, the reduced dimension d'_m is a hyperparameter that decreases computational complexity but also affects performance. Here, it is important to note that the resulting representation is sorted, since the ordered local feature set is used as the queries for the attention mechanism, which defines the ordering of the output. Therefore, a permutation-invariant vector of fixed size is obtained without employing set-pooling operations and the aforementioned bottleneck is circumvented while retaining shape information, regardless of the size of the latent feature vector. Finally, the result of the local-global attention mechanism can directly be used for common computer vision tasks.

4.3.4 Complete Network Architecture

To summarize, the Point Transformer architecture consists of two main branches: The local and global feature generation modules, where SortNet is an integral part to obtain permutation-invariant local features by selecting and sorting key points based on a learned score. The output of both branches is combined using the proposed local-global attention mechanism such that the resulting latent feature representation of the input point cloud is sorted and permutation-invariant as well. Finally, two exemplary output heads for common 3D computer vision tasks are introduced, namely the shape classification and the part segmentation task.

Shape Classification

The goal of shape classification is to assign one of C unique object class labels to the input point cloud. The shape classification module is visualized in Figure 4.3 a). The latent feature representation of size $K \cdot M \times d'_m$ is flattened to a permutation-invariant vector of fixed size $\mathbb{R}^{M \cdot K \cdot d'_m}$. The vector is then transformed to the output space $\mathbb{R}^C$ using a row-wise feedforward network (rFF) to produce a probability for each of the C classes. This is done by applying a final softmax layer (2.64) in order to obtain a valid probability distribution, cf. Section 2.2.1. Finally, the class with the highest probability is chosen as the predicted object label for the input point cloud.

Part Segmentation

For the task of part segmentation, the goal is to assign a label to each individual point of the point cloud, representing what part of the object the point belongs to. For that, one additional cross multi-head attention layer, see Equation (4.7), is employed to attend the output of the Point Transformer to each point of the input point cloud. Note that contrary to the shape classification task, an additional self-attention mechanism with a preceding rFF is applied in the global feature branch, illustrated in Figure 4.3 by an asterisk *, to project the input to higher dimensional space $\mathbb{R}^{N \times d''_m}$. This representation, together with the sorted output of Point Transformer, is used for the aforementioned cross multi-head attention block, which results in a sorted matrix of size $\mathbb{R}^{N \times d'_m}$. Here, it is evident that the input to a cross multi-head attention layer can be of different dimensions, but the output size corresponds to the shape of the first input matrix, i.e., the queries. Finally, the row-wise feedforward network (rFF) reduces the dimension of each point to the output space $\mathbb{R}^{N \times C}$, where again C denotes the number of possible classes for each point. Similar to the shape classification task, a final softmax layer applied dimension-wise produces a valid probability distribution over the class labels for each point.

4.4 Experiments and Evaluation

Based on the introduced shape classification head and the part segmentation head, two experiments are performed using the Point Transformer network on public and widely used benchmarks, namely the ModelNet40 [WSK+15] und ShapeNet [CFG+15] datasets. For both experiments, Point Transformer is compared to state-of-the-art approaches that use set-pooling methods [QSMG17; QYSG17; ZKR+17], approaches that are based on the attention mechanism [LHLZ19; LLK+19; TZWL21; YZN+19], as well as point cloud convolution networks [LBS+18; TQD+19; XFX+18]. Finally, an extensive model design analysis is provided with an ablation study of the local and global feature generation modules. Additionally, the robustness of SortNet against point cloud rotations is evaluated, and a visualization of the capability of selecting important key points is given. For the evaluations, Point Transformer is implemented using the popular PyTorch framework [PGM+19]. Additionally, the RAdam optimizer [LJH+20] from Section 2.4.5 is used for training the network and the Mish activation function from Equation (2.62) is used for all layers. Additional implementation details and a detailed overview of hyperparameters are provided in the Appendix A.1.

Table 4.1: Evaluation results of Point Transformer for object classification (ModelNet40) and part segmentation (ShapeNet) tasks. Provided values denote the overall accuracy for classification and the mean IoU for segmentation. −: Result not available.

Method	ModelNet40 [WSK+15]	ShapeNet [CFG+15]
PointNet [QSMG17]	89.2	83.7
PointNet++ [QYSG17]	91.9	85.1
ShapeContextNet [XLCT18]	89.8	84.6
Deep Sets [ZKR+17]	90.3	-
Point2Sequence [LHLZ19]	92.6	85.2
Set Transformer [LLK+19]	90.4	-
PAT [YZN+19]	91.7	-
Tao et al. [TZWL21]	87.5	75.2
Point Transformer	92.8	85.9
KPConv [TQD+19]	92.9	86.2
PointCNN [LBS+18]	92.2	86.1
SpiderCNN [XFX+18]	90.5	85.3

4.4.1 Shape Classification

For the shape classification task, Point Transformer is evaluated on the ModelNet40 dataset [WSK+15], where the modified version by Qi et al. [QYSG17] is used. The dataset provides point clouds from 40 different categories, thus $C = 40$, that were sampled from the mesh of various CAD models. For each object, 10.000 points are provided with normal vectors for each point. Therefore, the input dimension is $D = 6$, with $\boldsymbol{x}_i = [x, y, z, n_z, n_y, n_z]$, where $\boldsymbol{n} = [n_x, n_y, n_z]$ is the normal vector that is perpendicular to the objects surface at point $[x, y, z]$. The normal vectors are used for the evaluation to obtain a fair comparison with related approaches since it is a popular input format. However, it is important to note that it is possible to perform the classification task with coordinate information only, i.e., $\boldsymbol{x}_i = [x, y, z]$. In total, 9843 shapes are provided for the training set and 2468 for the test set. Additionally, random scaling in the range of $[0.8, 1.25]$ and random translating in the range of $[-0.1, 0.1]$ are used for augmentation purposes. Finally, the hyperparameters are set to $N = 1024$, $D = 6$, $d_m = 512$, $d'_m = 64$, $M = 4$ and $K = 64$. As mentioned above, additional information about the implementation can be found in Appendix A.1. The results are shown in Table 4.1, where the overall accuracy is provided; that is, the total percentage of correctly classified shapes. As the evaluation scores indicate, Point Transformer outperforms related approaches that use the attention mechanism and traditional set-pooling-based network archi-

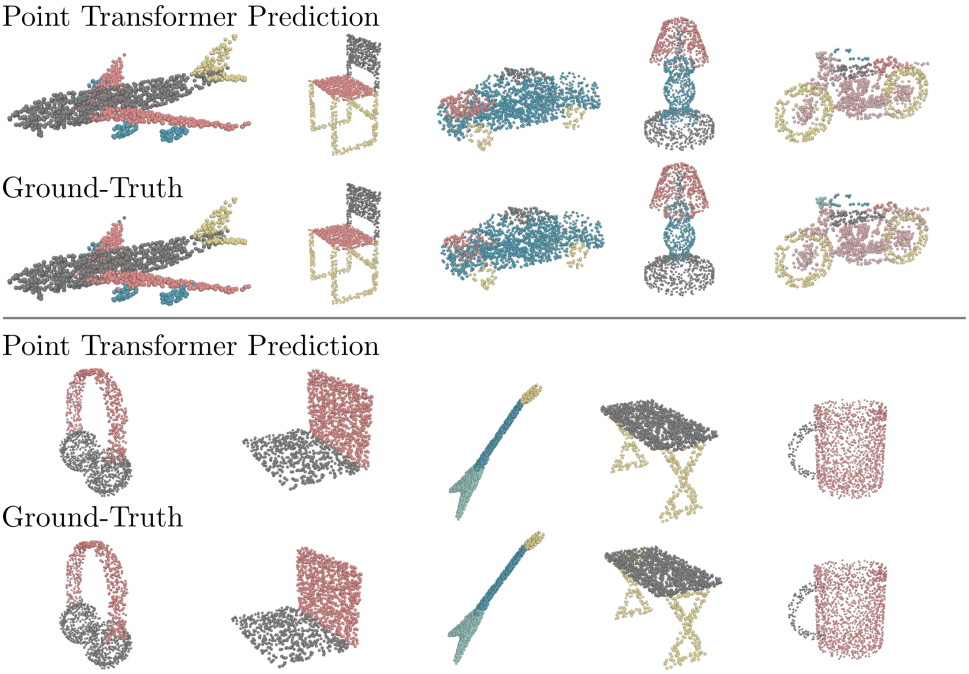


Figure 4.5: Exemplary visualizations of results from the part segmentation task. The predictions of Point Transformer are compared to the ground-truth labels. Here, every part is depicted with a different color. It is apparent, that Point Transformer accurately predicts the part labels.

tectures. Compared to point convolution approaches, Point Transformer achieves on-par performance, with a total classification accuracy of 92.8%.

4.4.2 Part Segmentation

The task of point cloud part segmentation involves the assignment of each point to a part class. For this, the popular ShapeNet dataset [CFG+15] is used, which provides $C = 50$ different part labels for 13.998 train samples and 2874 test samples in 16 different object categories, as exemplarily shown in Figure 4.5. For this challenging task, the network has to learn a deep understanding of the objects shape and inter-point relations. Therefore, the part segmentation head from Figure 4.3 b)

Table 4.2: Model complexity study. Here, the network size and the inference time is evaluated against related approaches for the results reported in Table 4.1. −: Result not available.

Method	# of params	Network size	Inference time
Point Transformer	13.5 M	51 MB	110 ms
SortNet	10 k	0.04 MB	1.25 ms
PAT [YZN+19]	-	5.8 MB	88 ms
KPConv [TQD+19]	15 M	-	210 ms
PointNet++ [QYSG17]	2.1 M	24 MB	160 ms
Point2Seq [LHLZ19]	2 M	-	-

is employed and the hyperparameters are set as follows: $N = 1024$, $D = 6$, $d_m = 512$, $d'_m = 256$, $d''_m = 256$, $M = 10$ and $K = 16$. Following the ideas of [QSMG17] and [QYSG17], a one-hot encoding of the object class is appended as additional input features. The mean Intersection over Union (mIoU) is used as the performance metric. As the name suggests, the IoU specifies the intersection area of each predicted part with the ground-truth part divided by the union area. For the case of per-point part segmentation, the intersection denotes the total number of correctly classified points for a single class. The union is obtained by calculating the total number of points in that class category from both the ground-truth and the prediction set combined, whereas correctly classified points are counted once, as illustrated in Figure 2.2. Evidently, a higher IoU denotes a more accurate prediction result and vice-versa. The mean IoU averages the IoU of all part classes of the object. For a more in-depth explanation of the concept, the reader is referred to, e.g., [QYSG17]. The final results are given in Table 4.1, where Point Transformer again outperforms related attention-based and set-pooling-based approaches with a mean IoU of 85.9. Finally, a visual comparison of the predictions of Point Transformer and the ground-truth is shown in Figure 4.5. Additional visualizations are provided in Appendix A.2.2.

4.4.3 Network Complexity Study

This section evaluates and compares the computational complexity to related approaches. All experiments regarding the inference time are performed on an Nvidia GeForce 1080Ti graphics card. The results are given in Table 4.2. First, the total number of learnable parameters is evaluated, which is a suitable metric for comparing network complexity, i.e., network depth and memory requirements. As shown in Table 4.2, Point Transformer has approximately 13.5 million learnable parameters which results in a memory requirement of about 51 MB. In comparison to a point

Table 4.3: Hyperparameter study results on the ModelNet40 dataset for different combinations of the hyperparameters M (number of SortNets) and K (top-k selections).

M	4	4	4	4	8	8	8	8	8	16	16	16	16	32	32	32
K	16	32	64	96	8	16	32	64	96	8	16	32	64	8	16	32
Accuracy	91.7	92.3	**92.8**	91.7	90.2	90.5	91.9	92.4	92.0	91.2	91.6	92.0	91.7	90.8	91.3	91.1

convolution approach, e.g., KPConv [TQD+19], Point Transformer requires fewer parameters and less memory. However, set-pooling based methods, for example, Point2Seq [LHLZ19] and PointNet++ [QYSG17], outperform related approaches with regard to the computational complexity. This is mainly due to the fact that the attention mechanism itself has a high demand for memory resources. Nevertheless, one SortNet module only accounts for about 10.000 learnable parameters, which shows that the module can be added to existing network architectures without negatively impacting the network complexity or the inference speed. In particular, Point Transformer achieves faster inference times compared to KPConv and PointNet++ because multi-head attention blocks are highly optimized regarding parallel computation. Finally, since the attention mechanism has the highest complexity of the Point Transformer architecture, the lower bound of the complexity can be estimated to $\mathcal{O}(N^2 \cdot d_m)$ [VSP+17], i.e., the complexity scales quadratically with respect to the point set cardinality N since d_m is a fixed hyperparameter.

4.4.4 Network Design Analysis

In the following, multiple experiments are performed to examine the performance and effectiveness of the Point Transformer approach.

Hyperparameter Ablation Study

Here, the impact of different choices for the number of deployed SortNet modules and the number of top-k selections is investigated. The result of the hyperparameter ablation study on the ModelNet40 dataset is shown in Table 4.3, where the accuracy is presented for different hyperparameter choices. Furthermore, an overview of the results of a hyperparameter grid search experiment is provided in Appendix A.1, where additional values are stated that achieved the best overall performance on the classification and part segmentation tasks.

Table 4.4: Results of the network design analysis. Different SortNet architectures are evaluated to highlight the benefit of employing a learnable score.

Experiment	Accuracy (%)
SortNet with learnable score	83.4
SortNet with FPS	74.8
SortNet with random point selection	60.1

SortNet Ablation Study

For this ablation study, only the SortNet module from Figure 4.4 is evaluated on the shape classification task with the classification head from Figure 4.3 a), without employing the global feature branch or the local-global attention mechanism. That way, it is guaranteed that the impact of different SortNet approaches is evaluated. In particular, this section aims to demonstrate that learning a score to select and sort important key points from the entire point cloud benefits the ability to learn shape and geometric information for different computer vision tasks. Since it is impossible to define important key points heuristically, the benefits are indicated with regard to the classification accuracy score. Specifically, SortNet is trained in three different variations with $M = 10$ and $K = 12$, thus selecting only a small subset of the entire input point cloud, i.e., $M \cdot K = 120$ and $N = 1024$. First, SortNet is evaluated as implemented in the Point Transformer pipeline with a learnable score and a top-k selection process that creates permutation-invariant latent features. In the second experiment, the learned selection process is replaced with the heuristic furthest point sampling (FPS) method, which randomly selects points while retaining as much of the original shape as possible. Finally, a random sampling approach is evaluated. Here, it is important to note that permutation-invariance is not guaranteed for the last two experiments since no symmetric set function is applied to the input set. However, this experiment aims to show that selecting key points based on a learned score increases performance over random selections or handcrafted sampling methods, e.g., FPS; therefore, these methods are used as baselines. Table 4.4 shows that the random selection only achieves 60% classification accuracy, which is expected because the random downsampling may discard important relations. Next, sampling points using the furthest point heuristic slightly improves the accuracy to 75%, which shows that retaining shape information helps the overall performance. Finally, SortNet without the Point Transformer pipeline, as mentioned above, already achieves a classification accuracy of 83%, which empirically supports the claim that the learned selection focuses on important local regions of the object. Therefore, deploying SortNet not only induces permutation-invariance but also helps improve the shape

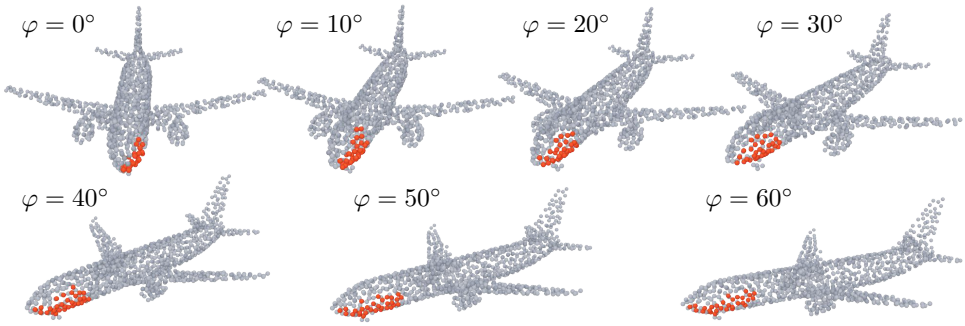


Figure 4.6: Visualization of the rotation robustness of SortNet. The top-k selections (•) are highlighted to show that SortNet focuses on similar local regions regardless of how the input is rotated.

understanding capability of the network. Further qualitative results are illustrated in Appendix A.2.1, where additionally, it is shown that the combination of all SortNet modules selects key points such that the shape of the original objects remains intact.

Rotation Robustness of SortNet

Additionally, SortNet is able to be robust against input point cloud rotations, which is an important characteristic of point processing networks. To support this statement, Point Transformer is again evaluated on the ModelNet40 dataset, where the input is randomly rotated around the z-axis. Note that no rotations were applied to the input during the training process; thus, the network has not seen any rotated samples. In order to present quantitative results, Point Transformer is compared to PointNet++. When the test set is rotated, the classification accuracy of Point Transformer drops from 92.8% to 92.3%, which is a decrease of only 0.5%. However, when the same rotations are applied to the input of PointNet++, the classification accuracy drops 3.3% from 91.9% to 88.6%, indicating that Point Transformer can better handle arbitrary rotations. The robustness against rotations is also visualized in Figure 4.6 and Appendix A.2.1, where the top-k selections of one SortNet are highlighted for different rotated input point clouds of the same object. The qualitative analysis shows that one SortNet focuses on the same spatial region of the object regardless of how the input is rotated, for example, the front right edge of the airplane marked red in Figure 4.6.

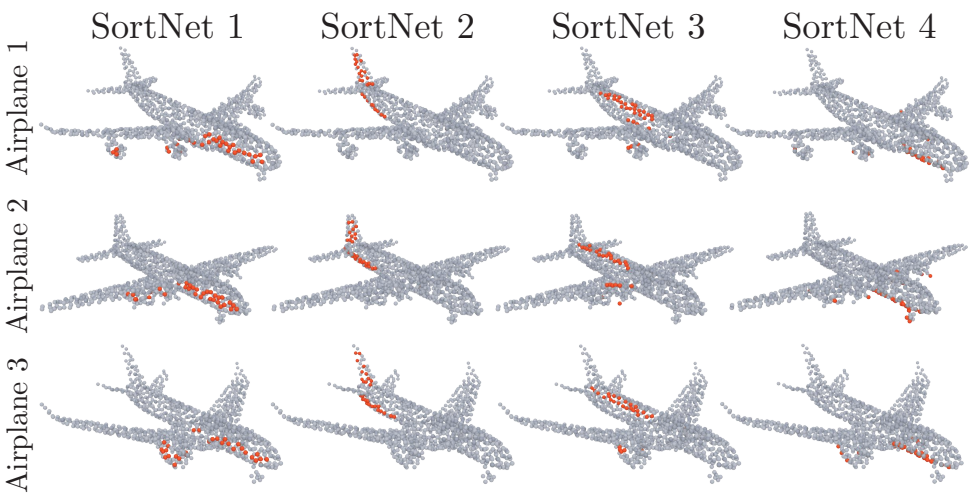


Figure 4.7: Shape awareness of SortNet: The top-k selections (•) of different SortNet modules are highlighted to show that each instance focuses on the same local region, even when the geometric shape is slightly different across different objects of the same class.

Visualizations of Learned Local Regions and Top-k Selections

Finally, this section focuses on the learned local regions and top-k selections of the SortNet modules to explain why SortNet resembles the receptive field of a CNN. For that, the top-k selections of four separate SortNet modules are highlighted in Figure 4.7. Additionally, Figure 4.7 also shows three different objects of the same object category (airplanes). It is apparent that SortNet 1 and SortNet 4 learned to focus on the front part of the airplane, regardless of a specific object instance. The other two SortNet modules (SortNet 2 and SortNet 3), however, select local regions in the rear end, even when the shape slightly varies, which is comparable to recent discoveries for the receptive field of convolutional neural networks, e.g., [LLUZ16]. The visualization of the top-k selections and the rotation robustness indicate that the SortNet module is aware of the object's shape and can focus on important key points. Again, additional visualizations of top-k selections for multiple object classes are shown in Appendix A.2.1.

Chapter 5

Localization Datasets

This chapter introduces the datasets that are used to train the deep learning-based localization approaches. For that, the test vehicle is presented that is used for automated driving scenarios at the Institute of Measurement, Control and Microtechnology at Ulm University. Moreover, the necessary algorithms and processes to generate the digital maps from recordings of sensor measurements are explained. In total, two datasets are created for this thesis: The DeepLocalization dataset and the Point Transformer Localization dataset. The first dataset uses a multi-modal sensor setup to create sparse 2D measurements and a generic landmark-based digital map. Then, a lidar dataset is presented that contains a high-density point cloud-based representation of the environment with semantic and geometric information of static and persistent objects. Consequently, each dataset is comprised of two parts: A collection of sensor measurements that are provided in the vehicle frame and a digital map in global coordinates, e.g., UTM coordinates. Moreover, the ground-truth pose is included for every sample as a reference for the evaluation.

5.1 Experimental Setup

The experimental test vehicle that is used to record the data samples in this thesis is shown in Figure 5.1. It can operate autonomously by perceiving the environment through a multi-modal sensor setup from which suitable driving strategies are inferred. Then, a combination of planning and control algorithms execute the calculated trajectories [KNW+15]. In order to ensure a safe and reliable operation, a multitude of software modules has to communicate and work intertwined, whereas the task of vehicle self-localization plays an important role, as explained in Section 2.3. For that, the test vehicle is equipped with different sensors to provide information about the environment, which is necessary to navigate safely in dynamic scenarios. In the following, the sensor setup is presented, which is used to obtain an accurate



Figure 5.1: Experimental vehicle equipped with an extensive sensor setup that is used to record and create the datasets for this thesis.

estimate of the vehicle's pose. The test vehicle shown in Figure 5.1 is equipped with a global navigation satellite system (GNSS) that provides information about the global pose of the vehicle, e.g., in geocentric coordinates (latitude and longitude) or projected coordinates (Universal Transverse Mercator, UTM). However, commercially available GPS systems usually do not meet the localization accuracy requirements for automated driving in urban scenarios due to atmospheric disturbances, multi-path effects, or line of sight occlusions of the satellites, cf. Section 2.3. Therefore, a differential GPS system (dGPS) is employed, that uses correction data from ground-based reference stations to correct the aforementioned interferences. The correction data is received through standard mobile communication networks. However, when the clear line of sight to the GNSS satellites is occluded, for example, by buildings or bridges, no pose estimate can be provided. For that, the system is additionally equipped with a fiber-optic gyroscope, which is a high-precision inertial measurement unit (IMU), that makes angular acceleration rates available to support the prediction of the vehicle's motion. In combination, the dGPS system together with the IMU can obtain pose information with an accuracy in the range of a few centimeters. In this thesis, the high-precision dGPS measurements are used as a reference for the evaluation and the creation of the digital map. Furthermore, the test vehicle perceives the environment through various sensors, which can be categorized by the type of collected data. First, time-of-flight sensors calculate the distance to another object by measuring the time the signal needs to return to the receiver, i.e., the time between the transmission and reception of a signal. Lidar and radar sensors are popular choices in the field of robotics and automated driving that belong into the time-of-flight category, whereas the difference lies in the electromagnetic spectrum, e.g., lidar sensors use laser pulses (infrared light) and radar employs radio

waves. Lidar or laser sensors can have one or more layers and either be static or involve rotating mirrors that distract the light to obtain a 360° representation of the environment. Besides the distance for each reflected object, lidar sensors often provide the reflectivity of the object as an additional feature. Compared to radar, lidar point clouds tend to be denser; however, it is possible to calculate the velocity of the reflected object from radar measurements, i.e., Doppler velocities. Finally, a stereo-camera setup is deployed that, besides providing a RGB image, also supplies a disparity image that assigns a depth to each pixel of the image (RGB-D). For that, a precise calibration between both cameras is required that affects the accuracy of the disparity image. A more in-depth introduction for the employed sensors can be found in [WHW09; YLCT20; YVBW21].

5.2 DeepLocalization Dataset

The DeepLocalization dataset is created using the mapping techniques introduced by Stübler et al. [SRD17; Stü18; SWD15]. In the following, the creation of measurements from the multi-modal sensor setup and the digital map is introduced. Again, a more in-depth explanation is provided in [EHH+19; Stü18]. The dataset covers rural and urban areas on a 5 km long test track in Ulm-Lehr, Germany, that is visualized in Figure 5.2. Additionally, multiple challenging scenarios for the task of automated driving are included, such as roundabouts, merging lanes, intersections, crowded areas, and traffic lights. Multiple runs were recorded to create the digital map and the measurements. Three recordings for the digital map were saved over a span of multiple days in December 2017. In order to obtain a diverse dataset that is close to real-world conditions, e.g., where notable differences between the map and the measurements occur, the recordings for the measurements were conducted approximately one year later, in November 2018. In total, eight runs were performed to collect the measurement dataset.

5.2.1 Measurements

The measurement part of the DeepLocalization dataset contains a total of 145.000 samples from the aforementioned eight runs on the test track that is shown in Figure 5.2. Each sample includes an arbitrary amount of processed measurements from lidar, radar and camera images, as well as high-precision pose information of the vehicle from the dGPS system. In the following, the processing steps are explained, which are employed to obtain the processed measurements from the raw sensor readings to achieve a consistent and uniform measurement representation.

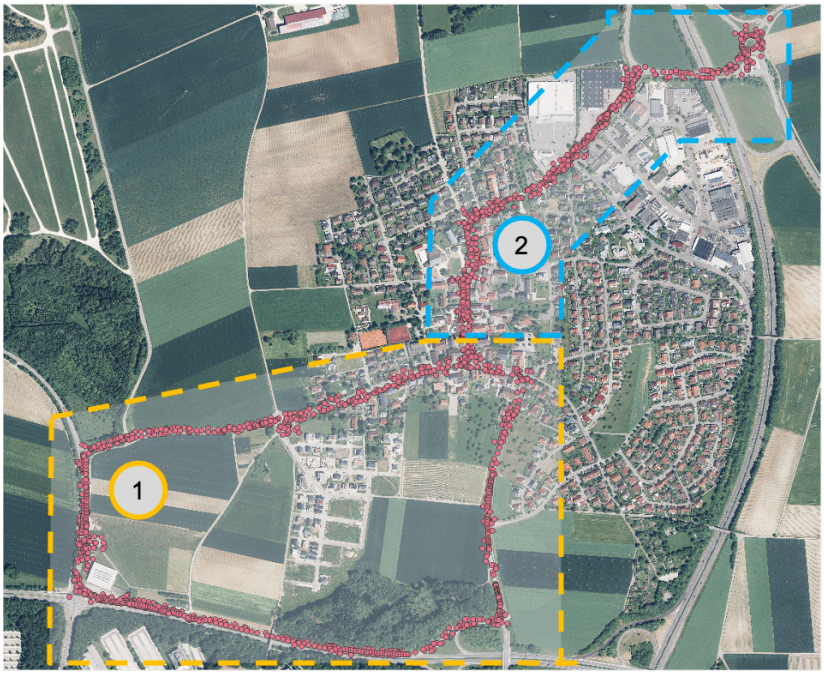


Figure 5.2: Test track in Ulm-Lehr with rural and urban scenarios. Landmarks are shown as red dots (•). The additional Train / Test split is illustrated as (1) / (2), respectively. [EBD21a] © 2021, IEEE. Geobasisdaten © LGL, `www.lgl-bw.de`.

A measurement that is provided directly from the sensor is considered to be a *raw measurement*. For the sake of simplicity, a raw measurement processed or altered by one of the following algorithms is called a *measurement* for the remainder of this thesis. For the generation of the DeepLocalization measurements, two different algorithms are employed. Since both the raw lidar and the raw radar measurements can be converted into a similar data format, e.g., range information for each reflected object, the preprocessing steps are identical. Considering that sensor measurements are subject to noise and clutter or ghost measurements and one target per object is desired for this approach, all raw lidar and radar measurements are clustered using the popular density-based spatial clustering of applications with noise (DBSCAN) algorithm [EKSX96]. The goal is to combine adjacent measurements with many neighboring points that belong to the same object and discard any noise. The algorithm has two hyperparameters: $minPts$ and ϵ and tries to assign each point of the input set to one of three categories: core points, reachable points, and outliers. Core points have at least $minPts$ points within the distance ϵ. Furthermore, a point

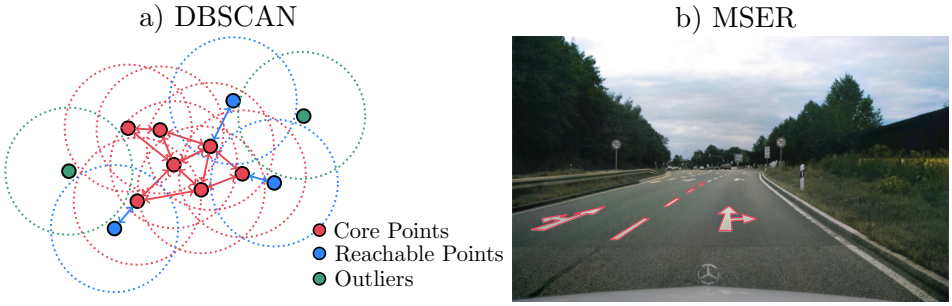


Figure 5.3: Overview of the DBSCAN algorithm [EKSX96] (left). Red points are core points that have at least $minPts = 2$ neighboring points within the distance ϵ. Reachable points are within the denoted ϵ distance of a core point and outliers can not be reached by any core points. The MSER algorithm [MCUP04] (right) finds white blobs in the image, e.g., road markings, denoted as red polygons.

is said to be reachable if it is within the distance ϵ of a core point. Then, clusters are formed using all core points that are in the neighborhood ϵ, together with all reachable points from that cluster. Consequently, if a point is neither a core point nor reachable from a cluster, it is discarded as an outlier. The algorithm is also illustrated in Figure 5.3a). Using the DBSCAN algorithm, the raw measurements from the lidar and radar sensors are clustered individually. For each cluster, the 2D mean and variance are calculated. The mean $\boldsymbol{\mu} = [x, y]^T$ is then used as the position and the variance as the spatial extent of the processed measurement. The position is then stored in the dataset. The lidar measurements set $\mathcal{M}^{\mathrm{l}}$ and radar measurement set $\mathcal{M}^{\mathrm{r}}$ are given as follows:

$$\mathcal{M}^{\mathrm{l}} = \{\boldsymbol{m}^{\mathrm{l}}_1, \ldots, \boldsymbol{m}^{\mathrm{l}}_{\pi^{\mathrm{l}}}\},\ \mathcal{M}^{\mathrm{r}} = \{\boldsymbol{m}^{\mathrm{r}}_1, \ldots, \boldsymbol{m}^{\mathrm{r}}_{\pi^{\mathrm{r}}}\}, \boldsymbol{m}^{\mathrm{l}}, \boldsymbol{m}^{\mathrm{r}} \in \mathbb{R}^2, \tag{5.1}$$

where π^{l} and π^{r} denote the total number of lidar and radar measurements per scan, respectively.

Next, the preprocessing steps for obtaining measurements from camera images are explained. The goal is to extract features from the image that describe static and easily recognizable regions. In the literature, many popular feature extraction algorithms exists, e.g., SIFT [Low04] or ORB [RRKB11]. However, this thesis follows the ideas of [Stü18], where distinct blobs are located inside the image using the maximally stable extremal region (MSER) algorithm [MCUP04]. Here, it is important to note that the image is converted to grayscale prior to processing. Without going into further detail, the MSER algorithm extracts so-called co-variant regions or blobs $\mathcal{Q}$, i.e., maximally stable extremal regions, which are differentiable

and stable components of a gray-level region of a binary image. Binarization converts an image to either black or white values depending on a binarization threshold, e.g. pixels with a gray value above the threshold are set to white and vice-versa. Here, $\mathcal{Q}$ is characterized by uniform intensities surrounded by a background with high contrast. Furthermore, the boundary of $\mathcal{Q}$ is defined as δQ, i.e., the set of pixels that do not belong to Q but are adjacent to at least one pixel of $\mathcal{Q}$. Next, a region $\mathcal{Q}$ is said to be a maximally stable extremal region if the boundary remains invariant after applying different binarization thresholds [MCUP04; Stü18]. For example, it is possible to adapt the parameters to detect white blobs in the image. In the case of automated driving scenarios, these white regions could be lane or road markings, which are static and almost always recognizable, as shown in Figure 5.3b). Another advantage of detecting road markings is that the location of road markings is on the ground plane with $z = 0$. Therefore, requiring a disparity image is avoided since the detected blobs can be projected to the ground plane, cf. [Stü18]. After finding all features in the current image, the hulls and image moments are calculated to extract the center of each blob, which acts as the 2D position of the measurement. Consequently, the camera measurements are given as follows:

$$\mathcal{M}^{\mathrm{c}} = \{\boldsymbol{m}_1^{\mathrm{c}}, \ldots, \boldsymbol{m}_{\pi^{\mathrm{c}}}^{\mathrm{c}}\}, \boldsymbol{m}^{\mathrm{c}} \in \mathbb{R}^2, \tag{5.2}$$

where π^{c} denotes the number of extracted measurements from one grayscale image. Finally, the total measurement set for one sample is the union of all sensor measurement sets

$$\mathcal{M} = \mathcal{M}^{\mathrm{l}} \cup \mathcal{M}^{\mathrm{r}} \cup \mathcal{M}^{\mathrm{c}}, \mathcal{M} \in \mathbb{R}^{\pi \times 2}, \tag{5.3}$$

with $\pi = \pi^{\mathrm{l}} + \pi^{\mathrm{r}} + \pi^{\mathrm{c}}$. It is apparent, that the DeepLocalization dataset is comprised of all three sensor modalities. As mentioned above, besides the measurement set $\mathcal{M}$, the dataset also provides the current ground-truth pose $\boldsymbol{p}_{\mathrm{GT}}$ obtained from the dGPS system. Finally, when obtaining a measurement sample $\mathcal{M}$ for time step t, it is denoted by $\mathcal{M}_t$.

5.2.2 Mapping

This section serves as a broad overview of the map creation process. A thorough introduction can be found in [SRD17; Stü18]. The idea of the mapping algorithm is to collect preprocessed measurements as explained in Section 5.2.1 and store perceived objects that are static, persistent, and easily recognizable as landmarks. The method described in [Stü18] uses a temporal filtering approach to track possible landmark candidates over time using one or more runs on the test track. For that, a Bernoulli filtering approach is employed [SRD17] that sets up new landmarks and

associates and tracks measurements to existing landmarks. Moreover, the algorithm assigns an existence probability p_{ex} to every possible landmark that increases if subsequent measurements confirm the existence of that landmark. Similarly, the existence probability decreases if the object is not associated even though it should be visible. After processing all samples, unlikely landmarks can be discarded using a threshold p_{thres} for the existence probability, and all landmark candidates are stored in the digital map

$$\mathcal{L} = \{\boldsymbol{l}_1, \ldots, \boldsymbol{l}_\nu\}, \boldsymbol{l} \in \mathbb{R}^2, \tag{5.4}$$

where $p_{\text{ex}} > p_{\text{thres}}$. Thus, it is possible to discard measurements that originated from dynamic objects or were caused by faulty measurements. During the entire mapping process, the precise ground-truth pose $\boldsymbol{p}_{\text{GT}}$ from the dGPS system is used to provide the algorithm with centimeter-level position accuracy of possible landmarks in global coordinates. The pose is used to transform the identified landmarks from sensor coordinates into the global map frame.

In summary, the landmarks for the DeepLocalization dataset represent easily recognizable and static objects in the environment, that were detected by either radar, lidar or camera sensors, e.g., traffic lights, poles, or trees. However, no semantic information about the type is associated with a landmark; therefore, a landmark is defined as a generic representation of a static object. Radar and lidar measurements are clustered using the DBSCAN algorithm and white blobs, e.g., road markings, are extracted from camera images using the MSER algorithm. Furthermore, a Bernoulli filtering approach accumulates landmarks from multiple scans and discards landmark candidates with a low existence probability. After the map creation process, all suitable landmarks are persisted, whereas the position is obtained from the mean of the cluster (for lidar and radar landmarks) or the center of the hull (for camera landmarks). Finally, the position of each landmark is transformed to global coordinates using the high-precision dGPS measurements.

The final digital map with the generic landmarks is shown in Figure 5.2. In total, the map is comprised of 3860 landmarks, where 1731 were obtained from lidar measurements, 1411 from camera images, and 718 from radar targets. The landmarks on the 5 km long test track require approximately 600 kB of memory storage. In total, the dataset provides two different test and train splits. First, a uniform split where six runs on the test track ($\approx$ 105.000 samples) are used for training, and the remaining two runs ($\approx$ 40.000 samples) are used for testing. In this variation, the same environment and spatial regions are used for training and testing. Second, to evaluate the generalization capabilities, the test area is split according to the regions marked ① and ② in Figure 5.2. Here, ① denotes the area used for training the network and ② defines the area from which test samples are obtained. Thus, it is possible to train a network in one region and evaluate the performance in a

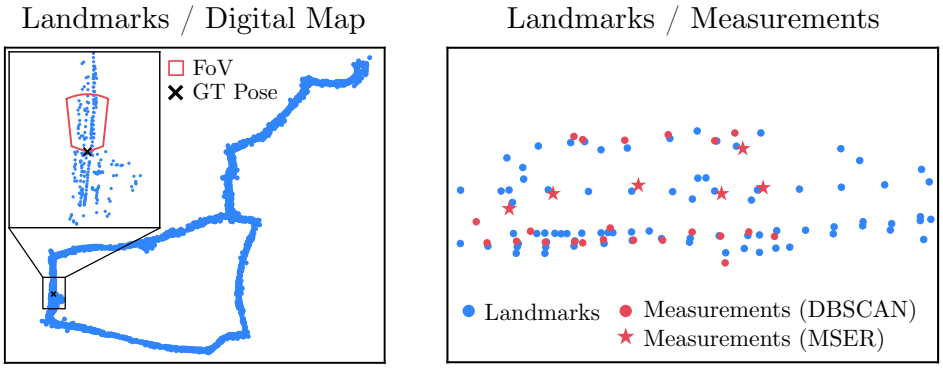


Figure 5.4: Overview of one input sample that contains the measurement and landmark sets. The landmarks are loaded from the digital map based on the ground-truth position (left). The right image depicts the landmarks in the vehicle's field of view and the multi-modal sensor measurements both transformed into the vehicle frame. Additionally, measurements originating from radar and lidar sensors (DBSCAN) and camera images (MSER) are distinguished.

different region the network has not seen before. Finally, an overview of an exemplary landmark and measurement sample is shown in Figure 5.4 for one time step t. The landmarks in the vehicle's FoV are loaded from the digital map based on the ground-truth position. The resulting measurement and landmark sets contain the 2D representations after the preprocessing pipeline (DBSCAN and MSER), where both point sets are transformed into the vehicle coordinate system, as shown in the right image. Here, a distinction is made for the visualization that highlights the fact, that measurements from camera images are more likely to originate from the middle of the road, e.g., from road markings.

5.3 Point Transformer Localization Dataset

As explained earlier, the Point Transformer Localization dataset is comprised of high-density lidar scans obtained from a 32-layer Velodyne VLP32 lidar sensor. Therefore, both the measurement and the digital map creation need to be adapted to handle dense point clouds, where the total number of points is greater than 10.000. Additionally, the goal is to include semantic and geometric information about the environment; thus, the Point Transformer Localization dataset does not contain generic and single landmarks but point sets reflected from the surface

of each static and persistent object. The increase of the point cloud cardinality correlates with an increase in computational requirements as well as a high workload to assign semantic information to each point. In order to solve this problem, the semantic labels of the lidar scans are obtained by employing a pretrained deep neural network. The processing steps for the measurement generation and the mapping process are outlined in the following. In total, four recordings in August 2020 on the same test track in Ulm-Lehr, cf. Figure 5.2, are used for the Point Transformer Localization dataset. The recordings include the lidar scans as well as the ground-truth pose from the dGPS system, which is used for the map generation and evaluation.

5.3.1 Measurements

This section introduces the creation of the measurement point clouds, including the semantic segmentation network that detects static environment information. An important requirement for the deep neural network that provides semantic labels is that the inference time is as low as possible since it should have a low impact on the overall processing time. Recently, Hu et al. proposed RandLA-Net [HYX+20], a lightweight architecture that handles large-scale point clouds as input. The method uses random point sampling in combination with a feature aggregation component that incorporates local neighborhoods to preserve geometric information while providing fast inference times of about 22 frames per second, cf. Section 3.1. Therefore, RandLA-Net is an ideal network architecture choice for the dataset. The ideas for the measurement generation follow the principles of the DeepLocalization dataset: only static and persistent objects in the environment are of interest for the localization task. Hence, the goal is to discard any points originating from dynamic objects by relying on semantic segmentation for each scan. The labeled dataset that is used to train RandLA-Net for the task of semantic segmentation is the SemanticKITTI dataset [BGM+19; BGM+21] that provides per-point class labels for various lidar sequences. Here, it is important to note that the network was trained on a different dataset and the inference is applied on lidar scans that were recorded using the test vehicle introduced in Section 5.1. However, a qualitative evaluation of the segmentation results on the recorded scans showed satisfying results that are well-suited for creating the Point Transformer Localization dataset. A list of all classes that are provided by the SemanticKITTI dataset is available in Appendix B.1. Since only static objects are relevant, points that are assigned dynamic class labels in the segmentation phase are discarded, e.g., car, truck, motorcycle, or person. Moreover, lidar points that belong to the ground plane are discarded as well, as they do not provide useful information for the localization task. Consequently, the resulting classes kept for obtaining points for the measurement dataset are as follows: Building, vegetation, trunk, fence, pole, and

traffic sign. Finally, the points of each static class are clustered using the DBSCAN algorithm introduced in Section 5.2.1 [EKSX96] to assign each point to a unique object. This object of a specific static class is considered one landmark with semantic and geometric information in the Point Transformer Localization dataset. Then, the preprocessed scans are saved, where each point is comprised of the 3D position, a class label, and a unique object id. Thus, a measurement sample from the Point Transformer Localization dataset is given as

$$\mathcal{M} = \{\boldsymbol{m}_1, \ldots, \boldsymbol{m}_\pi\}, \boldsymbol{m} \in \mathbb{R}^5. \tag{5.5}$$

Similar to the DeepLocalization dataset, each measurement sample is accompanied by the high-precision pose from the dGPS system for evaluation purposes.

5.3.2 Mapping

The map creation process requires semantic labels for the measurements as input, similar to the previous section. Points classified as static and persistent are used to build the high-density lidar map. Then, the scans from one run on the test track are accumulated and transformed into global coordinates (UTM) using the pose information from the dGPS system, which provides centimeter-level accuracy. The result is a high-density map with millions of points per class. However, since the perceived objects overlap between subsequent scans, multiple points may be duplicates, increasing the overall memory requirement without providing exploitable information for the localization task. One possible solution is to aggregate lidar points that are in the vicinity of each other. A voxel downsampling approach is employed for the dataset that discretizes the space into cubes with edge length l_v. Then, all points that are within one voxel are combined. The algorithm creates exactly one new point per occupied voxel, where the new position is averaged over all points inside the voxel. Therefore, the size of the point cloud can be greatly reduced. Consequently, the resulting resolution of the high-density map can be adapted by changing l_v; however, at some point, the voxelization can cause a loss of geometric information. Thus, l_v is set close to the resolution of the lidar scan to preserve the level of detail, e.g., $l_v = 0.06$. At this point, the map consists of semantically labeled point clouds, that are divided according to the class label. Nevertheless, it is still possible that outliers from faulty measurements exist. In order to counteract this issue, the next processing step involves a radius outlier removal algorithm that is applied to every class separately. The algorithm removes all points with less than n_b points within a given sphere of radius r_b, where n_b and r_b are adapted to fit each class individually. After downsampling the map point cloud and removing outliers, the creation of the digital map is complete. At this point, the high-density digital

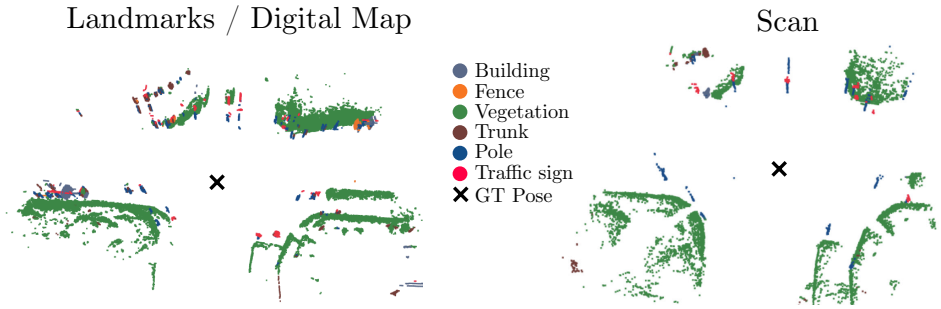


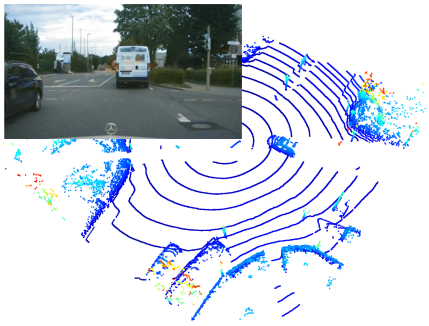
Figure 5.5: Overview of one input sample that contains the scan and landmark point clouds with semantic class information. Here, all dynamic objects are discarded to only include static and persistent objects.

map is represented by

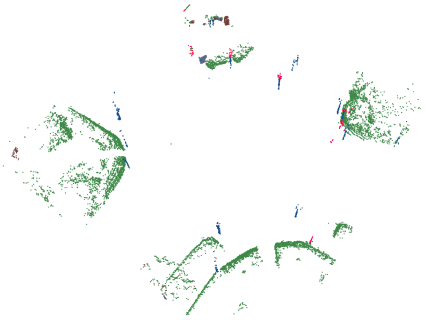
$$\mathcal{L} = \{\boldsymbol{l}_1, \ldots, \boldsymbol{l}_\nu\}, \boldsymbol{l} \in \mathbb{R}^4, \tag{5.6}$$

where each point contains the 3D position as well as the corresponding static class label. In comparison with the DeepLocalization dataset, landmarks in the Point Transformer Localization dataset also represent static objects in the environment, such as buildings, fences, poles, or traffic signs. However, contrary to the previous dataset, a landmark is not defined by 2D coordinates. Here, the static environment is represented by a dense 3D lidar point cloud with class labels for each point, that were obtained from a deep semantic segmentation approach, i.e., RandLA-Net. Consequently, the digital map point cloud is a high-definition discrete replica of the static environment, that contains geometric, shape, and semantic information about each static object. Similar to the DeepLocalization dataset, one exemplary data sample for one time step t is shown in Figure 5.5 after the preprocessing pipeline, which includes the semantic segmentation and the extraction of static and persistent object classes.

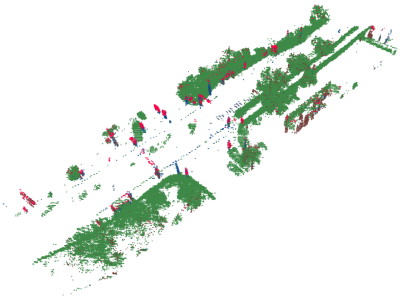
The entire measurement and map creation process is also outlined in Figure 5.6. To summarize: The first step extracts all lidar scans for the desired map location, whereas the field of view is adapted to include only static objects in the proximity of the vehicle. Then, the semantic segmentation network, RandLA-Net, is applied to each scan to extract static objects and discard ground and dynamic points. Next, the individual scans are aggregated by transforming the point clouds from sensor coordinates to the global map frame using the high-precision dGPS pose. In order to reduce the overall size of the map point cloud, several downsampling and outlier removal algorithms are applied to create the final high-density lidar map. Further information about the dataset is provided in Appendix B.1.



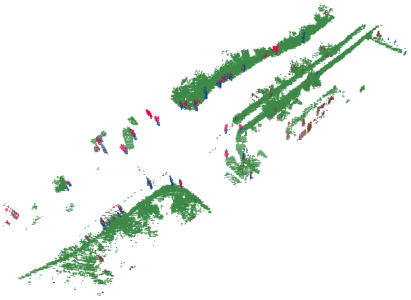
1) Extraction of Lidar Scans.



2) Semantic Segmentation using RandLA Net. Dynamic objects (e.g. cars) and ground points are discarded.



3) Aggregation of consecutive scans that are registered using the high-precision dGPS measurements.



4) Downsampling using a voxel downsample approach and outlier removal.



5) Finished digital high-density lidar map with semantic information.

Figure 5.6: Overview of the high-density lidar map creation process. For the measurement dataset, steps 1) and 2) are applied to the raw lidar measurements to discard dynamic objects.

Chapter 6

Deep Learning for Vehicle Self-Localization

This section introduces the main contributions of this thesis, which are two novel deep learning approaches to tackle the problem of vehicle self-localization in the field of automated driving in highly dynamic and unstructured environments. More precisely, DeepLocalization and two network extensions are proposed that can be used for sparse input point clouds to register measurements and landmarks and infer an accurate pose. On the other hand, Point Transformer Localization, which is based on the Point Transformer architecture introduced in Chapter 4, can be deployed for dense point clouds obtained from lidar scans. As mentioned in the previous chapters, only landmark-based localization methods are considered, where the inputs are unordered point sets with arbitrary cardinality. In the following, a brief motivation is given why solving the localization problem using deep learning approaches promises to achieve increased accuracy. Then, the fundamental problem formulation and the requirements are stated. Finally, the methodology and network architectures for both approaches are presented, and different inference pipelines are outlined.

6.1 Motivation

The task of vehicle self-localization is crucial for the safe and reliable operation of automated agents, including self-driving cars [KNW+15; Thr02]. For that, multi-modal sensor measurements are usually compared with a-priori landmark-based feature maps, where relevant environment characteristics are stored with high-precision accuracy to ultimately infer the vehicle's pose, i.e., the position and the orientation from Equation (2.36). In most cases, the vehicle's pose is estimated relative to a global cartesian coordinate system, e.g., the Universal Transverse Mercator (UTM).

When the ego position is known, information from the digital map, like road geometries and semantic and scene information, can be used to enrich and support the environment perception of automated vehicles, especially in highly dynamic and unstructured scenarios [GDD18; GPBD20]. Therefore, vehicle self-localization can be characterized as one of the backbones of the automated driving processing chain that many subsequent modules rely on, like human-vehicle interaction [WBKB20], trajectory planning and prediction [SBM+20] and object tracking [EHHD18]. For these applications, the required accuracy of the position lies in the range of about 30 cm to 50 cm [LMT07] and for the orientation around 5° [EHH+19]. A widely known approach for obtaining pose estimates is the deployment of global navigation satellite systems (GNSS), for example, the global positioning system (GPS), Galileo, or BeiDou. Usually, the position is estimated for these localization methods by calculating runtime differences obtained from multiple satellites. For that, the receiver needs a clear line of sight to at least four satellites to receive positioning data, from which it is possible to calculate the ego pose on the surface of the earth. A more in-depth introduction to the principles of global navigation satellite systems can be found in [TM17]. Here, it is important to note that in most cases, the achieved accuracy for these systems is not sufficient for the requirements of automated driving. In the worst-case scenario, it may be impossible to obtain a pose estimate when not enough satellites are visible. Since this thesis deals with solving the task of self-localization in urban scenarios with a high amount of dynamic objects, buildings, and tunnels that can occlude the line of sight to the satellites, this problem can severely impact safety and reliability and render the vehicle inoperable. Moreover, additional effects and scenarios can significantly reduce the GNSS accuracy to over 10 m [WEK05], for example, due to atmospheric disturbances or multi-path propagations from narrow passages between tall buildings, which are also called street canyons. One possible commercially available solution to increase the GPS accuracy, is to equip the vehicle with a differential GPS (dGPS) system [SYF02]. Here, correction data, which is collected from fixed ground-based reference stations calibrated with high-precision, is used to counteract atmosphere disturbances and other effects, providing centimeter-level accuracy. Additionally, this method can be combined with inertial measurement units (IMU) to further improve the pose estimate. However, the acquisition of these systems, together with calibration and maintenance, can cost up to 100.000 €, rendering this approach unsustainable for mass production. Moreover, the usage of the correction data is commonly billed per minute, therefore further increasing operational costs.

The prevalent solution is to employ traditional filter and heuristic-based algorithms that utilize built-in sensors of the vehicle, e.g., camera, radar or lidar, in combination with the aforementioned digital maps to infer a position and orientation inside the map coordinate system, also called the map frame. The creation of the maps is one of the fundamental problems of robotics [Thr02]. In general, there are two types of how maps are usually created: offline and online based methods. The

online based approach is most commonly known as simultaneous localization and mapping (SLAM) and involves constructing the digital map while localizing in an unknown environment. Since a thorough discussion about SLAM is out of scope for this thesis, a comprehensive survey on the topic can be found in [APSL08; BAYG17]. Nevertheless, these algorithms often suffer from error propagation and require loop closures to simultaneously optimize the pose and the digital map [BAC15]. In urban scenarios, this error propagation can be particularly severe due to the high amount of dynamic objects, e.g., cars or pedestrians, that interfere with the map building process. Therefore, it is still common to rely on a-priori digital maps, i.e., maps built offline, to solve the vehicle localization problem. As mentioned in Chapter 5, the digital maps are created using high-precision reference sensors, for example, dGPS systems that use correction data to improve the standard GPS measurements and achieve centimeter-level map accuracy. Consequently, localization approaches are categorized and distinguished by the type of map representation; however, the goal is always the same: collect landmarks, i.e., environmental information from easily **recognizable** and **static** objects, such as vegetation, traffic lights, traffic signs or buildings. Then, the map can be represented using geometric information [SWD15; WDN+14], e.g., depth, points, voxels or semantic and instance information [CWL+20]. During the localization phase, the perceived environment is then matched with the digital map to infer the ego position inside the map frame; in other words, suitable associations between landmarks and measurements are searched. For this, traditional methods usually rely on filter-based algorithms, such as the Extended Kalman-Filter (EKF) [CHM12; TSK11] or variations of the particle filter [FTBD01; Stü18; TFBD01]. These methods usually model the measurement-to-landmark associations probabilistically, which either requires a one-to-one or a one-to-many mapping between measurements and landmarks from the digital map [TBF98]. Based on this, the landmark-based vehicle self-localization process is divided into three subtasks for the remainder of this thesis:

1. **Matching:** The task of finding the best and most probable measurement-to-landmark associations.

2. **Registration:** Finding the optimal transformation that minimizes the distance of the landmarks to the matched measurements based on some error function.

3. **Pose Estimation:** Deriving the vehicle's pose based on the transformation from the registration process.

Despite the success of traditional filter-based localization methods in the field of robotics in the past decades, many methods still struggle to distinguish noise from static landmarks, especially in dynamic and unstructured environments [CWL+20]. Additionally, each of the three tasks usually requires a different heuristic or fine-grained algorithm. For example, for the matching process, a kNN search can be

deployed, and the point cloud registration can be obtained from, e.g., the iterative closest point (ICP) algorithm. Here, it becomes apparent that each subtask and algorithm may require hyperparameters tuning, which also depends on different scenarios and sensor configurations.

However, recent advances in the field of machine learning and deep learning have enabled neural networks, which are naturally more resistant to noisy input data, to solve many tasks in the field of automated vehicles and achieve state-of-the-art results, including localization [EBD21a; EHH+19; LZW+19], odometry [LCW+19], point cloud registration [HEB+20] or object tracking [EHHD18]. Moreover, impressive progress has been made in point processing neural networks, such as PointNet [QSMG17], PointNet++ [QYSG17] or KPConv [TQD+19], as mentioned in Section 3.1. Nevertheless, deploying these point processing networks for the task of vehicle self-localization has not been fully addressed in the literature, mainly due to the complex requirements regarding the input data and large memory and computational demands of high-density point clouds, e.g., from lidar sensors. Following these observations, several methods are presented in this thesis that build upon the innovations in the field of deep neural networks and 3D computer vision and apply the advances for the task of landmark-based localization. Moreover, this thesis explores how point-processing neural network architectures can be adapted to deal with different input point densities.

In subsequent sections, two novel architectures are presented that use 2D point features as input due to their manageable memory requirements compared to voxel-based representations and easy maintenance capabilities [BWWN19]. Moreover, a 3D point-based approach is introduced that utilizes semantic information from high-density lidar sensors. Consequently, the main contributions of this theses can be summarized as follows:

1. **DeepLocalization**: A landmark-based localization method that relies on sparse measurement and landmark point clouds as input. By using novel permutation-invariant network architectures, the approach accurately predicts the vehicle's pose in global coordinates using either an initial GPS estimate or a previous pose.

2. **Point Transformer Localization**: A high-density 3D point cloud localization approach, based on the proposed Point Transformer architecture that matches lidar scans with semantic labels to a digital map to infer the vehicle's pose. For that, the popular attention mechanism is employed to perform the point cloud registration and matching tasks.

6.2 Problem Formulation

This section serves as an overview of the requirements of both deep learning-based vehicle self-localization approaches, namely DeepLocalization and Point Transformer Localization. As mentioned before, the difference between the two methods is the density of the input point cloud. This results in noticeable differences in memory and computational requirements that need to be taken into account for the network architecture design. Nevertheless, the input to both methods is jointly described by the two input sets

$$\mathcal{M}_t = \{\boldsymbol{m}_1, \boldsymbol{m}_2, \ldots, \boldsymbol{m}_\pi\},\ \boldsymbol{m} \in \mathbb{R}^{D_m}, \tag{6.1a}$$

$$\mathcal{L} = \{\boldsymbol{l}_1, \boldsymbol{l}_2, \ldots, \boldsymbol{l}_\nu\},\ \boldsymbol{l} \in \mathbb{R}^{D_l}, \tag{6.1b}$$

which follow the notation introduced in Section 2.3, i.e., $\mathcal{M}_t$ denotes the set of all measurements with cardinality π at time step t and, equally, $\mathcal{L}$ denotes the set of all map landmarks with cardinality ν. Hereby, the dimension of the measurements and landmarks, D_m and D_l respectively, depend on the self-localization approach and will be specified later on.

In the last section, another distinction was introduced. DeepLocalization processes low-density input point clouds, while Point Transformer Localization uses high-density point clouds. The difference between low-density and high-density is the order of magnitude of the input set cardinality; in other words

$$\nu^{\mathrm{PTLoc}} \gg \nu^{\mathrm{DeepLoc}}, \tag{6.2a}$$

$$\pi^{\mathrm{PTLoc}} \gg \pi^{\mathrm{DeepLoc}}. \tag{6.2b}$$

More precisely, the number of processed points for DeepLocalization, ν^{DeepLoc} and π^{DeepLoc}, lies in the range of 20 to 100, while the number of points processed by Point Transformer Localization, ν^{PTLoc} and π^{PTLoc} is in the range of 20 000 to 40 000 points per time step. Depending on the number of input points, the overall network structure has to be adapted to handle the memory and computational requirements of large point sets.

Finally, the goal of both methods follows the definition of Equation (2.44), where the symmetric pose set function

$$f_{\mathrm{loc}}(\mathcal{M}_t, \mathcal{L}) \rightarrowtail \hat{\boldsymbol{p}}, \tag{6.3}$$

maps the measurement set $\mathcal{M}_t$ and landmark set $\mathcal{L}$ to the vehicle's pose $\hat{\boldsymbol{p}}$ inside the map frame. In this thesis, the localization function f_{loc} from Equation (6.3) is approximated using a deep neural network $h^{\boldsymbol{\theta}}_{\mathrm{loc}}$ following the universal approximation

theorem from Equation (2.46) and (2.47), which is given as follows:

$$h_{\text{loc}}^{\boldsymbol{\theta}}(\mathcal{M}_t, \mathcal{L}) \approx f_{\text{loc}}(\mathcal{M}, \mathcal{L}), \tag{6.4}$$

with $h_{\text{loc}}^{\boldsymbol{\theta}}(\mathcal{M}_t, \mathcal{L}) \rightarrowtail \hat{\boldsymbol{p}}$ and $\hat{\boldsymbol{p}} \in \mathbb{R}^3$. Here, $\boldsymbol{\theta}$ denotes the set of learnable parameters, i.e., the weights of the neural network. Depending on the network architecture, $h_{\text{loc}}^{\boldsymbol{\theta}}(\cdot)$ approximates one or more subtasks that were introduced in the previous section, in particular matching, registration, and pose estimation.

6.3 DeepLocalization

This section introduces the DeepLocalization method [EBD21a; EHH+19], which builds upon recent advances made by Qi et al. [QSMG17; QYSG17] in the field of 2D and 3D point processing networks. The main novelties of this contribution are threefold:

1. Formulation of the general localization problem in terms of deep neural networks.

2. Processing of two self-contained point clouds, i.e., the measurements and landmarks, with one end-to-end trainable neural network that is permutation-invariant and can independently handle arbitrary cardinality of both input sets.

3. Proposition of a novel training framework, which can, theoretically, generate an infinite amount of training samples without the need for manual labeling of the data points.

Besides the standard DeepLocalization network configuration introduced in [EHH+19], this section also presents two additional improvements that greatly benefit the overall localization accuracy. This is done by proposing network architectures and data processing techniques that better adapt to the task of vehicle self-localization and the three subtasks defined in Section 6.1.

6.3.1 Methodology

The input to DeepLocalization are preprocessed multi-modal measurements $\mathcal{M}_t$ and landmarks from the digital map $\mathcal{L}$. The sensor measurements are obtained from lidar,

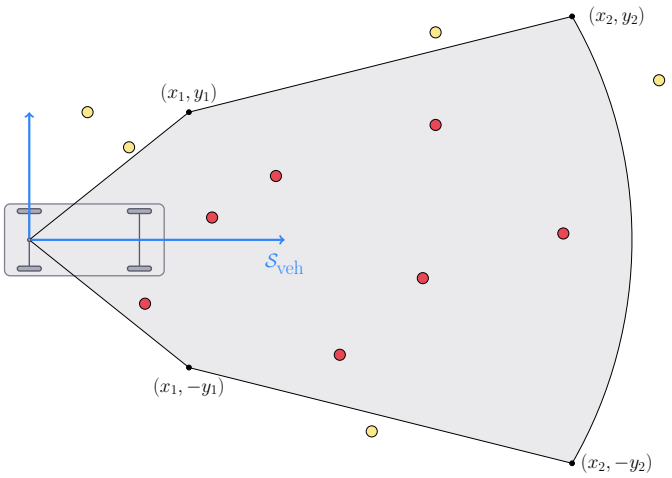


Figure 6.1: Illustration of the vehicle's field of view (FoV). Red points are inside the FoV while yellow points are discarded.

radar and camera sensors, where each measurement $\boldsymbol{m}$ represents one perceived object from the environment. Similarly, the landmarks from the digital map $\mathcal{L}$ are static and easily recognizable objects, as introduced in Section 5.2.2. The landmarks are comprised of preprocessed measurements from all three sensor modalities (DBSCAN for radar and lidar, and MSER for camera landmarks), whereas each landmark $\boldsymbol{l}$ denotes a generic 2D representation of one object, such as a traffic light, traffic sign, pole, or tree. However, no semantic information is used for this approach. Both the measurements and landmarks are specified using a x and a y coordinate that denotes the position in the vehicle's coordinate system (measurements) and the UTM coordinate system (landmarks), respectively. Thus, the dimension of the input points is given by $D_m = D_l = 2$. Moreover, only a subset of the original map point set is used; in particular, only landmarks that are in the vehicle's field of view (FoV) are selected because the vehicle can not detect and match landmarks outside the range of the sensors; thus

$$\mathcal{L}^{\text{FoV}} \subseteq \mathcal{L}. \tag{6.5}$$

Here, the field of view is selected heuristically, as shown in Figure 6.1, with bounding box vertices $x_1 = 10\,\text{m}$, $x_2 = 80\,\text{m}$, $y_1 = 5\,\text{m}$, $y_2 = 15\,\text{m}$. The field of view was found by approximating a convex hull over the entire measurement dataset that encloses about 90 % of the points to discard any outliers. Here, it is apparent that the cardinality of the landmark point set changes, i.e., $\nu^{\text{FoV}} \leq \nu$. Moreover, it is implied that ν^{FoV} is associated with the current time step t. However, for the sake of

simplicity, the superscript and the time index are omitted unless specified otherwise.

Considering the goal of aligning the measurements to the landmarks from the digital map, see Equation (6.4), regressing the pose directly in global coordinates, e.g., the UTM coordinates system, may induce numerical stability issues [EHH+19]. Additionally, requiring the output to cover a wide range of possible values decreases the generalization capabilities of the network. This would force the network to learn coordinates that are only in the valid range of the training dataset, which makes a general deployment to unseen areas impossible. To solve this problem, the self-localization task is reformulated as local self-localization introduced in Section 2.3. Therefore, instead of regressing the pose in a global coordinate system, a noisy and inaccurate pose is assumed as the initial starting pose $\boldsymbol{p}_{\text{init}}$, which can either be obtained from a previous time step or a commercially available GPS sensor. Here, it is important to note that the initial pose deviates from the ground-truth pose $\boldsymbol{p}_{\text{GT}}$, whereas the difference or the pose offset $\Delta\boldsymbol{p}_{\text{init}}$ can be obtained as follows:

$$\Delta\boldsymbol{p}_{\text{init}} = \boldsymbol{p}_{\text{GT}} - \boldsymbol{p}_{\text{init}}. \tag{6.6}$$

Thus, when $\boldsymbol{p}_{\text{init}}$ is known, the pose estimate from the localization network from Equation (6.4) can be expressed as

$$h^{\boldsymbol{\theta}}_{\text{loc}}(\mathcal{M}_t, \mathcal{L}^{\text{FoV}}) \rightarrowtail \Delta\hat{\boldsymbol{p}} = [\Delta\hat{x},\, \Delta\hat{y},\, \Delta\hat{\varphi}]^T, \tag{6.7}$$

where the goal is to approximate $\Delta\boldsymbol{p}_{\text{init}}$ instead of the global pose $\boldsymbol{p}_{\text{GT}}$. Following this approach, the pose estimate in global coordinates $\hat{\boldsymbol{p}}$ can easily be reconstructed by

$$\hat{\boldsymbol{p}} = \boldsymbol{p}_{\text{init}} + \Delta\hat{\boldsymbol{p}}. \tag{6.8}$$

The idea is that during the inference process, only the inaccurate initial GPS pose $\boldsymbol{p}_{\text{init}}$ is known and required. Then, using the field of view defined in Equation (6.5), all landmarks in the vicinity of the vehicle are loaded based on $\boldsymbol{p}_{\text{init}}$. Each landmark is transformed from the global coordinate system $\mathcal{S}_{\text{UTM}}$ into the vehicle coordinate system $\mathcal{S}_{\text{veh}}$, see Section 2.3, using the inverse homogenous transformation matrix H^{-1} (2.18) with

$$H = \begin{bmatrix} \cos(\varphi_{\text{init}}) & -\sin(\varphi_{\text{init}}) & x_{\text{init}} \\ \sin(\varphi_{\text{init}}) & \cos(\varphi_{\text{init}}) & y_{\text{init}} \\ 0 & 0 & 1 \end{bmatrix}, \tag{6.9}$$

where the entries are taken from the initial pose estimate $\boldsymbol{p}_{\text{init}}$. It is apparent that using an inaccurate pose for the transformation induces a shift and a rotation to the landmarks, that is equal to the difference $\Delta\boldsymbol{p}_{\text{init}}$ between the initial pose $\boldsymbol{p}_{\text{init}}$ and the ground-truth pose $\boldsymbol{p}_{\text{GT}}$, cf. Equation (6.6). At this point, both the landmarks

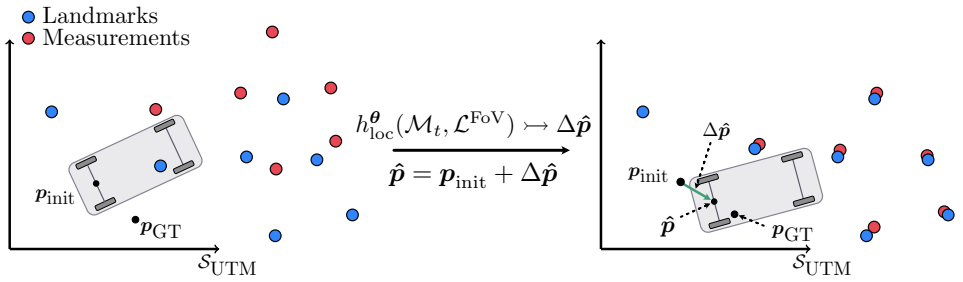


Figure 6.2: Formulation of the localization task in terms of DeepLocalization. The landmarks from the digital map are loaded with respect to an initial or previous inaccurate pose $\boldsymbol{p}_{\text{init}}$ that induces a shift and rotation between landmarks • and current sensor measurements • (left image). After applying the estimated pose offset $\Delta\hat{\boldsymbol{p}}$ from the network, the pose $\hat{\boldsymbol{p}}$ is corrected towards the ground-truth $\boldsymbol{p}_{\text{GT}}$ (right image).

and measurements are provided in the same local coordinate system and can be processed by the deep neural network. Subsequently, the network learns the spatial offset between landmarks and measurements, which can be translated to a global pose estimate as demonstrated in Equation (6.8). The process is also visualized in Figure 6.2, where the network output is applied to the initial pose estimate. Here, it becomes apparent that the measurements align with the landmarks from the digital map after applying the correction. Consequently, the aforementioned difficulties of learning a pose in global coordinate space can be circumvented by reformulating the problem as a local localization task, however, with the disadvantage of requiring an initial pose estimate at every time step. For this reason, an additional inference approach is presented in the following subsection that mitigates this restriction to require only one noisy and inaccurate GPS measurement at initialization time and then relies on the previous pose estimate for further operation. Next, the network architecture and the objective function, i.e., the loss, will be introduced in detail.

Network Architecture

DeepLocalization, which is illustrated in Figure 6.3, consists of two input branches: one feature generation module for the preprocessed multi-modal measurements $\mathcal{M}_t = \{\boldsymbol{m}_1, \boldsymbol{m}_2, \ldots, \boldsymbol{m}_\pi\}, \boldsymbol{m} \in \mathbb{R}^2$ and one feature generation module for the landmarks from the digital map $\mathcal{L}^{\text{FoV}} = \{\boldsymbol{l}_1, \boldsymbol{l}_2, \ldots, \boldsymbol{l}_\nu\}, \boldsymbol{l} \in \mathbb{R}^2$, that operate independently. The goal of the two branches is to produce a global feature signature, that describes the spatial characteristics of each input point cloud. For that, two

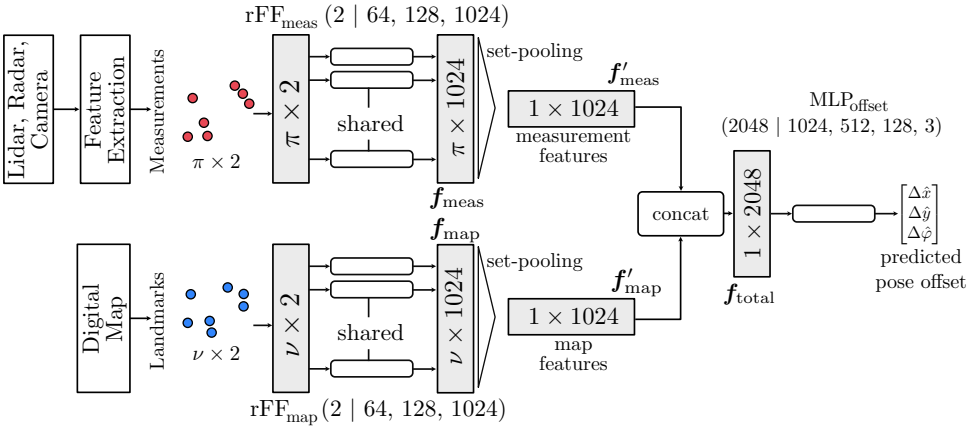


Figure 6.3: Overview of DeepLocalization network architecture. The measurement and landmark point sets are processed individually to obtain two latent feature vectors. The features are concatenated and fed to a final MLP to predict a pose offset. Based on [EHH+19].

separate rFFs are deployed to transform the input points to higher dimensional space $\mathbb{R}^{1024}$, in particular $\text{rFF}_{\text{meas}}(2\,|\,64, 128, 1024)$ and $\text{rFF}_{\text{map}}(2\,|\,64, 128, 1024)$, with $\text{rFF}_{\text{meas}} : \mathbb{R}^{\pi\times 2} \to \mathbb{R}^{\pi\times 1024}$ and $\text{rFF}_{\text{map}} : \mathbb{R}^{\nu\times 2} \to \mathbb{R}^{\nu\times 1024}$. Here, it is important to note that, although the weights of network are shared among the individual point sets as explained in Section 2.4.4, each rFF is optimized independently, i.e., rFF_{meas} and rFF_{map} have separate weights. After obtaining the latent features of the measurements $\boldsymbol{f}_{\text{meas}} \in \mathbb{R}^{\pi\times 1024}$ and the landmarks from the vehicle's field of view $\boldsymbol{f}_{\text{map}} \in \mathbb{R}^{\nu\times 1024}$, a max-pooling operation is applied feature-wise over the latent representations, extending the ideas of [QSMG17] to the case of two separate input point clouds. In other words, the set-pooling operation defined in Equation (2.69) is used to produce two independent feature vectors of fixed size that are permutation-invariant and do not depend on the number of input points

$$\boldsymbol{f}'_{\text{meas}} = \text{Setpool}(\boldsymbol{f}_{\text{meas}}), \tag{6.10a}$$

$$\boldsymbol{f}'_{\text{map}} = \text{Setpool}(\boldsymbol{f}_{\text{map}}), \tag{6.10b}$$

where Setpool is set to the max-pooling function (2.66) and $\boldsymbol{f}'_{\text{meas}}, \boldsymbol{f}'_{\text{map}} \in \mathbb{R}^{1024}$. At this point, each of the two latent feature representations of fixed size contains a global signature of the input points; that is, relations and dependencies among all points of each point cloud are considered. Since the goal is to find the offset from the initial pose, i.e., the offset between the measurement and the landmarks in the vehicle frame, see Equation (6.7), the global measurement features have to be correlated with the global map features. For this, the outputs of the set-pooling

operations are concatenated

$$\boldsymbol{f}_{\text{total}} = \boldsymbol{f}'_{\text{meas}} \oplus \boldsymbol{f}'_{\text{map}}, \tag{6.11}$$

where $\boldsymbol{f}_{\text{total}} \in \mathbb{R}^{2048}$ denotes the resulting feature representation of both input point clouds and $\oplus$ is the concatenation operation. Finally, an additional multilayer perceptron (MLP) is deployed to predict the offset of the input measurement point set and the landmarks from the vehicle's field of view: $\text{MLP}_{\text{offset}}(2048 \,|\, 1024, 512, 128, 3)$, with $\text{MLP}_{\text{offset}} : \mathbb{R}^{2048} \rightarrow [\Delta\hat{x}, \Delta\hat{y}, \Delta\hat{\varphi}]^T$. Moreover, the ReLU activation function (2.59) is deployed after each layer to induce non-linearities, with the exception of the last layer of $\text{MLP}_{\text{offset}}$ in order to obtain valid positive and negative outputs for the pose offset prediction.

Learning Objective

The learning objective is to minimize the localization error of the predicted pose offset $\Delta\hat{\boldsymbol{p}}$ and the ground-truth pose offset $\Delta\boldsymbol{p}$ [EHH+19]. For that, the position and the orientation offsets are treated independently. This is done because the difference in magnitude between the position error, which is given in meters, and the orientation error, usually specified in radians, can become quite large. Thus, the MSE Loss (2.72) is employed for the x and y components (L_{pos}) and the φ component (L_{ori}) as follows:

$$L_{\text{pos}}^{\text{MSE}} = \text{E}\left[(\Delta x - \Delta\hat{x})^2\right] + \text{E}\left[(\Delta y - \Delta\hat{y})^2\right], \tag{6.12a}$$

$$L_{\text{ori}}^{\text{MSE}} = \text{E}\left[(\Delta\varphi - \Delta\hat{\varphi})^2\right]. \tag{6.12b}$$

As mentioned earlier, finding a suitable weighting factor for L_{pos} and L_{ori} to obtain gradients of equal magnitude is a tedious task that also depends on the input data distribution. Therefore, the geometric loss function (2.74) introduced by Kendall et al. [KC17] is used to learn a balanced weighting of the individual loss parts from Equations (6.12), such that

$$L_{\text{total}} = L_{\text{pos}}^{\text{MSE}} \cdot \exp(-s_{\text{pos}}) + s_{\text{pos}} + L_{\text{ori}}^{\text{MSE}} \cdot \exp(-s_{\text{ori}}) + s_{\text{ori}}, \tag{6.13}$$

where s_{pos} and s_{ori} contain the task specific uncertainty, i.e., homoscedastic uncertainty, that does not depend on the input data. This can be interpreted as learnable task-dependent weighting [EHH+19; KC17]. Therefore, the manual tuning process for the weighting factors is circumvented. Finally, the parameters are updated during the training process, which will be introduced in the next section, using the popular Adam optimizer (2.77) proposed by Kingma et al. [KB15]. In other words, the uncertainties s_{pos} and s_{ori} are implemented as variable parameters of the network

that are included in the optimization process. Thus, the parameters can be freely adapted by the optimization algorithm to support the overall learning objective of minimizing the position and orientation error from Equation (6.12).

Training Process

One of the key contributions of DeepLocalization is the novel training process that allows for automatic label generation during training. Theoretically, it is possible to retrieve an infinite amount of unique training samples from a dataset. Therefore, the training process increases the generalization capabilities of the network while reducing the risk of overfitting the training data distribution.

The dataset introduced in Chapter 5 provides the ground-truth pose information of the ego-vehicle obtained from dGPS reference sensors. However, as explained in Section 6.3.1, DeepLocalization assumes a noisy and inaccurate initial starting position and orientation and infers a pose offset that corrects the initial estimate in the direction of the ground-truth. This approach can be leveraged to generate training samples automatically as follows:

1. First, a training sample is loaded from the dataset, which contains the current sensor measurements in the vehicle coordinate system $\mathcal{S}_{\text{veh}}$, the landmarks in the vehicle's FoV from the digital map in global coordinates (UTM), and the high-precision pose from the dGPS system $\boldsymbol{p}_{\text{GT}}$.

2. Then, in order to simulate the inaccurate pose as mentioned earlier, an artificial pose offset $\Delta\tilde{\boldsymbol{p}}$ is subtracted from the ground-truth pose $\boldsymbol{p}_{\text{GT}}$, such that

$$\tilde{\boldsymbol{p}}_{\text{init}} = \boldsymbol{p}_{\text{GT}} - \Delta\tilde{\boldsymbol{p}}, \tag{6.14}$$

   where $\tilde{\boldsymbol{p}}_{\text{init}}$ indicates that the initial pose estimate is obtained synthetically. Here, $\Delta\tilde{\boldsymbol{p}}$ can be generated using an arbitrary probability distribution that best describes the measurement noise of the commercially available GPS sensor, which can often be retrieved from the sensor data-sheet. In practice, drawing the artificial pose offset $\Delta\tilde{\boldsymbol{p}}$ from either a uniform distribution (2.26) or a normal distribution (2.28), e.g.,

$$\Delta\tilde{\boldsymbol{p}} \sim \mathcal{U}_{[a,b]} \text{ or} \tag{6.15a}$$
$$\Delta\tilde{\boldsymbol{p}} \sim \mathcal{N}(\mu, \sigma^2), \tag{6.15b}$$

   has proved to be a realistic modeling of the measurement uncertainty [EBD21a; EHHD18]. Here, the lower and upper bound of the uniform distribution, a

and b respectively, and the mean μ and variance σ^2 of the normal distribution depend on the sensor and can be adapted for different scenarios as well.

3. The generated pose offset $\Delta\tilde{\boldsymbol{p}}$ is applied to the ground-truth $\boldsymbol{p}_{\mathrm{GT}}$ according to Equation (6.14), to obtain the artificially generated initial pose $\tilde{\boldsymbol{p}}_{\mathrm{init}}$. It is then used for the inverse homogenous transformation matrix $\tilde{H}^{-1}$, that transforms the landmarks from the global coordinate system to the vehicle frame $\mathcal{S}_{\mathrm{veh}}$, where $\tilde{H}$ is given as follows:

$$\tilde{H} = \begin{bmatrix} \cos(\varphi_{\mathrm{GT}} - \tilde{\varphi}) & -\sin(\varphi_{\mathrm{GT}} - \tilde{\varphi}) & x_{\mathrm{GT}} - \tilde{x} \\ \sin(\varphi_{\mathrm{GT}} - \tilde{\varphi}) & \cos(\varphi_{\mathrm{GT}} - \tilde{\varphi}) & y_{\mathrm{GT}} - \tilde{y} \\ 0 & 0 & 1 \end{bmatrix}. \tag{6.16}$$

   As mentioned before, using an inaccurate pose induces a noticeable shift and rotation between the measurements and landmarks. The network is trained to detect and correct this offset.

4. Following Equations (6.6), (6.7) and (6.14), it is apparent that the label or target vector equals the artificial pose offset $\Delta\tilde{\boldsymbol{p}}$. In other words, the network learns to correct the synthetic shift and rotation that is applied to the landmarks. Therefore, the label can be generated on-the-fly while training the network. Moreover, since $\Delta\tilde{\boldsymbol{p}}$ is subject to a random probability distribution, e.g., Equation (6.15a) or Equation (6.15b), it is possible to obtain a different label every training step, thus minimizing the risk of overfitting and improving the generalization capabilities of the network.

5. Finally, data augmentation techniques can be applied to each input sample, that replicate naturally occurring disturbances like clutter measurements, missed detections, and measurement noise. This, combined with the generated labels, results in unique data samples every iteration, further validating the aforementioned claim of low overfitting risks. However, this comes with the drawback that convergence usually takes longer due to the input distribution spread that the network has to adapt to. With this training strategy, real-world conditions are simulated, where the environment and spatial distribution of the landmarks vary in different scenarios. This effect will be explained in subsequent sections.

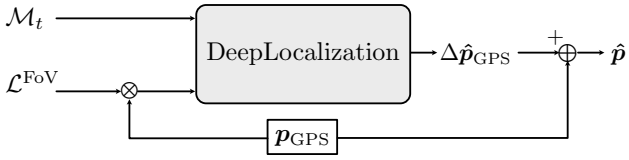
The presented data preparation method for the training samples allows for efficient deployment of the DeepLocalization approach: A relatively small dataset needs to be collected using high-precision reference sensors, e.g., a dGPS system, in addition to the preferred sensor setup, such as lidar, radar or camera. Then, using a mapping approach, a digital map is created offline from the sensor recordings, for example, using the algorithms introduced in Section 5.2.2. Again, manual labeling of the

training data is not required since it can be derived from the dGPS pose and the artificial offsets. Moreover, the workload to create a diverse dataset is reduced without increasing operational costs. Additionally, since the data is augmented thoroughly, the network can be deployed in various scenarios and geographic regions, even when the network has not seen the location before. Since generic landmarks are used, i.e., semantic information is abstracted into simple coordinate features, the risk of overfitting to local characteristics of landmarks is negligible. Consequently, the presented approach provides a low-cost, easy-to-use, and extendible framework, which is validated in extensive evaluations in the next chapter.

At this stage, one unique training sample is generated. However, in most cases, it is desirable to employ the mini-batch principle by applying the optimization and the backpropagation algorithm jointly over multiple data points to obtain smoother gradients. This reduces the risk of getting stuck in a non-optimal local minimum during optimization [GBC16, Chapter 8.1.3]. Therefore, several samples, also called the mini-batch, are fed to the network simultaneously, and the gradients are accumulated and averaged. Commonly, the number of samples in the mini-batch is denoted by B. Since the samples are stacked together, one requirement is that the number of points for each sample in the mini-batch must be equal. However, since the network processes point clouds, this can not be guaranteed. The sensors can detect an arbitrary number of landmarks at each time step. Additionally, the number of landmarks in the vehicle's field of view changes depending on the current position. In order to circumvent this restriction, one sample containing the multi-modal sensor measurement, the landmarks from the digital map, and the ground-truth pose for one time step is loaded from the dataset and is duplicated B times. Then, the automatic label generation mechanism using the artificial pose offset and all data augmentation techniques are applied to each duplicated sample individually, considering that the sum of deleted points from missed detections and added points from clutter is the same. This again creates B unique data samples with different pose offsets with the same dimension that can be processed by the network.

Here, it is important to note that due to the way the samples in the mini-batch are created, it is not possible to employ the popular batch normalization principle (2.86) [IS15]. The shift and rotation applied to the ground-truth pose are sampled from either a uniform or a normal distribution. Normalizing over the mini-batch would ultimately reverse the artificially generated noise since subtracting the mean and dividing by the variance of the shifted and rotated measurements or landmarks would move the points closer to the ground-truth. This effect correlates with the batch size B, i.e., more samples result in a statistically more accurate estimate of the mean and variance. Therefore, applying batch normalization provides the network with additional information that is not available during inference, where the batch size is $B = 1$. Consequently, no batch normalization layers are employed in the DeepLocalization architecture.

a) GPS-based Inference:



b) Filter-based Inference:

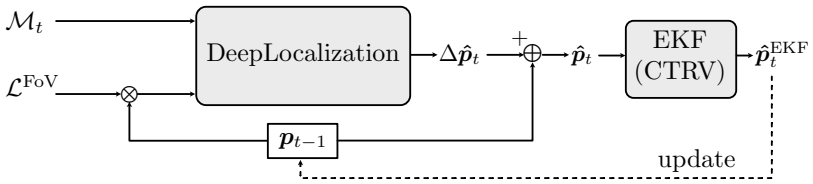


Figure 6.4: Overview of system architecture for the GPS-based and filter-based inference methods. [EBD21a] © 2021, IEEE.

Inference

In the following, two inference approaches are presented that allow for different modes of operation based on the availability of GPS signals. First, a pure GPS-based method is introduced, which is inspired by the training process. However, this inference approach relies on a GPS measurement for every time step, which can not be guaranteed, especially in urban scenarios. Therefore, a filter-based approach is proposed that requires only one initial pose estimate and employs an Extended Kalman-Filter (EKF) that uses the network's output to predict the vehicle's pose [EHH+19]. Both systems are visualized in Figure 6.4 and are explained in the following.

The **GPS-based inference** is aimed to improve a noisy measurement from commercially available GPS sensors that achieve a localization accuracy of about 1 m to 10 m, depending on environmental influences like atmosphere disturbances or multi-path effects. Nowadays, almost all vehicles are equipped with a GPS sensor for navigation purposes; therefore, inaccurate pose estimates can be obtained without additional cost. However, as mentioned in Section 2.3, the precision is not sufficient for the safe operation of the vehicle in urban scenarios, where the required accuracy lies in the range of about 20 cm to 50 cm [EHH+19]. Due to these specifications, it is necessary to improve the signal to enable its usage for automated driving applications. The GPS-based inference follows the same steps as the training process explained in Section 6.3.1: Starting from an inaccurate pose measurement

$\boldsymbol{p}_{\mathrm{GPS}}$ obtained from the GNSS system, landmarks in the vehicle's field of view are loaded from the digital map in global coordinates. Then, the GPS measurement is used to transform the landmarks into the vehicle coordinate system $\mathcal{S}_{\mathrm{veh}}$ using the homogenous transformation matrix H^{-1} from Equation (6.16), whereas the entries are taken directly from $\boldsymbol{p}_{\mathrm{GPS}}$, instead of the artificial pose estimate $\tilde{\boldsymbol{p}}_{\mathrm{init}}$. Since in most cases $\boldsymbol{p}_{\mathrm{GPS}} \neq \boldsymbol{p}_{\mathrm{GT}}$, this again induces a shift and a rotation compared to the ground-truth that is equal to the localization error

$$\Delta \boldsymbol{p}_{\mathrm{GPS}} = \boldsymbol{p}_{\mathrm{GT}} - \boldsymbol{p}_{\mathrm{GPS}} \tag{6.17}$$

of the GPS sensor. When the parameters of the label generation algorithm from Equation (6.15) are chosen similar to the accuracy specifications of the sensor, the trained network can directly be used to predict the correction vector that improves the localization accuracy, i.e., $\Delta\hat{\boldsymbol{p}}_{\mathrm{GPS}}$. To obtain a global pose prediction $\hat{\boldsymbol{p}}$, the output of the network $\Delta\hat{\boldsymbol{p}}_{\mathrm{GPS}}$ is applied to the initial measurement, such that $\hat{\boldsymbol{p}} = \boldsymbol{p}_{\mathrm{GPS}} + \Delta\hat{\boldsymbol{p}}_{\mathrm{GPS}}$. Thus, the GPS-based inference is conceptually the same as the training process, with the only difference that the initial pose is taken directly from the GPS sensor, instead of artificially generating a pose offset and can directly be used for deployment.

The **filter-based inference** introduces a concept that enables the overall system to only require one initial GPS pose estimate at startup, instead of relying on it for every time step. For that, an Extended Kalman-Filter [SSM62] is employed that receives the network's output as measurement input and uses an underlying constant turn rate and velocity (CTRV) motion model to predict the vehicle's pose [Thr02, Chapter 5], [Stü18]. The EKF considers the state space $\boldsymbol{x} \in \mathbb{R}^d$, which for the task of localization contains the vehicle's pose, i.e., $\boldsymbol{x} = [x^p,\, y^p,\, v,\, \varphi,\, \dot{\varphi}]^T$, where x^p, y^p are the position, v denotes the velocity, φ the orientation and $\dot{\varphi}$ the angular velocity. The filter algorithm is comprised of two parts, first a prediction step followed by an update or innovation step. The prediction step is performed according to the CTRV motion model. Finally, after obtaining a new observation $\boldsymbol{z}$, which is equal to the network's output, the predicted pose is updated to consider the current observation with regard to the motion model. A complete overview and derivation of the EKF algorithm is given in Appendix B.2. Here, it becomes apparent, that only one GPS measurement is required to initialize the Extended Kalman-Filter, as shown in Figure 6.4. Using the GPS pose $\boldsymbol{p}_{t-1}$, the landmarks are loaded and transformed to the vehicle coordinate system as explained in the last paragraph for the GPS-based inference. The DeepLocalization network infers the pose correction vector $\Delta\hat{\boldsymbol{p}}_t$ that is then applied to the initial or previous pose to obtain a global pose estimate $\hat{\boldsymbol{p}}_t$. The EKF predicts the pose according to the motion model and uses the global pose estimate as measurement input $\hat{\boldsymbol{p}}_t$ in the innovation step to update the overall pose prediction. Here, the EKF produces a smoothed pose according to the motion model, which is a supplementary benefit of the filter-based inference method. The output of

the Extended Kalman-Filter $\hat{\boldsymbol{p}}_t^{\mathrm{EKF}}$ is then used as the initial pose estimate for the next time step. An additional advantage of deploying the EKF is that additional pose measurements, for example, from commercially available GPS sensors or inertial measurement units (IMU), can easily be fused, increasing the overall localization accuracy without altering the system architecture.

Discussion

Revisiting the steps of the localization task from Section 6.1, the matching and the registration subtasks in the DeepLocalization architecture are performed on a global scale, which means that the network tries to align both input point clouds in one step. Furthermore, the pose estimation is combined with the matching and registration task in $\mathrm{MLP}_{\mathrm{offset}}$, which receives the latent feature representation of the input point sets. Therefore, no specific local relations are considered to predict the pose offset. However, as mentioned before, in the optimal case, i.e., not considering clutter measurement and missed detections, a sensed object in the environment is likely to originate from a landmark in the digital map. Thus, it is preferable to consider local dependencies between landmarks and measurements. Moreover, combining all three subtasks into only one multilayer perceptron increases the complexity of the learned tasks, thus negatively impacting the performance of the network. In order to combine these observations, two modified versions of the DeepLocalization network architecture are presented in subsequent sections that divide the localization subtasks such that a separate network module is responsible for the matching, registration, and pose prediction. Additionally, methods that consider the dependency between landmarks and measurements during the matching task are introduced.

6.3.2 DeepLocalization2

The second iteration of DeepLocalization extends the network architecture to tackle the aforementioned difficulties by considering the local dependencies between measurements and landmarks during the matching process. For that, the localization task is divided into the proposed three subtasks, as shown in Figure 6.5, such that an individual neural network can solve each task. The network is based on the work of [LQG19; QYSG17] and incorporates the ideas into the DeepLocalization system. The overall goal for the matching process is to take each measurement and find landmarks in the vicinity and associate the most likely landmarks from which the measurement originated. This concept stems from the fact that measurements are created from landmarks in the ideal case. Assuming the case that only static landmarks are perceivable, the sensors do not produce faulty measurements, and

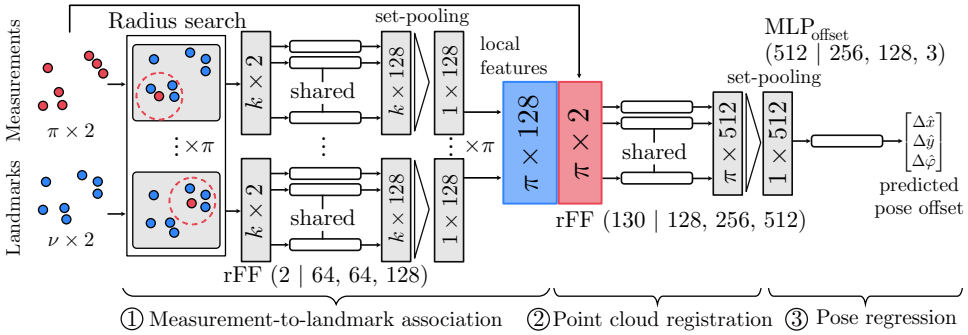


Figure 6.5: DeepLocalization2 network architecture. Instead of generating latent features for each input separately, a measurement-to-landmark association process matches nearby landmarks to each measurement, which are then processed by row-wise feedforward networks that produce a feature vector for each landmark. Then, the features are concatenated and a subsequent rFF generates a global signature of the input. Finally, the pose correction vector is estimated.

no dynamic objects are present in the current scene, each landmark should produce exactly one measurement. However, due to the dynamic nature of urban and rural scenarios, sensor noise, clutter measurements, and missed detection occur frequently, as explained in Section 2.3. Thus, assigning a measurement to a single landmark is not a trivial task, and sometimes no optimal association exists. Therefore, the network has to cope with finding the most probable associations and simultaneously consider noise in the form of clutter measurements and missed detections. Additionally, the pose inference becomes more difficult since the sensors are subject to measurement noise, and the landmarks are transformed into the vehicle frame suboptimally using an inaccurate initial estimate. Hence, the idea of this extension is to employ separate networks in the pipeline, where each module can focus on solving one of the localization subtasks.

To summarize, the localization steps involve finding suitable associations between measurements and landmarks. Then, using the result, the point clouds are registered to find the optimal transformation that minimizes the distance between the associations to infer the vehicle's pose. For the first task, a radius search algorithm is employed, as shown in Figure 6.5. Instead of learning a global latent feature representation as proposed in the DeepLocalization architecture, the goal is to find spatial relations among the measurements and landmarks, similar to traditional localization methods. However, rather than relying on one or more hand-crafted associations, the network learns the most probable assignments by finding all landmarks in the vicinity

of one measurement. The Euclidean distance to every landmark is calculated for each measurement, and only landmarks within a predefined radius r are considered. Then, the selected landmarks are processed by a row-wise feedforward network (rFF), which is a multilayer perceptron (MLP) with shared weights, that projects the spatial distances into latent feature space $\mathbb{R}^{\nu_r \times 128}$. Here, ν_r represents the variable number of associated landmarks. If no landmark is found in the neighborhood, the feature vector is set to zero. A subsequent max-pooling operation produces a local feature vector of fixed length for every measurement that contains all landmarks in the vicinity. This way, the network is able to learn the most probable associations. Finally, the feature vectors are concatenated to the corresponding measurement. Note that the rFF shown in the association process in Figure 6.5 is the same for every radius search. The measurement-to-landmark association produces local features of all landmarks in the vicinity of each measurement. After concatenating the local features $\mathbb{R}^{\pi \times 128}$ with the input points $\mathbb{R}^{\pi \times 2}$, the local feature matrix $\mathbb{R}^{\pi \times 130}$ is used to learn a global latent representation. This operation abstractly resembles the point cloud registration process since the entirety of both point clouds is considered in this step. Again, a row-wise feedforward network is employed to transform the local features into higher dimensional space $\mathbb{R}^{\pi \times 512}$. Then, a max-pooling operation generates a feature vector of fixed size that now contains global information about the measurements and landmarks. Finally, the network estimates the vehicle's pose using the aggregated local and global features. Since the global representation is now both invariant to input point permutations and unknown set cardinality, the multilayer perceptron $\text{MLP}_{\text{offset}}$ is used to regress a pose correction vector $\Delta\hat{\boldsymbol{p}}$.

During the measurement-to-landmark association process, a predefined radius r is assumed for the radius search algorithm. However, finding a universal radius for the neighborhood search that only assigns plausible landmarks and avoids erroneous associations is a tedious task that depends on the input data distribution. Additionally, if the selected radius is too small, it is possible that measurements far away from the vehicle are not associated with any landmark, as small pose inaccuracies have a more significant impact on the position offset of distant landmarks and vice versa. Therefore, a distance-dependant radius search algorithm is proposed that considers the uncertainties of the initial GPS measurement or the previous pose estimate, which is illustrated in Figure 6.6. The goal is to find the upper and lower error bounds caused by an inaccurate estimate. Assuming a GPS uncertainty for the orientation of α_{GPS}, the distance-dependent radius r_d can be calculated as follows:

$$r_d = \sin(\alpha_{\text{GPS}}) \cdot \sqrt{m_x^2 + m_y^2}, \tag{6.18}$$

where the position of the measurement in the vehicle coordinate frame is denoted by (m_x, m_y). Additionally, it is possible to integrate the position uncertainty into Equation (6.18) to obtain a more comprehensive radius calculation.

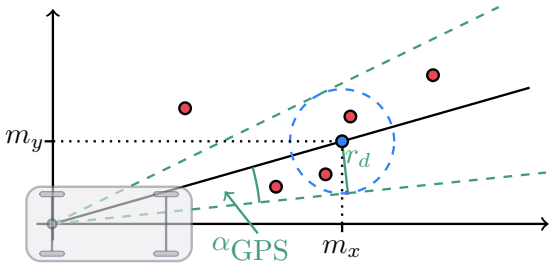


Figure 6.6: Overview of the radius search algorithm that adapts the radius for the ball query search according to the distance of the current measurement (•). The parameter α_{GPS} acts as an upper and lower bound for the calculation of the radius r_d.

Discussion

At this point, the network architecture is more adapted to the specific localization tasks and considers the characteristics of traditional landmark-based self-localization methods, as explained earlier. Instead of relying on individual global feature signatures of both input point sets, DeepLocalization2 now associates nearby landmarks to each measurement since a measurement is likely to originate from a static object, i.e., a landmark. In case of clutter measurements and missed detections that are most prevalent in highly dynamic scenarios, the network has to identify and separate noise from real measurements. The previously introduced radius search algorithm discards any landmarks outside the predefined radius, reducing the number of false associations since the network has to consider fewer possibilities. As mentioned in the previous section, the distance-dependant radius is used to incorporate the fact that orientation offsets induced by the inaccurate initial pose estimate have a more severe impact on distant measurements and landmarks. Consequently, choosing a radius that is too small would cause the radius search to find no suitable landmarks in the vicinity of a measurement. On the other hand, defining a radius that is too big may induce a loss of performance. This effect occurs since it would neglect the idea of pre-selecting appropriate landmarks in the vicinity, i.e., the local feature vector of the measurement would contain information about landmarks that are unlikely to be the origin. Moreover, since the local feature generation is based on the Euclidean distance of the specific landmark to the current measurement, the proximity defines the weight of each association. However, being the closest landmark is not always a guarantee for the correct choice of assignment, as clutter measurements, measurement noise, or inaccurate initial pose estimates can cause erroneous associations.

Due to the stated reasons, it is desirable to circumvent the manual selection procedure of the radius search by incorporating these steps into the learning and optimization process. Therefore, in the next section, a final improvement of the DeepLocalization network architecture is proposed that leverages the advantages of the attention mechanism, see Section 4.2.1, to learn a data-driven weighting of possible measurement-to-landmark associations.

6.3.3 Attention-based DeepLocalization

This section introduces the attention-based vehicle self-localization method [EBD21a], which is a network extension of the original DeepLocalization approach. The goal is to introduce the attention mechanism [VSP+17] for the localization task, as attention and the Transformer architecture have shown impressive results in the field of natural language processing (NLP), computer vision, e.g., vision transformer [DBK+21], and point cloud processing [EBD21b; LLK+19]. The changes compared to the DeepLocalization and DeepLocalization2 architecture and the necessary fundamentals of the proposed algorithms are highlighted in the following.

Revisiting the advances and insights from previous methods, DeepLocalization uses both input point clouds to produce global features in latent space using two separate row-wise feedforward networks (rFF). In the last section, the idea of associating nearby landmarks to each measurement to consider local relations was incorporated into the processing pipeline using the radius search algorithm. However, this required a manual tuning of the radius parameter r, which is prone to the initial and inaccurate pose estimate and can depend on the data distribution. Therefore, the distant-dependant radius r_d, see Equation (6.18), was introduced, that adapts the radius according to the distance of the measurement. Nevertheless, the parameter α_{GPS}, which denotes the orientation accuracy that can, for example, be obtained from the sensor data-sheet, is still a hyperparameter that may be adjusted in different scenarios. Moreover, increasing the radius for distant measurement can induce negative side effects, e.g., erroneous associations, which hinders generalization capabilities. Due to these restrictions, the attention mechanism is employed to approximate the localization task, which is split into the following three subtasks: 1) The association process that learns to match measurements and landmarks in higher dimensional space where the attention mechanism scores each possible measurement-to-landmark association to produce a local latent feature representation. 2) The global point cloud registration task that uses the learned features from the association process. And finally, 3) the pose prediction, in other words, the spatial transformation that describes the mapping between landmarks and measurements. Here, the measurement-to-landmark association process is given by $h_{\boldsymbol{\phi}}(\cdot)$ and the point cloud registration together with the final pose regression is given by $g_{\boldsymbol{\theta}}(\cdot)$,

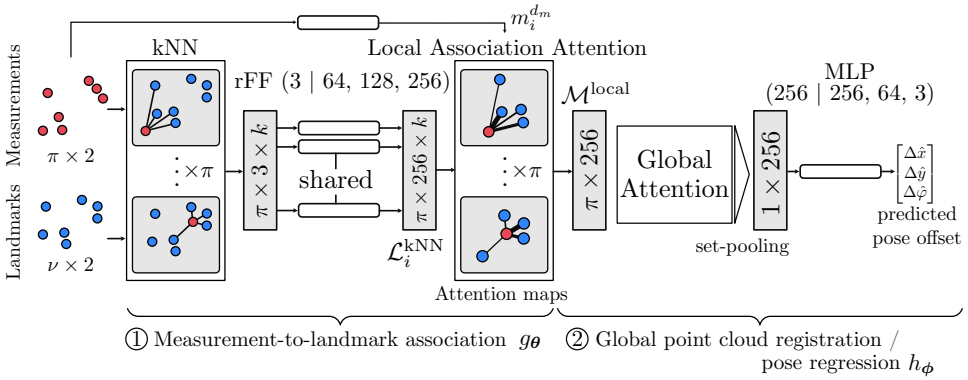


Figure 6.7: Attention-based DeepLocalization network architecture. Instead of heuristically selecting the closest landmarks for each measurement, the network learns to associate the most probable combination using the attention mechanism. Instead of heuristically weighting each possible association, the attention mechanism learns to score the assignments, as depicted in the attention maps. The results of the association processes of every measurement are related using a subsequent global attention operation to then predict the pose offset. Adapted from [EBD21a] © 2021, IEEE.

which in combination approximates the localization function

$$f(\mathcal{M}_t, \mathcal{L}^{\text{FoV}}) \approx g_{\boldsymbol{\theta}}(h_{\boldsymbol{\phi}}(\mathcal{M}_t, \mathcal{L}^{\text{FoV}})), \tag{6.19}$$

whereas the set of learnable parameters is denoted by $\boldsymbol{\theta}$ and $\boldsymbol{\phi}$. As described in the original DeepLocalization network architecture, the multi-modal sensor measurements $\mathcal{M}_t = \{\boldsymbol{m}_1, \boldsymbol{m}_2, \ldots, \boldsymbol{m}_\pi\}, \boldsymbol{m} \in \mathbb{R}^2$ and the landmarks from the digital map $\mathcal{L}^{\text{FoV}} = \{\boldsymbol{l}_1, \boldsymbol{l}_2, \ldots, \boldsymbol{l}_\nu\}, \boldsymbol{l} \in \mathbb{R}^2$ serve as input sets, which are unordered and can have arbitrary cardinality, as illustrated in Figure 6.7. In the first step, a k-Nearest Neighbor (kNN) search is employed to reduce the total number of possible associations, similar to the radius search in Section 6.3.2 that acts as a grouping mechanism. Again, the idea is to consider the spatial relationship between measurements and landmarks and take into account, that only landmarks in the vicinity are possible origins of sensor measurements. However, the DeepLocalization2 architecture only considers distances between landmarks and measurements; therefore, it is beneficial to implement the radius search heuristically. The kNN algorithm is used for this approach because the attention mechanism can score combinations of measurement-to-landmark associations, thus learning to select probable pairs based on the underlying localization task. Here, it is important to note that employing the

kNN is possible without changing the system architecture as the attention mechanism can handle an arbitrary number of input points [VSP+17]. However, choosing the k parameter for the grouping process instead of a radius r or r_d is usually an easier task which is also more robust to different input data distributions, especially since the network learns to score and weigh suitable associations [EBD21a].

For each of the π measurements at the current time step, the k nearest landmarks are selected, whereas k is a hyperparameter. Then, the Euclidean distances are calculated which, together with the x and y coordinated, serve as the input features $\in \mathbb{R}^{\pi \times 3 \times k}$ for a row-wise feedforward network (rFF), that transforms the selected landmarks to latent space $\mathbb{R}^{\pi \times d_m \times k}$ to capture higher order dependencies [EBD21a; QYSG17]. Here, the model dimension d_m is set to $d_m = 256$. Each projected local kNN neighborhood of landmarks is represented by $\mathcal{L}_i^{\text{kNN}}$, with $i \in \{1, \ldots, \pi\}$, as shown in Figure 6.7. At the same time, the measurements are also transformed to higher dimensional space $\boldsymbol{m}_i^{d_m} \in \mathbb{R}^{256}$, $i \in \{1, \ldots, \pi\}$. Next, in order to relate the measurements to the landmarks, the multi-head attention mechanism is used [VSP+17], that was introduced in Chapter 4. For that, the attention function $\mathcal{A}(\cdot)$ is employed, that defines a mapping of N queries $Q \in \mathbb{R}^{N \times d}$ and N_k key-value pairs $K \in \mathbb{R}^{N_k \times d}$, $V \in \mathbb{R}^{N_k \times d}$ to the latent output space $\mathbb{R}^{N \times d}$ as follows:

$$\mathcal{A}(Q, K, V) = \sigma\left(\frac{QK^T}{1/\sqrt{d}}\right) V, \tag{6.20}$$

where the latent feature dimension is given by d. Moreover, the activation function $\sigma(\cdot)$ that denotes the learned score is set to the softmax function from Equation (2.64). As previously mentioned, Equation (6.20) is also known as scaled dot-product attention $\mathcal{A}(\cdot) : \mathbb{R}^{N \times d}, \mathbb{R}^{N_k \times d}, \mathbb{R}^{N_k \times d} \rightarrow \mathbb{R}^{N \times d}$. Instead of applying one attention operation on the input features, this approach follows the ideas of the multi-head attention head introduced by Vaswani et al. [VSP+17]. Here, the queries, keys, and values are linearly projected h times using separate rFFs to incorporate spatial relations in different subspaces. To each of the h projections, Equation (6.20) is applied in parallel and the result is concatenated and fed to a subsequent row-wise feedforward network

$$\text{Multihead}(Q, K, V) = (\text{head}_1 \oplus ... \oplus \text{head}_h) W^O, \tag{6.21}$$

where $\oplus$ denotes matrix concatenation. The individual attention head modules are given by

$$\text{head}_i = \mathcal{A}(QW_i^Q, KW_i^K, VW_i^V), \tag{6.22}$$

with learnable parameter matrices $W_i^Q \in \mathbb{R}^{d \times d}$, $W_i^K \in \mathbb{R}^{d \times d}$, $W_i^V \in \mathbb{R}^{d \times d}$ and

$W^O \in \mathbb{R}^{d \times d}$. Then, the multi-head attention block is formally defined as [VSP+17]

$$\mathcal{A}^{\mathrm{MH}}(X, Y) = \mathrm{LayerNorm}(S + \mathrm{rFF}(S)), \tag{6.23}$$

which is made up of the attention mechanism (6.20) and residual connections that are normalized using layer normalization from Equation (2.88) [BKH16] with

$$S = \mathrm{LayerNorm}(X + \mathrm{Multihead}(X, Y, Y)). \tag{6.24}$$

To summarize, two arbitrary input sets X and Y are scored in latent feature space $\mathbb{R}^d$, with $\mathcal{A}^{\mathrm{MH}} : \mathbb{R}^{N \times d}, \mathbb{R}^{N_k \times d} \to \mathbb{R}^{N \times d}$.

This mechanism is used for the measurement-to-landmark association process, as visualized in Figure 6.7, where the local association attention $\mathcal{A}^{\mathrm{la}}$ is given as follows:

$$\mathcal{A}_i^{\mathrm{la}} := \mathcal{A}^{\mathrm{MH}}(\boldsymbol{m}_i^{d_m}, \mathcal{L}_i^{\mathrm{kNN}}),\ i = 1, \ldots, \pi. \tag{6.25}$$

The operation is defined such that each measurement $\boldsymbol{m}_i$ that is transformed to model dimension $d = d_m = 256$ using a rFF is attended with the projected k associated landmarks from the kNN algorithm. Therefore, the result is given by $\mathcal{A}_i^{\mathrm{la}} \to \mathbb{R}^{1 \times d_m}$ with $\boldsymbol{m}_i^{d_m} \in \mathbb{R}^{1 \times d_m}$ and $\mathcal{L}_i^{\mathrm{kNN}} \in \mathbb{R}^{k \times d_m}$, where the local association attention (6.25) produces features for every measurement containing the latent representation of the scored landmarks. Consequently, the network is able to learn the most suitable associations, which is visualized as the weighted connections of the attention maps in Figure 6.7. Finally, the results of $\mathcal{A}_i^{\mathrm{la}}$, with $i \in \{1, \ldots, \pi\}$ are concatenated to obtain the local feature matrix $\mathcal{M}^{\mathrm{local}} \in \mathbb{R}^{\pi \times d_m}$, which is comprised of latent features describing the associated landmarks and measurements. However, at this point, only local information is processed, therefore, another self-attention operation, which is called global attention in Figure 6.7, is employed to attend each of the measurement representations

$$\mathcal{A}^{\mathrm{global}} := \mathcal{A}^{\mathrm{MH}}(\mathcal{M}^{\mathrm{local}}, \mathcal{M}^{\mathrm{local}}), \tag{6.26}$$

with $\mathcal{A}^{\mathrm{global}} : \mathbb{R}^{\pi \times d_m}, \mathbb{R}^{\pi \times d_m} \to \mathbb{R}^{\pi \times d_m}$. Regarding the localization subtasks, Equation (6.26) represents the point cloud registration process, as features from both input sets are combined and related with the goal of finding the spatial transformation that optimizes the landmarks to measurement associations from the first step. Subsequently, a max-pooling operation (2.66) follows to obtain a global feature representation of fixed size that is invariant to permutations of the input point sets following the ideas of [QYSG17; VSP+17]. Finally, a MLP (2.50) is employed to predict the pose correction vector $\Delta\hat{\boldsymbol{p}}$.

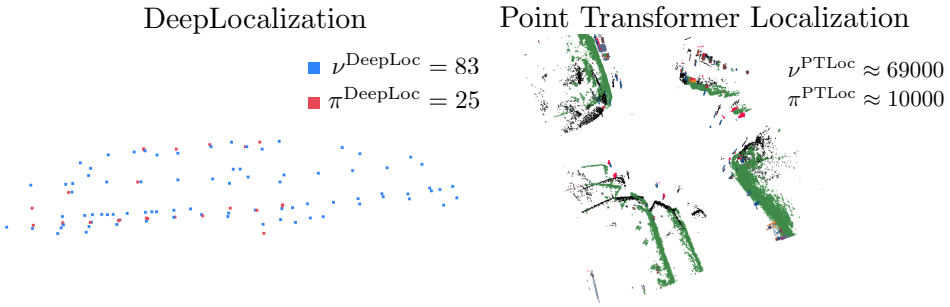


Figure 6.8: Comparison of point cloud densities. The cardinality of the input sets for the Point Transformer Localization is higher by multiple orders of magnitude compared to DeepLocalization.

6.4 Point Transformer Localization

This sections provides an introduction on implementing the Point Transformer approach from Chapter 4, for the task of vehicle self-localization. As explained in the introduction, the DeepLocalization network architecture is well-suited to handle sparse point clouds, where each static and easily recognizable object in the environment is represented by one generic landmark. Moreover, the multi-modal sensor measurements are clustered to obtain a similar representation that can be related to the landmarks from the digital map, as shown in the previous section. However, it is evident that the presented sparse definition of the environment using static 2D objects without any semantic or geometric features disregards and neglects information that can be useful for a deep neural network and may improve the matching, registration, and pose regression task. Recent technological advances in point cloud processing are fueled by increasingly more powerful sensors that are becoming more performant while acquisition and operational costs are decreased. The development is supported by increased competition of various hardware manufacturers and increased research interest. Therefore, the exploration of high-density 3D-based localization methods is a key contribution of this thesis. While the sparse representation of landmarks in the DeepLocalization architecture allows for a one-to-one association of measurements and landmarks, this is no longer possible with high-density point clouds. This becomes apparent when the sparse and the dense point clouds are compared to each other in Figure 6.8, where the cardinality of both inputs differ by multiple orders of magnitude, see Equation (6.2). Assigning one point from the lidar point cloud to one point of the high-density digital map is impractical. In this case, it is more reasonable to compare geometric structures, such as trees or buildings and finds spatial relations among local regions of the lidar scan point cloud and the 3D

points from the digital map that are in the vehicle's field of view. This task is more similar to the presented Point Transformer approach from Chapter 4, where spatial information is used to obtain a local and global latent feature representation to solve the task of object classification and part segmentation. As this section will show, the ideas of the Point Transformer network architecture can be adapted to tackle the difficult task of registering multiple dense point clouds to obtain the vehicle's pose.

6.4.1 Introduction

In general, the overall system setup is related to the DeepLocalization approach, as explained in Section 6.3. In the following, the individual steps are revisited shortly, and the differences are laid out: The input to the network is again comprised of two unordered point sets with arbitrary cardinality. In order to highlight the fact that not individual points are considered, but entire scenes with semantic information, the measurements are called the **scan** in this section $\mathcal{M}_t = \{\boldsymbol{m}_1, \boldsymbol{m}_2, \ldots, \boldsymbol{m}_\pi\}, \boldsymbol{m} \in \mathbb{R}^D$, and the landmarks are called the **map** $\mathcal{L}^{\text{FoV}} = \{\boldsymbol{l}_1, \boldsymbol{l}_2, \ldots, \boldsymbol{l}_\nu\}, \boldsymbol{l} \in \mathbb{R}^D$. Here, it is important to note that the magnitude of the point set cardinality is higher by multiple orders compared to the landmark-based localization from the previous section, as given in Equation (6.2). Moreover, three-dimensional input points are processed, whereby additional features are appended as required, thus $D \geq 3$. For example, the Point Transformer Localization approach uses semantic class information for each 3D input point ($D = 4$), as introduced in Section 5.3. As mentioned before, it is not possible to find or define a one-to-one measurement-to-landmark association. Therefore, in this approach, the localization task is reduced to the point cloud registration subtask and the subsequent pose estimation. However, the training approach follows the same ideas: Starting from an initial inaccurate pose estimate, points from the high-density digital map are loaded in the vehicle's field of view and transformed from global coordinates into the vehicle frame. In the next step, the network predicts the offset from the initial pose to the ground-truth by aligning both input point clouds, i.e., the scan and the map, and finds the spatial transformation that best describes the registration between $\mathcal{M}_t$ and $\mathcal{L}^{\text{FoV}}$. The novelties of the Point Transformer Localization approach can be summarized as follows:

1. Introduction of a novel network architecture that is able to register two dense 3D lidar point clouds, namely the scan and the map, to infer the vehicle's pose.

2. Incorporation of semantic and geometric information to support the registration task.

3. Proposition of a local and global attention operation as introduced in Chapter 4 to aggregate spatial information.

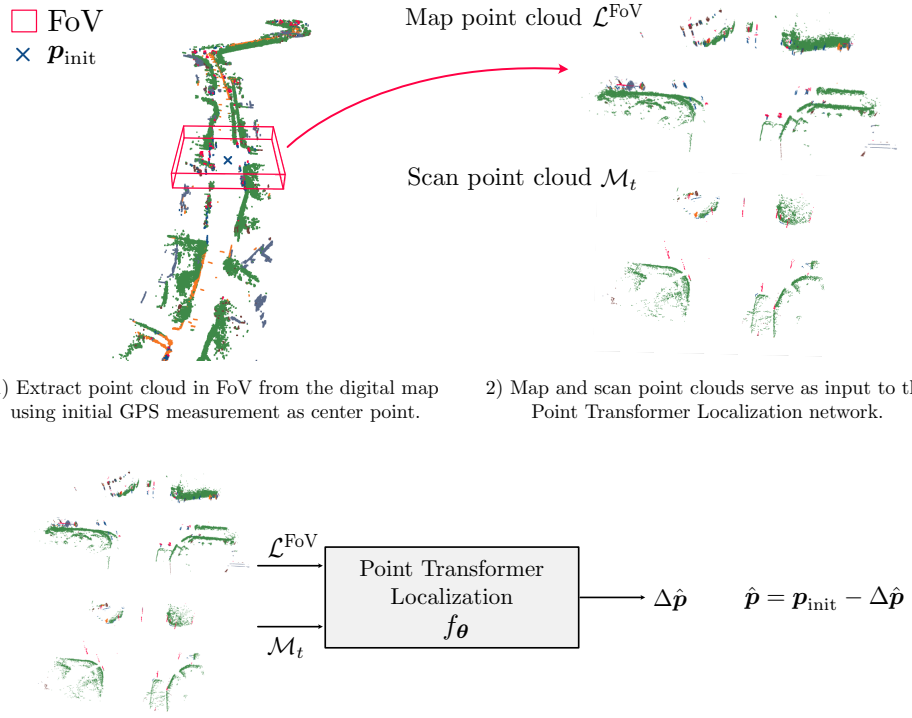


Figure 6.9: Overview of the processing steps for the Point Transformer Localization pipeline. First, the points in the vehicle's field of view are loaded from the digital map, based on the initial pose estimate $\boldsymbol{p}_{\text{init}}$. This, together with a scan point cloud with semantic class labels, where only static objects are extracted, serve as input to the Point Transformer Localization architecture.

In the subsequent sections, the methodology is introduced for the individual steps of the Point Transformer Localization approach, which are also illustrated in Figure 6.9. Here, the initial pose estimate $\boldsymbol{p}_{\text{init}}$ from a standard GPS system is used to extract the map point cloud $\mathcal{L}^{\text{FoV}}$ in the field of view of the vehicle. The scan point cloud $\mathcal{M}_t$ is processed by a semantic segmentation network and only points with static class labels are extracted. Then, the scan and the map points are fed to the network to infer the pose correction vector. Apart from the network architecture, the objective, training steps, and inference methods are equal to the DeepLocalization approach as explained in Section 6.3.1.

6.4.2 Methodology

The Point Transformer Localization methods combines the discoveries and conclusions of the DeepLocalization and the Point Transformer approaches and introduces an architecture that is capable of learning semantic and geometric shape information between two dense lidar point clouds. The input is comprised of two point clouds that are taken from the Point Transformer Localization dataset introduced in Section 5.3, in particular the scan $\mathcal{M}_t \in \mathbb{R}^{\pi \times D}$ and the map $\mathcal{L}^{\mathrm{FoV}} \in \mathbb{R}^{\nu \times D}$, whereas each point is comprised of a 3D position and a class label, thus $D = 4$. Similar to the DeepLocalization methodology, the scan and the map are used to infer the pose offset using a deep neural network

$$f_{\boldsymbol{\theta}}(\mathcal{M}_t, \mathcal{L}^{\mathrm{FoV}}) \rightarrowtail \hat{\boldsymbol{p}}, \tag{6.27}$$

where $\boldsymbol{\theta}$ denotes the set of learnable parameters. Here, it is important to note that both point clouds $\mathcal{M}_t$ and $\mathcal{L}^{\mathrm{FoV}}$ are preprocessed using a semantic segmentation neural network, e.g., RandLA-Net [HYX+20], where a class label is associated to each point, cf. Chapter 5. Furthermore, since the localization task ideally requires static and recognizable objects as input, i.e., the landmarks, all points are discarded that are assigned to a dynamic class as stated in Chapter 5 and Appendix B.1. Consequently, only points originating from the static environment are used as input to the network; however, noise is still prevalent due to misclassifications of the segmentation network. The classification result can additionally be used to separately process each class individually such that only the same object classes are used at a time. For example, points from the lidar scan that belong to the class *pole* are only matched and registered against poles from the high-density digital map to reduce erroneous associations. The following section introduces the network architecture and a novel attempt to incorporate and model a localization pose uncertainty using a new loss function.

Network Architecture

The processing steps of the Point Transformer Localization can be described as follows: First, both point clouds are divided according to the class labels and fed to the Point Transformer Localization network, which is illustrated in Figure 6.10. As mentioned earlier, each class c is then processed individually to avoid matching scan points and map points that are in the vicinity of each other but originate from another object class. Following the ideas of the DeepLocalization2 and attention-based localization approach, a heuristic downsampling approach first reduces the scan point cloud size of the current class to N' points $\boldsymbol{m}_j, j \in \{1, \ldots, N'\}$ to decrease computational complexity for subsequent algorithms. Here, the points of

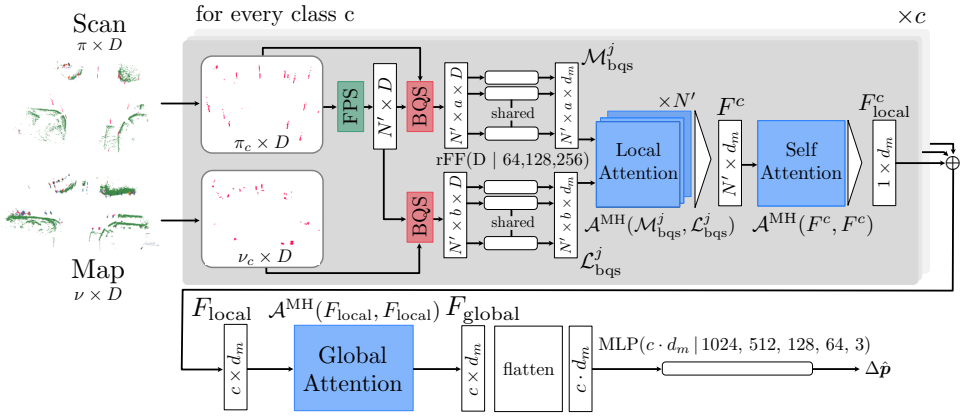


Figure 6.10: Overview of the Point Transformer Localization network architecture. The illustrated process is repeated for every static object class c. First, a FPS algorithm downsamples the scan point cloud to N' points. The subsampled points are used as center points for a ball query search algorithm (BQS) to select scan points and map points in the vicinity. Then, a rFF transforms the selected points to latent feature space d_m. A local attention operation follows to relate the scan and map points (measurement-to-landmark association). Afterward, set-pooling operations produce a local feature vector of fixed size. Finally, the result for every class is concatenated and related again using a global attention mechanism (point cloud registration), where a subsequent MLP generates the output vector. Here, it is important to note that the rFF is applied to every $\mathcal{M}^j_{\text{bqs}}$ and $\mathcal{L}^j_{\text{bqs}}$ individually, but is illustrated together to increase readability.

the downsampled point cloud are called key points. It is desirable to cover as much of the original scan point cloud as possible; therefore, the furthest point sampling algorithm (FPS) from Section 3.1 is employed to downsample the input while equally distributing the key points [QYSG17]. It is important to note that other algorithms are also imaginable, such as the learned top-k selection process introduced in the Point Transformer method. However, in order to keep the approach feasible, the popular and established FPS algorithm is used. Starting from each of the N' scan points $\boldsymbol{m}_j$, a ball query search algorithm (BQS), cf. Section 6.3.2, is employed on both the scan point cloud and the map point cloud of the current segmentation class. The goal of the BQS is to find all points in the neighborhood that have a minimum distance of r_{bqs} to each key point. Then, the relative Euclidean distances of the neighboring points to the key point are calculated, which are used to generate latent

local features using a row-wise feedforward network that transforms each point into a higher dimensional space $\mathbb{R}^{d_m}$. The result of the ball query search and the rFF for the j-th scan key point is denoted as $\mathcal{M}^j_{\text{bqs}} = \{\boldsymbol{m}^j_0, \ldots, \boldsymbol{m}^j_a\}, \boldsymbol{m}^j_{(\cdot)} \in \mathbb{R}^{d_m}$ for all scan points in the vicinity and $\mathcal{L}^j_{\text{bqs}} = \{\boldsymbol{l}^j_0, \ldots, \boldsymbol{l}^j_b\}, \boldsymbol{l}^j_{(\cdot)} \in \mathbb{R}^{d_m}$ for all map points in the vicinity, whereas a and b are the total number of points inside the radius r_{bqs}, respectively. The latent features of the scan and the map points inside the radius r_{bqs} are then fed to a cross-attention layer

$$\mathcal{A}^{\text{MH}}(\mathcal{M}^j_{\text{bqs}}, \mathcal{L}^j_{\text{bqs}}), \tag{6.28}$$

that relates the selected scan points and map points and imitates the association process. The result of the cross-attention layer (6.28), where $\mathcal{A}^{\text{MH}}(\cdot) : \mathbb{R}^{a \times d_m}, \mathbb{R}^{b \times d_m} \rightarrow \mathbb{R}^{a \times d_m}$, aggregates the local dependencies and spatial relations between the scan and the map. In order to produce permutation-invariant features of fixed size, a max-pooling operation (2.66) is applied feature-wise, i.e., set-pooling (2.69) for every m_j, which are then concatenated. Therefore, for every segmentation class c, a local feature vector $F^c \in \mathbb{R}^{N' \times d_m}$ is obtained, where the features from the FPS algorithm are related against each other using a second self-attention operation

$$\mathcal{A}^{\text{MH}}(F^c, F^c), \tag{6.29}$$

that is followed by a set-pooling operation (2.69) to produce a permutation-invariant latent feature vector of fixed size that aggregates the local features of the current class $F^c_{\text{local}} \in \mathbb{R}^{1 \times d_m}$. The presented approach that constitutes the scan to map association process is repeated for every class, whereas the networks share their respective weights to decrease memory requirements. The results are concatenated to obtain the local feature vector that describes the local associations for every class

$$F_{\text{local}} = F^1_{\text{local}} \oplus \cdots \oplus F^c_{\text{local}},\ F_{\text{local}} \in \mathbb{R}^{c \times d_m}. \tag{6.30}$$

Following the ideas of the attention-based self-localization from Section 6.3.3, global features $F_{\text{global}} \in \mathbb{R}^{c \times d_m}$ are generated by employing a global attention operation, similar to Equation (6.29)

$$\mathcal{A}^{\text{MH}}(F_{\text{local}}, F_{\text{local}}). \tag{6.31}$$

Here, the features of all classes are related to create a global signature of the map and the scan point clouds, where all possible associations are considered. Finally, the result is flattened, i.e., transformed to a 1D vector of fixed size $\mathbb{R}^{c \cdot d_m}$. As shown in Figure 6.10, a subsequent output head transforms the global features into the output space using a multilayer perceptron (MLP).

To summarize: The Point Transformer Localization method processes each static

class separately. For that, both the scan and the map point cloud of the current class are fed to the network. Then, the furthest point sampling (FPS) algorithm reduces the number of scan points to N', that act as center points for a subsequent ball query search (BQS). The BQS finds all scan points ($\mathcal{M}^j_{\text{bqs}}$) and map points ($\mathcal{L}^j_{\text{bqs}}$) that are in the vicinity of each center point from the FPS. Then, a rFF transforms the local neighborhoods to latent feature space d_m. A local attention operation follows to relate the selected scan and map points of the current center point from the BQS. Afterward, a set-pooling method generates a feature vector of fixed size. The result of the local attention operations for every local neighborhood is concatenated (F^c) and processed by an additional self-attention module with a subsequent set-pooling operation (F^c_{local}). This process is repeated for every class and the results are concatenated again (F_{local}). A final global attention operation relates the latent features of every class to subsequently infer the pose correction vector.

Pose Uncertainty Loss

In many real-world applications in the field of automated robots, it is desirable to obtain or estimate the uncertainty that is associated with the current task, such as object detection and tracking [EHHD18], human pose estimation [WBKB20] and vehicle self-localization [Stü18]. Until now, the DeepLocalization-based approaches always assumed a fixed and distinct pose estimate obtained from the network output, i.e., the pose correction vector. However, traditional probabilistic localization methods that rely on a filter-based implementation, such as the RFS-MCL [Stü18], provide an uncertainty estimate out-of-the-box because it is part of the underlying mathematical model. By obtaining an uncertainty measure, subsequent software modules can utilize the information to increase the overall reliability and performance. For example, when the localization software module provides a pose subject to high uncertainty, an object detection algorithm can apply the information to the global position estimates of the detected objects, such as other road participants. Following these ideas and motives, a pose uncertainty loss is introduced. In the literature, it is common practice to model the vehicles's position as a two dimensional multivariate normal distribution from Equation (2.32) [Thr02], where the mean denotes the x and y position and the covariance the position uncertainty σ_x and σ_y, respectively. Thus, it follows that the vehicle's position p^{pos} is modeled as

$$p^{\text{pos}} \triangleq \mathcal{N}(\boldsymbol{\mu}, \Sigma), \tag{6.32}$$

with $\boldsymbol{\mu} = [x,\, y]^T$ and $\Sigma = \text{diag}(\sigma_x, \sigma_y)$. For the case of deep neural networks, two position distributions are necessary, namely a ground-truth position distribution $p^{\text{pos}}_{\text{GT}}$ and a predicted position distribution $\hat{p}^{\text{pos}}$. In this case, the predicted distribution is

estimated by the deep neural network. On the other hand, the ground-truth pose is obtained from a dGPS reference system as explained in Chapter 5; thus, it is possible to define a ground-truth distribution where the mean is equal to the ground-truth pose and the covariance is approximated by the measurement noise of the dGPS sensor, which is usually provided in the manufacturers' data-sheet. Apparently, the learning objective is to approximate the ground-truth distribution. This can be done by employing the Kullback-Leibler (KL) divergence (2.34) as a loss function

$$L_{\mathrm{kl}}^{\mathrm{pos}}(\hat{p}^{\mathrm{pos}}, p_{\mathrm{GT}}^{\mathrm{pos}}) \triangleq \mathbb{KL}(\hat{p}^{\mathrm{pos}} \,||\, p_{\mathrm{GT}}^{\mathrm{pos}}) = \sum \hat{p}^{\mathrm{pos}} \log \frac{\hat{p}^{\mathrm{pos}}}{p_{\mathrm{GT}}^{\mathrm{pos}}}, \tag{6.33}$$

that takes both position distributions as input, i.e., $\hat{p}^{\mathrm{pos}}$ and $p_{\mathrm{GT}}^{\mathrm{pos}}$. One important characteristic of the Kullback-Leibler divergence is, that the result is zero if $\hat{p}^{\mathrm{pos}} = p_{\mathrm{GT}}^{\mathrm{pos}}$ and $\mathbb{KL}(\hat{p}^{\mathrm{pos}} \,||\, p_{\mathrm{GT}}^{\mathrm{pos}}) > 0$ if $\hat{p}^{\mathrm{pos}} \neq p_{\mathrm{GT}}^{\mathrm{pos}}$. Furthermore, the Kullback-Leibler divergence is differentiable; thus, it is well suited to be directly employed as a loss function. In practice, this change leads to a different output dimension, in other words, the network predicts the following vector $[\hat{x},\, \hat{\sigma}_x,\, \hat{y},\, \hat{\sigma}_y,\, \hat{\varphi}]^T$. As mentioned earlier, the covariance matrix of the normal distribution for the ground-truth is constructed according to the data-sheet of the dGPS manufacturer, which in this case is $\sigma_{x,\mathrm{GT}} = \sigma_{y,\mathrm{GT}} = 0.05\,\mathrm{m}$. To summarize, a multivariate normal distribution is generated for the predicted pose from the network ($\hat{p}^{\mathrm{pos}}$), which requires an extension of the output space to include the covariance estimates as well. Similarly, a multivariate ground-truth normal distribution is constructed ($p_{\mathrm{GT}}^{\mathrm{pos}}$), which has a relatively small covariance or uncertainty, since the the dGPS measurement is assumed to be accurate. Then, the learning objective of the network is to predict a multivariate normal distribution ($\hat{p}^{\mathrm{pos}}$) that is equal or similar to the constructed ground-truth distribution ($p_{\mathrm{GT}}^{\mathrm{pos}}$). For that, the Kullback-Leibler divergence (6.33) is employed, which is a statistical distance measure, i.e., it forces the network to approximate the ground-truth distribution. In that case, the parameters of the network are optimized with respect to minimizing the difference between the position estimate $\hat{\boldsymbol{\mu}} = [\hat{x},\, \hat{y}]^T$ and the ground-truth position $\boldsymbol{\mu}_{\mathrm{GT}} = [x_{\mathrm{GT}},\, y_{\mathrm{GT}}]^T$, as well as the covariance estimate $\hat{\sigma}_x, \hat{\sigma}_y$ and the ground-truth covariance.

Here, it is also possible to employ the Jensen-Shannon divergence from Equation (2.35) to obtain a symmetric loss function. Finally, the covariance matrix of the multivariate normal distribution is required to be positive definite, which can be guaranteed if $\hat{\sigma}_x, \hat{\sigma}_y > 0$. Therefore, the following activation function is applied after the last output neuron for the predicted variances $\tilde{\sigma}_x$ and $\tilde{\sigma}_y$

$$\hat{\sigma}_x = 1 + R(\tilde{\sigma}_x), \tag{6.34a}$$

$$\hat{\sigma}_y = 1 + R(\tilde{\sigma}_y), \tag{6.34b}$$

where $R(\tilde{\sigma})$ is the exponential linear unit, also called the elu activation function,

that is defined as

$$R(\tilde{\sigma}) = \begin{cases} \tilde{\sigma} & \tilde{\sigma} > 0, \\ \alpha(\exp(\tilde{\sigma}) - 1) & \tilde{\sigma} \leq 0, \end{cases} \tag{6.35}$$

and the scaling factor is set to $\alpha = 1$. Applying the differentiable elu activation function to $\tilde{\sigma}_x, \tilde{\sigma}_y$ guarantees the positive definiteness of the predicted covariance matrix, which is a required property for the calculation of the Kullback-Leibler loss from Equation (6.33). The overall method of employing the pose uncertainty loss (6.33) forces the network to predict higher uncertainties if the pose estimate $\hat{\boldsymbol{\mu}} = [\hat{x}, \hat{y}]^T$ is subject to a low confidence, because it leads to a lower Kullback-Leibler divergence and vice versa. Consequently, σ_x and σ_y can be interpreted as a learned pose uncertainty. Finally, the same concept is applied for the orientation, however, here a one dimensional normal distribution, i.e., Equation (2.28), is assumed for the orientation $p^{\text{ori}} \triangleq \mathcal{N}(\varphi, \sigma_\varphi)$. Therefore, the output of the network expands to $[\hat{x},\, \hat{\sigma}_x,\, \hat{y},\, \hat{\sigma}_y,\, \hat{\varphi},\, \hat{\sigma}_\varphi]^T$ and the orientation loss is given as follows:

$$L_{\text{kl}}^{\text{ori}}(\hat{p}^{\text{ori}}, p_{\text{GT}}^{\text{ori}}) \triangleq \mathbb{KL}(\hat{p}^{\text{ori}} \,||\, p_{\text{GT}}^{\text{ori}}). \tag{6.36}$$

Consequently, the complete loss function that is used for optimizing the Point Transformer Localization network is defined as

$$L_{\text{total}} = L_{\text{kl}}^{\text{pos}} + L_{\text{kl}}^{\text{ori}}, \tag{6.37}$$

where the Kullback-Leibler loss for the position from Equation (6.33) and the Kullback-Leibler loss for the orientation from Equation (6.36) equally contribute to the total loss. The advantage of this approach is that due to modeling the localization error as the statistical distance or similarity of two probability distributions, the position and orientation error are provided in the same order of magnitude, regardless of the specific units, e.g., meters and radians. Moreover, since the two-dimensional position error is combined into one dependent loss, it forces the network to optimize for both position components x and y simultaneously, decreasing training time. Since no learnable weighting terms are involved, the pose uncertainty loss's aforementioned characteristics lead to faster convergence and increased accuracy. However, at this point, it is important to note that the pose uncertainty introduced in this section is comparable to the aleatoric uncertainty proposed by [KG17]. Aleatoric uncertainty captures observation noise; that is, the uncertainty stems from a limited data distribution. In order to obtain a thorough uncertainty measure, it is also necessary to include the epistemic uncertainty, i.e., model uncertainty, for example, by employing Monte-Carlo dropout simulations.

Linear Attention Transformer

This section introduces an additional implementation detail that is different from the original Point Transformer architecture due to the increase of the point cloud cardinality while processing lidar scans. The goal is to decrease the computational complexity by employing optimized variants of the Transformer architecture. Here, it is important to note that the following method is optional and can be seen as a drop-in replacement mechanism for the traditional attention operation.

As mentioned in previous chapters, the original Transformer architecture proposed by Vaswani et al. [VSP+17] has $\mathcal{O}(n^2)$ complexity, where n denotes the sequence length. In the case of Point Transformer and Point Transformer Localization, this means that the complexity scales quadratically with respect to the number of points. This limitation is negligible for artificial datasets, like the ModelNet40 or ShapeNet dataset, where the point set cardinality is known and adaptable, e.g., $N = 1024$. However, when the number of points varies significantly between different scans, e.g., as prevailing in the Point Transformer Localization dataset, this can lead to unforeseen increases in inference times that violate real-time requirements. In recent years, several improvements have been proposed to the original Transformer architecture that achieve linear complexity $\mathcal{O}(n)$ with respect to the total number of points. This is usually done by making use of various properties of matrix multiplication and beneficial reformulations of the attention operation, for example, by approximating the attention mechanism using a low-rank matrix [WLK+20] or by expressing the attention as a linear dot-product of kernel feature maps [KVPF20]. This allows for employing the powerful attention mechanism as proposed in the Point Transformer architecture to operate on large-scale point clouds, such as the Point Transformer Localization dataset while meeting real-time requirements and avoiding decreases in the overall localization performance. In practice, the attention and Transformer modules are replaced by Linear Attention Transformer implementations [KVPF20]. This can easily be achieved without further alterations to the network architecture while leading to significant performance increases such that the adapted model is real-time capable. In summary, the presented Transformer modules from Chapter 4 are replaced by optimized Transformer variants that achieve linear complexity with respect to the number of input points. An important characteristic of this substitution is, that the overall methodology does not change and all important and beneficial properties of employing attention for point cloud processing still hold.

As mentioned before, this is purely an implementation detail that decreases inference time for the presented Point Transformer approach by adapting the attention operation without altering the overall network architecture.

Chapter 7

Experiments and Evaluation

This chapter introduces the experiments and evaluations of the presented approaches for vehicle self-localization on the datasets created for this thesis. For this, the following sections are divided into two parts: First, the DeepLocalization approach and the proposed network improvements are evaluated in multiple ways to highlight the effectiveness of each iteration. Moreover, current state-of-the-art methods that employ a filter-based approach are compared to show superior performance of DeepLocalization. In the second part, the dense lidar-based dataset is used to evaluate the Point Transformer Localization architecture. All methods are implemented using the popular PyTorch framework [PGM+19] and are performed on an Nvidia GeForce 1080Ti graphics card.

7.1 DeepLocalization

As mentioned, DeepLocalization is evaluated in two different ways: A synthetic experiment is conducted, where different sensor deterioration effects are applied to the measurements systematically, in order to highlight the robustness of the presented learning-based localization methods against common noise sources. Moreover, real-world experiments are carried out that utilize the sparse point clouds obtained from real clustered measurements and generic 2D landmarks from the digital map. Here, the proposed network architectures are compared against popular filter-based approaches that achieve state-of-the-art results and traditional localization methods, e.g., the iterative closest point algorithm (ICP), that act as a baseline for the evaluation.

7.1.1 Synthetic Experiments

The goal of the simulated experiment that uses synthetic input data is to systematically evaluate the robustness of the DeepLocalization approaches. In detail, the process involves the piecewise deterioration of sensor measurements, whereas the severity of the worsening effects is controlled to examine the effect on the localization accuracy. In the following, the process of automatically generating the synthetic input data samples is explained. Then, the three network architectures are evaluated to confirm the claim that each iteration improves the network's capability to handle noisy inputs.

Suppose the ideal case is considered and no occlusions or dynamic objects are present. In that case, the multi-modal sensors capture every landmark from the digital map in the vehicle's field of view. In this thesis, three types of possible sensor impairments are introduced in Section 2.3, that occur in realistic scenarios: 1) When a landmark is occluded, not visible from the current sensor position, or not detected due to sensor failure, this effect is called a **missed detection**. 2) **Clutter measurements** are caused by dynamic objects that are falsely classified as landmarks or erroneous sensor detections, which are also called *ghost detections*. Finally, 3) all sensors are subject to **measurement noise** that cause inaccurate position estimates. Identifying the sensor impairments is a useful insight that helps evaluate the robustness of the proposed localization methods. Since each of the three effects can be considered separately, it is possible to draw conclusions regarding each impairment. For that, mathematical models for each of the introduced sensor limitations are introduced. Then, the goal is to incrementally increase each effect and apply it to the measurements separately. Finally, the effects are applied simultaneously and, again, the severity is increased to observe the influence on the accuracy.

Synthetic Measurement and Landmark Generation

The generation of the synthetic measurements and landmarks relies on the dataset from Section 5.2, which is introduced for the DeepLocalization method. Next, the processing steps are outlined that result in the creation of synthetic measurements $\mathcal{M}^{\text{syn}}$ and landmarks $\mathcal{L}^{\text{syn}}$. First, a data sample from the dataset is loaded, and only the ground-truth pose $\boldsymbol{p}_{\text{GT}}$ is extracted. Originating from $\boldsymbol{p}_{\text{GT}}$, all landmarks $\mathcal{L}^{\text{FoV}}$ are loaded from the digital map that are inside the vehicle's field of view. As mentioned in the previous section, the synthetic experiments assume ideal sensor measurements initially. That is, every landmark from the digital map is detected at the correct position without any measurement noise, and no clutter measurements occur in the current scene. This is done by simply duplicating the landmarks

for the measurements, i.e., $\mathcal{M}_t = \mathcal{L}^{\text{FoV}}$. Then, three separate experiments are conducted where each proposed deterioration effect is applied incrementally to $\mathcal{M}$. It is necessary to define mathematical models for the impairments that can also be adapted using well-defined hyperparameters. Since both the clutter measurements and missed detections are events that can happen multiple times in one scan, the Poisson distribution (2.27) is employed

$$\text{Pois}_{\text{m}}(k_m; \lambda_m) = \frac{\lambda_m^{k_m} e^{-\lambda_m}}{k_m!}, \tag{7.1a}$$

$$\text{Pois}_{\text{c}}(k_c; \lambda_c) = \frac{\lambda_c^{k_c} e^{-\lambda_c}}{k_c!}, \tag{7.1b}$$

where λ_m and λ_c denote the average number or occurrences of missed detections and clutter measurements per scan, respectively. Using Equations (7.1), it is possible to control the severity of each of the impairments individually by adapting the hyperparameter λ. For every time step in the synthetic experiment, the number of missed detections k_m is randomly drawn from the Poisson distribution (7.1a) with λ_k. Then, k_m measurements are deleted from the original measurement set $\mathcal{M} \in \mathbb{R}^{\pi \times 2}$ that is loaded from the dataset. Here, the deletions are performed randomly, which additionally acts as a data augmentation method. Similarly, for the clutter measurement, the total number of false detections k_c is drawn from another Poisson distribution (7.1b). Consequently, the k_c additional measurements are placed randomly inside the vehicle's field of view where the position follows a multivariate Gaussian distribution (2.32)

$$\mathcal{N}(\boldsymbol{x}; \boldsymbol{\mu}, \Sigma) = \frac{1}{\sqrt{(2\pi)^k |\Sigma|}} \exp\left(-\frac{1}{2}(\boldsymbol{x} - \boldsymbol{\mu})^T \Sigma^{-1} (\boldsymbol{x} - \boldsymbol{\mu})\right), \tag{7.2}$$

where $\boldsymbol{\mu}$ and Σ are set close to the mean and variance of the landmarks in the vehicle's field of view of all sample in the dataset [EBD21a]. In this particular case, a gaussian mixture is used, i.e., a combination of multiple gaussians

$$\mathcal{N}(\boldsymbol{x}_1, \boldsymbol{x}_2; \boldsymbol{\mu}_1, \boldsymbol{\mu}_2, \Sigma_1, \Sigma_2) = \lambda_1 \mathcal{N}(\boldsymbol{x}_1; \boldsymbol{\mu}_1, \Sigma_1) + \lambda_2 \mathcal{N}(\boldsymbol{x}_2; \boldsymbol{\mu}_2, \Sigma_2), \tag{7.3}$$

where λ_1, λ_2 denote weighting parameters in order to scale each component of the gaussian mixture. To obtain realistic clutter measurements that mostly occur at the side of the road, the synthetic measurements to be added to $\mathcal{M}$ are sampled from the distribution from Equation (7.3) with parameters $\boldsymbol{\mu}_1 = [20, -2]^T$, $\boldsymbol{\mu}_2 = [20, 2]^T$, $\Sigma_1 = \Sigma_2 = \text{diag}(120, 1)$ and $\lambda_1 = 1, \lambda_2 = 0.6$ [EBD21a]. The resulting distribution is visualized in Figure 7.1. Here, it is important to note that both the number of clutter measurements and missed detections is limited, such that the resulting measurement set is valid and in the range of realistic scenarios, i.e., $0 < \pi_{\text{min}}^{\text{syn}} \leq \pi^{\text{syn}} \leq \pi_{\text{max}}^{\text{syn}}$. This means, that the total number of synthetically

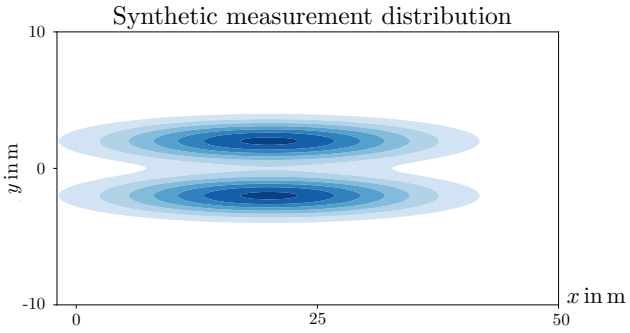


Figure 7.1: Multivariate Gaussian mixture distribution that is used to sample synthetic clutter measurements.

generated measurements is realistic compared to real-world scenarios. Finally, the last step is to apply measurement noise to the artificially generated data samples. Measurement noise simply represents a displaced landmark localization due to inaccurate sensors, which can be simulated by randomly shifting a measurement. Consequently, for each of the synthetic measurements, noise is drawn from a uniform distribution (2.26) $\boldsymbol{\sigma}_i^{\text{meas}} \sim \mathcal{U}_{[-\sigma,\sigma]}$, $\boldsymbol{\sigma}_i^{\text{meas}} \in \mathbb{R}^2$ and is applied to the position $\tilde{\boldsymbol{m}}_i = \boldsymbol{m}_i + \boldsymbol{\sigma}_i^{\text{meas}}$, whereas the noise is sampled separately for each measurement. Therefore, the final synthetic measurement set is given as follows:

$$\mathcal{M}_{\text{syn}} = \{\tilde{\boldsymbol{m}}_1, \tilde{\boldsymbol{m}}_2, \ldots, \tilde{\boldsymbol{m}}_{\pi^{\text{syn}}}\}, \quad \pi_{\text{syn}} = \nu_{\text{syn}} + k_c - k_m, \tag{7.4}$$

and the synthetic landmark set is denoted by

$$\mathcal{L}_{\text{syn}} = \mathcal{L}^{\text{FoV}}, \tag{7.5}$$

where $\mathcal{L}^{\text{FoV}}$ contains all landmarks that are currently in the vehicle's field of view. Note, that the real measurements from the dataset are not used in this experiment. The synthetic data generation process is also visualized in Figure 7.2.

Results

As explained earlier, the synthetic experiments aim to investigate the robustness of the presented localization approaches. The previous section introduced the generation process of the artificial measurements for this experiment, where the severity of the deterioration effects can easily be adapted to simulate different dynamic scenarios. Therefore, the following analysis evaluates the impact of the

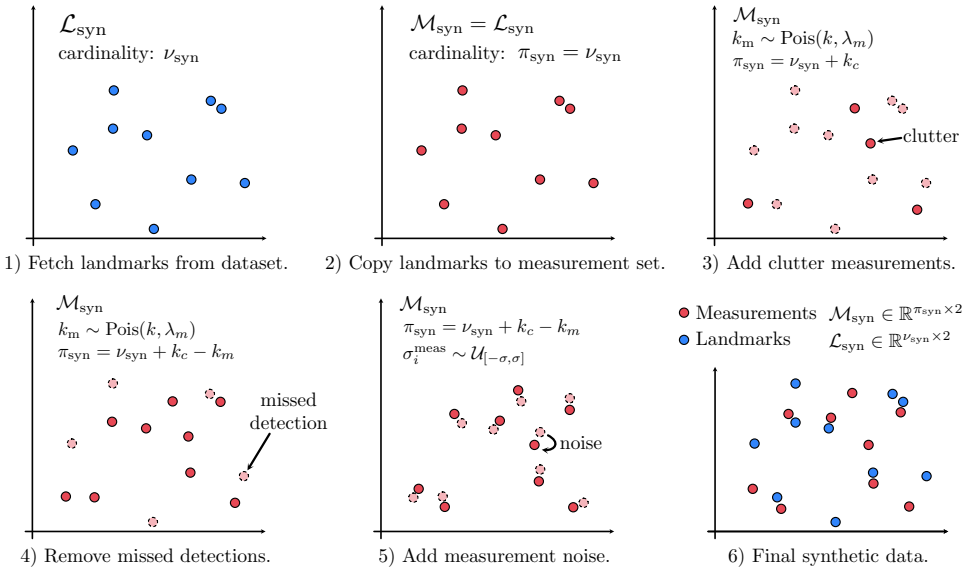


Figure 7.2: Overview of the synthetic measurement generation process. The additional clutter measurements in step three are placed according the the bivariate normal distribution shown in Figure 7.1. After applying the clutter measurements, missed detections and measurement noise, a realistic measurement sample is created.

parameters λ_m, λ_c, and $\boldsymbol{\sigma}$ on the localization accuracy. In total, four separate experiments are conducted: one for each of the impairments and a final experiment where all effects are combined. Additionally, each experiment is repeated several times, whereas the evaluated parameter is increased consecutively. This allows for a detailed and structured analysis, where the corresponding localization accuracy is shown in Figure 7.3 for the three presented DeepLocalization approaches. Here, the root mean square error (2.76) is visualized for each experiment. Apparently, each DeepLocalization iteration improves the overall robustness against noise, confirming the claim that concentrating on local regions and spatial relationships between the measurements and the landmarks helps the network improve the measurement-to-landmark association process and the overall pose prediction performance. Moreover, DeepLocalization2 and the attention-based DeepLocalization method achieve a more precise localization accuracy in an ideal setting and tend to be less prone to increased dynamic effects like clutter measurements and missed detections. Notable, in some cases, the improved network architectures achieve better accuracy with high deterioration effects applied compared to the original DeepLocalization architecture in the ideal case, cf. missed detections and clutter in Figure 7.3. Finally, the

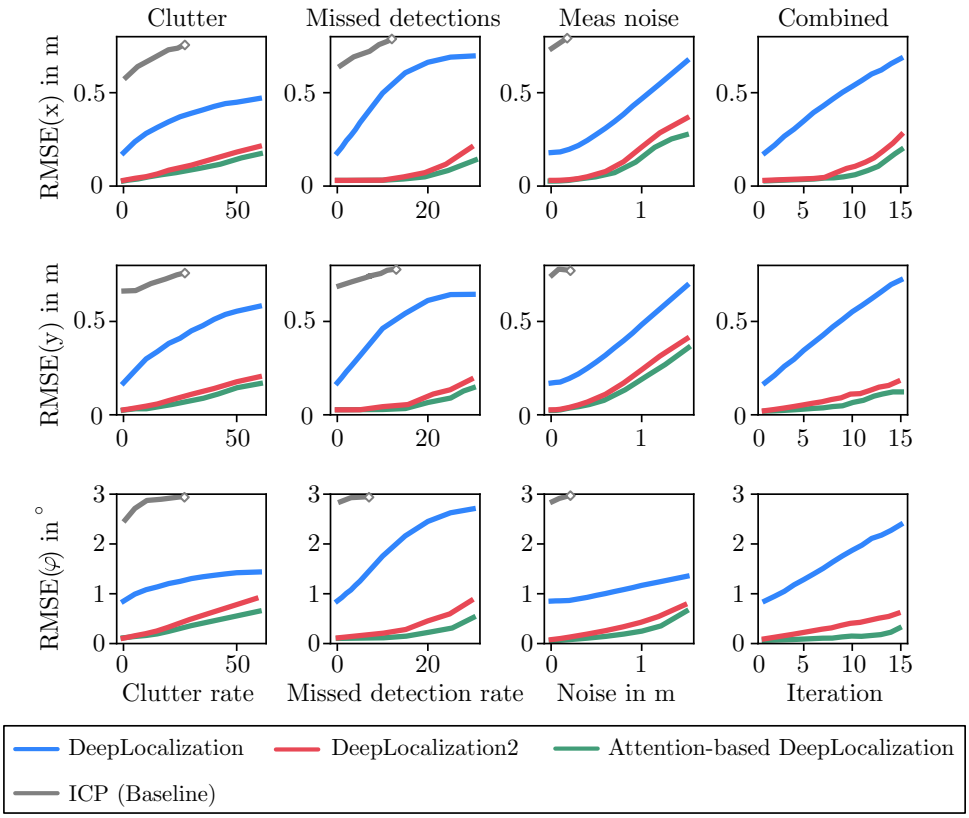


Figure 7.3: Results of the synthetic experiments for the three proposed DeepLocalization approaches. Each network iteration improves the overall robustness against the deterioration effects. Note, that the baseline evaluation (ICP) diverged after several iterations, which is illustrated as ⋄.

strengths of the attention-based DeepLocalization approach are demonstrated in the experiment where all impairments are combined. Increasing λ_c, λ_m, and $\boldsymbol{\sigma}$ has a negligible effect on the network performance when compared to the previous methods, whereas the overall localization accuracy can be improved up to 68 %. For comparison, the ICP is also evaluated as a baseline, however, it is apparent that the algorithm diverges after several iterations as the associations between the synthetic measurements and landmarks are too dissimilar with increased impairments. Consequently, the experiment where all three deterioration effects are combined is omitted as no satisfying solution could be found.

7.1.2 Real-world Experiments

After evaluating the proposed methods on artificial input data, the next step involves experiments on real-world samples with measurements obtained from the multi-model sensor setup introduced in Chapter 5. For that, multiple experiments are conducted to explore different aspects that are important for the task of vehicle self-localization. This includes evaluating the localization accuracy depending on different noise parameters of the initial GPS measurement, the generalization capabilities in unseen environments, and the performance of the different inference approaches introduced in Section 6.3.1. Moreover, further evaluations are performed on a custom and simplified version of the popular KITTI dataset [GLU12] to show that the presented approaches can be adapted according to the availability of sensor data. For this, the camera images are used to extract static objects using the MSER algorithm, as explained in Chapter 5. Since this experiment only includes landmarks from one sensor modality, the localization accuracy is expected to be lower compared to a multi-modal dataset. Finally, the results of the real-world experiments are compared against state-of-the-art approaches related to the task of self-localization. For that, the prevalent ICP algorithm [BM92], that registers two point clouds iteratively, is used as a baseline. Then, traditional filter-based localization methods are evaluated on the dataset to provide extensive analysis highlighting the advantages of deploying deep learning-based approaches for vehicle self-localization.

Similar to the previous section, all provided results denote the root mean square error (RMSE) unless specified otherwise. First, the GPS-based inference method is evaluated on the dataset from Chapter 5, where the accuracy of all three DeepLocalization approaches are compared against each other. As mentioned in Section 6.3 this process always assumes a standard initial GPS measurement, which is then used as the starting pose estimate for the network to correct. In order to systematically evaluate the performance under various signal strength conditions and effects, three experiments are conducted with different GPS noise parameters. Following the ideas of the training strategy from Section 6.3.1, noise is added to each of the pose components, x, y and φ, from a uniform distribution $\mathcal{U}_{[-\sigma,\sigma]}$, cf. Equation (2.26), where the lower and upper bounds are given by σ_x, σ_y and σ_φ, respectively. For example, the noise that is added to the x position of the vehicle is sampled from $\mathcal{U}_{[-\sigma_x,\sigma_x]}$. The GPS noise parameters are set to resemble realistic measurement uncertainties of commercially available GPS sensors [WEK05].

In this thesis, three different scenarios are considered, that each denote different GPS signal qualities, for example, due to atmosphere disturbances or multi-path effects. In particular, the noise ranges are defined as follows: 1) $\sigma_x = \sigma_y = 2\,\mathrm{m}$, $\sigma_\varphi = 10°$, 2) $\sigma_x = \sigma_y = 1\,\mathrm{m}$, $\sigma_\varphi = 4°$ and 3) $\sigma_x = \sigma_y = 0.5\,\mathrm{m}$, $\sigma_\varphi = 2°$.

Table 7.1: RMSE results of real-world experiments using the GPS-based inference method.

	Ulm-Lehr			Adapted KITTI [GLU12]		
Method	x	y	φ	x	y	φ
1) $\sigma_x = \sigma_y = 2\,\text{m}$, $\sigma_\varphi = 10°$						
DeepLocalization	0.77 m	0.70 m	2.5°	0.84 m	0.82 m	3.1°
DeepLocalization2	0.53 m	0.58 m	1.7°	0.61 m	0.64 m	2.3°
Attention-based	0.41 m	0.51 m	1.7°	0.49 m	0.55 m	2.1°
2) $\sigma_x = \sigma_y = 1\,\text{m}$, $\sigma_\varphi = 4°$						
DeepLocalization	0.44 m	0.37 m	1.3°	0.48 m	0.44 m	2.3°
DeepLocalization2	0.26 m	0.30 m	0.9°	0.33 m	0.36 m	1.9°
Attention-based	0.20 m	0.23 m	0.9°	0.29 m	0.32 m	1.7°
3) $\sigma_x = \sigma_y = 0.5\,\text{m}$, $\sigma_\varphi = 2°$						
DeepLocalization	0.27 m	0.23 m	0.8°	0.35 m	0.33 m	1.4°
DeepLocalization2	0.18 m	0.20 m	0.6°	0.20 m	0.24 m	1.2°
Attention-based	0.17 m	0.18 m	0.6°	0.21 m	0.24 m	1.1°

The results are given in Table 7.1. Here, it is apparent that each network improvement increases the overall localization accuracy. In general, DeepLocalization2 achieves a performance increase of about 40 % and the attention-based model variation a performance increase of about 54 % compared to the original model architecture. Therefore, the model adaptions that were employed to more realistically model the steps of the self-localization task and consider local spatial relationships and geometric dependencies between measurements and landmarks prove to be an efficient enhancement to the DeepLocalization network architecture. In all three experiments, the attention-based localization approach achieves sufficient accuracy results for the safe and reliable operation of automated vehicles in urban and dynamic scenarios, as defined in Section 2.3. When the GPS signal is disturbed, i.e., the initial pose estimate is subject to a high degree of uncertainty ($\sigma_x = \sigma_y = 2\,\text{m}$, $\sigma_\varphi = 10°$), the network achieves a mean localization accuracy of about 40 cm to 50 cm and about 2°. Accordingly, when the initial GPS measurement is more accurate ($\sigma_x = \sigma_y = 0.5\,\text{m}$, $\sigma_\varphi = 2°$), the localization accuracy increases to about 20 cm and below 1°. As expected, the results on the adapted KITTI odometry dataset, which only consists of camera-based landmarks and measurements, slightly deviate. The overall localization accuracy is lower compared to a multi-modal sensor setup. However, this experiment is conducted to highlight the capability of the presented approaches to adapt to different sensor configurations and system requirements. Consequently, the network performance is subject to the availability of sensor measurements and the quality of the environment perception.

Table 7.2: RMSE results of real-world experiments using the filter-based inference method. −: Result not available.

	Ulm-Lehr			KITTI [GLU12]		
Method	x	y	φ	x	y	φ
ICP [BM92]	1.17 m	1.46 m	4.9°	1.91 m	1.84 m	6.1°
EKF + GPS	0.59 m	0.54 m	6.5°	1.07 m	1.12 m	6.5°
FastSLAM+GPS [MTKW02]	0.34 m	0.32 m	2.1°	0.73 m	0.77 m	2.9°
PHDSlam+GPS [MVA10]	0.30 m	0.32 m	1.7°	–	–	–
RFS-MCL+GPS [Stü18]	0.28 m	0.26 m	1.9°	–	–	–
DeepLocalization + EKF	0.57 m	0.34 m	1.3°	–	–	–
DeepLocalization2 + EKF	0.49 m	0.33 m	1.0°	–	–	–
Attn-based + EKF	0.39 m	0.28 m	1.0°	–	–	–
With additional GPS measurements						
DeepLocalization + EKF	0.27 m	0.24 m	0.8°	0.45 m	0.44 m	2.1°
DeepLocalization2 + EKF	0.21 m	0.20 m	0.7°	0.38 m	0.40 m	1.6°
Attn-based + EKF	0.18 m	0.16 m	0.6°	0.31 m	0.25 m	1.4°

Next, the filter-based inference method is evaluated and compared against popular traditional localization methods in Table 7.2. For that, the measurement noise matrix of the EKF is heuristically set to $R = \text{diag}(2, 2, 0.5)$. The process noise matrix can directly be derived from the CTRV motion model, considering that acceleration and yaw rate changes are treated as Gaussian distributed white noise. For the comparison, the related approaches are introduced in the following. However, a detailed and fundamentally sound explanation of each method is out of scope for this thesis. The reader is referred to the corresponding publication for further details. As explained above, the iterative closes point algorithm (ICP) serves as a baseline for the evaluation process. The algorithm takes two point clouds as input, the source and the target, and transforms the source to best match the target point cloud. This is done by minimizing an error metric, for example, the euclidean distances between source and target points, whereas the optimization process is performed iteratively [BM92]. Due to its simplicity, the ICP is a widely used algorithm in the field of robotics and computer vision. As a second baseline evaluation, a standalone Extended Kalman-Filter (EKF) is employed that uses the same CTRV motion model as the filter-based inference for the DeepLocalization approaches. However, the EKF only receives the inaccurate and noisy GPS measurements as input. Therefore, the actual benefit of employing the deep neural networks in combination with a temporal filtering approach can be highlighted. Then, two popular simultaneously localization and mapping approaches (SLAM) are evaluated on the localization dataset: the FastSLAM [MTKW02] and the PHDSlam [MVA10] algorithm. Here,

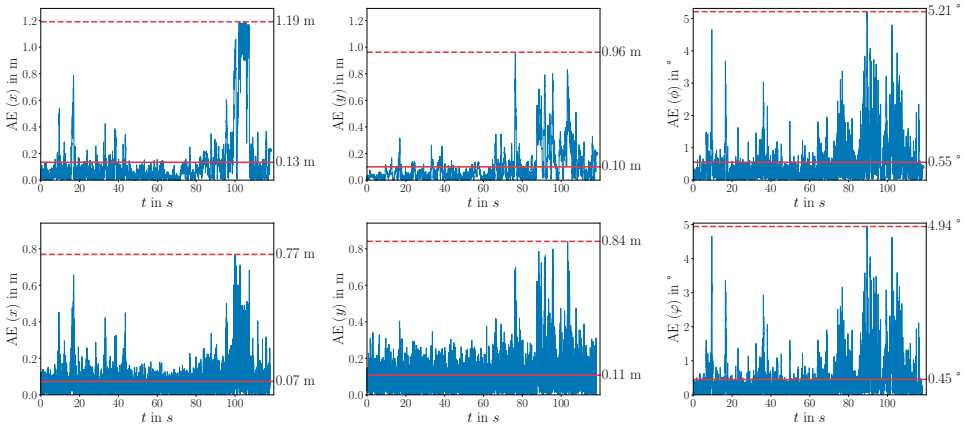


Figure 7.4: Here, the absolute localization errors (AE) on the test track in Ulm-Lehr, the RMSE (—) and the maximum error (- - -) for the filter-based inference of DeepLocalization2 (top) and the attention-based DeepLocalization (bottom) are shown.

it is important to note that, usually, SLAM approaches include the map creation process as well. However, since a fair comparison is intended, the algorithms are adapted to include the localization steps only. The mapping algorithm is discarded and the landmarks from the digital map are made available. Finally, the Random Finite Set Monte-Carlo localization approach (RFS-MCL), that was introduced by Stübler [Stü18] at the Institute of Measurement, Control and Microtechnology at Ulm University, is evaluated and compared to the proposed methods in this thesis. The filter-based DeepLocalization approaches rely on one noisy GPS measurement for initialization. Then, the network predicts a pose correction that is applied to the initial pose estimate and fed to the EKF with an underlying CTRV motion model to output a temporally filtered pose result. This process is explained in detail in Section 6.3. Furthermore, since standard GPS measurements from commercially available GNSS sensors are usually available at no cost, further experiments are conducted where the GPS measurements are used as additional inputs for the EKF.

The results of all real-world filter-based experiments are shown in Table 7.2. Apparently, the baselines perform poorly on the dataset as the measurement samples are subject to noise and the aforementioned deterioration effects that negatively impact the association and optimization algorithms. In contrast, the three traditional filter-based approaches, that additionally use noisy GPS measurements, show satisfying results in the range of 26 cm to 34 cm. Compared to the filter-based inference of the proposed DeepLocalization architectures that operate without additional GPS signals, the traditional methods achieve slightly better accuracy results. However,

Table 7.3: Network complexity study of the DeepLocalization network architectures.

Method	Inference time	Number of parameters
DeepLocalization	1.7 ms	1.8 million
DeepLocalization2	11 ms	1.6 million
Attention-based	26 ms	8.4 million

when GPS measurements are supplied to the EKF similar to the related approaches in Table 7.2 for a fair comparison, e.g., FastSLAM + GPS, PHDSlam + GPS and RFS-MCL + GPS, the deep learning approaches outperform the related work and show superior accuracy, with a position accuracy of 16 cm to 27 cm and an orientation accuracy of below 1°. This supports the claim that the data-based training and optimization process is more robust to noise than handcrafted algorithms. Moreover, as expected, each improvement of the network architecture further increases the overall localization accuracy.

Finally, two exemplary evaluation drives of about 2 min on the test track in Ulm-Lehr are shown in Figure 7.4. Here, the absolute localization error is plotted over the drive duration for each pose component. Additionally, the respective RMSE and maximum error values are marked. It is apparent that both DeepLocalization2 and the attention-based localization approach achieve satisfying results, where the absolute localization error is below 20 cm for most of the examined scenarios. Moreover, only a few outliers are present, demonstrating promising results for safe and reliable deployment in diverse environments.

7.1.3 Network Design and Ablation Studies

In this section, the DeepLocalization network design choices are further examined. First, the model complexity, the network size, and the inference times are compared. Then, network-specific analyses are performed, such as different radius choices for the DeepLocalization2 method or the generalization capabilities in unseen environments.

Model complexity

In order to evaluate the complexity of the proposed networks, the total number of learnable parameters, which provides information about the size and memory

Table 7.4: Evaluation of the influence of different radius search parameters with respect to the localization accuracy.

Radius	x	y	φ
$r = 2\,\mathrm{m}$	0.30 m	0.32 m	1.0°
$r = 5\,\mathrm{m}$	0.29 m	0.34 m	1.3°
$r = 10\,\mathrm{m}$	0.36 m	0.36 m	1.3°
$r = \infty$	0.40 m	0.39 m	1.5°
variable r_d, cf. Equation (6.18)	0.26 m	0.30 m	0.9°

requirements, and the inference time is shown in Table 7.3. DeepLocalization and DeepLocalization2 have a comparable number of total trainable parameters; however, the inference time is about ten times as high for DeepLocalization2. This is mainly due to the radius search, which can be further optimized by using concurrency capabilities of the implementation framework. In contrast, the attention-based localization approach has about 8.4 million parameters with an inference time of 26 ms. The increase in model size and inference time stems from the attention operation, which has $\mathcal{O}(n^2)$ complexity [VSP+17]. However, the results are still within the realtime requirement which is below 100 ms [EBD21a; EHH+19]. Moreover, by employing optimized attention operations, e.g., the linear attention transformer from Section 6.4, the complexity and inference time can be improved further. In comparison, the ICP algorithm takes about 7.6 ms and the RFS-MCL about 29 ms per processing step [EHH+19]. In summary, the model complexity and inference time of all presented approaches are sufficient and meet the required real-time capabilities.

Radius Search

In the following, different choices for the radius search algorithm introduced in Section 6.3.2 are outlined in Table 7.4. First, three constant values for the radius are evaluated, i.e., 2 m, 5 m and 10 m. Then the radius is set to $r = \infty$ to consider the case that all landmarks are possible associations for each measurement, similar to the original DeepLocalization method. Finally, the variable radius search algorithm from Equation (6.18) is presented. The evaluation proves that adapting the radius according to the distance of each measurement to the origin of the vehicle's frame improves the overall localization accuracy by minimizing possible erroneous associations.

Table 7.5: Evaluation of generalization capabilities using the different train and test splits. RMSE values are reported.

① / ② split	x	y	φ
DeepLocalization2	0.33 m	0.36 m	2.4°
Attention-based	0.29 m	0.33 m	1.9°

Generalization Capabilites

Finally, DeepLocalization2 and the attention-based localization approach are evaluated on the train and test splits visualized in Figure 5.2. Previously, the training and the evaluation were performed in the same spatial environments, whereas different recordings were used to obtain different input data distributions. In order to make a statement about the generalization capabilities of the two network architectures, the models are trained and evaluated on different spatial regions, ① and ② , in Figure 5.2 respectively. The results are shown in Table 7.5, where the filter-based inference is evaluated, similar to the results presented in Table 7.2. As expected, the performance decreases when the network is evaluated in unseen environments. The localization accuracy for DeepLocalization2 drops to 33 cm and 36 cm and for the attention-based localization method to 29 cm and 33 cm for the x and y position. However, these results show promising generalization capabilities, which allow for widespread deployment of the presented localization approaches without the need to collect training data from all areas of application, which in turn reduces the operational costs.

7.2 Point Transformer Localization

This section evaluates the Point Transformer Localization approach in several real-world experiments. The evaluation pipeline is identical as introduced for the DeepLocalization method; that is, the GPS and the filter-based inference are examined in conjunction with the Point Transformer Localization network. Following the processes explained in Section 7.1.2, Point Transformer Localization is first evaluated using the GPS-based inference method, which is similar to how the network is trained. Then, the filter-based inference is evaluated, where an EKF with a CTRV motion model is deployed to obtain a smoothed estimate of the vehicle's pose. Finally, the novel pose uncertainty loss (6.37) is analyzed and compared to the original geometric loss function (6.13). In terms of comparability, Point Transformer Localization can not fairly be benchmarked against the DeepLocalization approach because the input

Table 7.6: Results of Point Transformer Localization on the high-density lidar dataset.

	RMSE		
	x	y	φ
GPS-based inference with geometric loss			
$\sigma_x = \sigma_y = 0.5\,\text{m}, \sigma_\varphi = 1°$	0.20 m	0.14 m	0.58°
$\sigma_x = \sigma_y = 1.0\,\text{m}, \sigma_\varphi = 5°$	0.35 m	0.28 m	1.58°
$\sigma_x = \sigma_y = 2.0\,\text{m}, \sigma_\varphi = 10°$	0.60 m	0.48 m	2.91°
GPS-based inference with pose uncertainty loss			
$\sigma_x = \sigma_y = 0.5\,\text{m}, \sigma_\varphi = 1°$	0.17 m	0.10 m	0.49°
$\sigma_x = \sigma_y = 1.0\,\text{m}, \sigma_\varphi = 5°$	0.30 m	0.22 m	1.24°
$\sigma_x = \sigma_y = 2.0\,\text{m}, \sigma_\varphi = 10°$	0.54 m	0.40 m	2.47°
Filter-based inference with pose uncertainty loss			
with additional GPS measurements	0.17 m	0.13 m	0.55°
without additional GPS measurements	0.23 m	0.30 m	0.85°

data distribution and input modalities are too different. Furthermore, related approaches from the literature include the L3Net [LZW+19] and PointLoc [WWZ+21] methods. Unfortunately, the implementation and the localization dataset of the L3Net publication are not available to the public, which makes a comparison not possible. Moreover, the PointLoc approach is not scene-agnostic, i.e., the network overfits to a specific dataset and can only estimate a global pose inside the scene of the dataset. This is in contrast to the proposed methods in this thesis, which allow for a widespread deployment due to the presented formulation of the local localization task that uses pose offsets instead of global coordinates.

The results of the GPS-based inference and the filter-based inference are presented in Table 7.6. Additionally, the impact of the geometric loss function and the pose uncertainty loss on the localization accuracy is evaluated. The noise parameters for the evaluation of the GPS-based inference are taken from the previous experiments using the DeepLocalization architecture and represent common uncertainty values under different quality assumptions of commercially available GPS sensors, e.g., [WEK05]. It is apparent, that the Point Transformer Localization approach achieves satisfying results on the high-density lidar dataset, where the localization accuracy of the first two experiments, i.e., $\sigma_x = \sigma_y = 0.5\,\text{m}, \sigma_\varphi = 1°$ and $\sigma_x = \sigma_y = 1.0\,\text{m}, \sigma_\varphi = 5°$, meet the localization requirements in urban scenarios, as stated in Section 2.3. Employing the novel pose uncertainty loss from Equation (6.33) leads to improved accuracy in all three experiments. Moreover, the training process converged notably faster since the geometric loss first requires an optimization of the learned

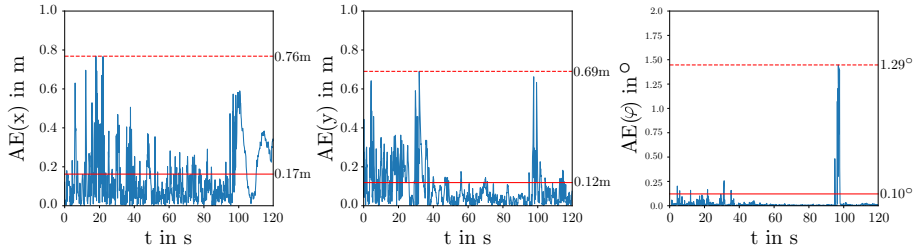


Figure 7.5: Results of a 2 min test drive using the filter-based inference approach and the Point Transformer Localization, where the absolute localization errors (AE), the RMSE (—) and the maximum error (- - -) are shown.

weighting factors before the optimizer focuses on minimizing the actual pose loss function, cf. Equation (6.13). As previously explained, the pose uncertainty loss directly optimizes for the position and orientation distribution with the additional benefit of providing aleatoric uncertainties. Finally, the filter-based inference that is similar to the DeepLocalization method, achieves a localization accuracy of about 0.13 m to 0.17 m when used with additional GPS measurements as input and 0.23 m to 0.30 m without GPS measurements. When the performance of the GPS-based inference is compared to DeepLocalization, the Point Transformer Localization approach achieves slightly worse results. However, the filter-based inference improves overall localization accuracy, indicating that using semantic and geometric shape information benefits the robustness since the filter-based localization task is usually more difficult, as localization errors are propagated over time. The results of a 2 min test drive using the Point Transformer Localization is visualized in Figure 7.5. Moreover, Figure 7.6 illustrates the localization process in various scenes of the Point Transformer Localization dataset. In the depicted scenarios, the shift and rotation of the measurements are apparent, which is caused by the inaccurate initial localization estimate. However, after processing the scan and the map with the proposed Point Transformer Localization method, the point clouds are better aligned, as shown in the highlighted areas.

Finally, the network complexity is evaluated. Overall, the network size of about 24.3 MB with 5.2 million parameters is comparable to the DeepLocalization methods from Table 7.3. The inference time of about 65 ms is still within the bounds of the real-time requirements, i.e., below 100 ms. However, it is possible to further decrease the inference time by optimizing the implementation. For example, the point clouds of every class are processed sequentially. Moreover, multiple kNN algorithms are performed on large-scale point clouds. In both cases, an improved parallel and GPU-based implementation promises to reduce processing times further.

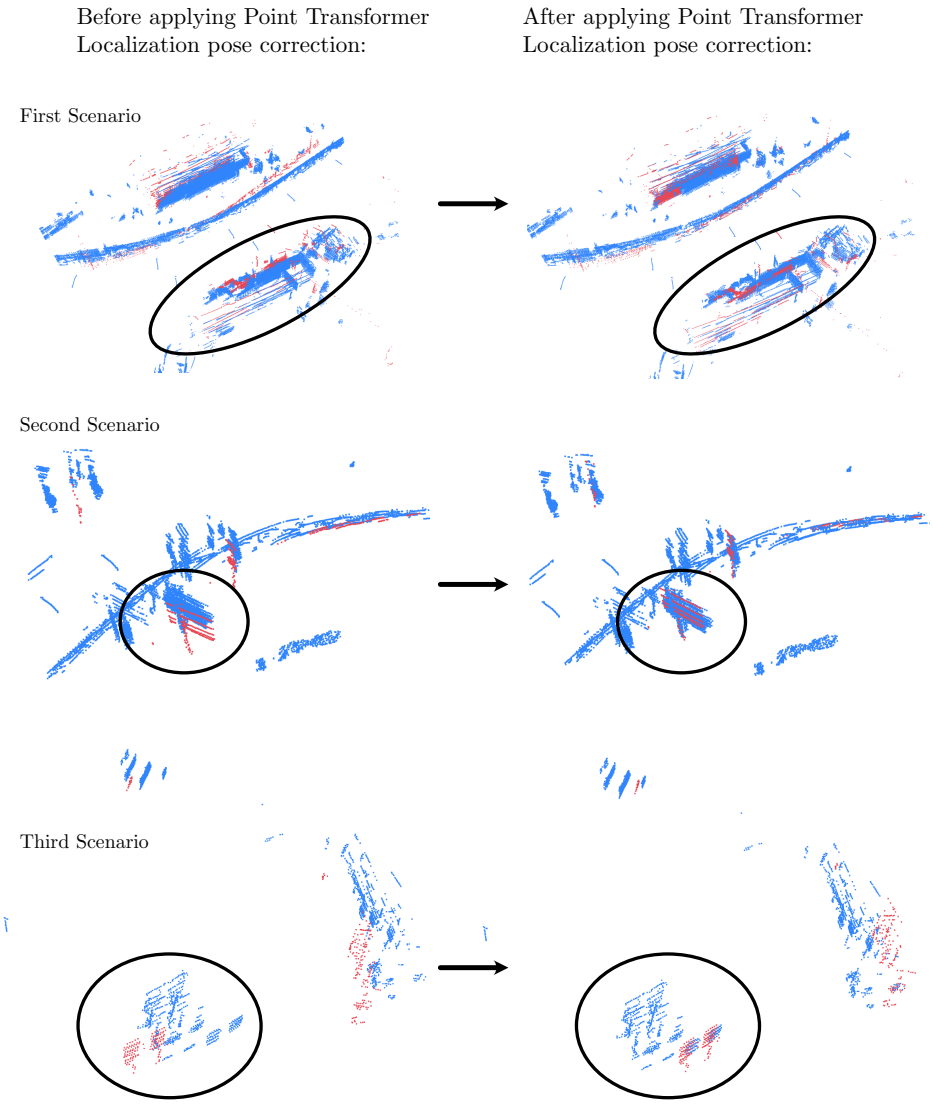


Figure 7.6: Three different scenes where the application of the learned pose correction is shown. On the left-hand side, the map point cloud (blue) is transformed into the vehicle frame using the inaccurate initial GPS measurement. Here, it is apparent that there is an offset between the map and the scan point cloud (red). The network learns to correct the translation and rotation offsets, which is shown on the right-hand side. Circles denote areas where the improvement is most evident.

Chapter 8

Summary and Outlook

8.1 Summary

This thesis deals with the challenging task of 2D and 3D point cloud processing with deep neural networks and applications in vehicle self-localization. For that, two new approaches are presented: The first main contribution proposes Point Transformer, a novel architecture that adapts the popular attention mechanism for the task of 3D point cloud processing. The attention mechanism itself is permutation-invariant, i.e., it does not depend on the ordering of the input set; however, the output is still unordered. Therefore, the SortNet module is presented that produces an ordered set by learning a score used to select important key points from the shape of the object. Then, latent local features are produced and aggregated that are related to the global signature of the object. Depending on the desired task, different output heads are employed. The attention mechanism, as well as the related Transformer network architecture, are well-suited for the problem of processing 3D point clouds, as it relates each element of the point cloud to all other elements. Applied to the proposed Point Transformer architecture, this allows finding local and geometric relationships within the point cloud, leveraging the spatial properties of three-dimensional shapes. Additionally, it is also possible to attend one set of points to a different point set which is an essential characteristic if the method is adapted for vehicle self-localization, where two independent input sets, e.g., measurements and landmarks, are registered to find the pose. Finally, a comprehensive evaluation of Point Transformer is performed that demonstrates competitive results on popular 3D computer vision benchmarks. Furthermore, an extensive network ablation study is conducted to investigate the strengths of the proposed method.

Then, the second main contribution includes the successful introduction of two novel deep learning-based localization approaches that tackle the prevalent challenges and difficulties of point-based vehicle self-localization with digital maps. As mentioned

multiple times, the localization process is subject to different types of noise, including clutter or missed detections caused by the dynamic environment or faulty sensors and measurement noise of commercially available ranging sensors. Furthermore, the representation of point clouds implies unordered and unstructured input data that the network has to cope with. The presented approaches solve this problem by relying on the widely popular set-pooling methods (DeepLocalization) and the ideas of the Point Transformer architecture (Point Transformer Localization) to induce permutation-invariance. Then, data-based approaches are naturally more robust to noise since the training data distribution closely resembles the real-world scenarios in the optimal case. Therefore, the generalization capabilities are one of the key performance measures besides the localization accuracy. DeepLocalization and Point Transformer Localization are specifically designed to learn the most probable measurement-to-landmark association and consider and deal with noisy data. Two contributions ensure that the proposed method is easily deployable in unseen environments: 1) The design of the network and the corresponding formulation of the local localization task that is not dependent on the scenes of the dataset. And 2) the novel training strategy that can generate an infinite amount of labeled training samples online during training. The network is conditioned to infer a pose offset relative to either a previous pose (filter-based inference) or a noisy GPS measurement (GPS-based inference) instead of directly inferring the global pose. That way, the training scene is not explicitly encoded in the weights of the network, allowing the network to learn a relative offset from the perceived environment, i.e., the sensor measurements and the landmarks from the high-precision digital map. Therefore, a large-scale operation can be achieved. Finally, the proposed localization methods are evaluated in various real-world scenarios on different datasets. The results demonstrate state-of-the-art performance that meet the required localization accuracy in dynamic environments. Additionally, a synthetic ablation study is conducted to investigate the robustness of the DeepLocalization approaches against the introduced deterioration effects in real automated driving applications.

8.2 Outlook

Several improvements can be incorporated into the proposed localization framework in the future to improve the overall localization accuracy further. In this thesis, two different approaches are presented, namely DeepLocalization and Point Transformer Localization, that operate on sparse 2D and dense 3D data, respectively. As explained in the previous chapters, DeepLocalization performs best when applied to sparse 2D measurements, whereas Point Transformer Localization is designed to work on dense lidar point clouds. Furthermore, different inference approaches are presented. One possible improvement includes deploying specific network architectures for different

input modalities instead of employing one network to process multi-modal measurements. One solution could be the usage of the Point Transformer Localization for lidar point clouds exclusively, and a second network, e.g., DeepLocalization, for sparse radar measurements. Then, an Extended Kalman-Filter (EKF) could be employed that receives alternating pose corrections from both network architectures to estimate the global pose. Therefore, the strengths of each network can be leveraged individually with the downside of increased computational requirements.

To counteract the overall memory consumption and reduce the overall inference time, optimized network architectures, such as the Linear Attention Transformer that is deployed in the Point Transformer Localization approach, can be incorporated in the DeepLocalization architecture as well, especially the attention-based localization network from Section 6.3.3.

Finally, in real-world applications, i.e., vehicle self-localization for automated driving, additional data is usually provided through the CAN (Controller Area Network) or FlexRay interface. This includes the vehicle's speed obtained from the wheel encoders or the steering angle. The filter-based inference method proposed in Chapter 6 uses a motion model, such as the CTRV motion model, to predict the vehicle's state at the next time step. Here, it is possible to incorporate the ego-motion data in the prediction step to achieve a more accurate pose prediction that is additionally decoupled from the network output. Thus, in the case of erroneous pose estimates, the filtered output is not as severely affected compared to relying solely on the prediction of the network.

Appendix A

Point Transformer

A.1 Implementation Details

In the following, specific implementation details of the Point Transformer architecture are outlined. The hyperparameters and model parameters that were used to achieve the reported results for the classification and segmentation task are given in Table A.1. Moreover, the experiments were performed on an Nvidia GeForce 1080Ti graphics card, 64 GB of RAM and a Intel® Xeon® CPU E5-1660 v4 @ 3.20 GHz. Furthermore, all presented methods are implemented in the popular PyTorch [PGM+19] framework that provides all major deep learning methods as well as the Transformer architecture.

Table A.1: Hyperparameters of Point Transformer for the classification and the part segmentation task.

Parameter	Explanation	Classification	Part Segmentation
General Hyperparameters			
B	Batch size	11	8
N	Number of input points	1024	1024
D	Input dimension	6	6
l_r	Learning rate	0.001	0.005
w_d	Weight decay	$1e{-}6$	0.0001
d_m	Latent feature dimension	512	512
-	Weight initializer	Kaiming Normal [HZRS15]	
Local Feature Generation			
-	rFF feature dimension	$(64, 128, 512)$	$(64, 128, 512)$
-	Dropout	0.4	0.3
n_{head}	Number of local attention heads	8	8
n_{layers}	Number of local attention layers	1	1
SortNet			
M	Number of SortNets	4	10
K	Number of top-k selections	64	16
-	rFF feature dimension	$(128, 256, 512)$	$(64, 128, 512)$
-	Dropout	0.4	0.3
Global Feature Generation			
N'	Reduced point set	128	64
-	Segmentation rFF	-	$(64, 128, 256)$
$d_m^{''}$	Segmentation feature dimension	-	256
n_{head}	Number of local attention heads	8	8
n_{layers}	Number of local attention layers	1	1
-	Dropout	0.4	0.3
Local-Global Attention			
$d_m^{'}$	Reduced feature dimension	64	256
n_{head}	Number of local attention heads	8	8
n_{layers}	Number of local attention layers	4	4
Classification Head			
C	Number of classes	40	
-	Fully connected dimension	$(4096, 1024, 512, 128, 40)$	
-	Dropout	0.4	
Segmentation Head			
C	Number of classes	50	
-	Output rFF	$(256, 128, 50)$	
n_{head}	Number of local attention heads	8	8
n_{layers}	Number of local attention layers	1	1

A.2 Additional Evaluation Results

A.2.1 SortNet

In the first experiment, the shape awareness of SortNet is examined, and the results are shown in Figure A.1. Here, the top-k selections of all SortNet modules from a trained Point Transformer network with $M = 8$ and $K = 32$ are illustrated, whereas each SortNet is depicted in a different color. Additionally, the entire point cloud is given as well. Even though the receptive field of a single SortNet is small with $K = 32$ compared to the input point cloud with $N = 1024$, it focuses on a small region of the shape. However, when all selections are combined, it is apparent that the network learns to cover as much of the shape as possible such that the local feature generation module can attend to each local region that is important for the specific task.

In the second experiment, the top-k selection of a single SortNet is shown for various classes in Figure A.2, where the visualized object is rotated consecutively. The selection is highlighted in a dark color. Here, it is important to note that the network was not trained with rotated input and has only seen the first sample, where no rotation has been applied. The qualitative analysis demonstrates that a single SortNet selects points from the same local region of the object, regardless of rotation. This suggests that the network is robust against rotation and learns shape information, that is used to support the specific task.

Finally, Figure A.3 depicts the top-k selections from multiple SortNet modules for different objects of the same class (chair and table). Similar to the previous experiment, it is apparent that a single SortNet focuses on similar local regions across different objects. For example, the first SortNet attends to the front edge and the legs of the table objects, implicating that the receptive field is aware of the underlying shape.

A.2.2 Part Segmentation

Similar to Section 4.4.2, the predictions from the part segmentation task on the ShapeNet dataset are visualized in Figure A.4 for three additional class selections (airplane, bag, car). Here, the segmentation from the Point Transformer network is compared to the ground-truth, where each class label is shown in a different color.

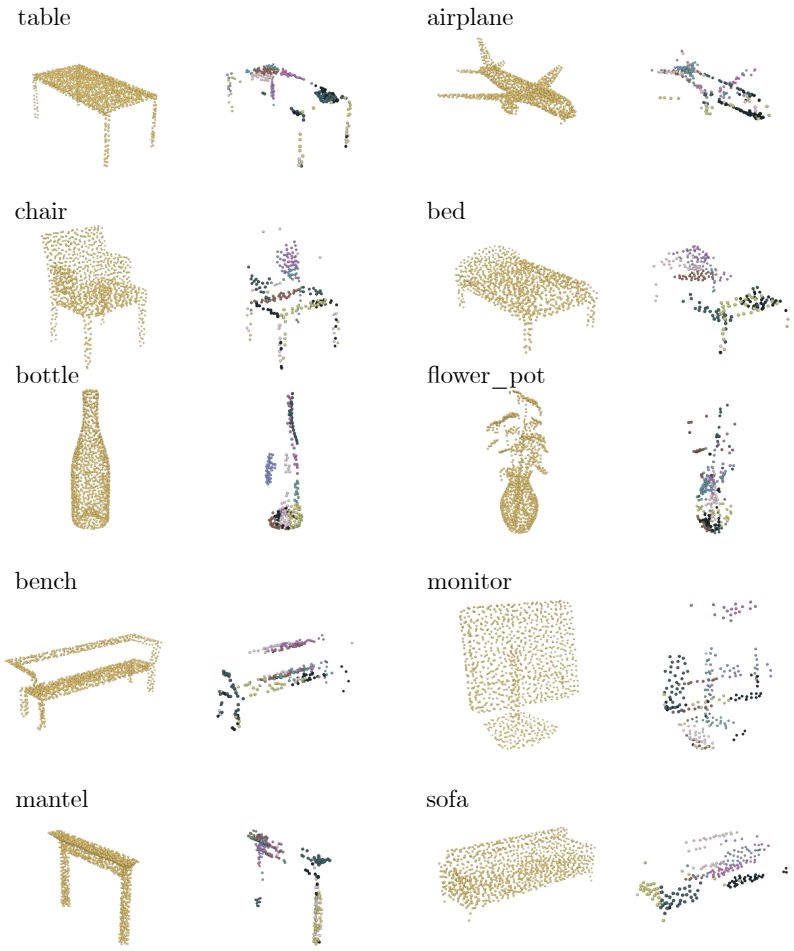


Figure A.1: Here, all selected points from the local feature generation branch (right) are shown in comparison with the complete input point cloud (left). The selected points of each SortNet are shown in the same color. It is clear that every SortNet focuses on different local regions of the object. When the selected points are visualized together, the input point cloud is still recognizable, suggesting that in combination, all SortNets try to retain as much of the underlying shape as possible [EBD21b]. Licensed under CC BY 4.0 - `https://creativecommons.org/licenses/by/4.0/`.

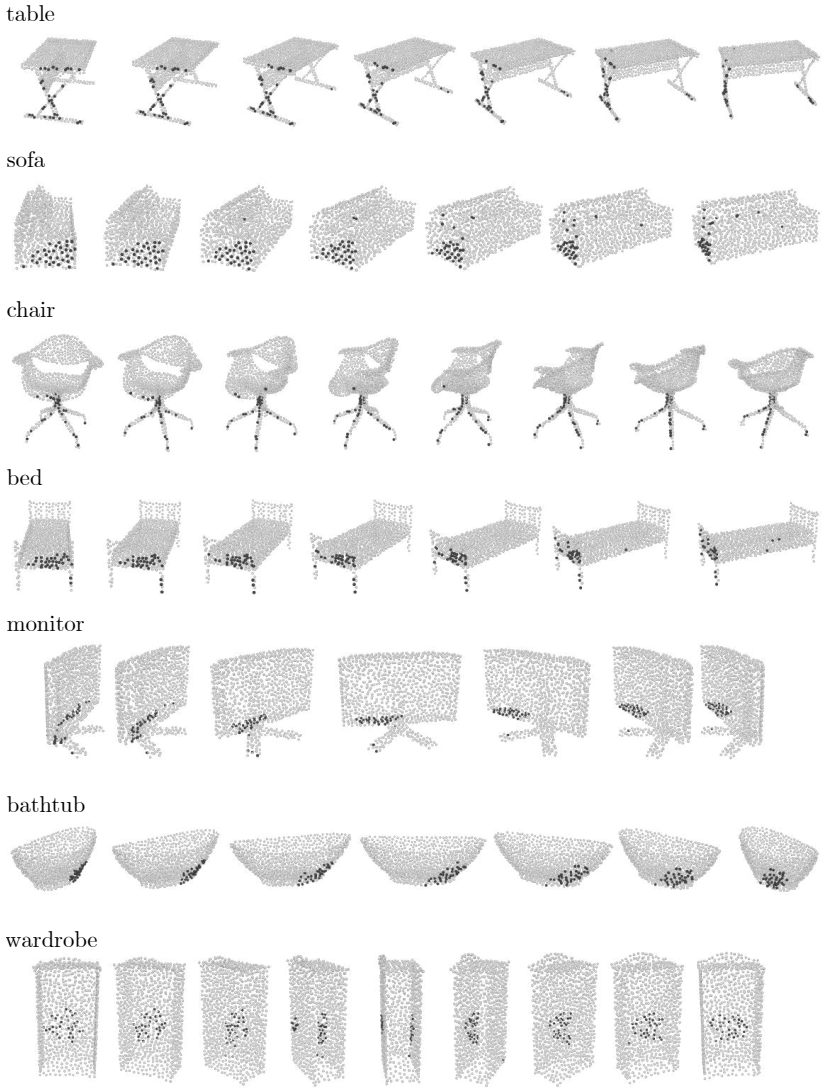


Figure A.2: Influence of input point rotations on the Top-K selection process. • Top-K selection, • Input points. When the input point cloud is rotated, SortNet still focuses on similar local regions of the underlying shape [EBD21b]. Licensed under CC BY 4.0 - `https://creativecommons.org/licenses/by/4.0/`.



Figure A.3: Top-K selections for different chair and table models. • Top-K selection (dark points), • Input points (light points). SortNet selects points from similar local regions when applied to objects of the same category, suggesting that it is aware of the underlying shape [EBD21b]. Licensed under CC BY 4.0 - `https://creativecommons.org/licenses/by/4.0/`.

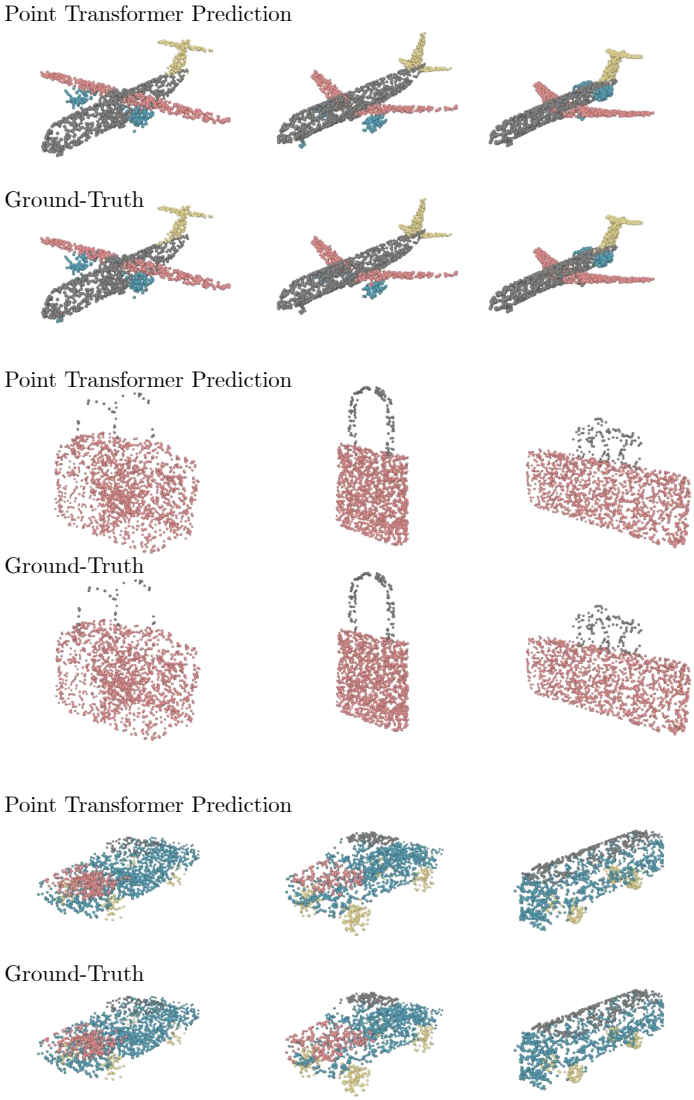


Figure A.4: Additional results for task of part segmentation for different object categories (airplane, bag and car). The prediction of Point Transformer is shown in the first row and the ground-truth in the second row for every object class, respectively [EBD21b]. Licensed under CC BY 4.0 - `https://creativecommons.org/licenses/by/4.0/`.

Appendix B

DeepLocalization and Point Transformer Localization

B.1 Datasets

This section introduces further details about the localization datasets created for this thesis. First, Table B.1 specifically highlights important characteristics and differences of the DeepLocalization dataset and the Point Transformer Localization dataset. Then, the classes of the SemanticKITTI dataset used for the semantic segmentation of lidar scans are given in Table B.2. Since the goal is to extract static and recognizable objects of the environment for the localization task, all classes marked with ✗ are discarded as dynamic objects, whereas the remaining classes are used for both the mapping process and the segmentation for the scans. Here, it is important to note that ground points are removed as well, as experiments have shown that ground points do not benefit the localization accuracy.

Table B.1: Further information and details about the Localization datasets.

	DeepLocalization Dataset		**Point Transformer Localization Dataset**	
	map	measurements	map	scan
Number of Recordings	3	8	1	3
Date of Recording	December 2017	November 2018	September 2020	August 2020
Track Length	5 km		5 km	
Sensor Modalities	Radar, Lidar, Camera		Lidar	
Mapping Algorithm	Bernoulli Filter		Registration, Voxel Downsampling, Outlier removal	
Sensor Preprocessing	DBSCAN, MSER		Semantic Segmentation, Outlier removal	
Number of samples	145.000	-	15.000	
dGPS Pose	✓		✓	
Number of Landmarks	3860	(600 kB)		
Radar	718	(111 kB)		
Lidar	1731	(269 kB)		
Camera	1411	(220 kB)		
Number of Map Points			4.17M	(100 MB)
Building			832K	(20 MB)
Fence			186K	(4.6 MB)
Pole			29K	(832 kB)
Traffic-Sign			29K	(753 kB)
Trunk			150K	(3.8 MB)
Vegetation			2.94M	(70 MB)

Table B.2: Semantic KITTI classes used for the Point Transformer Localization dataset.

	Class Name	**included in Point Transformer Localization Dataset**
ground	road	✗
	sidewalk	✗
	parking	✗
	other ground	✗
	building	✓
	other structure	merged with *building* class
ground	car	✗
	truck	✗
	bicycle	✗
	motorcycle	✗
	other vehicle	✗
nature	vegetation	✓
	trunk	✓
	terrain	✗
human	person	✗
	bicyclist	✗
	motorcyclist	✗
object	fence	✓
	pole	✓
	traffic-sign	✓
	other object	✗

B.2 Filter-based Inference

The following section introduces the discrete Extended Kalman Filter (EKF) equations that are employed in the filter-based inference method. The EKF is an extension to the traditional Kalman Filter that is able to handle non-linear state and observation models [Thr02, Chapter 3.3]. The idea is to linearize the non-linear function around the current estimate. For that, the Jacobians of the state transition and the measurement model are required for every time step in order to obtain the covariances [AM12]. The state-space $\boldsymbol{x}$ for the vehicle's pose is given as follows:

$$\boldsymbol{x} = \begin{bmatrix} x^p \\ y^p \\ v \\ \varphi \\ \dot{\varphi} \end{bmatrix}, \tag{B.1}$$

where x^p and y^p denote the position, v is the velocity, φ the orientation and $\dot{\varphi}$ the angular velocity of the orientation. Furthermore, the measurement vector $\boldsymbol{z}$ is defined as

$$\boldsymbol{z} = \begin{bmatrix} x^p \\ y^p \\ \varphi \end{bmatrix}, \tag{B.2}$$

such that the network's prediction applied to the previous pose estimate $\hat{\boldsymbol{p}} = [\hat{x}, \hat{y}, \hat{\varphi}]^T$ can directly be used as an observation for the filter. In general, the Extended Kalman filter is comprised of two alternating steps: First a prediction step followed by an innovation step to update the predicted state with respect to an observation $\boldsymbol{z}$. In the following, the notation $\hat{\boldsymbol{x}}_{n|m}$ denotes the predicted state $\boldsymbol{x}$ at time n including all observation up to time step m. Furthermore, the non-linear state transition f and measurement model h are given as follows:

$$\boldsymbol{x}_k = f(\boldsymbol{x}_{k-1}, \boldsymbol{u}_{k-1}, \boldsymbol{w}_{k-1}), \tag{B.3a}$$

$$\boldsymbol{z}_k = h(\boldsymbol{x}_{k-1}, \boldsymbol{v}_{k-1}), \tag{B.3b}$$

where $\boldsymbol{u}$ is the control vector and $\boldsymbol{w}$ and $\boldsymbol{v}$ are the process and measurement noise, respectively. Additionally, $\boldsymbol{w}$ and $\boldsymbol{v}$ are assumed to be zero-mean multivariate Gaussian white noise with covariance matrices Q and R. Before the EKF equations can be applied, the filter has to be initialized, that is, the state vector $\boldsymbol{x}_0$ at time step zero needs to be set. For the presented filter-based inference method, this can be done by relying on an inaccurate GPS measurement.

Prediction step: For the prediction step that only uses the previous estimate without a control vector, the state transition model is applied such that

$$\hat{\boldsymbol{x}}_{k|k-1} = f(\hat{\boldsymbol{x}}_{k-1|k-1}) + \boldsymbol{w}. \tag{B.4}$$

As explained, the filter-based inference assumes a constant turn rate and velocity motion model (CTRV) with additive noise. Therefore, f from Equation (B.4) is given as

$$f(\hat{\boldsymbol{x}}_{k-1|k-1}) = \begin{bmatrix} x^p_{k-1|k-1} + \frac{v_{k-1|k-1}}{\dot{\varphi}_{k-1|k-1}} \left(\sin\left(\varphi_{k-1|k-1} + \dot{\varphi}_{k-1|k-1} \cdot T\right) - \sin\left(\varphi_{k-1|k-1}\right)\right) \\ y^p_{k-1|k-1} + \frac{v_{k-1|k-1}}{\dot{\varphi}_{k-1|k-1}} \left(\cos\left(\varphi_{k-1|k-1}\right) - \cos\left(\varphi_{k-1|k-1} + \dot{\varphi}_{k-1|k-1} \cdot T\right)\right) \\ v_{k-1|k-1} \\ \varphi_{k-1|k-1} + \left(\dot{\varphi}_{k-1|k-1} \cdot T\right) \\ \dot{\varphi}_{k-1|k-1} \end{bmatrix}, \tag{B.5}$$

where the sampling time is denoted by T. Here, a problem arises when the yaw rate $\dot{\varphi}$ becomes too small. This causes a singularity, therefore, a threshold is employed in case of $\dot{\varphi} \approx 0$ that reduces the motion model to

$$f^*(\hat{\boldsymbol{x}}_{k-1|k-1}) = \begin{bmatrix} x^p_{k-1|k-1} + v_{k-1|k-1} \cdot \cos\left(\varphi_{k-1|k-1}\right) \cdot T \\ y^p_{k-1|k-1} + v_{k-1|k-1} \cdot \sin\left(\varphi_{k-1|k-1}\right) \cdot T \\ v_{k-1|k-1} \\ \varphi_{k-1|k-1} + \left(\dot{\varphi}_{k-1|k-1} \cdot T\right) \\ \dot{\varphi}_{k-1|k-1} \end{bmatrix}. \tag{B.6}$$

Next, the predicted covariance follows as

$$\hat{P}_{k|k-1} = F_k \hat{P}_{k-1|k-1} F_k^T + Q_k, \tag{B.7}$$

where F_k is the Jacobian of the state transition and Q_k the process noise. Here, F_k can be obtained numerically for the CTRV case, cf. Equation (B.5),

$$F_k = \left.\frac{\partial f}{\partial \boldsymbol{x}}\right|_{\boldsymbol{x}_{k-1|k-1}} = \left.\begin{bmatrix} 1 & 0 & \frac{\sin(\varphi+\dot{\varphi}\cdot T)-\sin(\varphi)}{\dot{\varphi}} & \frac{v\cdot(\cos(\varphi+\dot{\varphi}\cdot T)-\cos(\varphi))}{\dot{\varphi}} & F_k^{(1,5)} \\ 0 & 1 & \frac{\cos(\varphi)-\cos(\varphi+\dot{\varphi}\cdot T)}{\dot{\varphi}} & \frac{v\cdot(-\sin(\varphi)+\sin(\varphi+\dot{\varphi}\cdot T))}{\dot{\varphi}} & F_k^{(2,5)} \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & T \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}\right|_{\boldsymbol{x}_{k-1|k-1}}, \tag{B.8}$$

with

$$F_k^{(1,5)} = -\frac{v \cdot (\sin(\varphi + \dot{\varphi} \cdot T) - \sin(\varphi))}{\dot{\varphi}^2} + \frac{v \cdot \cos(\varphi + \dot{\varphi} \cdot T) \cdot T}{\dot{\varphi}}, \tag{B.9a}$$

$$F_k^{(2,5)} = -\frac{v \cdot (\cos(\varphi) - \cos(\varphi + \dot{\varphi} \cdot T))}{\dot{\varphi}^2} + \frac{v \cdot \sin(\varphi + \dot{\varphi} \cdot T) \cdot T}{\dot{\varphi}}. \tag{B.9b}$$

Consequently, for the case of small yaw rates, cf. Equation (B.6), the Jacobian is given by

$$F_k^* = \left.\frac{\partial f^*}{\partial \boldsymbol{x}}\right|_{\boldsymbol{x}_{k-1|k-1}} = \begin{bmatrix} 1 & 0 & T \cdot \cos\left(\varphi_{k-1|k-1}\right) & -v_{k-1|k-1} \cdot T \cdot \sin\left(\varphi_{k-1|k-1}\right) & 0 \\ 0 & 1 & T \cdot \sin\left(\varphi_{k-1|k-1}\right) & v_{k-1|k-1} \cdot T \cdot \cos\left(\varphi_{k-1|k-1}\right) & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & T \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}. \tag{B.10}$$

In order to obtain the process covariance matrix Q_k, several assumptions are required. First, the constant turn rate and velocity motion model is considered, where the acceleration and the yaw rate changes are assumed to be zero-mean, uncorrelated Gaussian white noise. Here, the variance of the acceleration and the yaw rate changes are given as σ_a^2 and $\sigma_{\ddot{\varphi}}^2$, respectively [Die21]. Considering the simplified motion model (B.6), the noise gains are given as

$$\Gamma_a = \begin{bmatrix} \frac{T^2 \cos(\varphi)}{2} \\ \frac{T^2 \sin(\varphi)}{2} \\ T \end{bmatrix}, \tag{B.11a}$$

$$\Gamma_{\ddot{\varphi}} = \begin{bmatrix} \frac{T^2}{2} \\ T \end{bmatrix}. \tag{B.11b}$$

Finally, Q_k can be obtained as follows:

$$Q_k = \begin{bmatrix} Q_a & 0 \\ 0 & Q_{\ddot{\varphi}} \end{bmatrix}, \tag{B.12}$$

with

$$\begin{aligned} Q_a &= \Gamma_a \sigma_a^2 \Gamma_a^T \\ &= \sigma_a^2 \cdot \begin{bmatrix} \frac{1}{4}T^4 \cos^2(\varphi) & \frac{1}{4}T^4 \sin(\varphi)\cos(\varphi) & \frac{1}{2}T^3 \cos(\varphi) \\ \frac{1}{4}T^4 \sin(\varphi)\cos(\varphi) & \frac{1}{4}T^4 \sin^2(\varphi) & \frac{1}{2}T^3 \sin(\varphi) \\ \frac{1}{2}T^3 \cos(\varphi) & \frac{1}{2}T^3 \sin(\varphi) & T^2 \end{bmatrix} \end{aligned} \tag{B.13a}$$

and

$$\begin{aligned} Q_{\ddot{\varphi}} &= \Gamma_{\ddot{\varphi}} \sigma_{\ddot{\varphi}}^2 \Gamma_{\ddot{\varphi}}^T \\ &= \sigma_{\ddot{\varphi}}^2 \begin{bmatrix} \frac{1}{4}T^4 & 0 \\ \frac{1}{2}T^3 & T^2 \end{bmatrix}. \end{aligned} \tag{B.13b}$$

Innovation step: After performing the prediction steps (B.4) and (B.7), the state $\boldsymbol{x}_{k|k-1}$ is updated by incorporating a current observation $\boldsymbol{z}_k$. For that, a predicted measurement is obtained as follows:

$$\hat{\boldsymbol{z}}_{k|k-1} = H_k \hat{\boldsymbol{x}}_{k|k-1}, \tag{B.14}$$

where H_k is the measurement matrix that corresponds to the network output $\hat{\boldsymbol{p}} = [\hat{x}, \hat{y}, \hat{\varphi}]^T$

$$H_k = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix}. \tag{B.15}$$

Furthermore, $\tilde{\boldsymbol{z}}_k = \boldsymbol{z}_k - \hat{\boldsymbol{z}}_{k|k-1}$ is defined as the measurement residual. Next, the innovation or system uncertainty S_k is updated

$$S_k = H_k \hat{P}_{k|k-1} H_k^T + R_k. \tag{B.16}$$

Here, the state uncertainty or measurement noise matrix R_k has to be defined. In the case of the filter-based inference, the matrix was found heuristically and set to a diagonal matrix, where the entries are dependent on the specific network architecture that is deployed. At this point, the updated state estimate $\hat{\boldsymbol{x}}_{k|k}$ can be retrieved as follows:

$$\hat{\boldsymbol{x}}_{k|k} = \hat{\boldsymbol{x}}_{k|k-1} + K_k \tilde{\boldsymbol{y}}_k, \tag{B.17}$$

where

$$K_k = \hat{P}_{k|k-1} H_k^T S_k^{-1} \tag{B.18}$$

is called the Kalman gain. Finally, the covariance estimate is updated as well

$$\hat{P}_{k|k} = (I_5 - K_k H_k) \hat{P}_{k|k-1}, \tag{B.19}$$

where I_5 denotes a 5×5 identity matrix.

List of Symbols

General Notation

x	Scalar
$\boldsymbol{x}$	Vector
$\boldsymbol{x}^T$	Transpose of a vector
X	Matrix
X^T	Transpose of a matrix
X^{-1}	Inverse of a matrix
$\det X$ or $\lvert X \rvert$	Determinant of a matrix
$\mathcal{X}$	Set
$\mathcal{P}$	Point cloud
$\triangleq$	is equal by definition
$\oplus$	Concatenation operation
$\otimes$	Multiplication operation
$\langle \cdot \rangle$	Pair

Probability Theory

$p(\cdot)$	Probability distribution
$\mathcal{N}(\cdot)$	Normal distribution
$\mathcal{U}\cdot$	Discrete uniform distribution
$\mathcal{U}_{[\cdot]}$	Continuous uniform distribution
$\mathrm{Pois}(\cdot)$	Poisson distribution
μ or $\mathrm{E}[\cdot]$	Mean or expectation of a probability distribution
σ	Standard deviation
σ^2 or $\mathrm{Var}[\cdot]$	Variance of a probability distribution
Σ	Covariance matrix
$\mathrm{Cov}(\cdot)$	Covariance between two random variables
$\sim$	Drawn from a distribution / Follows a distribution
$\mathbb{KL}(\cdot \,\|\|\, \cdot)$	Kullback-Leibler divergence

Localization

$\boldsymbol{p}$	Pose
$\Delta\boldsymbol{p}$	Pose offset / error
$\boldsymbol{p}_{\text{init}}$	Initial pose
$\boldsymbol{p}_{\text{GT}}$	Ground-truth pose
$\hat{\boldsymbol{p}}$	Predicted pose
$\boldsymbol{p}_{\text{GPS}}$	GPS pose
$\tilde{\boldsymbol{p}}$	Synthetic pose
$\mathcal{M}$	Measurement or scan set
$\mathcal{M}_t$	Measurement or scan set at time step t
$\mathcal{L}$	Landmark set
$\mathcal{L}^{\text{FoV}}$	Landmarks in vehicle's field of view
$\boldsymbol{m}$	Measurement
$\boldsymbol{l}$	Landmark
π	Cardinality of measurement or scan set
ν	Cardinality of landmark set
$\mathcal{S}$	Coordinate system
$\mathcal{S}_{\text{veh}}$	Vehicle coordinate system
$\mathcal{S}_{\text{UTM}}$	UTM coordinate system

Deep Learning

$\text{MLP}(\cdot\,\vert\,\cdot)$	Multilayer perceptron
$\text{CNN}(\cdot)$	Convolutional neural network
$\text{rFF}(\cdot\,\vert\,\cdot)$	Row-wise feedforward network
$\mathcal{A}(\cdot)$	Attention mechanism
$\sigma(\cdot)$	Activation function
$L(\cdot)$	Loss function
$f_{\boldsymbol{\theta}}$	Network with learnable parameters $\boldsymbol{\theta}$

Bibliography

[Agg18] Aggarwal, Charu C.: *Neural Networks and Deep Learning*. Springer, 2018, page 497.

[AM12] Anderson, Brian DO and Moore, John B: *Optimal filtering*. Courier Corporation, 2012.

[APSL08] Aulinas, Josep; Petillot, Yvan R.; Salvi, Joaquim; and Lladó, Xavier: *The SLAM problem: a survey*. In: *Artificial Intelligence Research and Development, Proceedings of the 11th International Conference of the Catalan Association for Artificial Intelligence, CCIA 2008, October 22-24, 2008, Sant Martí d'Empúries, Spain*. Volume 184, pages 363–371. *Frontiers in Artificial Intelligence and Applications*, IOS Press, 2008.

[BAC15] Bresson, Guillaume; Aufrère, Romuald; and Chapuis, Roland: *Improving SLAM with Drift Integration*. In: *IEEE 18th International Conference on Intelligent Transportation Systems, ITSC 2015, Gran Canaria, Spain, September 15-18, 2015*, pages 2700–2706, IEEE, 2015.

[BAYG17] Bresson, Guillaume; Alsayed, Zayed; Yu, Li; and Glaser, Sebastien: *Simultaneous Localization and Mapping: A Survey of Current Trends in Autonomous Driving*. In: *IEEE Trans. Intell. Veh.*, volume 2, no. 3, pages 194–220, 2017.

[BCB15] Bahdanau, Dzmitry; Cho, Kyunghyun; and Bengio, Yoshua: *Neural Machine Translation by Jointly Learning to Align and Translate*. In: *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

[BGLL17] Britz, Denny; Goldie, Anna; Luong, Minh-Thang; and Le, Quoc V.: *Massive Exploration of Neural Machine Translation Architectures*. In: *CoRR*, volume abs/1703.03906, 2017.

[BGM+18] Beltrán, Jorge; Guindel, Carlos; Moreno, Francisco Miguel; et al.: *BirdNet: A 3D Object Detection Framework from LiDAR Information*. In: *21st International Conference on Intelligent Transportation Systems, ITSC 2018, Maui, HI, USA, November 4-7, 2018*, pages 3517–3523, IEEE, 2018.

[BGM+19] Behley, Jens; Garbade, Martin; Milioto, Andres; et al.: *SemanticKITTI: A Dataset for Semantic Scene Understanding of LiDAR Sequences*. In: *2019 IEEE/CVF International Conference on Computer Vision, ICCV 2019, Seoul, Korea (South), October 27 - November 2, 2019*, pages 9296–9306, IEEE, 2019.

[BGM+21] Behley, Jens; Garbade, Martin; Milioto, Andres; et al.: *Towards 3D LiDAR-based semantic scene understanding of 3D point cloud sequences: The SemanticKITTI Dataset*. In: *Int. J. Robotics Res.*, volume 40, no. 8-9, 2021.

[BKH16] Ba, Jimmy Lei; Kiros, Jamie Ryan; and Hinton, Geoffrey E: *Layer Normalization*. In: *arXiv preprint arXiv:1607.06450*, 2016.

[BLRW16] Brock, André; Lim, Theodore; Ritchie, James M.; and Weston, Nick: *Generative and Discriminative Voxel Modeling with Convolutional Neural Networks*. In: *CoRR*, volume abs/1608.04236, 2016.

[BM92] Besl, Paul J. and McKay, Neil D.: *A Method for Registration of 3-D Shapes*. In: *IEEE Trans. Pattern Anal. Mach. Intell.*, volume 14, no. 2, pages 239–256, 1992.

[BPB+18] Barea, Rafael; Perez, Carlos; Bergasa, Luis Miguel; et al.: *Vehicle Detection and Localization using 3D LIDAR Point Cloud and Image Semantic Segmentation*. In: *21st International Conference on Intelligent Transportation Systems, ITSC 2018, Maui, HI, USA, November 4-7, 2018*, pages 3481–3486, IEEE, 2018.

[BR19] Brachmann, Eric and Rother, Carsten: *Neural-Guided RANSAC: Learning Where to Sample Model Hypotheses*. In: *2019 IEEE/CVF International Conference on Computer Vision, ICCV 2019, Seoul, Korea (South), October 27 - November 2, 2019*, pages 4321–4330, IEEE, 2019.

[BWWN19] Berrio, Julie Stephany; Ward, James R.; Worrall, Stewart; and Nebot, Eduardo M.: *Identifying robust landmarks in feature-based maps*. In: *2019 IEEE Intelligent Vehicles Symposium, IV 2019, Paris, France, June 9-12, 2019*, pages 1166–1172, IEEE, 2019.

[Cen08] Censi, Andrea: *An ICP variant using a point-to-line metric*. In: *2008 IEEE International Conference on Robotics and Automation, ICRA 2008, May 19-23, 2008, Pasadena, California, USA*, pages 19–25, IEEE, 2008.

[CFG+15] Chang, Angel X.; Funkhouser, Thomas A.; Guibas, Leonidas J.; et al.: *ShapeNet: An Information-Rich 3D Model Repository*. In: *CoRR*, volume abs/1512.03012, 2015.

[CHM12] Chen, Ling; Hu, Huosheng; and McDonald-Maier, Klaus D.: *EKF Based Mobile Robot Localization*. In: *2012 Third International Conference on Emerging Security Technologies, Lisbon, Portugal, September 5-7, 2012*, pages 149–154, IEEE Computer Society, 2012.

[CLF+21] Chen, Siheng; Liu, Baoan; Feng, Chen; Vallespi-Gonzalez, Carlos; and Wellington, Carl: *3D Point Cloud Processing and Learning for Autonomous Driving: Impacting Map Creation, Localization, and Perception*. In: *IEEE Signal Process. Mag.*, volume 38, no. 1, pages 68–86, 2021.

[Cov99] Cover, Thomas M: *Elements of Information Theory*. John Wiley & Sons, 1999.

[CWL+20] Chen, Changhao; Wang, Bing; Lu, Chris Xiaoxuan; Trigoni, Niki; and Markham, Andrew: *A Survey on Deep Learning for Localization and Mapping: Towards the Age of Spatial Machine Intelligence*. In: *CoRR*, volume abs/2006.12567, 2020.

[DBK+21] Dosovitskiy, Alexey; Beyer, Lucas; Kolesnikov, Alexander; et al.: *An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale*. In: *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*, OpenReview.net, 2021.

[Dev12] Devlin, Keith: *The Joy of Sets: Fundamentals of Contemporary Set Theory*. Springer Science & Business Media, 2012.

[DFBT99] Dellaert, Frank; Fox, Dieter; Burgard, Wolfram; and Thrun, Sebastian: *Monte Carlo Localization for Mobile Robots*. In: *1999 IEEE International Conference on Robotics and Automation, Marriott Hotel, Renaissance Center, Detroit, Michigan, USA, May 10-15, 1999, Proceedings*, pages 1322–1328, IEEE Robotics and Automation Society, 1999.

[DHS11] Duchi, John C.; Hazan, Elad; and Singer, Yoram: *Adaptive Subgradient Methods for Online Learning and Stochastic Optimization*. In: *J. Mach. Learn. Res.*, volume 12, pages 2121–2159, 2011.

[Die21] Dietmayer, Klaus: *Lecture notes in Filter und Trackingverfahren*. 2021.

[DL18] Deng, Li and Liu, Yang: *Deep Learning in Natural Language Processing*. Springer, 2018.

[DLMP15] D'Alfonso, Luigi; Lucia, Walter; Muraca, Pietro; and Pugliese, Paolo: *Mobile robot localization via EKF and UKF: A comparison based on real data*. In: *Robotics Auton. Syst.*, volume 74, pages 122–127, 2015.

[DRD15] Deusch, Hendrik; Reuter, Stephan; and Dietmayer, Klaus: *The Labeled Multi-Bernoulli SLAM Filter*. In: *IEEE Signal Process. Lett.*, volume 22, no. 10, pages 1561–1565, 2015.

[EBD21a] Engel, Nico; Belagiannis, Vasileios; and Dietmayer, Klaus: *Attention-based Vehicle Self-Localization with HD Feature Maps*. In: *24th IEEE International Intelligent Transportation Systems Conference, ITSC 2021, Indianapolis, IN, USA, September 19-22, 2021*, pages 76–83, IEEE, 2021.

[EBD21b] Engel, Nico; Belagiannis, Vasileios; and Dietmayer, Klaus: *Point Transformer*. In: *IEEE Access*, volume 9, pages 134826–134840, 2021.

[ED20] Engel, Nico and Dietmayer, Klaus: *Selbstlokalisierung fur automatisierte Fahrzeuge mit tiefen neuronalen Netzen*. In: *Uni-Das Workshop 2020*, 2020.

[EH20] Elhousni, Mahdi and Huang, Xinming: *A Survey on 3D LiDAR Localization for Autonomous Vehicles*. In: *IEEE Intelligent Vehicles Symposium, IV 2020, Las Vegas, NV, USA, October 19 - November 13, 2020*, pages 1879–1884, IEEE, 2020.

[EHH+19] Engel, Nico; Hoermann, Stefan; Horn, Markus; Belagiannis, Vasileios; and Dietmayer, Klaus: *DeepLocalization: Landmark-based Self-Localization with Deep Neural Networks*. In: *2019 IEEE Intelligent Transportation Systems Conference, ITSC 2019, Auckland, New Zealand, October 27-30, 2019*, pages 926–933, IEEE, 2019.

[EHHD18] Engel, Nico; Hoermann, Stefan; Henzler, Philipp; and Dietmayer, Klaus: *Deep Object Tracking on Dynamic Occupancy Grid Maps Using RNNs*. In: *21st International Conference on Intelligent Transportation Systems, ITSC 2018, Maui, HI, USA, November 4-7, 2018*, pages 3852–3858, IEEE, 2018.

[EKSX96] Ester, Martin; Kriegel, Hans-Peter; Sander, Jörg; and Xu, Xiaowei: *A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise*. In: *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining (KDD-96), Portland, Oregon, USA*, pages 226–231, AAAI Press, 1996.

[FHR+21] Feng, Di; Haase-Schütz, Christian; Rosenbaum, Lars; et al.: *Deep Multi-Modal Object Detection and Semantic Segmentation for Autonomous Driving: Datasets, Methods, and Challenges*. In: *IEEE Trans. Intell. Transp. Syst.*, volume 22, no. 3, pages 1341–1360, 2021.

[Fox01] Fox, Dieter: *KLD-Sampling: Adaptive Particle Filters*. In: pages 713–720, 2001.

[FRS07] Fontanelli, Daniele; Ricciato, Luigi; and Soatto, Stefano: *A Fast RANSAC-Based Registration Algorithm for Accurate Localization in Unknown Environments using LIDAR Measurements*. In: *IEEE Conference on Automation Science and Engineering, CASE 2007, September 22-25, 2007. Scottsdale, Arizona, USA*, pages 597–602, IEEE, 2007.

[FTBD01] Fox, Dieter; Thrun, Sebastian; Burgard, Wolfram; and Dellaert, Frank: *Particle Filters for Mobile Robot Localization.* In: *Sequential Monte Carlo Methods in Practice.* From the series *Statistics for Engineering and Information Science*, pages 401–428, Springer, 2001.

[FTZ+18] Fathy, Mohammed E.; Tran, Quoc-Huy; Zia, M. Zeeshan; Vernaza, Paul; and Chandraker, Manmohan: *Hierarchical Metric Learning and Matching for 2D and 3D Geometric Correspondences.* In: *Computer Vision - ECCV 2018 - 15th European Conference, Munich, Germany, September 8-14, 2018, Proceedings, Part XV.* Volume 11219, pages 832–850. *Lecture Notes in Computer Science*, Springer, 2018.

[GBC16] Goodfellow, Ian; Bengio, Yoshua; and Courville, Aaron: *Deep Learning.* The MIT Press, 2016.

[GDD18] Gies, Fabian; Danzer, Andreas; and Dietmayer, Klaus: *Environment Perception Framework Fusing Multi-Object Tracking, Dynamic Occupancy Grid Maps and Digital Maps.* In: *21st International Conference on Intelligent Transportation Systems, ITSC 2018, Maui, HI, USA, November 4-7, 2018*, pages 3859–3865, IEEE, 2018.

[GLU12] Geiger, Andreas; Lenz, Philip; and Urtasun, Raquel: *Are we ready for autonomous driving? The KITTI vision benchmark suite.* In: *2012 IEEE Conference on Computer Vision and Pattern Recognition, Providence, RI, USA, June 16-21, 2012*, pages 3354–3361, IEEE Computer Society, 2012.

[GPBD20] Gies, Fabian; Posselt, Joachim; Buchholz, Michael; and Dietmayer, Klaus: *Extended Existence Probability Using Digital Maps for Object Verification.* In: *IEEE 23rd International Conference on Information Fusion, FUSION 2020, Rustenburg, South Africa, July 6-9, 2020*, pages 1–7, IEEE, 2020.

[GSZD18] Graf, Maximilian; Speidel, Oliver; Ziegler, Julius; and Dietmayer, Klaus: *Trajectory Planning for Automated Vehicles using Driver Models.* In: *21st International Conference on Intelligent Transportation Systems, ITSC 2018, Maui, HI, USA, November 4-7, 2018*, pages 1455–1460, IEEE, 2018.

[GWH+21] Guo, Yulan; Wang, Hanyun; Hu, Qingyong; et al.: *Deep Learning for 3D Point Clouds: A Survey.* In: *IEEE Trans. Pattern Anal. Mach. Intell.*, volume 43, no. 12, pages 4338–4364, 2021.

[HEB+20] Horn, Markus; Engel, Nico; Belagiannis, Vasileios; Buchholz, Michael; and Dietmayer, Klaus: *DeepCLR: Correspondence-Less Architecture for Deep End-to-End Point Cloud Registration.* In: *23rd IEEE International Conference on Intelligent Transportation Systems, ITSC 2020, Rhodes, Greece, September 20-23, 2020*, pages 1–7, IEEE, 2020.

[HM15] Hirschberg, Julia and Manning, Christopher D: *Advances in Natural Language Processing*. In: *Science*, volume 349, no. 6245, pages 261–266, 2015.

[HSW89] Hornik, Kurt; Stinchcombe, Maxwell B.; and White, Halbert: *Multilayer feedforward networks are universal approximators*. In: *Neural Networks*, volume 2, no. 5, pages 359–366, 1989.

[HW68] Hubel, David H and Wiesel, Torsten N: *Receptive Fields and Functional Architecture of Monkey Striate Cortex*. In: *The Journal of physiology*, volume 195, no. 1, pages 215–243, 1968.

[HYX+20] Hu, Qingyong; Yang, Bo; Xie, Linhai; et al.: *RandLA-Net: Efficient Semantic Segmentation of Large-Scale Point Clouds*. In: *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2020, Seattle, WA, USA, June 13-19, 2020*, pages 11105–11114, Computer Vision Foundation / IEEE, 2020.

[HZRS15] He, Kaiming; Zhang, Xiangyu; Ren, Shaoqing; and Sun, Jian: *Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification*. In: *2015 IEEE International Conference on Computer Vision, ICCV 2015, Santiago, Chile, December 7-13, 2015*, pages 1026–1034, IEEE Computer Society, 2015.

[HZRS16] He, Kaiming; Zhang, Xiangyu; Ren, Shaoqing; and Sun, Jian: *Deep Residual Learning for Image Recognition*. In: *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*, pages 770–778, IEEE Computer Society, 2016.

[IKP10] Ivanjko, Edouard; Kitanov, Andreja; and Petrovic, Ivan: *Model based Kalman Filter Mobile Robot Self-Localization*. In: *Robot Localization and Map Building*, pages 59–89, 2010.

[IS15] Ioffe, Sergey and Szegedy, Christian: *Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift*. In: *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015*. Volume 37, pages 448–456. *JMLR Workshop and Conference Proceedings*, JMLR.org, 2015.

[JAA95] Jain, P.K.; Ahmad, K.; and Ahuja, O.P.: *Functional Analysis*. New Age International, 1995.

[KB15] Kingma, Diederik P. and Ba, Jimmy: *Adam: A Method for Stochastic Optimization*. In: *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

[KC16] Kendall, Alex and Cipolla, Roberto: *Modelling uncertainty in deep learning for camera relocalization.* In: *2016 IEEE International Conference on Robotics and Automation, ICRA 2016, Stockholm, Sweden, May 16-21, 2016*, pages 4762–4769, IEEE, 2016.

[KC17] Kendall, Alex and Cipolla, Roberto: *Geometric Loss Functions for Camera Pose Regression with Deep Learning.* In: *2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, Honolulu, HI, USA, July 21-26, 2017*, pages 6555–6564, IEEE Computer Society, 2017.

[KFK+18] Kuutti, Sampo; Fallah, Saber; Katsaros, Konstantinos; et al.: *A Survey of the State-of-the-Art Localization Techniques and Their Potentials for Autonomous Vehicle Applications.* In: *IEEE Internet Things J.*, volume 5, no. 2, pages 829–846, 2018.

[KG17] Kendall, Alex and Gal, Yarin: *What Uncertainties Do We Need in Bayesian Deep Learning for Computer Vision?* In: *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pages 5574–5584, 2017.

[KGC15] Kendall, Alex; Grimes, Matthew; and Cipolla, Roberto: *PoseNet: A Convolutional Network for Real-Time 6-DOF Camera Relocalization.* In: *2015 IEEE International Conference on Computer Vision, ICCV 2015, Santiago, Chile, December 7-13, 2015*, pages 2938–2946, IEEE Computer Society, 2015.

[KGC18] Kendall, Alex; Gal, Yarin; and Cipolla, Roberto: *Multi-Task Learning Using Uncertainty to Weigh Losses for Scene Geometry and Semantics.* In: *2018 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2018, Salt Lake City, UT, USA, June 18-22, 2018*, pages 7482–7491, Computer Vision Foundation / IEEE Computer Society, 2018.

[KMN18] Kanezaki, Asako; Matsushita, Yasuyuki; and Nishida, Yoshifumi: *RotationNet: Joint Object Categorization and Pose Estimation Using Multiviews From Unsupervised Viewpoints.* In: *2018 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2018, Salt Lake City, UT, USA, June 18-22, 2018*, pages 5010–5019, Computer Vision Foundation / IEEE Computer Society, 2018.

[KND+21] Kulkarni, Abhilash Sachin; Nayak, Jyothi S; Desai, Aditi; Singh, Jahnavi; and Murali, Shraddha: *Deep Semantic Segmentation for Self-driving Cars.* In: *Advances in Signal and Data Processing*, pages 1–14, Springer, 2021.

[KNW+15] Kunz, Felix; Nuss, Dominik; Wiest, Jürgen; et al.: *Autonomous driving at Ulm University: A modular, robust, and sensor-independent fusion approach*. In: *2015 IEEE Intelligent Vehicles Symposium, IV 2015, Seoul, South Korea, June 28 - July 1, 2015*, pages 666–673, IEEE, 2015.

[Kra11] Kraus, Karl: *Photogrammetry: Geometry from Images and Laser Scans*. Walter de Gruyter, 2011.

[KVPF20] Katharopoulos, Angelos; Vyas, Apoorv; Pappas, Nikolaos; and Fleuret, François: *Transformers are RNNs: Fast Autoregressive Transformers with Linear Attention*. In: *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*. Volume 119, pages 5156–5165. *Proceedings of Machine Learning Research*, PMLR, 2020.

[LBS+18] Li, Yangyan; Bu, Rui; Sun, Mingchao; et al.: *PointCNN: Convolution On X-Transformed Points*. In: *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*, pages 828–838, 2018.

[LCW+19] Li, Qing; Chen, Shaoyang; Wang, Cheng; et al.: *LO-Net: Deep Real-Time Lidar Odometry*. In: *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2019, Long Beach, CA, USA, June 16-20, 2019*, pages 8473–8482, Computer Vision Foundation / IEEE, 2019.

[LD18] Le, Truc and Duan, Ye: *PointGrid: A Deep Network for 3D Shape Understanding*. In: *2018 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2018, Salt Lake City, UT, USA, June 18-22, 2018*, pages 9204–9214, Computer Vision Foundation / IEEE Computer Society, 2018.

[LHLZ19] Liu, Xinhai; Han, Zhizhong; Liu, Yu-Shen; and Zwicker, Matthias: *Point2Sequence: Learning the Shape Representation of 3D Point Clouds with an Attention-Based Sequence to Sequence Network*. In: *The Thirty-Third AAAI Conference on Artificial Intelligence, AAAI 2019, The Thirty-First Innovative Applications of Artificial Intelligence Conference, IAAI 2019, The Ninth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2019, Honolulu, Hawaii, USA, January 27 - February 1, 2019*, pages 8778–8785, AAAI Press, 2019.

[LJH+20] Liu, Liyuan; Jiang, Haoming; He, Pengcheng; et al.: *On the Variance of the Adaptive Learning Rate and Beyond*. In: *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*, OpenReview.net, 2020.

[LLK+19] Lee, Juho; Lee, Yoonho; Kim, Jungtaek; et al.: *Set Transformer: A Framework for Attention-based Permutation-Invariant Neural Networks*. In: *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*. Volume 97, pages 3744–3753. *Proceedings of Machine Learning Research*, PMLR, 2019.

[LLUZ16] Luo, Wenjie; Li, Yujia; Urtasun, Raquel; and Zemel, Richard S.: *Understanding the Effective Receptive Field in Deep Convolutional Neural Networks*. In: *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain*, pages 4898–4906, 2016.

[LMT07] Levinson, Jesse; Montemerlo, Michael; and Thrun, Sebastian: *Map-Based Precision Vehicle Localization in Urban Environments*. In: *Robotics: Science and Systems III, June 27-30, 2007, Georgia Institute of Technology, Atlanta, Georgia, USA*, The MIT Press, 2007.

[Low04] Lowe, David G.: *Distinctive Image Features from Scale-Invariant Keypoints*. In: *Int. J. Comput. Vis.*, volume 60, no. 2, pages 91–110, 2004.

[LP17] Lynch, Kevin M and Park, Frank C: *Modern Robotics*. Cambridge University Press, 2017.

[LPM15] Luong, Thang; Pham, Hieu; and Manning, Christopher D.: *Effective Approaches to Attention-based Neural Machine Translation*. In: *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing, EMNLP 2015, Lisbon, Portugal, September 17-21, 2015*, pages 1412–1421, The Association for Computational Linguistics, 2015.

[LQG19] Liu, Xingyu; Qi, Charles R.; and Guibas, Leonidas J.: *FlowNet3D: Learning Scene Flow in 3D Point Clouds*. In: *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2019, Long Beach, CA, USA, June 16-20, 2019*, pages 529–537, Computer Vision Foundation / IEEE, 2019.

[LTLH19] Liu, Zhijian; Tang, Haotian; Lin, Yujun; and Han, Song: *Point-Voxel CNN for Efficient 3D Deep Learning*. In: *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pages 963–973, 2019.

[LVC+19] Lang, Alex H.; Vora, Sourabh; Caesar, Holger; et al.: *PointPillars: Fast Encoders for Object Detection From Point Clouds*. In: *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2019, Long Beach, CA, USA, June 16-20, 2019*, pages 12697–12705, Computer Vision Foundation / IEEE, 2019.

[LWZ+19] Lu, Weixin; Wan, Guowei; Zhou, Yao; et al.: *DeepVCP: An End-to-End Deep Neural Network for Point Cloud Registration.* In: *2019 IEEE/CVF International Conference on Computer Vision, ICCV 2019, Seoul, Korea (South), October 27 - November 2, 2019*, pages 12–21, IEEE, 2019.

[LZW+19] Lu, Weixin; Zhou, Yao; Wan, Guowei; Hou, Shenhua; and Song, Shiyu: *L3-Net: Towards Learning Based LiDAR Localization for Autonomous Driving.* In: *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2019, Long Beach, CA, USA, June 16-20, 2019*, pages 6389–6398, Computer Vision Foundation / IEEE, 2019.

[Mah07] Mahler, Ronald PS: *Statistical Multisource-Multitarget Information Fusion.* Artech House, Inc., 2007.

[MCUP04] Matas, Jiri; Chum, Ondrej; Urban, Martin; and Pajdla, Tomás: *Robust wide-baseline stereo from maximally stable extremal regions.* In: *Image Vis. Comput.*, volume 22, no. 10, pages 761–767, 2004.

[MD95] Maksarov, D and Durrant-Whyte, H: *Mobile Vehicle Navigation in Unknown Environments: A Multiple Hypothesis Approach.* In: *IEE Proceedings-Control Theory and Applications*, volume 142, no. 4, pages 385–400, 1995.

[MHW+19] Mohamed, Sherif Abdelmonem Sayed; Haghbayan, Mohammad Hashem; Westerlund, Tomi; et al.: *A Survey on Odometry for Autonomous Navigation Systems.* In: *IEEE Access*, volume 7, pages 97466–97486, 2019.

[Mis19] Misra, Diganta: *Mish: A Self Regularized Non-Monotonic Neural Activation Function.* In: *CoRR*, volume abs/1908.08681, 2019.

[MS15] Maturana, Daniel and Scherer, Sebastian A.: *VoxNet: A 3D Convolutional Neural Network for real-time object recognition.* In: *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2015, Hamburg, Germany, September 28 - October 2, 2015*, pages 922–928, IEEE, 2015.

[MTKW02] Montemerlo, Michael; Thrun, Sebastian; Koller, Daphne; and Wegbreit, Ben: *FastSLAM: A Factored Solution to the Simultaneous Localization and Mapping Problem.* In: *Proceedings of the Eighteenth National Conference on Artificial Intelligence and Fourteenth Conference on Innovative Applications of Artificial Intelligence, July 28 - August 1, 2002, Edmonton, Alberta, Canada*, pages 593–598, AAAI Press / The MIT Press, 2002.

[MTKW03] Montemerlo, Michael; Thrun, Sebastian; Koller, Daphne; and Wegbreit, Ben: *FastSLAM 2.0: An Improved Particle Filtering Algorithm for Simultaneous Localization and Mapping that Provably Converges.* In: *IJCAI-03, Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence, Acapulco, Mexico, August 9-15, 2003*, pages 1151–1156, Morgan Kaufmann, 2003.

[Mur12] Murphy, Kevin P: *Machine Learning: A Probabilistic Perspective*. MIT press, 2012.

[Mur22] Murphy, Kevin P.: *Probabilistic Machine Learning: An Introduction*. MIT Press, 2022.

[MVA10] Mullane, John; Vo, Ba-Ngu; and Adams, Martin David: *Rao-Blackwellised PHD SLAM.* In: *IEEE International Conference on Robotics and Automation, ICRA 2010, Anchorage, Alaska, USA, 3-7 May 2010*, pages 5410–5416, IEEE, 2010.

[MVAV11] Mullane, John; Vo, Ba-Ngu; Adams, Martin David; and Vo, Ba-Tuong: *A Random-Finite-Set Approach to Bayesian SLAM.* In: *IEEE Trans. Robotics*, volume 27, no. 2, pages 268–282, 2011.

[Nie15] Nielsen, Michael A: *Neural networks and deep learning*. Volume 2018. Determination Press San Francisco, CA, 2015.

[NRT+18] Nuss, Dominik; Reuter, Stephan; Thom, Markus; et al.: *A random finite set approach for dynamic occupancy grid maps with real-time application.* In: *Int. J. Robotics Res.*, volume 37, no. 8, pages 841–866, 2018.

[PCS15] Pomerleau, François; Colas, Francis; and Siegwart, Roland: *A Review of Point Cloud Registration Algorithms for Mobile Robotics.* In: *Found. Trends Robotics*, volume 4, no. 1, pages 1–104, 2015.

[PGM+19] Paszke, Adam; Gross, Sam; Massa, Francisco; et al.: *PyTorch: An Imperative Style, High-Performance Deep Learning Library.* In: *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pages 8024–8035, 2019.

[PLH+19] Peng, Sida; Liu, Yuan; Huang, Qixing; Zhou, Xiaowei; and Bao, Hujun: *PVNet: Pixel-Wise Voting Network for 6DoF Pose Estimation.* In: *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2019, Long Beach, CA, USA, June 16-20, 2019*, pages 4561–4570, Computer Vision Foundation / IEEE, 2019.

[Qi18] Qi, Ruizhongtai: *Deep Learning on Point Clouds for 3D Scene Understanding.* Dissertation, 2018.

[QSMG17] Qi, Charles Ruizhongtai; Su, Hao; Mo, Kaichun; and Guibas, Leonidas J.: *PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation.* In: *2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, Honolulu, HI, USA, July 21-26, 2017*, pages 77–85, IEEE Computer Society, 2017.

[QSN+16] Qi, Charles Ruizhongtai; Su, Hao; Nießner, Matthias; et al.: *Volumetric and Multi-view CNNs for Object Classification on 3D Data.* In: *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*, pages 5648–5656, IEEE Computer Society, 2016.

[QYSG17] Qi, Charles Ruizhongtai; Yi, Li; Su, Hao; and Guibas, Leonidas J.: *PointNet++: Deep Hierarchical Feature Learning on Point Sets in a Metric Space.* In: *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pages 5099–5108, 2017.

[RHW86] Rumelhart, David E; Hinton, Geoffrey E; and Williams, Ronald J: *Learning Representations by Back-Propagating Errors.* In: *nature*, volume 323, no. 6088, pages 533–536, 1986.

[Ros12] Rosenkrantz, Roger D: *ET Jaynes: Papers on Probability, Statistics and Statistical Physics.* Volume 158. Springer Science & Business Media, 2012.

[Ros58] Rosenblatt, Frank: *The perceptron: A Probabilistic Model for Information Storage and Organization in the Brain.* In: *Psychological review*, volume 65, no. 6, page 386, 1958.

[RRKB11] Rublee, Ethan; Rabaud, Vincent; Konolige, Kurt; and Bradski, Gary R.: *ORB: An efficient alternative to SIFT or SURF.* In: *IEEE International Conference on Computer Vision, ICCV 2011, Barcelona, Spain, November 6-13, 2011*, pages 2564–2571, IEEE Computer Society, 2011.

[RUG17] Riegler, Gernot; Ulusoy, Ali Osman; and Geiger, Andreas: *OctNet: Learning Deep 3D Representations at High Resolutions.* In: *2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, Honolulu, HI, USA, July 21-26, 2017*, pages 6620–6629, IEEE Computer Society, 2017.

[RVB18] Radwan, Noha; Valada, Abhinav; and Burgard, Wolfram: *VLocNet++: Deep Multitask Learning for Semantic Visual Localization and Odometry.* In: *IEEE Robotics Autom. Lett.*, volume 3, no. 4, pages 4407–4414, 2018.

[RVVD14] Reuter, Stephan; Vo, Ba-Tuong; Vo, Ba-Ngu; and Dietmayer, Klaus: *The Labeled Multi-Bernoulli Filter*. In: *IEEE Trans. Signal Process.*, volume 62, no. 12, pages 3246–3260, 2014.

[SBM+20] Strohbeck, Jan; Belagiannis, Vasileios; Müller, Johannes; et al.: *Multiple Trajectory Prediction with Deep Temporal and Spatial Convolutional Neural Networks*. In: *IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2020, Las Vegas, NV, USA, October 24, 2020 - January 24, 2021*, pages 1992–1998, IEEE, 2020.

[SBZB15] Shi, Baoguang; Bai, Song; Zhou, Zhichao; and Bai, Xiang: *DeepPano: Deep Panoramic Representation for 3-D Shape Recognition*. In: *IEEE Signal Process. Lett.*, volume 22, no. 12, pages 2339–2343, 2015.

[SEJ+20] Senior, Andrew W.; Evans, Richard; Jumper, John; et al.: *Improved protein structure prediction using potentials from deep learning*. In: *Nat.*, volume 577, no. 7792, pages 706–710, 2020.

[SGWM18] Su, Jong-Chyi; Gadelha, Matheus; Wang, Rui; and Maji, Subhransu: *A Deeper Look at 3D Shape Classifiers*. In: *Computer Vision - ECCV 2018 Workshops - Munich, Germany, September 8-14, 2018, Proceedings, Part III*. Volume 11131, pages 645–661. *Lecture Notes in Computer Science*, Springer, 2018.

[SHK+14] Srivastava, Nitish; Hinton, Geoffrey E.; Krizhevsky, Alex; Sutskever, Ilya; and Salakhutdinov, Ruslan: *Dropout: a simple way to prevent neural networks from overfitting*. In: *J. Mach. Learn. Res.*, volume 15, no. 1, pages 1929–1958, 2014.

[SHM+16] Silver, David; Huang, Aja; Maddison, Chris J.; et al.: *Mastering the game of Go with deep neural networks and tree search*. In: *Nat.*, volume 529, no. 7587, pages 484–489, 2016.

[SHS+18] Silver, David; Hubert, Thomas; Schrittwieser, Julian; et al.: *A General Reinforcement Learning Algorithm that Masters Chess, Shogi, and Go through self-play*. In: *Science*, volume 362, no. 6419, pages 1140–1144, 2018.

[Sin15] Singh, Santokh: *Critical Reasons for Crashes Investigated in the National Motor Vehicle Crash Causation Survey*. Tech. rep. 2015.

[SMKL15] Su, Hang; Maji, Subhransu; Kalogerakis, Evangelos; and Learned-Miller, Erik G.: *Multi-view Convolutional Neural Networks for 3D Shape Recognition*. In: *2015 IEEE International Conference on Computer Vision, ICCV 2015, Santiago, Chile, December 7-13, 2015*, pages 945–953, IEEE Computer Society, 2015.

[SRD17] Stuebler, Manuel; Reuter, Stephan; and Dietmayer, Klaus: *A continuously learning feature-based map using a bernoulli filtering approach.* In: *Sensor Data Fusion: Trends, Solutions, Applications, SDF 2017, Bonn, Germany, October 10-12, 2017*, pages 1–6, IEEE, 2017.

[SSM62] Smith, Gerald L; Schmidt, Stanley F; and McGee, Leonard A: *Application of Statistical Filter Theory to the Optimal Estimation of Position and Velocity on Board A Circumlunar Vehicle.* National Aeronautics and Space Administration, 1962.

[STIM18] Santurkar, Shibani; Tsipras, Dimitris; Ilyas, Andrew; and Madry, Aleksander: *How Does Batch Normalization Help Optimization?* In: *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*, pages 2488–2498, 2018.

[Stü18] Stübler, Manuel: *Self-Assessing Localization and Long-Term Mapping using Random Finite Sets.* Dissertation, Universität Ulm, 2018.

[SV05] Shirali, Satish and Vasudeva, Harkrishan Lal: *Metric Spaces.* Springer Science & Business Media, 2005.

[SVL14] Sutskever, Ilya; Vinyals, Oriol; and Le, Quoc V.: *Sequence to Sequence Learning with Neural Networks.* In: *Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems 2014, December 8-13 2014, Montreal, Quebec, Canada*, pages 3104–3112, 2014.

[SWD15] Stuebler, Manuel; Wiest, Jürgen; and Dietmayer, Klaus: *Feature-based mapping and self-localization for road vehicles using a single grayscale camera.* In: *2015 IEEE Intelligent Vehicles Symposium, IV 2015, Seoul, South Korea, June 28 - July 1, 2015*, pages 267–272, IEEE, 2015.

[SYF02] Shuxin, Chen; Yongsheng, Wang; and Fei, Chen: *A Study of Differential GPS Positioning Accuracy.* In: *2002 3rd International Conference on Microwave and Millimeter Wave Technology, 2002. Proceedings. ICMMT 2002.* Pages 361–364. IEEE, 2002.

[SZAB17] Sedaghat, Nima; Zolfaghari, Mohammadreza; Amiri, Ehsan; and Brox, Thomas: *Orientation-boosted Voxel Nets for 3D Object Recognition.* In: *British Machine Vision Conference 2017, BMVC 2017, London, UK, September 4-7, 2017*, BMVA Press, 2017.

[TBF98] Thrun, Sebastian; Burgard, Wolfram; and Fox, Dieter: *A Probabilistic Approach to Concurrent Mapping and Localization for Mobile Robots.* In: *Auton. Robots*, volume 5, no. 3-4, pages 253–271, 1998.

[TFBD01] Thrun, Sebastian; Fox, Dieter; Burgard, Wolfram; and Dellaert, Frank: *Robust Monte Carlo localization for mobile robots.* In: *Artif. Intell.*, volume 128, no. 1-2, pages 99–141, 2001.

[Thr02] Thrun, Sebastian: *Probabilistic Robotics*. In: *Communications of the ACM*, volume 45, no. 3, pages 52–57, 2002.

[TM17] Teunissen, Peter JG and Montenbruck, Oliver: *Springer Handbook of Global Navigation Satellite Systems*. Volume 1. Springer, 2017.

[TQD+19] Thomas, Hugues; Qi, Charles R.; Deschaud, Jean-Emmanuel; et al.: *KPConv: Flexible and Deformable Convolution for Point Clouds*. In: *2019 IEEE/CVF International Conference on Computer Vision, ICCV 2019, Seoul, Korea (South), October 27 - November 2, 2019*, pages 6410–6419, IEEE, 2019.

[TSK11] Teslic, Luka; Skrjanc, Igor; and Klancar, Gregor: *EKF-Based Localization of a Wheeled Mobile Robot in Structured Environments*. In: *J. Intell. Robotic Syst.*, volume 62, no. 2, pages 187–203, 2011.

[TZWL21] Tao, Zhiyong; Zhu, Yixin; Wei, Tong; and Lin, Sen: *Multi-Head Attentional Point Cloud Classification and Segmentation Using Strictly Rotation-Invariant Representations*. In: *IEEE Access*, volume 9, pages 71133–71144, 2021.

[VBK16] Vinyals, Oriol; Bengio, Samy; and Kudlur, Manjunath: *Order Matters: Sequence to Sequence for Sets*. In: *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*, 2016.

[VRB18] Valada, Abhinav; Radwan, Noha; and Burgard, Wolfram: *Deep Auxiliary Learning for Visual Localization and Odometry*. In: *2018 IEEE International Conference on Robotics and Automation, ICRA 2018, Brisbane, Australia, May 21-25, 2018*, pages 6939–6946, IEEE, 2018.

[VSP+17] Vaswani, Ashish; Shazeer, Noam; Parmar, Niki; et al.: *Attention is All you Need*. In: *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pages 5998–6008, 2017.

[WBKB20] Wiederer, Julian; Bouazizi, Arij; Kressel, Ulrich; and Belagiannis, Vasileios: *Traffic Control Gesture Recognition for Autonomous Vehicles*. In: *IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2020, Las Vegas, NV, USA, October 24, 2020 - January 24, 2021*, pages 10676–10683, IEEE, 2020.

[WCL+20] Wang, Bing; Chen, Changhao; Lu, Chris Xiaoxuan; et al.: *AtLoc: Attention Guided Camera Localization*. In: *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020, New York, NY, USA, February 7-12, 2020*, pages 10393–10401, AAAI Press, 2020.

[WDN+14] Wiest, Jürgen; Deusch, Hendrik; Nuss, Dominik; et al.: *Localization based on region descriptors in grid maps*. In: *2014 IEEE Intelligent Vehicles Symposium Proceedings, Dearborn, MI, USA, June 8-11, 2014*, pages 793–799, IEEE, 2014.

[WEK05] Wing, Michael G; Eklund, Aaron; and Kellogg, Loren D: *Consumer-grade Global Positioning System (GPS) Accuracy and Reliability*. In: *Journal of forestry*, volume 103, no. 4, pages 169–173, 2005.

[WFE+19] Wagstaff, Edward; Fuchs, Fabian; Engelcke, Martin; Posner, Ingmar; and Osborne, Michael A.: *On the Limitations of Representing Functions on Sets*. In: *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*. Volume 97, pages 6487–6494. *Proceedings of Machine Learning Research*, PMLR, 2019.

[WFE+21] Wagstaff, Edward; Fuchs, Fabian B.; Engelcke, Martin; Osborne, Michael A.; and Posner, Ingmar: *Universal Approximation of Functions on Sets*. In: *CoRR*, volume abs/2107.01959, 2021.

[WHW09] Winner, Hermann; Hakuli, Stephan; and Wolf, Gabriele: *Handbuch Fahrerassistenzsysteme: Grundlagen, Komponenten und Systeme für aktive Sicherheit und Komfort*. Springer-Verlag, 2009.

[WLG+17] Wang, Peng-Shuai; Liu, Yang; Guo, Yu-Xiao; Sun, Chun-Yu; and Tong, Xin: *O-CNN: Octree-based Convolutional Neural Networks for 3D Shape Analysis*. In: *ACM Trans. Graph.*, volume 36, no. 4, 72:1–72:11, 2017.

[WLK+20] Wang, Sinong; Li, Belinda Z.; Khabsa, Madian; Fang, Han; and Ma, Hao: *Linformer: Self-Attention with Linear Complexity*. In: *CoRR*, volume abs/2006.04768, 2020.

[WSK+15] Wu, Zhirong; Song, Shuran; Khosla, Aditya; et al.: *3D ShapeNets: A Deep Representation for Volumetric Shapes*. In: *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2015, Boston, MA, USA, June 7-12, 2015*, pages 1912–1920, IEEE Computer Society, 2015.

[WWZ+21] Wang, Wei; Wang, Bing; Zhao, Peijun; et al.: *PointLoc: Deep Pose Regressor for LiDAR Point Cloud Localization*. In: *IEEE Sensors Journal*, 2021.

[XBK+15] Xu, Kelvin; Ba, Jimmy; Kiros, Ryan; et al.: *Show, Attend and Tell: Neural Image Caption Generation with Visual Attention*. In: *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015*. Volume 37, pages 2048–2057. *JMLR Workshop and Conference Proceedings*, JMLR.org, 2015.

[XFX+18] Xu, Yifan; Fan, Tianqi; Xu, Mingye; Zeng, Long; and Qiao, Yu: *SpiderCNN: Deep Learning on Point Sets with Parameterized Convolutional Filters*. In: *Computer Vision - ECCV 2018 - 15th European Conference, Munich, Germany, September 8-14, 2018, Proceedings, Part VIII*. Volume 11212, pages 90–105. *Lecture Notes in Computer Science*, Springer, 2018.

[XLCT18] Xie, Saining; Liu, Sainan; Chen, Zeyu; and Tu, Zhuowen: *Attentional ShapeContextNet for Point Cloud Recognition*. In: *2018 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2018, Salt Lake City, UT, USA, June 18-22, 2018*, pages 4606–4615, Computer Vision Foundation / IEEE Computer Society, 2018.

[YBT+19] Yang, Luwei; Bai, Ziqian; Tang, Chengzhou; et al.: *SANet: Scene Agnostic Network for Camera Localization*. In: *2019 IEEE/CVF International Conference on Computer Vision, ICCV 2019, Seoul, Korea (South), October 27 - November 2, 2019*, pages 42–51, IEEE, 2019.

[YLCT20] Yurtsever, Ekim; Lambert, Jacob; Carballo, Alexander; and Takeda, Kazuya: *A Survey of Autonomous Driving: Common Practices and Emerging Technologies*. In: *IEEE Access*, volume 8, pages 58443–58469, 2020.

[YVBW21] Yeong, De Jong; Velasco-Hernández, Gustavo Adolfo; Barry, John; and Walsh, Joseph: *Sensor and Sensor Fusion Technology in Autonomous Vehicles: A Review*. In: *Sensors*, volume 21, no. 6, page 2140, 2021.

[YZN+19] Yang, Jiancheng; Zhang, Qiang; Ni, Bingbing; et al.: *Modeling Point Clouds With Self-Attention and Gumbel Subset Sampling*. In: *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2019, Long Beach, CA, USA, June 16-20, 2019*, pages 3323–3332, Computer Vision Foundation / IEEE, 2019.

[ZC88] Zhou, Yi-Tong and Chellappa, Rama: *Computation of optical flow using a neural network*. In: *Proceedings of International Conference on Neural Networks (ICNN'88), San Diego, CA, USA, July 24-27, 1988*, pages 71–78, IEEE, 1988.

[ZKR+17] Zaheer, Manzil; Kottur, Satwik; Ravanbakhsh, Siamak; et al.: *Deep Sets*. In: *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pages 3391–3401, 2017.

[ZSC+12] Zhang, Feihu; Stahle, Hauke; Chen, Guang; et al.: *A sensor fusion approach for localization with cumulative error elimination*. In: *IEEE International Conference on Multisensor Fusion and Integration for Intelligent Systems, MFI 2012, Hamburg, Germany, September 13-15, 2012*, pages 1–6, IEEE, 2012.

[ZZD+20] Zhang, Yang; Zhou, Zixiang; David, Philip; et al.: *PolarNet: An Improved Grid Representation for Online LiDAR Point Clouds Semantic Segmentation*. In: *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2020, Seattle, WA, USA, June 13-19, 2020*, pages 9598–9607, Computer Vision Foundation / IEEE, 2020.

Own Publications

Parts of this thesis were pre-published on various conferences:

Engel, Nico; Belagiannis, Vasileios; and Dietmayer, Klaus: *Attention-based Vehicle Self-Localization with HD Feature Maps*. In: *24th IEEE International Intelligent Transportation Systems Conference, ITSC 2021, Indianapolis, IN, USA, September 19-22, 2021*, pages 76–83, IEEE, 2021.

Engel, Nico; Belagiannis, Vasileios; and Dietmayer, Klaus: *Point Transformer*. In: *IEEE Access*, volume 9, pages 134826–134840, 2021.

Engel, Nico and Dietmayer, Klaus: *Selbstlokalisierung fur automatisierte Fahrzeuge mit tiefen neuronalen Netzen*. In: *Uni-Das Workshop 2020*, 2020.

Engel, Nico; Hoermann, Stefan; Horn, Markus; Belagiannis, Vasileios; and Dietmayer, Klaus: *DeepLocalization: Landmark-based Self-Localization with Deep Neural Networks*. In: *2019 IEEE Intelligent Transportation Systems Conference, ITSC 2019, Auckland, New Zealand, October 27-30, 2019*, pages 926–933, IEEE, 2019.

Engel, Nico; Hoermann, Stefan; Henzler, Philipp; and Dietmayer, Klaus: *Deep Object Tracking on Dynamic Occupancy Grid Maps Using RNNs*. In: *21st International Conference on Intelligent Transportation Systems, ITSC 2018, Maui, HI, USA, November 4-7, 2018*, pages 3852–3858, IEEE, 2018.

As co-author:

Forstenhäusler, Marc; Engel, Nico; and Dietmayer, Klaus: *Temporal Filtering to Stabilize Features for SLAM*. In: *IEEE/ASME International Conference on Advanced Intelligent Mechatronics, AIM 2019, Hong Kong, SAR, China, July 8-12, 2019*, pages 1592–1597, IEEE, 2019.

Horn, Markus; Engel, Nico; Belagiannis, Vasileios; Buchholz, Michael; and Dietmayer, Klaus: *DeepCLR: Correspondence-Less Architecture for Deep End-to-End Point Cloud Registration*. In: *23rd IEEE International Conference on Intelligent Transportation Systems, ITSC 2020, Rhodes, Greece, September 20-23, 2020*, pages 1–7, IEEE, 2020.

Supervised Theses

Bachelor Theses:

Dehler, Nikolas: *Plausibilisierung von digitalen Karten und Sensordaten.* Bachelor thesis, Ulm University, 2020.

Raisch, Fabian Constantin Nepomuk: *Landmarkenbasierte Lokalisierung für das autonome Fahren mittels Convolutional Neural Networks.* Bachelor thesis, Ulm University, 2019.

Wittmann, Mario: *Erweiterte Methoden zur Plausibilisierung von digitalen Karten.* Bachelor thesis, Ulm University, 2021.

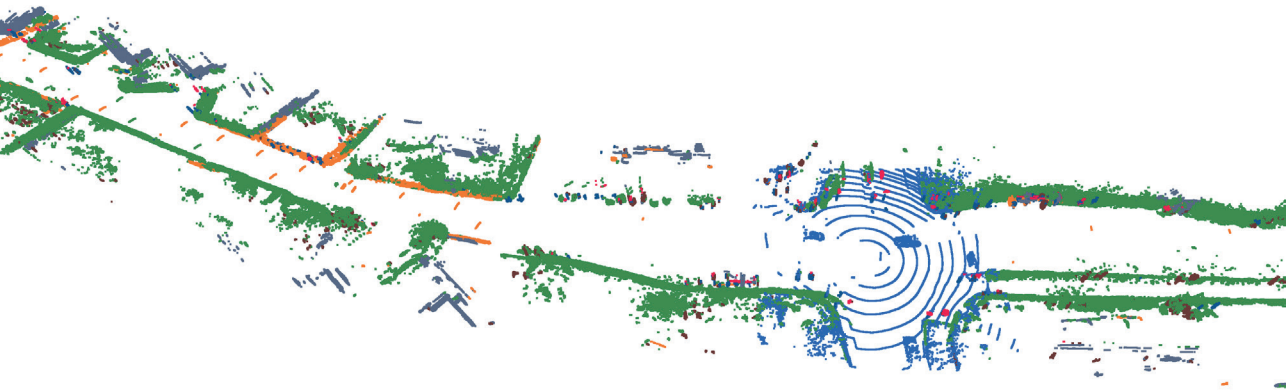
Master Theses:

Hertenberger, Johannes: *LiDAR Mapping mit tiefen neuronalen Netzen.* Master thesis, Ulm University, 2021.

Volz, Robin: *Verfahren zur landmarkenbasierten Selbstlokalisierung für das autonome Fahren mittels DeepLocalization.* Master thesis, Ulm University, 2020.

The curriculum vitae was removed in the online version for privacy reasons.

The curriculum vitae was removed in the online version for privacy reasons.



ISBN 978-3-941543-69-0

ISBN 978-3-941543-70-6 (E-Book)