universität ulm



Joint Multi-Object Tracking and Object Detection with Transformer Models

Felicia Ruppel

Band 60

MRM

Schriftenreihe des Instituts für Mess-, Regel- und Mikrotechnik

Felicia Ruppel

Joint Multi-Object Tracking and Object Detection with Transformer Models

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

Herausgeber:
Prof. Dr.-Ing. Klaus Dietmayer

Band 60

Felicia Ruppel

Joint Multi-Object Tracking and Object Detection with Transformer Models

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, 2025

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 A 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 2025
Print on Demand

ISBN 978-3-941543-99-7 ISBN 978-3-911223-00-3 (E-Book)



universität
uulm

Joint Multi-Object Tracking and Object Detection with Transformer Models

DISSERTATION

zur Erlangung des akademischen Grades einer

DOKTOR-INGENIEURIN

(Dr.-Ing.)

der Fakultät für Ingenieurwissenschaften, Informatik
und Psychologie der Universität Ulm

von

Felicia Ruppel
aus Frankfurt am Main

Gutachter:	Prof. Dr.-Ing. Klaus Dietmayer
	Prof. Dr.-Ing. Marcus Baum
Amtierender Dekan:	Prof. Dr. Matthias Tichy

Ulm, 07.04.2025

Acknowledgements

First and foremost, I would like to thank my academic supervisor, Prof. Dr.-Ing. Klaus Dietmayer, director of the Institute of Measurement, Control and Microtechnology at Ulm University, for the continuous support and valuable feedback to my work. His insights and experience have helped me grow and enabled me to do this work. I would also like to thank Prof. Dr.-Ing. Marcus Baum, head of the Data Fusion Lab at Georg-August-Universität Göttingen, for serving as the second examiner of my thesis, and for the interest in my work.

This work has been carried out in collaboration between Ulm University and Bosch Research. At Bosch, I would like to thank my department head, Axel Stamm, and former department head, Dr. Alexander Kleiner, for providing me an environment in which I could focus on my research, while being immersed in an innovative and fruitful setting. Also, I'd like to thank my group lead Dr. Yakov Miron, and former group leads Dr. Claudius Gläser and Dr. Fabian Timm for our valuable discussions, their advice, support and encouragement to collaborate and present my work.

A special thanks goes to my PhD advisor from the company's side at Bosch, Dr. Florian Faion, who has helped me find and shape this topic, has provided me with insightful input and feedback, and has accompanied me through the great and the difficult times. Especially in the beginning, in a time of contact restrictions, lockdown and home office, he has taken the time to make sure that I got the support and guidance that I needed.

At Bosch, I would also like to thank my colleagues Florian Drews, Jiaying Lin, Elizabeth De Benedictis, Annika Meyer, Rafael Salgado, Jasmine Richter, Faris Janjoš, Michael Ulrich, Marvin Klimke, Johannes Müller, Marcel Schreiber and many more for the insightful discussions and also for the fun activities outside of work, as well as the Master's students I (co-)supervised, especially Mathijs van Geerenstein and Yinzhe Shen, for their valuable contributions. I would also like to thank the colleagues at the MRM in Ulm for the discussions and exchange.

I'm thankful to my family and friends, my parents Wolfgang and Isolde, my brother Julian, as well as Rosi and Georg for motivating and accompanying me. Finally, a very special thanks goes to my partner Jonas, for his love and support, and for always being there for me.

Abstract

Autonomous driving has been an active field of research for many years, and may revolutionize the way humans and goods travel. Still, there are long ways to go towards fully autonomous vehicles that do not require human supervision. An autonomous driving pipeline typically consists of a perception, prediction, planning and control module. This work focuses primarily on perception, in particular object detection and tracking, though the presented methods may be scaled beyond that. Perception translates the measured sensor data into a representation of the world around the vehicle. In the case of multi-object tracking and detection, the observed environment is expressed as bounding boxes around nearby objects e.g. cars and pedestrians, each characterized by a location, shape and orientation, as well as a unique object ID that marks the same object over time.

The field of object detection has developed rapidly in the past decade, fueled by the success of deep learning (DL) methods. While developed for the two-dimensional (2D) image domain at first, the approaches were quickly advanced to 3D, and to multiple sensor modalities, such as radar and lidar. Since then, DL has become the standard approach to object detection and replaced classical methods in many applications, such as autonomous driving. Still, some questions are open, especially in safety-critical scenarios. How can sufficient performance be guaranteed, even in previously unseen situations?

Motivated by the success of learned models for object detection, this work aims to explore similar approaches in the field of multi-object tracking. Does a tracker benefit from the rich feature space that an object detector already has? Can the two components support one another and perform a detection with temporal consistency directly, i.e. joined object detection and tracking? To answer these questions, a deep detection and tracking model is developed and thoroughly analyzed. Different from commonly used hybrid methods, it does not have a narrow interface between a deep detector to a classical tracker, but allows both tasks full access to the encoded input sensor data, as well as the track history.

Transformer models are utilized to facilitate the deep tracking in this work, since their attention mechanism is a natural fit to the tracking problem: An entity, such as an existing track, can interact with the measured sensor information through

attention and extract the relevant information it needs to update its state. Therefore, the learned attention alleviates the need for an explicit data-to-track association. Besides this, observing the focus of attention may give insights into the model's uncertainty.

Main contributions of this work are the development of a transformer based object detector applicable to the autonomous driving use-case, i.e. it can cope with the necessarily large field of view, both regarding the perception output as well as the point cloud input. This method then serves as a basis for the joined detector and tracker that is developed in the following. Both models are evaluated in comparison to Bayesian trackers, and developed iteratively towards increased performance. Besides this, the attention spread is explored as a possible measure of detection and tracking uncertainty.

Kurzfassung

Bereits seit vielen Jahren ist autonomes Fahren ein Forschungsthema, das sich stetig weiterentwickelt und das Potenzial hat, die Mobilität zu revolutionieren, sowohl bezüglich des Waren- als auch des Personenverkehrs. Ein vollständig autonomes Fahrzeug, welches sich frei von menschlicher Kontrolle auf beliebigen Straßen fortbewegen kann, liegt allerdings noch in der Zukunft und erfordert erhebliche technische Fortschritte auf mehreren Gebieten. Die Verarbeitungskette für autonomes Fahren besteht üblicherweise aus einer Umfelderfassung (Perception), gefolgt von Prädiktions-, Planungs- und Steuerungs- (Control) Modulen. Die vorliegende Arbeit beschäftigt sich primär mit der Umfelderfassung, insbesondere mit Objektdetektion und Tracking. Dennoch lassen sich die vorgestellten Ansätze auch darüber hinaus skalieren. Die Perception umfasst das Verarbeiten der gemessenen Sensordaten in eine Repräsentation der Umgebung des Ego-Fahrzeugs. Im Fall von Objektdetektion und Tracking werden dazu üblicherweise Begrenzungsboxen (Bounding Boxes) geschätzt, die jeweils ein Objekt beschreiben. Eine solche Box besteht aus einer Position, Ausdehnung und Orientierung, sowie einer Identifikationsnummer, die dasselbe Objekt über die Zeit markiert. Außerdem kann sie einer Kategorie zugeordnet werden, beispielsweise Fahrzeugen oder Fußgängern.

Im vergangen Jahrzehnt hat sich das Gebiet der Objektdetektion maßgeblich weiterentwickelt, getrieben durch den Erfolg von Deep Learning (DL). Zunächst wurden DL-Methoden für Detektion im 2D-Bildbereich entwickelt, im Folgenden jedoch auch für dreidimensionale Anwendungsfälle portiert und mit verschiedenen Arten von Sensoren betrieben. In vielen Anwendungsbereichen haben DL-Modelle klassische Methoden für Objektdetektion ersetzt und können als Standard betrachtet werden, auch im Bereich autonomes Fahren. Dennoch sind manche Fragen bisher unzureichend erforscht, beispielsweise: Wie kann man solche Methoden in sicherheitskritischen Bereichen absichern?

Im Gegensatz zur Detektion wird Tracking üblicherweise weiterhin mit klassischen Verfahren gelöst. Die vorliegende Arbeit hat das Ziel zu erforschen, inwiefern auch Multi-Objekt-Tracking von Deep Learning profitieren kann: Kann ein Tracker von dem hoch-dimensionalen Feature-Space profitieren, den ein Detektor kodiert? Können die beiden Komponenten sich gegenseitig unterstützen und voneinander profitieren, sodass die Ausgabe von Detektionen mit zeitlich konsistenter Identität

in einem Modell ermöglicht wird? Um diese Fragen zu beantworten, wird ein gemeinsamer Detektor und Tracker mit DL-Methoden entwickelt und ausführlich analysiert. Im Gegensatz zu sonst üblichen hybriden Verfahren haben in diesem Modell sowohl der Tracker als auch der Detektor uneingeschränkten Zugriff auf die kodierten Sensordaten und die Track-Historie, sodass keine niedrig-dimensionale Schnittstelle zwischen den Komponenten benötigt wird.

Um Deep Tracking umzusetzen werden Transformer-Modelle verwendet, da deren Attention-Mechanismus eine Möglichkeit bietet das Assoziationsproblem abzubilden und zu lernen: Eine Entität, beispielsweise ein bestehender Track, kann durch Attention mit den Eingabe-Daten interagieren und die für ihn relevanten Informationen abgreifen, um seinen Zustand zu aktualisieren. Durch diesen Mechanismus ist keine zusätzliche explizite Daten-zu-Track Assoziation notwendig. Eine weitere Eigenschaft des vorgestellten Ansatzes ist, dass die gelernten Attention-Gewichte betrachtet werden können und Rückschlüsse auf die Unsicherheit der Beobachtung zulassen.

Diese Arbeit leistet mehrere wissenschaftliche Beiträge. Zunächst wird ein Transformer-basierter Objekt-Detektor entwickelt, der im Kontext von autonomem Fahren einsetzbar ist, d.h., dass er ein ausreichend großes Sichtfeld unterstützt sowie ausreichend große Punktwolken als Eingabe. Dieser Detektor dient im Folgenden als Grundlage für die Entwicklung eines gemeinsamen Detektors und Trackers. Beide Modelle werden im Vergleich zu bayesschen Tracking-Verfahren evaluiert, analysiert und schrittweise verbessert. Außerdem wird die Verteilung der Attention-Gewichte als möglicher Indikator für Detektions- und Tracking-Unsicherheit betrachtet und untersucht.

Contents

1 Introduction **1**

2 Fundamentals of Perception for Autonomous Driving **5**
2.1 Data Acquisition 5
2.1.1 Sensor Modalities 6
2.1.2 Public Datasets 6
2.2 Object Detection 8
2.2.1 Image-based 2D Detection 8
2.2.2 3D Detection in Urban Environments 11
2.2.3 Performance Metrics 12
2.3 Multi-Object Tracking 14
2.3.1 Bayesian Methods 14
2.3.2 Deep Learning based Methods 16
2.3.3 Performance Metrics 17
2.4 Downstream Tasks 19
2.4.1 Prediction 20
2.4.2 Planning 20

3 Fundamentals of Deep Learning Architectures **21**
3.1 Model Training and Inference 21
3.1.1 Loss Functions 22
3.1.2 Backpropagation and Optimization 23
3.2 Comparison to Conventional Statistical Methods 24
3.2.1 Feature Selection 24
3.2.2 Bias-Variance Trade-Off 25
3.3 Deep Learning Architectures 26
3.3.1 Feed-Forward Networks 26
3.3.2 Convolutional Neural Networks 27
3.3.3 Recurrent Models 27
3.4 Transformer Models 28
3.4.1 Attention Mechanism 30
3.4.2 Transformer Model Architecture 31
3.4.3 Training Peculiarities 31

4 A Joint Approach to Object Detection and Tracking 33
4.1 Motivation and Aim . . . 33
4.2 Transfer of Classical Tracking Principles . . . 34
4.2.1 Prediction and Update . . . 34
4.2.2 Robustness to Sensor and Synchronization Issues . . . 36
4.2.3 Sequential vs Stacked Input . . . 37
4.3 Novel Characteristics of a Deep Tracker . . . 38
4.3.1 Track Association Method . . . 38
4.3.2 Access to Latent Features . . . 39
4.3.3 High-Dimensional Track State . . . 40
4.4 Transformer-based Multi-Object Detection and Tracking . . . 41
4.4.1 Contributions to Object Detection . . . 42
4.4.2 Contributions to Tracking . . . 43
4.5 Conceptual Comparison to Filter-based Approaches . . . 43

5 Transformer-based Object Detection for Tracking 47
5.1 Motivation and Aim . . . 47
5.2 Input Data Encoding . . . 48
5.2.1 Backbone and Voxelization . . . 49
5.2.2 Accommodation of Large Input Sequence Length . . . 50
5.3 Object Embedding . . . 51
5.3.1 Learned Queries . . . 51
5.3.2 Sampled Anchor Locations . . . 52
5.4 Iterative Positional Refinement . . . 53
5.5 Object Extraction and Multi-Class Detection . . . 55
5.5.1 Unified Bounding Box Regression . . . 56
5.5.2 Multi-Group Regression . . . 57
5.6 Further Advances Towards Competitive Performance . . . 60
5.6.1 Data Augmentation . . . 60
5.6.2 Enhanced Backbone . . . 61
5.6.3 Feature-Informed Initialization . . . 62
5.7 Detection Evaluation . . . 64
5.7.1 Quantitative Results of TransLPC . . . 64
5.7.2 Query Refinement Evaluation . . . 66
5.7.3 Multi-Class Detection . . . 69
5.7.4 Feature-Informed Query Initialization . . . 73
5.8 Discussion . . . 75

6 Multi-Object Tracking Based on Attention 77
6.1 Track Life Cycle . . . 77
6.1.1 Training Mode . . . 78
6.1.2 Inference mode . . . 79

6.2 Ego-Motion Correction in Feature Space . . . 80
6.2.1 Method Overview . . . 81
6.2.2 Training Strategy . . . 83
6.3 Tracker Training Procedure . . . 84
6.3.1 Pair-wise Training . . . 84
6.3.2 Mixed Sequence- and Pair-wise Training . . . 85
6.4 Multi-Class Tracking . . . 87
6.5 Physics-Aware Deep Tracking . . . 88
6.5.1 Short-Term Prediction . . . 88
6.5.2 Orientation Queries . . . 89
6.5.3 Deterministic Ego-Motion Correction . . . 90
6.5.4 Track Update Layer . . . 91
6.6 Evaluation . . . 92
6.6.1 Quantitative Results of TransMOT . . . 92
6.6.2 Ego-Motion Correction in Feature Space . . . 94
6.6.3 Susceptibility to Skipped Frames . . . 96
6.6.4 Multi-Class Tracking . . . 97
6.6.5 Track Update Layer . . . 99
6.6.6 TransMOT v2 Evaluation . . . 101
6.7 Discussion . . . 104

7 Uncertainty Estimation from Attention Weights 109
7.1 Introduction . . . 109
7.2 Method Overview . . . 111
7.3 Definition of Attention Spread . . . 111
7.4 Evaluation . . . 113
7.4.1 Spatial Distribution . . . 114
7.4.2 Relation to Detection Accuracy . . . 114
7.4.3 Behavior during Tracking . . . 116
7.5 Conclusion . . . 117

8 Summary and Outlook 119

A Additional Parameters 123

B List of Publications 127

C Supervised Theses and Projects 129

Acronyms 131

List of Symbols 133

Bibliography 135

Chapter 1

Introduction

Autonomous driving has the potential to revolutionize human mobility: For individual transport, robot drivers can enable people to make better use of travel times, e.g. for work or relaxation. In addition, those people who are currently not able or allowed to drive a car could benefit from new possibilities, e.g. children. Commercial transportation, on the other hand, may become cheaper and more efficient. Besides this, autonomous driving aims to improve road safety and reduce the number of accidents. However, there is still a long road ahead in development: Currently, no fully autonomous system, which can drive on any road, is available. The most promising projects in this direction involve robo-taxis, which operate on selected, well-mapped areas, and autopilots for certain scenarios, e.g. on a highway when driving slower than 60 kilometers per hour [Heb22]. Notably, the latter liberates the driver from their liability in case of an accident, i.e. the driver does not need to supervise the autonomous system.

An autonomous system is commonly divided into multiple sub-tasks: *Perception* involves perceiving the world around the vehicle and detecting relevant structures and objects, as well as tracking them over time. *Prediction* aims to predict the surrounding agents' future motion, whereas *planning* reacts to the given information, deciding on an action for the autonomous vehicle and submitting it to *control* the car accordingly. This work focuses on the challenging perception task: Both the detection, as well as the tracking of relevant objects is of interest and builds the basis of all future components. 3D object detection commonly entails finding, localizing and classifying objects around the ego-vehicle. The result is a list of classified bounding boxes that contain a three-dimensional (3D) location, as well as size and orientation information for each object. In the scope of this work are pre-defined classes, such as pedestrians, cars and buses and others, whereas unseen object categories are not detected. In addition to this, a tracker provides a track identifier that is supposed to be consistent over time for each unique object entity, as well as, potentially, also a track uncertainty.

Oftentimes, the sub-tasks tracking and detection are researched disjunct from one another. In such a *chained system*, the tasks can be handled by different teams of experts with little communication among one another. However, such an approach entails unused potential: Firstly, the interface between the components in a chained system is typically narrow, e.g. low-dimensional bounding boxes, discarding information that the previous component had. Second, while object detection is commonly performed by deep learning (DL) based methods, classical algorithms are still very prevalent for object tracking. However, DL gained some traction this field more recently. How can a tracker benefit from the rich feature space of an object detector? Is it beneficial to use a broader interface between the two, or to combine them into a single model? How can some advantages of classical, Bayesian methods be retained?

This thesis proposes a modular, innovative and Bayesian-inspired deep multi-object detection and tracking method, which is called TransMOT and aims to answer the questions posed above, among others. TransMOT is based on a transformer model, which allows for a direct update of each track by accessing the current input data through cross-attention. This makes an explicit association between new measurements and previously detected objects obsolete, since the transformer learns to compute a soft association. Rather than performing detection first, tracking second, both tasks are performed at the same stage, simultaneously. The tracker has the potential to access any information from the sensor data that may be beneficial to the task, e.g. the appearance of specific entities of objects.

When this work was commenced, transformers had become popular for natural language tasks, such as translation and text generation. Researchers were starting to encompass their benefit also into other fields, such as computer vision. In this regard, both a detector as well as a tracker were developed to process camera data from stationary cameras [CMS+20; MKLF21]. However, such methods were not compatible with the challenges that come with autonomous driving, e.g. the need to cover a large field of view (FoV) and to compensate for the ego-motion when measuring from a moving car. One contribution of this work is to enable transformer-based joint object detection and tracking for point cloud data, suitable for the autonomous driving task, which utilizes lidar data that spans a range of ca. 100 times 100 meters.

The proposed method entails a track state for each track in form of a high-dimensional feature vector, which is called a track query. While such a vector does not contain explicit information, e.g. an object location, it encodes knowledge about its track implicitly. Therefore, it has the potential to capture more information than a simple bounding box; such as the track history or the appearance of the respective object. The propagated track queries are presented to the model in the following timestamp, allowing them to update their state with direct access to the measured sensor data.

Since the sensor data is measured from a moving platform, i.e. the ego-vehicle, the propagated track states need to be ego-motion corrected accordingly. This is a challenge in the high-dimensional feature space, where geometric rules do not apply. Therefore, a learned module is developed to perform the task, which is inspired from transformations in the 3D domain.

Many properties of Bayesian trackers are beneficial and shall not be lost, even when shifting to a DL paradigm. For this reason, TransMOT aims to be modular. It contains an update and a prediction step, and, while they are applied to high-dimensional feature vectors, it is possible to extract a bonding box from the track queries at any time, to follow their movement. Another challenge in a DL based system is that accuracy guarantees or uncertainties are difficult to obtain, whereas that is a strength of classical methods. However, transformers offer unique insights into their inner working when accessing the attention weights: For each forward pass, these weights show which input data was accessed to compute the result. To make use of this, an analysis of uncertainty based on attention weights is performed.

Over the course of this work, the proposed joined tracker and detector is improved continuously, which involves adding the capability to track multiple different classes, methods to increase detection accuracy, such as a location refinement, and incorporating some physical constrains that guide the model behavior. Besides this, findings from concurrent work are incorporated, as well as improved upon. The developed tracker is compared with classical methods, i.e. to a tracker based on a Kalman filter, as well as to a Poisson multi-Bernoulli mixture (PMBM) filter, which is based on random finite sets.

The thesis is structured as follows: Chapter 2 introduces the fundamentals of perception for autonomous driving, including object detection and tracking, as well as commonly used public datasets and evaluation metrics. In chapter 3, deep learning paradigms that are relevant to this work are described: Both a general understanding of learned methods as well as concrete model architectures are presented, that serve as building blocks for modern machine learning systems. This includes transformer models, as well as convolutional neural networks, among others. Chapter 4 presents the transformer based multi-object tracker and detector that is the main subject of this work. It aims to facilitate a higher-level understanding, and discusses the implicit track association and the high dimensional track state, as well as which aspects of the proposed methods are reminiscent of the classical domain. The following chapters, on the other hand, provide more technical detail: Chapter 5 focuses on the contributions towards object detection. A detector is developed that is applicable to autonomous driving, which begins as a proof of concept and is improved towards state-of-the-art (SOTA) performance. This detector serves as a basis for the joined detector and tracker that is detailed in chapter 6. All components are introduced that have been necessary to achieve competitive multi-class tracking performance, such as the

ego-motion compensation. Both chapters provide experimental results that provide insights into the model behavior, as well as allow for a comparison with alternative approaches. Finally, chapter 7 explores the attention weight behavior within the transformer and determines whether its characteristics can serve as a measure of uncertainty. For this, a novel indicator is introduced, which is called attention spread. The thesis is concluded with a summary and an outlook in chapter 8.

Chapter 2

Fundamentals of Perception for Autonomous Driving

The perception task involves reasoning about the environment, which is a skill that humans learn at a young age without explicit training, whereas it does not come as naturally to machines. This chapter introduces the fundamentals of machine perception for autonomous driving as well as related work in this field. A perception module extracts information about the environment from sensor data. This entails reasoning about the existence, type, location, orientation and size of objects as well as their movement over time. Some methods additionally estimate free-space regions and unobserved areas or perform a semantic segmentation, i.e. a categorization of each data point in the sensor image. Since the first step in any perception processing chain is sensor data acquisition, it is introduced in the following. After this, the chapter focuses on the tasks object detection and tracking and finally gives an outlook to further downstream tasks in an autonomous driving pipeline.

2.1 Data Acquisition

An autonomous vehicle uses sensors to perceive the world around it. While companies oftentimes acquire their own datasets, some are publicly available and can be used to train and evaluate perception methods as well as to achieve comparability between different approaches.

2.1.1 Sensor Modalities

Different types of vehicle-mounted sensors are common: Cameras measure a dense picture of the environment, but do not directly capture depth information. Light detection and ranging (lidar) sensors, on the other hand, emit laser light and observe the time of return, therefore being able to measure depth. For autonomous driving, spinning lidars are commonly used, so that they cover a FoV of 360 degrees horizontally, while vertical coverage is achieved with fixed scan lines, e.g. at 32 different vertical angles [LI20]. The sensor's output is a point cloud, i.e. a set of 3D points, typically with an intensity value associated with each of them. Another common sensor modality is a radio detection and ranging (radar) system, which transmits electromagnetic waves in the radio frequency range and receives the resulting echo. Radar devices installed in vehicles are typically frequency-modulated continuous-wave (FMCW) radars, which are able to measure radial velocities as well as distances to surrounding objects. Such measurements can be processed into a point cloud, though a much sparser one compared to lidar, with a smaller angular accuracy [KJD18]. To their advantage, on the other hand, radar sensors are commonly less expensive than lidar sensors and are less affected by fog and rain.

This work mainly relies on lidar sensor data to perform perception, because of their relatively dense image, combined with an accurate depth measurement. Note that, due to the similar representation as a point cloud, deep learning methods for lidar signals can also be applied to radar point clouds, with some adaptations. Besides this, one can think of a multi-sensor setup, in which the different modalities are fused and the sensors complement each other. The proposed perception method is compatible with such a fusion setting, which is explored in section 5.6.3.

2.1.2 Public Datasets

Deep learning methods require a large amount of training data, which can be categorized into labeled and unlabeled data. The latter only contains raw sensor measurements and is very easy to produce in large amounts by driving with a sensor-equipped vehicle. It can be used to train deep learning methods in a self-supervised fashion. However, for current state-of-the-art (SOTA) perception methods, at least a certain amount of supervised training is necessary. To create a labeled dataset for this, ground truth annotations are added manually to the sensor data, which is both tedious and expensive to perform.

Publicly available datasets benefit the research community by providing such labeled data, as well as serve as a benchmark to compare the performance of different

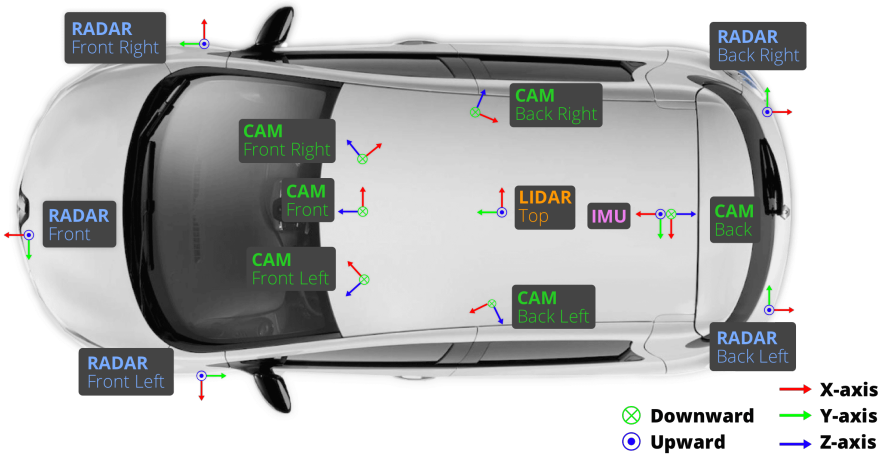


Figure 2.1: Sensor setup of the nuScenes dataset. [CBL+20] © 2020 IEEE

methods. The most common public datasets for general perception tasks are the Waymo Open Dataset [SKD+20], nuScenes [CBL+20] and KITTI [GLSU13]. KITTI was one of the first urban driving datasets of considerable size and features 1.5 hours of driving time, though more recent perception methods require a larger set of training data, either from nuScenes (5.5 hours) or the Waymo dataset (6.4 hours). nuScenes contains 40k labeled frames at a rate of 2 Hz and Waymo has 230k labeled frames at a rate of 10 Hz. The latter differentiates between 4 types of object classes, while nuScenes accommodates 23 different classes. A labeled frame in this context refers to a timestamp, for which ground truth bounding boxes are available, as well as synchronized sensor data. Since this work mainly uses the nuScenes dataset and especially their lidar sensor, it is described in more detail in the following.

The data was acquired with two Renault Zoe cars with a sensor setup as pictured in Figure 2.1. Both left- and right-hand traffic was captured in Singapore and Boston, respectively. The lidar sensor is a Velodyne HDL32E mounted on top of the vehicle, which captured sweeps at 20 Hz. Each frame contains 32 beams with ca. 1080 points per horizontal line (360° horizontal FoV, +10° to −30° vertical FoV). Its range is up to 100 meters, though the region of interest for both the object detection and tracking task is limited to 50 meters, i.e. the performance metrics are only computed in this range. Since the dataset is labeled at 2 Hz, it is common practice to aggregate 10 lidar sweeps into one frame, which both creates a denser point cloud as well as allows to make use of the temporal context of 0.5 seconds [CBL+20]. For the aggregation, each point is ego-motion corrected and marked with a time index. From the temporal context, object velocities can be estimated, even with 'single'-frame

object detectors. The following classes are annotated for the object detection task: car, bus, truck, trailer, construction vehicle*, pedestrian, bicycle, motorcycle, traffic cone* and barrier*. Those marked with * are excluded in the nuScenes *tracking* challenge. Only objects that have been observed with at least one lidar or radar point are annotated.

The ego-vehicle is localized with both a global positioning system (GPS) and an inertial measurement unit (IMU). To achieve an accuracy of 20 mm for the position as well as 0.2° for the orientation of ego-poses, a detailed map of lidar points was created in post-processing and combined with the IMU and GPS measurements [CBL+20]. The dataset is split into a `train`, `val` and `test` set with 700, 150 and 150 scenes, respectively. For the latter, the annotations are not publicly available, so that models can only be evaluated on it by uploading to the nuScenes test server. Therefore, it is common practice to train on the `train` split and evaluate on the `val` split.

2.2 Object Detection

The task *object detection* refers to finding and localizing objects of interest in a measured sensor frame. Early methods used signal processing methods to detect objects, e.g. in a radar system with a matched filter [Woo14]. In recent years, however, deep learning methods have become leading in the field. This chapter provides an introduction into state-of-the-art (SOTA) object detection approaches as well as their history, i.e. the methods that they were based on.

2.2.1 Image-based 2D Detection

Image Classification For many years, neural networks were a known technology, but they performed inferior to well-known, classical signal processing methods. One of the networks that disrupted the field was AlexNet [KSH12], which outperformed existing methods on the image classification benchmark ImageNet [DDS+09] by a large margin. Its main contribution was the combination of known components, such as convolutional layers and fully connected layers, into a *deeper* network (i.e. a stack of multiple layers) and to implement the network efficiently on a GPU. The task *image classification* is a preliminary stage of object detection, since each image to classify is known to picture exactly one object, whereas the type of object is to be determined. The network's output is therefore a classification score to distinguish between the known list of classes, but no localization of the objects is performed.

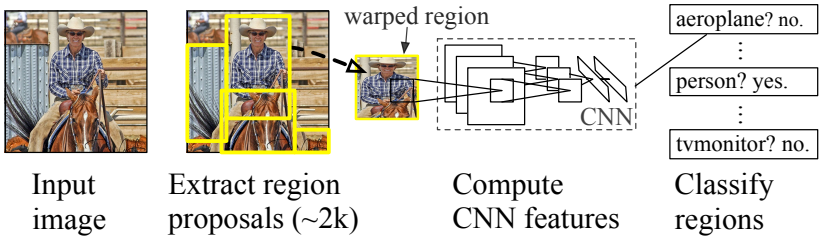


Figure 2.2: Overview of the R-CNN object detector [GDDM14] © 2014 IEEE

Object Detection Convolutional neural networks (CNNs) like AlexNet can be utilized to also perform object detection in the following way [GDDM14], which is pictured in Figure 2.2: Multiple regions of interest in an image are determined first, each with different positions, shapes, and sizes, using a region proposal algorithm. These can be understood as candidate boxes. Then, a neural network, e.g. AlexNet, is applied to each candidate region, to determine what type of object it contains (if any). Again, for this small region, the task is image classification, as well as computing a small location offset to position the box around the object. Bounding boxes for 2D object detection are axis-aligned rectangles:

$$\boldsymbol{b}_i = (u, v, w, h, \texttt{cls}) \,, \tag{2.1}$$

where u, v denote the box center in image coordinates, and w, h are the box's width and height. All of them are estimated relative to the initial anchor box. `cls` contains the class scores. R-CNN [GDDM14] requires about 2000 candidate regions, as well as a comparably long runtime, because each proposal region is passed through a network individually. [Gir15] improved on this by feeding the full image through a convolutional encoder to obtain a feature map. This is advantageous because the time-intensive feature-encoding with a CNN only has to be done once. The regions of interest, which can overlap, are then cut from the feature map, rather than the image. Another iteration is Faster R-CNN [RHGS15], which entails a trained region proposal network to find the regions of interest on the feature map, while previous methods still used classical methods for this. All approaches mentioned so far are considered *two-stage* approaches, because they consist of a proposal stage as well as a refinement stage.

YOLO (you only look once) [RDGF16] follows a *one-stage* concept: Its regions of interest are pre-defined by a regular grid of size $S \times S$, $S \in \mathbb{N}$ and by placing multiple candidate boxes at each grid center. While this method is faster than previous *two-stage* approaches, it struggles with some types of objects, such as small

ones. Which paradigm to follow is still subject to debate. Besides, there exist methods that can be considered 'in-between' one- and two stage methods, such as CenterNet [ZWK19], which uses keypoint detection to find probable object locations, and regresses a bounding box from there in a joined step.

Transformer Methods While CNNs have been considered the method of choice for object detection for a long time, the transformer architecture (cf. section 3.4) led to novel approaches. The detection transformer (DETR) replaces the anchor boxes with learned vectors, which they call *object queries*. Through the transformer's *cross-attention*, each query can access the input feature map in locations that were deemed interesting via an attention score. The queries are not bound to a location, but can, potentially, find an object anywhere on the image. It can, however be observed that queries each tend to develop a certain area of interest during the training. This method is *one-stage* in nature. Note that it still features a CNN backbone to compute the feature map. One of its benefits is that it eases the region proposal step with learned queries, which require no manual tuning. Besides this, the transformer allows an interaction of the queries among one another (*self-attention*), that enables them to suppress duplicate boxes, which would otherwise need to be removed in a post-processing step via non-maximum suppression (NMS).

Transformers are, however, known for their need of large amounts of data, as well as their slow convergence speed, which is also a shortcoming of DETR. One way to remedy this is DeformableDETR [ZSL+21], which replaces the cross-attention by an approximation called deformable attention. Rather than allowing each query access to the full feature map, deformable attention limits this to a set number of locations around the query location. Bounding boxes are estimated relative to the query locations. While this approach achieves superior convergence time as well as performance to DETR, it does so by constricting the model, posing the question if this superiority is mainly caused by a limited amount of training data and compute resources.

EfficientDETR [YALZ21] is a DETR-variant that belongs to the *two-stage* approaches: In a first stage, a heatmap is computed from the feature map, following Centerpoint [YZK21], and queries are placed at its peaks, as these are probable object locations. The authors find that they need fewer transformer decoder layers to refine the object locations compared to other, less informed, query placement.

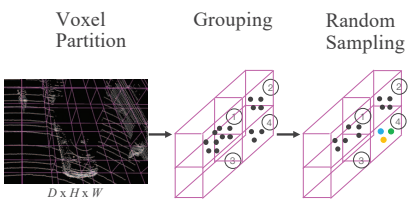
The joint detection and tracking model, that is the subject of this work, is based on the line of work started by DETR. Therefore, some of the mentioned design aspects are going to be discussed in more detail in the later chapters.

2.2.2 3D Detection in Urban Environments

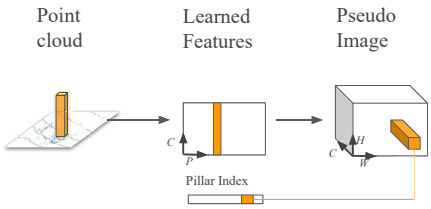
In contrast to 2D, 3D object detection requires detecting bounding boxes with more degrees of freedom. Besides a location in 3D and a width, length and height, an orientation is estimated so that the boxes are not axis-aligned. Typically, the orientation is expressed as a yaw angle, i.e. a rotation in the bird's-eye view (BEV) plane. As introduced above, multiple different sensor modalities are feasible for this task: Lidar and radar sensors measure 3D point clouds with explicit depth information. Cameras on the other hand do not, though 3D detection is still possible via context information, multi-modal sensor fusion and/or access to multiple perspectives.

In recent history, most 3D detection approaches are built upon knowledge from the 2D domain. Different concurrent lines of work exist to achieve this: Early works [CMW+17] use projections of a point cloud onto the BEV to then process them with 2D CNNs. In order to not loose too much height information, they slice the point cloud into multiple height groups and process a projected image from each. Why should one not keep the 3D structure and use 3D convolutions? It is possible to divide the field of view into voxels, i.e. 3D pixels, and to move a 3D filter on it to perform convolutions. This is, however, very inefficient. VoxelNet [ZT18] remedies this by only processing non-empty voxels in a sparse tensor, since the authors argue that 90% of voxels typically do not contain any points (cf. Figure 2.3 left). Another option is to find a more suitable point cloud representation for efficient convolutions. One example for this is PointPillars [LVC+19], pictured in Figure 2.3 (right), which divides the 3D space into a 2D grid observed from the BEV. This means that points belonging to the same cell can have different heights, forming pillars. The height dimension is not explicitly modeled by the cell structure, but rather encoded in feature space, resulting in a 2D grid of feature vectors that can be processed with standard CNNs similar to those designed for the image domain. PointNet++ [QYSG17] omits a grid structure entirely but aggregates and encodes nearby points via a distance-based thresholding, therefore grouping spheres of neighboring points. Which approach is preferable depends, since a voxel-based strategy tends to yield better performance at the cost of inference speed and methods using 2D convolutions tend to achieve less accurate results, but faster.

The 3D detection methods mentioned so far follow the *one-stage* paradigm. Since they lack prior information, they cover the field of view densely with candidate (anchor) boxes of varying orientation and sizes. Typically, one anchor box per class and orientation is placed in each grid cell of 30 to 80 centimeters side length. However, transformer models broke with this tradition, since they replace anchor boxes with more versatile object queries that can move farther distances so that fewer of them are required to cover the entire space. After transformer models became popular for



Voxelnet [ZT18]. © 2018 IEEE



PointPillars [LVC+19]. © 2019 IEEE

Figure 2.3: Examples of point cloud discretization methods. *Left:* 3D voxel-based grouping and sub-sampling. *Right:* Transformation to a 2D pseudo-image via separation into pillars.

2D detection, this concept has been transferred to 3D [RFGD22b; MGJ21]. While it is feasible for smaller point clouds, such as those measured indoors in [SLX15], to employ a full transformer-based pipeline, larger point clouds are necessary for urban object detection, e.g. for autonomous driving. This poses a challenge, as well as the small size of objects relative to the field of view, which can be more than a hundred meters wide and long. This work introduces one of the first transformer-based 3D detection methods that is feasible for autonomous driving lidar data, which is presented in chapter 5. It uses a transformer decoder head for object detection combined with a feature encoder, such as PointPillars [LVC+19] or VoxelNet [ZT18].

2.2.3 Performance Metrics

Average precision (AP) is the most common metric for object detection. To compute this metric, an assignment between the ground truth (GT) and the detections needs to be performed first. It classifies each detection as a true positive or false positive. Unmatched GT objects are considered false negatives. Since objects observed in urban scenes are small relative to the measurement range, nuScenes [CBL+20] opts for a distance-based metric for matching: Those objects with the smallest 2D center distance to a GT object as observed from BEV are matched, up to a certain maximum distance d. The number of true and false positives as well as false negatives are counted. After obtaining the matching, a *precision* metric is computed as follows:

$$\mathrm{Pr} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FP}}, \tag{2.2}$$

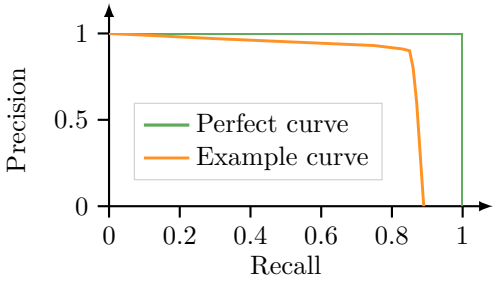


Figure 2.4: Example of a precision-recall curve. Increased recall typically comes at the cost of decreased precision, except for a perfect curve, as depicted in green.

where TP and FP denote the number of true and false positives, respectively. The *recall*

$$\mathrm{Re} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FN}}, \tag{2.3}$$

on the other hand, describes the share of GT objects that were successfully detected. Note that both precision and recall can be varied by setting a confidence threshold, above which detections are considered. Intuitively, a higher (stricter) confidence threshold entails fewer boxes in total, with lower recall but higher precision, while including low-confidence boxes can achieve a higher recall at the cost of more false positives. An example precision-recall curve is pictured in Figure 2.4. To achieve comparability, despite this ambiguity, precision scores are computed for multiple recall values, and averaged to obtain the average precision AP_d for the given matching distance d. In the nuScenes evaluation, four different matching distances are defined, $d \in \{0.5, 1, 2, 4\}$ meters. The results from each are averaged to compute the average precision (AP). AP is computed for all object classes of interest and, again, averaged, which is called mean average precision (mAP).

Besides the well-known mAP metric, nuScenes has introduced a custom detection score, which is named nuScenes detection score (NDS). It aims to quantify the *quality* of true positive detections in terms of their box parameters, such as location, orientation, size and velocity, in addition to the mAP and is defined as follows [CBL+20]:

$$\mathrm{NDS} = \frac{1}{2}\mathrm{mAP} + \frac{1}{10} \sum_{\mathrm{mTP} \in \mathcal{T}_\mathcal{P}} \left(1 - \min\left(1,\ \mathrm{mTP}\right)\right). \tag{2.4}$$

The sum on the right includes a set of five true positive metrics $\mathrm{mTP} \in \mathcal{T}_\mathcal{P}$, which are the average translation, scale, orientation, velocity and attribute errors and are each computed from the true positive detections compared to their matched GT

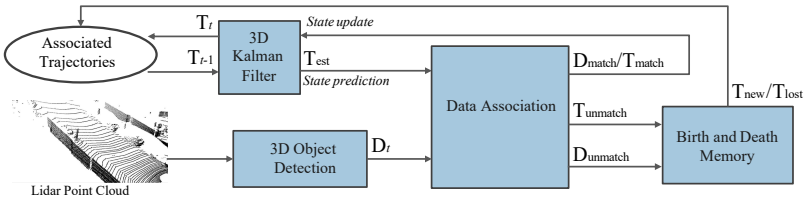


Figure 2.5: Overview of AB3DMOT [WWHK20], a tracker based on a Kalman filter that uses detections from a DL detector as input. © 2020 IEEE

boxes per class, then averaged over all classes. Their units are omitted in the sum.

2.3 Multi-Object Tracking

For autonomous driving, both past and current information about dynamic objects in the vicinity of the ego-vehicle is required. Multi-object tracking (MOT) aims to estimate the state of dynamic objects over time, such as, their location and velocity, as well as their cardinality, i.e. how many objects exist, given the current measurements and the accumulated history. Since observed objects can leave the field of view and new ones can enter, the respective tracks have to be terminated and newly spawned accordingly, also known as object death and birth. Bayesian methods are commonly used to model the MOT problem and are introduced in the following. More recently, MOT with deep learning has become a field of research, which is described afterwards.

2.3.1 Bayesian Methods

MOT is a complex problem: Measurements are reliable up to a certain uncertainty only. Given n_0 detected objects, how many and which of them actually exist and where are they located? When new measurements are obtained, does a measurement belong to an object that has been observed before, or to a new one, or is it a false alarm? Modeling all possible hypotheses to compute an optimal solution becomes intractable quickly [Mah14].

Since the true state $\boldsymbol{x}_k$ of a tracked object at time k is not observable, a Bayesian filter estimates its posterior distribution $p_k(\boldsymbol{x}_k|\boldsymbol{z}_{1:k})$, given measurements up to

the current time, $\boldsymbol{z}_{1:k}$. It operates recursively, i.e. $p_k(\boldsymbol{x}_k|\boldsymbol{z}_{1:k})$ is predicted to the next time step, denoted by $p_{k+1|k}(\boldsymbol{x}_{k+1}|\boldsymbol{z}_{1:k})$. Then, it is updated given the novel measurement, obtaining $p_{k+1}(\boldsymbol{x}_{k+1}|\boldsymbol{z}_{1:k+1})$. MOT can be implemented with multiple Kalman filters (one filter per tracked object) [Kal60; CPLB20]. This approach is commonly used in practice [WWHK20; CPLB20] and an example of it is pictured in Figure 2.5: A DL object detector outputs detections D_t. They are matched to the predicted tracks $\mathrm{T}_{\mathrm{est}}$ from a Kalman Filter, using a distance-based cost. Then, the tracks that were matched are updated with the Kalman filter equations, while the others are saved in memory to be terminated. Unmatched detections on the other hand lead to the birth of a new track. In this example [WWHK20], a constant velocity motion model is used. While such Kalman filter trackers are useful for many applications, they have some drawbacks: Heuristics are needed to associate the measurements to existing objects, to spawn new tracks and to terminate expired ones. The number of existing objects is not modeled as a random variable, but rather a byproduct of the applied heuristics.

The random finite set (RFS) theory allows to model the cardinality of a set as a random variable [Mah14]. The tracked objects are part of a set that contains a finite, though unknown, number of state vectors. The posterior density of this set of object states $\mathcal{X} = \{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(n)}\}$ can be modeled as a whole, i.e. $p_k(\mathcal{X}_k|\mathcal{Z}_{1:k})$. Optionally, dependencies between the objects may also be taken into account. Multiple different approaches exist to perform MOT with RFS, and some of them are presented exemplarily in the following.

A *Bernoulli RFS* is a type of RFS that contains either one object or is empty, with probabilities r and $1-r$, respectively. Its probability density is [Mah14]:

$$\pi(\mathcal{X}) = \begin{cases} 1-r & \mathcal{X} = \emptyset \\ r \cdot p(x) & \mathcal{X} = \{\boldsymbol{x}\}, \end{cases} \tag{2.5}$$

i.e., if the set has one element $\boldsymbol{x}$, it is distributed according to $p(x)$. Multiple independent Bernoulli RFS $\mathcal{X}^{(i)}$ can be combined into a *multi-Bernoulli RFS* $\mathcal{X} = \mathcal{X}^{(1)} \cup \cdots \cup \mathcal{X}^{(n)}$, which is parameterized by $\{\big(r^{(i)}, p^{(i)}\big)\}_{i=0}^{n}$. In a *labeled multi-Bernoulli (LMB)* RFS, those Bernoulli components that are not empty are augmented with a label (consistent over time). Such an LMB component can be used to describe one multi-object hypothesis, e.g. one of many possible measurement-to-object assignments. A generalized labeled multi-Bernoulli (GLMB) filter uses multiple LMB components to model different hypotheses, though the number of hypotheses grows exponentially with the number of track labels [VV13].

For usage in practice, exponential growth is not feasible, therefore approximations and pruning are necessary. One such approximation is the LMB filter [RVVD14],

which approximates the posterior density with an LMB that matches the first moment of the GLMB density. Another method is called Poisson multi-Bernoulli mixture (PMBM) filter [GWGS18]. Different from an LMB filter, it uses a Poisson point process (PPP) birth model. The authors show that the PMBM is equivalent to a δ-GLMB when the hypotheses are expressed in a more elaborate way (in which each of them has a deterministic cardinality), the Poisson birth component is removed, and labels are added.

2.3.2 Deep Learning based Methods

In recent years, deep learning methods have dominated many research fields. As mentioned above, one example is object detection, where deep models have de facto become the standard. When tracking objects is of interest, however, one common approach is to use a deep *detector* first and feed its output into a conventional tracker. This chained approach has some benefits: It is modular, so that one can, e.g., conveniently swap the detection component with a different detector and benefit from latest developments. Any of the aforementioned Bayesian trackers can be used in a chained system with a deep detector.

However, the interface between the chained detector and tracker is low-dimensional compared to the rich latent feature space of a deep detector. Therefore, some of the detector knowledge is withheld from the tracker, e.g. information about the visual appearance of an object. Most similar to a chained approach, Chiu *et al.* [CLAB21] propose to allow for a higher-dimensional detector output, i.e. latent features in addition to bounding boxes. These latent features provide an additional cost-measure for the data association task, so that a match between an existing track and a new detection is more likely if they have a more similar latent feature representation.

Other researchers propose to combine tracking and detection into a joined task and solve it with one trained model. Fast and Furious [LYU18] operates on a stack of N measured frames, i.e. the current one in addition to $N-1$ past ones. Tracking is performed by predicting each detected agent one step into the future and then matching them to new measurements in a greedy (distance-based) fashion. While this method breaks the narrow interface between detection and tracking, it suffers from inefficiency due to its windowed processing: Each measured frame is processed N times; once the first time it is measured and then $N-1$ times as a part of historic context information. TrackFormer [MKLF21], on the other hand, follows a sequential approach: A measured frame is processed once when it is obtained. The model encodes historical information about each track in feature space. It is a temporal extension to the concept of DETR [CMS+20] and uses a transformer model. While

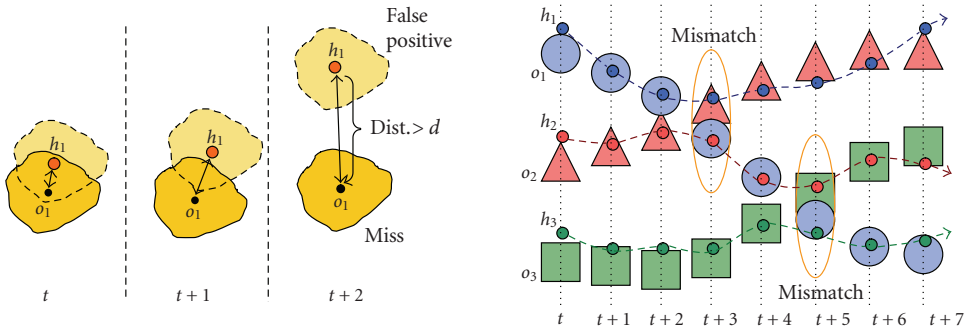


Figure 2.6: *Left:* Matching between an estimate (h_1) and a GT object (o_1). *Right:* Illustration of identity switches. [BS08] CC BY 2.0

TrackFormer is only suitable for image data obtained from a stationary platform, this thesis builds upon its concept and provides solutions and methods to apply it for autonomous driving.

2.3.3 Performance Metrics

To evaluate multi-object tracking, both the precision and accuracy of tracked objects compared to the GT is of interest, as well as their temporal consistency. The ClearMOT metrics were first proposed in [BS08]. They define a multi-object tracking accuracy (MOTA) as well as a precision. For this, the GT is matched to the estimates so that each estimate can be classified either as a true positive (TP) or a false positive (FP), cf. Figure 2.6 (left), given a certain maximum matching distance d. Missed GT objects are classified as false negatives (FN). Besides this, identity switches (IDS) are counted. They occur when a GT track is not covered by a single estimated track, but when its identity switches, which is illustrated in Figure 2.6 (right). The GT matching is determined so that the MOTA metric is maximized, defined as follows:

$$\text{MOTA} = 1 - \frac{\text{FP} + \text{FN} + \text{IDS}}{\text{TP} + \text{FN}}, \tag{2.6}$$

where TP + FN is the total number of GT objects. Note that this metric can become smaller than zero, e.g. if a lot of false positives occur. Besides MOTA, the multi-object tracking precision (MOTP) has been defined, which quantifies the

precision of GT-matched boxes and does not take temporal consistency into account:

$$\text{MOTP} = \frac{\sum_{(i,t)\in\mathcal{S}_{\text{TP}}} d_{i,t}}{\text{TP}}. \tag{2.7}$$

$\mathcal{S}_{\text{TP}}$ denotes the set of true positives and $d_{i,t}$ is their distance to GT. Similar to the object detection metrics, an average multi-object tracking accuracy (AMOTA) and average multi-object tracking precision (AMOTP) is obtained by averaging MOTA and MOTP at different recall values [WWHK20]. To achieve higher recall, more low-confident tracks are incorporated into the evaluation, typically resulting in both more true positives and more false positives. A lower recall, on the other hand, tends to achieve better precision at the cost of more missed detections or tracks. Through the averaging, AMOTA and AMOTP aim to remove the metrics' dependency on the recall value. The nuScenes [CBL+20] tracking challenge uses AMOTA and AMOTP as main tracking metrics, whereas they base them on a modified MOTA that does not reach below zero:

$$s\text{MOTA}_r = \max\left(0, 1 - \frac{\text{FP}_r + \text{FN}_r + \text{IDS}_r - (1-r)(\text{TP}_r + \text{FN}_r)}{\text{TP}_r + \text{FN}_r}\right), \tag{2.8}$$

where r is a given recall value. To obtain nuScenes' AMOTA, $s\text{MOTA}_r$ is averaged over L equidistant recall values

$$\text{AMOTA} = \frac{1}{L} \sum_{r\in\{\frac{1}{L},\frac{2}{L},\cdots,1\}} s\text{MOTA}_r, \tag{2.9}$$

just like AMOTP

$$\text{AMOTP} = \frac{1}{L} \sum_{r\in\{\frac{1}{L},\frac{2}{L},\cdots,1\}} \text{MOTP}_r. \tag{2.10}$$

In the remainder of this thesis, 'AMOTA' refers to the nuScenes metric as defined in Eq. (2.9). Some researchers argue that the aforementioned metrics fail to quantify long-term temporal consistency, since IDS are counted, which only take a short-term context into account. The higher order tracking accuracy (HOTA) metric [LOD+21] aims to remedy this by quantifying a longer context, though it has not found widespread use so far.

In the realm of classical Bayesian tracking, AMOTA and AMOTP are not common. Instead, tracking metrics are based on the optimal sub-pattern assignment (OSPA) distance. This distance compares two sets of point targets, i.e. the location estimates and the GT at a given time, as illustrated in Figure 2.7: The components of the two sets are matched to minimize a total cost, e.g. based on distance, which can be solved with the Hungarian method [Kuh55]. OSPA then consists of a cardinality error component that punishes missed detections, and a localization error component

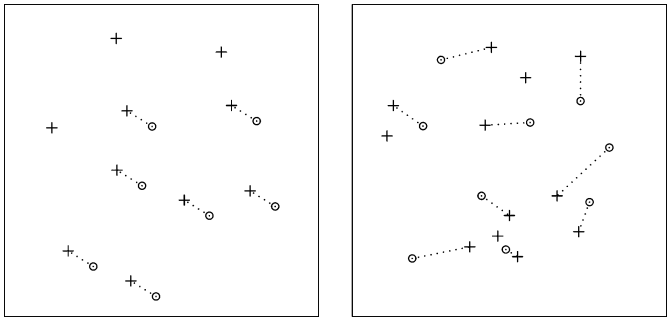


Figure 2.7: Examples of optimal assignments between two sets computed for the OSPA metric. While the left one could be matched by hand, the right one has been obtained with the Hungarian method. [SVV08] © 2008 IEEE

that describes how well the true positives were estimated [SVV08]:

$$\text{OSPA} = \left[\frac{1}{|\mathcal{Y}|} \left(\min_{\pi \in \Pi} \sum_{i=1}^{|\mathcal{X}|} d^{(c)}(x_i, y_{\pi(i)})^p + c^p \left(|\mathcal{Y}| - |\mathcal{X}| \right) \right) \right]^{1/p}, \tag{2.11}$$

where $\mathcal{X}$ and $\mathcal{Y}$ are the sets of estimates and GT, $|\cdot|$ is their cardinality, whereas $|\mathcal{Y}| \geq |\mathcal{X}|$, and $d^{(c)}(x, y) = \min(d(x, y), c)$ is the Euclidean distance between x and y up to a maximum distance c. Π is the set of possible assignments between $\mathcal{X}$ and $\mathcal{Y}$. In its original form, OSPA does not evaluate tracking performance over time, i.e. the temporal consistency of track labels is not considered. To extend this metric for tracking, [BVV17] introduce a metric called OSPA$^{(2)}$, which considers temporal context by defining a second order OSPA on a set of tracks over a period of time, while the distance between the tracks is defined as an averaged standard OSPA. OSPA for tracks [RVC10], on the other hand, introduces a track label error that punishes track identity switches. Both of them can also be computed in combination with a generalized optimal sub-pattern assignment (GOSPA) [RGS17], which is not normalized by the number of elements in the larger set.

2.4 Downstream Tasks

Both object detection and tracking are essential tasks for the following autonomous driving pipeline. While the full pipeline is not subject of this work, it is valuable to

reflect on the impact of the proposed model on the following components. Therefore, they are briefly introduced in the following. Besides this, the presented method can be extended to include further downstream tasks, taking the idea of end-to-end learning to the next level.

2.4.1 Prediction

This task entails the prediction of future trajectories of all surrounding agents of interest. Since this is an inherently stochastic problem, multiple hypotheses or modes are typically estimated. For one agent, the result may be that they are expected to drive straight ahead with a certain probability, or turn left or right. Each hypothesis is accompanied with a list of estimated waypoints for a given time horizon. Similar to the interface between detection and tracking, the interface between tracking and prediction is typically narrow. This means that, commonly, only track bounding boxes in combination with a track identity is propagated. The tracking model that is presented in this work, however, has the potential to also enable a broader interface to following components, which is discussed in chapter 6.

2.4.2 Planning

During planning, the ego vehicle's own trajectory is decided upon. For this, both the perception information and its respective prediction is taken into account as well as the target route. Especially in this stage, the importance of being able to quantify perception and prediction uncertainty becomes apparent, since a safe trajectory can only be planned on trustworthy input information, whereas one shall drive more cautious in a situation with limited visibility.

Prediction and planning can benefit from high definition (HD) map information in combination with high-precision ego localization. Since such detailed maps are not available for every street, map-less methods and hybrid methods are researched [LCW+23]. The latter utilize less detailed standard definition maps and both estimate HD maps from sensor inputs.

Chapter 3

Fundamentals of Deep Learning Architectures

A deep learning (DL) model is a structure to process data, characterized by its *deep* architecture, in the sense that it contains multiple processing layers stacked on top of one another. A layer can contain many learnable weights, each of which affects the output for a given input. With their structure of small entities acting according to learned behavior and interacting with nearby entities, DL models are reminiscent of the way neurons in the human brain behave. Even though artificial neural networks are not able to reach human-level capacities in an all-encompassing way (yet), the field of deep learning has developed rapidly in recent years. In certain tasks, such as image recognition and chess, deep models achieve on par or even super-human performance. Concerning multi-object tracking, both classical as well as DL approaches are in use. This thesis aims to explore the potential of DL methods in this regard and to advance the state of the art. In this chapter, selected fundamentals of deep learning are introduced, especially those that build the foundation for this work: Both model training and inference, as well as specific deep learning model architectures, such as transformer models and convolutional neural networks. Besides this, a comparison to conventional statistical methods is drawn, since the two fields are closely related.

3.1 Model Training and Inference

DL models are generic structures that only learn to fulfill a task of interest through training. Given a training sample, the model's output is compared to an expected result, and its weights are adapted accordingly, which is called *supervised* learning. For this, a loss function that quantifies the deviation between the result and the expectation is defined. Some loss functions, relevant to object detection and tracking,

are introduced in this section. Once the loss has been computed, it is propagated back through the model and the gradient direction is computed for all trainable weights to update them, which is described afterwards.

Note that, alternatively, models can be trained on unlabeled data in a *self-supervised* fashion. This is, e.g., achieved by only presenting selected parts of available data to the model and teaching it to predict what is has not seen. This work, however, only focuses on *supervised* training, using the manually labeled nuScenes [CBL+20] dataset.

3.1.1 Loss Functions

During training, the model's output is compared to the expected result with a loss function that quantifies the deviation into a single value. Note that the loss function used for training commonly differs from the one in evaluation [GBC16]. While the latter measures the performance at the task of interest, the former is chosen to be a suitable proxy for it while still able to be optimized efficiently.

For object detection, a regression as well as a classification loss is computed during training. The regression loss $\mathcal{L}_{\text{reg}}$ quantifies the accuracy of the estimated bounding box parameters $\hat{\boldsymbol{b}}_i$, i.e. the 3D location, size, orientation and velocity, if applicable. This loss is only applied to estimated boxes that have been matched to a GT bounding box $\boldsymbol{b}_i$, whereas those matched to background are unaffected by it. Loss functions commonly used for this are an ℓ_1-loss and an intersection-over-union (IoU)-loss, or a combination of both [CMS+20]:

$$\mathcal{L}_{\text{reg}}(i) = \begin{cases} \lambda_{\ell_1} ||\hat{\boldsymbol{b}}_i - \boldsymbol{b}_{m(i)}||_1 + \lambda_{\text{IoU}} \mathcal{L}_{\text{IoU}}(\hat{\boldsymbol{b}}_i, \boldsymbol{b}_{m(i)}) & c_i \neq \varnothing \\ 0 & \text{otherwise,} \end{cases} \tag{3.1}$$

where $m(i)$ denotes the index of the GT box that was matched to the i-th estimate and c_i denotes the target class, which is $\varnothing$ (*background*) for non-matched estimates. λ_{ℓ_1} and λ_{IoU} are weighting parameters. Many variations of this loss exist, such as a combination of ℓ_1 and ℓ_2 loss in the first term [Hub92] and modifications of IoU [Uni19].

Besides the box regression task, the model needs to learn to classify the boxes correctly. For this, it outputs one score $p(c)$ per possible class c, typically between 0 and 1, so that they can be understood as class probabilities, although they are not calibrated to behave like true probabilities [FRD18]. In a single class-setting, the classification is binary: *object* or *background* [RFGD22a]. When multiple classes are

to be detected, the classification is performed between all of them, besides *background*. A commonly used loss function is a weighted logarithmic loss

$$\mathcal{L}_{\text{cls}}(i) = \begin{cases} -\log \hat{p}_i(c_i) & c_i \neq \varnothing \\ -\frac{1}{10} \log \hat{p}_i(c_i) & \text{otherwise,} \end{cases} \tag{3.2}$$

where $\hat{p}_i(c_i)$ denotes an estimate's class score for the class it was matched to. Since background samples are much more common than true positives, the former are down-weighted to limit their influence on the parameter optimization [RHGS15]. The total loss for all estimates is additive:

$$\mathcal{L}_{\text{det}} = \sum_{i=1}^{M} \mathcal{L}_{\text{reg}}(i) + \mathcal{L}_{\text{cls}}(i). \tag{3.3}$$

Concerning deep multi-object tracking, there's no clear consensus yet which loss functions are most suitable for training. On the contrary, the loss heavily depends on how the tracking is implemented in the respective model. In a transformer based method, the matching between GT and estimated objects is mainly affected by the tracking, whereas once the matching is established, the same loss functions as for object detection can be used [MKLF21; RFGD22a]. In [LYU18], there is no explicit tracking loss, but only prediction and detection is supervised, so that the tracking task can be inferred from (in)consistencies between the two.

3.1.2 Backpropagation and Optimization

When the loss has been computed, a backpropagation step is performed to obtain a gradient of the loss function with respect to all learnable parameters of the model. This entails starting from the end of the network, i.e. the loss function and working backwards by continuously applying the chain rule $(f(g(x)))' = f'(g(x)) \cdot g'(x)$ to compute a gradient. Therefore, all processing steps in the network need to be differentiable (at least locally). In the case of a simple feed-forward network (cf. section 3.3.1), the backpropagation equations can be computed by hand using matrix multiplications. For more complex systems, the equations quickly become intractable, but are efficiently implemented in machine learning (ML) libraries such as *pytorch* and *tensorflow*. For each vector that requires a gradient, such ML libraries keep track of all operations performed on them and construct a computational graph that they can traverse backwards to compute gradients when needed [PGM+19].

Once the gradient has been computed, the network weights are modified according to the direction of steepest descent, i.e. gradient descent. In practice, modifications of gradient descent are commonly used, such as stochastic gradient descent: With

large datasets, it is typically infeasible to process all available data jointly and to compute one gradient from all of them at once. Instead, batches of samples from the randomly shuffled data are drawn. The gradient is computed from them and used as an approximation of the full gradient to perform a weights update. A learning rate determines how large the modification in one step can be. It can be set in advance or modified during the course of the training, e.g. with a certain schedule.

By design, optimization based on backpropagation is not guaranteed to reach a global minimum, but can only find local minima. This can lead to sub-optimal training results, to random improvements with different network initialization and can hinder reproducibility of results. However, [LBH15] argue this does not cause major issues because the optimization landscape tends to not exhibit many significantly sub-optimal local minima.

Ideally, one would want to not only find the weights with a globally minimum loss among the available data samples, but a global minimum among all possibly observable data. I.e., one doesn't want to just memorize the given data (over-fitting) but to generalize even on unseen data. Many strategies exist in this regard: Early stopping aims to stop the training before over-fitting can occur. An increase in the training data amount can also be beneficial, though it may entail expensive manual labeling. Besides this, data augmentation strategies are commonly used, which randomly modify the input data to increase their diversity. Examples for this are random cropping and rotation of images and point clouds.

3.2 Comparison to Conventional Statistical Methods

Deep learning methods were developed on a foundation of conventional statistical methods. This section aims to highlight similarities and differences between the two approaches. Many aspects of deep learning are reminiscent of conventional statistics, such as the aim to predict an outcome from given data and the choice of loss functions, whereas others are fundamentally different.

3.2.1 Feature Selection

Conventional methods typically do not process raw measured data directly, but extract relevant, hand-crafted features first. If one aims to detect edges in an image, for example, a filter would be designed to determine discontinuities in image

brightness. This means that such methods are very task- and data-specific. Their benefit is that, depending on the model used to parameterize the problem and the assumptions that were made, it is possible to provide performance guarantees. To improve a system, additional details can be added to the model to depict the real world better. In case of a failure, it is possible to determine its root cause by backtracking the algorithm's behavior.

Deep learning follows a fundamentally different approach: Feature extraction is left up to the model, with little human intervention. Typically, the measured data is processed in raw format, so that the model needs to learn to extract relevant information from it [LBH15]. It is, however, difficult to trace the cause of an error in inference and to prove that a model will perform well in unseen data. On the other hand, DL methods eliminate the need for labor-intensive, hand-crafted features and provide learned alternatives that may even include details, which were not apparent to human engineers.

3.2.2 Bias-Variance Trade-Off

From conventional statistics, it is known that the choice of too many parameters can lead to over-parameterization and over-fitting, especially when the number of parameters is larger than the number of data points. There is a trade-off between bias and variance: Too few parameters limit a model's expressiveness and thus introduce a bias, i.e. wrong predictions, on unseen data. Too many parameters on the other hand tend to cause the model to pick up on random perturbations in the presented data that do not generalize to unseen data [VS11]. An ideal model would exhibit minimal bias and variance, which is difficult to achieve in practice.

In the deep learning domain, there is a tendency to create larger models than one would expect with the bias-variance trade-off in mind. This is often justified by increasing amounts of available data. However, many models are provably over-parameterized, i.e., their number of parameters is large enough to fit the entire training data. It has been shown that over-parameterization allows for convergence to a global training loss minimum despite a non-convex loss function under some conditions [ALS19]. Techniques to pre-train an over-parameterized model and to prune it into smaller form have been developed [LYMP21]. In practice, many deep over-parameterized models do not suffer from large variance on unseen data, i.e. they still generalize reasonably well. Why that is, is subject to research [CBP21; NKB+21], as well as how similar results can be achieved with smaller-scale models.

3.3 Deep Learning Architectures

This section introduces deep learning components that are commonly used and relevant to this work: A feed-forward network (FFN) combines linear layers with non-linear activation functions and is applied to input vectors. A convolutional neural network (CNN) can process spatial data, i.e. its input are 2D or 3D matrices. A recurrent model can memorize past information and therefore process sequences of data. A transformer is a versatile structure that expects an unordered set of tokens as input, which can be enhanced with either spatial or temporal information [VSP+17].

3.3.1 Feed-Forward Networks

Feed-forward networks (FFNs) are also known as multi-layer perceptrons (MLPs). They are the simplest form of a neural network and consist of linear layers with a specified input length and output length and a non-linear activation function. A layer l is described by its weights $\boldsymbol{W}^l$, which are applied to the input vector $\boldsymbol{x}^l$

$$\boldsymbol{x}^{l+1} = \varphi_l(\boldsymbol{W}^l \boldsymbol{x}^l), \tag{3.4}$$

where $\boldsymbol{x}^{l+1}$ is the layer's output and φ is the activation function. The full network comprises of L layers, so that its output is [Nie15]

$$\boldsymbol{x}^{L+1} = \varphi_L(\boldsymbol{W}^L \varphi_{L-1}(\boldsymbol{W}^{L-1} \dots \varphi_1(\boldsymbol{W}^1 \varphi_0(\boldsymbol{W}^0 \boldsymbol{x}^0)))). \tag{3.5}$$

The activation functions φ_l needs to be non-linear to gain expressiveness from using multiple layers, since layers connected by a *linear* activation function could be combined into one equivalent single linear layer. It has been shown that a FFN with only two layers is a universal function approximator, i.e. it can approximate any function arbitrarily well in a given region, though it may require a very large network width [Cyb89]. Examples for commonly used activation functions are a sigmoid function (logistic function) $\varphi_{\text{sig}}(x) = (1 + e^{-x})^{-1}$ and a rectified linear unit (ReLU)

$$\varphi_{\text{ReLU}}(x) = \begin{cases} x & \text{if } x > 0 \\ 0 & \text{otherwise.} \end{cases} \tag{3.6}$$

The latter has some advantages for training neural networks, such as an efficient computation and its scale-invariance, i.e. $\varphi_{\text{ReLU}}(a \cdot x) = a \cdot \varphi_{\text{ReLU}}(x)$ [GBB11].

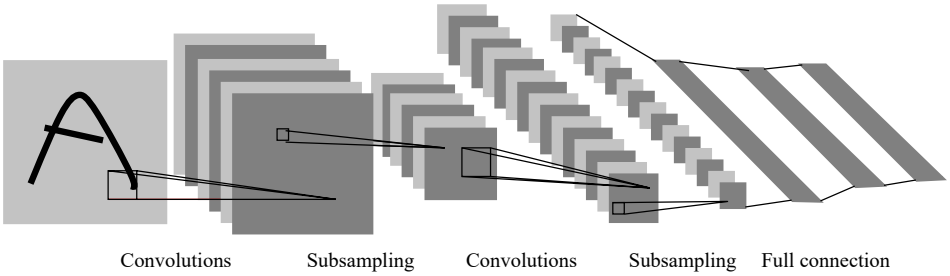


Figure 3.1: LeNet-5 CNN architecture with convolutional, subsampling and feed-forward layers [LBBH98]. © 1998 IEEE

3.3.2 Convolutional Neural Networks

Convolutional neural networks (CNNs) have been developed in the context of computer vision applications. They can process spatial data, e.g. 2D images and 3D voxelized point clouds, whereas the 2D case is introduced in the following. A convolutional layer is defined by a kernel size k_s, a stride s and an activation function φ. Optionally, the image can be padded with p pixels at each border to allow for processing to the outer-most pixels. A convolutional kernel window of size $k_s \times k_s$ is applied over all patches of the image with a stride of s. Its weights are trainable. Note that the same kernel weights are applied to the full image. Given an input image of size $U \times V$, the resulting output image is of size $\frac{U-k_s+2p}{s}+1 \times \frac{V-k_s+2p}{s}+1$.

A CNN typically combines different types of layers into one network. Figure 3.1 exemplarily depicts LeNet-5 [LBBH98], which is a small CNN designed to classify handwritten digits. It combines convolutional layers with fully connected and subsampling layers. The latter decrease the image size by applying a max-pooling filter with a stride larger than one on the image that selects the largest value in its scope. The fully connected layers at the end are used to convert the feature representation into a desired result, such as classification scores.

3.3.3 Recurrent Models

The methods discussed so far are unable to process measurements sequentially, since they don't have a memory to recall previously seen data and the model weights are fixed in inference. Recurrent neural networks (RNNs) were designed for sequential data, such as speech, and allow for a connection between the neurons from the previous to the current time step. For this, they contain a persistent internal state.

The current output depends on the current input as well the previous state. Due to the recurrent nature, the state can capture information from multiple previous time steps, though the influence of older ones typically fades over time [Elm90].

RNNs are known to be difficult to train, since the gradient of the output obtained for the kth frame of a sequence depends on all previous states, i.e. k forward passes. One training approach is called backpropagation through time, during which sequences of fixed length are processed. For all forward passes that belong to one sequence, the model parameters are held constant. A loss and gradient is computed for each intermediate result. At the end of the sequence, the model parameters are updated according to the gradients [Moz13]. Standard RNNs suffer from vanishing gradients for longer sequences, i.e. that the gradients become so small that they hinder the training process. One way to cope with this are long short-term memory (LSTM) networks [HS97] that include a gating structure to decide what to remember and what to forget.

3.4 Transformer Models

In the natural language processing (NLP) domain, LSTMs have allowed for processing written and spoken text, enabling chatbots and automatic translations. In RNNs, the current token (e.g. a word) interacts with all the previous tokens (previous words) via a joined history. What if each word could selectively interact with only those previous ones that are relevant to it and mask the remainder? What if all history is available in equal quality, no matter how long ago in the sequence it occurred? Transformer models [VSP+17] utilize an attention mechanism to allow each input token to interact with every other one and provide solutions for known issues with RNNs.

For sequential data processing, each input token is augmented with a temporal encoding that describes its location within the sequence. Then, the full history currently available is processed by the transformer in a single forward pass. This allows for understanding of a large context: If a name is mentioned in the first paragraph and then referred to in the third one, it can still be retrieved. At the same time, the information is only accessed selectively via attention weights so that it does not overwhelm the model. Transformers have revolutionized the natural language processing (NLP) domain, as they are the basis for large language models, such as ChatGPT [Ray23].

Not only for NLP, but also for other research fields, the transformer provides a way to elegantly model dependencies within data. This does not need to be a temporal

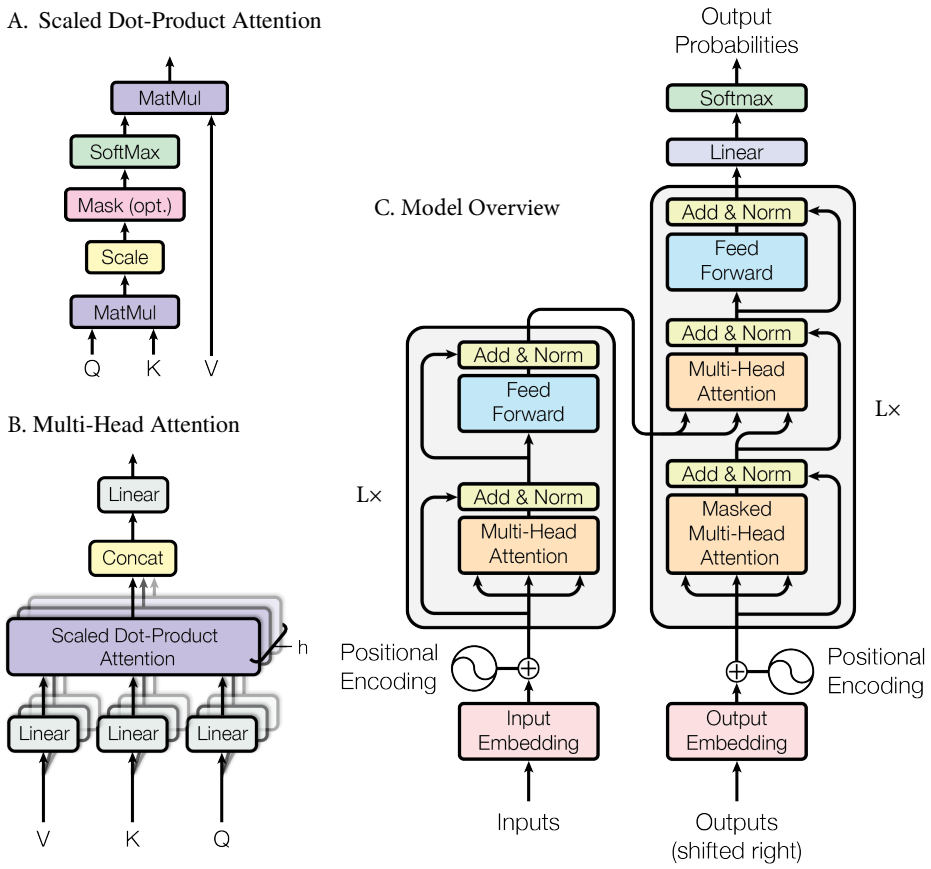


Figure 3.2: Transformer model and its components. [VSP+17] © 2017 Google

dependency, but can also be spatial or a mixture of different domains. Transformer models are versatile in this regard, since they only expect a set of tokens (vectors) as input and are invariant to their order [VSP+17]. A token can stem from any measured entity, such as a word or a pixel. The spatial or temporal relation of the tokens is not defined in their order, but by their content. Any order can therefore be incorporated by adding a corresponding encoding to the tokens. As mentioned previously, transformers have been applied to the 2D and 3D vision domain. This work aims to explore their potential in the context of MOT for autonomous driving.

The remainder of this section is structured as follows: The attention mechanism is

introduced, which is the main building block of the transformer. The full model architecture is described afterwards. Finally, since it can be challenging to train a transformer, some insights into training strategies are given.

3.4.1 Attention Mechanism

Each input token (typically a vector) is transformed into three different vectors: A query, a key and a value vector. The attention mechanism itself functions similar to a database: A token provides its query to communicate what information it is interested in. This query is compared to all keys provided by the other tokens, which results in attention weights. A key that is more similar to the query results in a higher weight, meaning that this interaction is of greater relevance. This is called scaled-dot product attention [VSP+17] and depicted in Figure 3.2 A, whereas the dot product serves as a similarity measure. With matrix notation, the attention weight matrix is computed for all tokens jointly:

$$\boldsymbol{W} = \text{softmax}\left(\frac{\boldsymbol{Q}\boldsymbol{K}^\top}{d}\right), \tag{3.7}$$

where $\boldsymbol{Q} \in \mathbb{R}^{N\times d}$ contains all N tokens' query vectors, and $\boldsymbol{K} \in \mathbb{R}^{N\times d}$ the keys. d is the channel dimension of the queries and keys and softmax denotes a normalization so that each line of the attention weight matrix $\boldsymbol{W}$ sums up to one. Finally, $\boldsymbol{W}$ is applied to the value matrix $\boldsymbol{V}$ to obtain the attention result

$$\text{Attention}(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}) = \boldsymbol{W}\boldsymbol{V} = \text{softmax}\left(\frac{\boldsymbol{Q}\boldsymbol{K}^\top}{d}\right)\boldsymbol{V}, \tag{3.8}$$

which is of the same size as $\boldsymbol{Q}$, i.e. $N \times d$.

When the queries, keys and values are computed from the same set of tokens, *self-attention* is performed, i.e. each token interacts with the others in its set. Another option is *cross-attention*, which entails two sets of tokens, one of length N, the other of length M. The keys and values are computed from one token set and the queries from the other. Those tokens from the latter set can then query information from all tokens in the former set, resulting in an attention matrix of size $M \times N$.

In a transformer, the attention is performed by h multiple heads in parallel, as pictured in Figure 3.2 B. This enables the individual heads to specialize on a certain aspect. Each attention head operates in a reduced channel dimension $\frac{d}{h}$ so that the results can be concatenated in the channel dimension.

3.4.2 Transformer Model Architecture

Figure 3.2 C. presents the full transformer model as introduced in [VSP+17]. It consists of an encoder (left) and a decoder (right). Since the transformer interprets its input as a set of unordered tokens, the input's order is preserved with an additive positional or temporal encoding first. Then, it is passed to the encoder, where multiple layers are stacked, each of which contains a (multi-head) self-attention component followed by a residual connection, a normalization and a small feed-forward network. While the attention is a one-to-one interaction in the first layer, higher order interactions are possible in the following ones, since all tokens aggregate context information. The output of the encoder is a set of tokens in the same shape as the input, but each of them was able to explore and aggregate its context.

In the decoder, both self- and cross attention is performed. The decoder input relates to the desired output. For example, if a machine translation task shall be performed from French to English, the English sequence produced so far is input to the decoder with an appended placeholder to decode the next word. This means that one forward pass is required for each additional output word. Object detection is a task that does not have a sequential output, therefore it is performed in one forward pass. As proposed by DETR [CMS+20], as many placeholders, i.e. object queries, as the maximal number of expected objects in a scene are input to the decoder.

The decoder input is first processed in a self-attention module, followed by a cross-attention, for which the decoder provides the queries and the encoder provides both key and value vectors. This means that each decoder token can query relevant information from the input that it needs to decode its desired result. Just like in the encoder, multiple decoder layers are stacked.

3.4.3 Training Peculiarities

Transformers are known for impressive performance on various tasks. However, they should not be misinterpreted as an easy solution to any problem. Transformer models are known to be difficult to train from empirical observation [LLG+20], for example, the final result is susceptible to certain initial conditions and the learning rate schedule needs to be chosen carefully. Besides this, convergence can be slow [ZSL+21] and the models require a comparatively large amount of data combined with a large batch size for successful training [CMS+20]. The latter means that a GPU with a large memory, or better multiple large GPUs, are required to facilitate such a training. Therefore, transformer training is quite cost- and resource intensive.

To make the technology more accessible and to achieve faster progress, multiple strategies have been developed. Large models can be pre-trained on a large dataset and the model weights are made accessible publicly so that many people can use them as a basis for their research [WDS+19]. In some domains, one can argue whether global attention is really necessary, or if it may be replaced with local attention to facilitate faster convergence [ZSL+21]. Placing the normalization at a different place in the transformer pipeline has also been shown to improve convergence [LLG+20]. Another option is to incorporate more prior knowledge into the model [YALZ21]. Still, recent history has shown that big breakthroughs, such as large language models, are achieved with very large models first, to then develop more tractable, smaller models in a second step [TLI+23].

Chapter 4

A Joint Approach to Object Detection and Tracking

4.1 Motivation and Aim

Commonly, trackers are viewed disjoint from object detection methods. While the field of object detection has benefited tremendously from recent deep learning advances, tracking has been less in the focus of machine learning research. The aim of this work is to explore the potential of deep learning with an explicit focus on tracking for autonomous driving. Therefore, a joined tracker and detector is developed, i.e. a model that follows the *tracking-and-detection* paradigm, as illustrated in Figure 4.1 (bottom). This grants the tracker access to both the rich latent feature space of the deep detector, as well as to the encoded input measurements themselves, which is much broader information compared to the low-dimensional bounding boxes that are often input to regular trackers in a chained system, also called *tracking-by-detection* (Figure 4.1 top). Can this additional information bring a benefit to tracking?

Another aspect that sets apart Bayesian from deep trackers is the incorporation of models and constraints. Bayesian methods rely on well-defined models and perform provably well under the assumed conditions. This can be a trade-off between a simpler model, which requires less manual tuning, and a more sophisticated one that contains many hand-crafted parameters. A deep tracker, on the other hand, has the potential to observe and learn multi-faceted models, when trained with a large and diverse set of real-world data, without the need of human support. On the other hand, it may lack explainability due to the black-box nature of the learned models. Does this improve the tracking performance, or is it beneficial to retain some constraints based on the laws of physics?

These questions, among others, motivate this research. In the following, the proposed

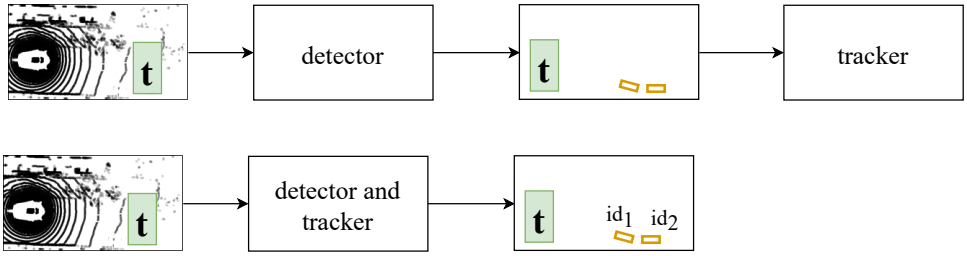


Figure 4.1: Visualization of different tracking paradigms. *Top:* Chained method. Low-dimensional bounding boxes are input to the tracker. *Bottom:* Joint detection and tracking grants the tracker access to detector knowledge.

transformer-based multi-object detector and tracker (TransMOT) is introduced. It is, in its core, inspired by Bayesian trackers, to retain some of their beneficial properties. It is also modular, so that many components can be exchanged and improved individually. This makes it very versatile: The name TransMOT denotes more of a concept than a certain model, which, depending on the components, can, for example, incorporate many physical constrains, or learn freely, only from given data. This work aims to find a balance between these opposing poles. In the following, the concepts that TransMOT is based on are introduced.

4.2 Transfer of Classical Tracking Principles

Bayesian trackers have many advantages. TransMOT, though it is based on deep learning, is designed to retain some beneficial properties of Bayesian methods. Some of them are discussed in the following.

4.2.1 Prediction and Update

Let $\mathcal{T} = \{\boldsymbol{t}_k\}_{k=1}^{K}$ denote a set of K tracks, with $\boldsymbol{t}_k = \{\boldsymbol{s}_j, t_j\}_{j=0}^{j_k}$. Each track describes one object instance over time and is a series of state estimates $\boldsymbol{s}_j$, each with a respective timestamp t_j. Tracks can have variable lengths j_k. In the proposed method, the track's state, i.e. all information that is currently known about it, is encoded in high-dimensional feature space with a *track query* $\boldsymbol{z}_j \in \mathbb{R}^d$. From it, a bounding box $\boldsymbol{b}_j$ can be computed so that $\boldsymbol{s}_j = \boldsymbol{z}_j, \boldsymbol{b}_j$. Unless specified otherwise, the bounding box locations are stated in an ego-centered coordinate system, in which

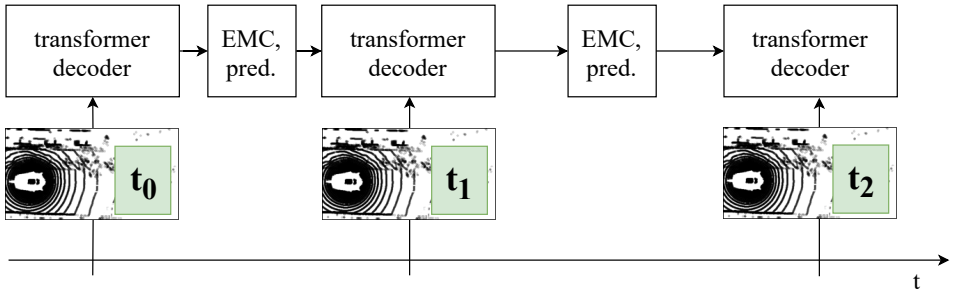


Figure 4.2: Time-continuous tracking system. A new measurement can be utilized any time it is observed, different from a time-discrete method.

the ego-vehicle is placed at the center, heading in y-direction, which is depicted in the sensor setup overview in Figure 2.1 as *LIDAR Top*.

In TransMOT, prediction and update steps are modular and can be viewed separately, similar to a Bayesian method.

Prediction For each non-terminated track, a prediction and ego motion compensation is performed to transform the latest estimated track state $\{\boldsymbol{s}_{j_k}, t_{j_k}\}$ into $\{\boldsymbol{s}'_{j_k}, t_{j_k+1}\}$, i.e. one step into the future. For this, both a transformed track query $\boldsymbol{z}'_{j_k}$ as well as bounding box $\boldsymbol{b}'_{j_k}$ are computed. No new measured sensor data is utilized, but only information available up to the time t_k, such as the agents' observed velocities as well as the measured ego pose change. Both a learned as well as a deterministic ego-motion compensation have been explored and compared, as well as a short-term prediction, which is detailed in chapter 6.

Update Based on the predictions $\{\boldsymbol{s}'_{j_k}, t_{j_k+1}\}$, the tracks are updated with newly measured sensor data at the time t_{j_k+1}, to obtain their latest state $\{\boldsymbol{s}_{j_k+1}, t_{j_k+1}\}$, as well as initial states of newly spawned tracks $\{\boldsymbol{z}_0, t_0\}$. In TransMOT, the update step is performed with a trained transformer decoder module, which allows the latest transformed track queries $\boldsymbol{z}'_{j_k}$ to access the measured input data via cross-attention and to aggregate updated information about their object. This paradigm is described in more detail in section 4.3.1. In case the update does not lead to a confident bounding box, the track is considered either terminated or temporarily occluded.

Compared to a Bayesian system, in the update step, rather than obtaining new

object locations and matching them to existing tracks, TransMOT understands each track as an entity that can access the input data it deems relevant to update itself. While the update step is performed differently from most Bayesian trackers, a common ground is the separation of prediction and update, which allows to swap components in a modular fashion, as well as to supervise the model's behavior.

4.2.2 Robustness to Sensor and Synchronization Issues

Sensors are physical systems that do not measure data at perfectly spaced time intervals. In case multiple sensors are involved, synchronization between them can also be an issue. Even with only one sensor, stochastic delays can occur, causing a deviation from the expected measurement interval, which a tracking system needs to cope with. Commonly, Bayesian filters are well-suited for accepting new measurements at any given time, since the prediction can be performed accordingly. Learning-based systems, on the other hand, often expect evenly spaced data, i.e. measurements from discrete points in time. Contrary to that, TransMOT aims to accommodate for continuous-time measurements by performing both the ego-motion correction as well as the prediction step in a continuous fashion, with any possible target timestamp, as illustrated in Figure 4.2. For this, the ego-motion correction module accepts the (continuous) ego pose change as an input, whereas the prediction operates directly with a continuous time delta. Experiments show that the system can even cope with a certain amount of dropped frames, i.e. missed measurements by projecting the track state accordingly, which is detailed in chapter 6.

This approach could be extended to a synchronization-agnostic multi-modal sensor fusion, as depicted in Figure 4.3. In this example, the tracker state is updated at different points in time with a different sensor modality. To realize such a system, each modality would need its own feature encoder before the features are passed to the transformer decoder. The asynchronous multi-modal sensor data may pose a challenge in typical perception training frameworks, since public benchmarks tend to focus on synchronized sensor data. Therefore, an intermediate goal can be a fusion model that accepts input from all sensor modalities at the same time. Such a model is introduced in section 5.6.3: It is based on FUTR3D [CZW+23] and accepts encoded image features and lidar features at the same time. The transformer can attend to both during its cross-attention and thereby learn an early fusion on feature-level.

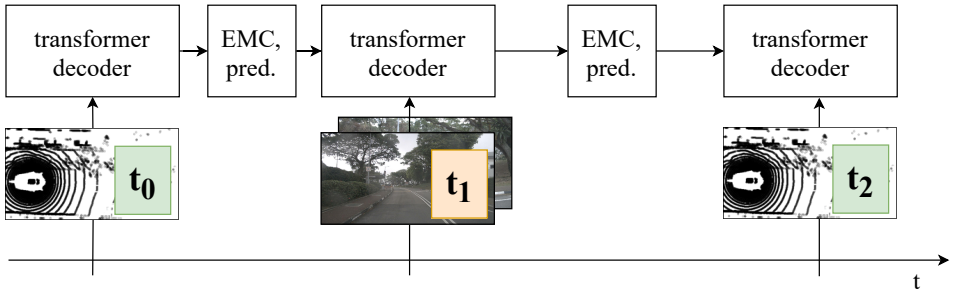


Figure 4.3: Extension of the proposed method to an asynchronous multi-modal system. Each modality is encoded with its own backbone and passed to the same transformer decoder to update the tracks.

4.2.3 Sequential vs Stacked Input

A tracker needs to be able to access past information, as this is crucial to extrapolate where an instance of an object is expected to appear next and to match new measurements to existing tracks, accordingly. Bayesian trackers encode the past information in a track state that commonly contains a current position as well as past motion characteristics, such as a velocity and an acceleration. Besides this, uncertainty parameters are usually estimated. For deep learning methods, two fundamentally different approaches to access past information are detailed in the following: One option is frame stacking [LYU18]. A set number of past measured frames is accumulated and input to the model as a stack, together with the newest measurement at time t, i.e. N frames from timestamps $\{t, t-1, \ldots, t-(N-1)\}$. This method entails, however, that the history is stored in an uncompressed fashion, i.e. as raw sensor data. Besides this, most of the input window overlaps with the one from the previous timestamp, causing the model to encode each measured frame multiple times (N times, until it is phased out from the history). To avoid such inefficiency, TransMOT follows a recurrent paradigm: The very first frame is input to the model and encoded in a per-track manner, i.e. the model aggregates information about each object in the scene, resulting in one high-dimensional feature vector per initialized track, $\boldsymbol{z}_j$. Such a vector can e.g. have $d = 256$ entries, which is larger than the track states typically used with Bayesian trackers, and can potentially encode additional information observed from the input data, such as a shape, motion or location. Note that the information is not stored in human-readable format, but a regression head can be applied to the feature vectors to extract box parameters from them. The set of track states is smaller than the raw input data so that it is a form of compression.

4.3 Novel Characteristics of a Deep Tracker

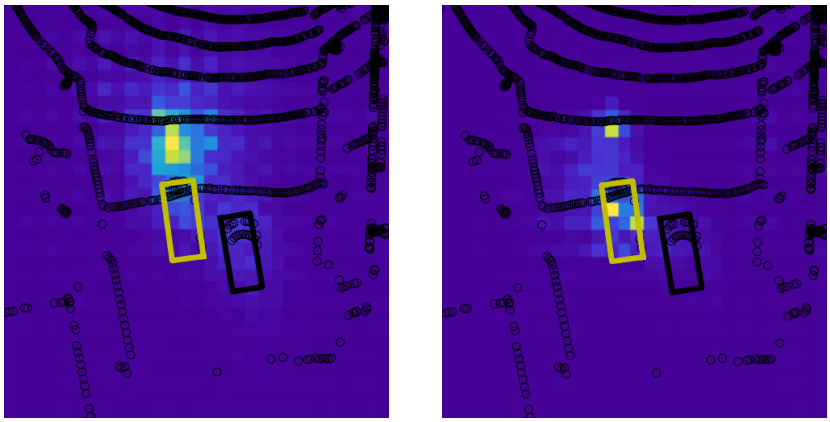
This section focuses on aspects that are different from Bayesian trackers, because they are enabled by deep tracking in the first place. These new possibilities also motivate the research efforts into deep tracking, since it is, e.g., promising to benefit from granting the tracker access to the detector's latent state, from maintaining a high-dimensional tracker state as well as from learning a soft association rather than following a matching heuristic.

4.3.1 Track Association Method

Whenever a new measurement is received, it needs to be associated to the existing tracks. This can either be a *one-to-one* matching, i.e. each measurement is assigned to at most one track, or a soft assignment that can accommodate a *one-to-many* matching. In both cases, to compute an affinity between a measurement and each track, a distance metric is commonly used, e.g. the Mahalanobis distance [CPLB20].

In the proposed deep tracker on the other hand, an association is performed that can be described as *all-to-many*. The track update is performed through a transformer decoder that allows each track query (i.e. the existing tracks) to interact freely with the new sensor measurement. In this case, 'new measurement' refers to a new lidar point cloud that is encoded through a feature encoder, as introduced in section 2.2.2, also known as a backbone. This results in a 2D feature map in bird's-eye view. Each query can, in principle, access any grid cell on this map. In practice, however, one can observe that queries learn to attend to certain regions of interest, which is detailed in chapter 7. Note that no explicit association is necessary in the proposed model: On the contrary, each track is able to update itself by freely accessing relevant parts of the input data. Since it may be difficult to deduce which parts are relevant in one pass, several decoder layers are stacked, and each of them contains one instance of cross-attention. This allows the attention to shift and to focus over the course of the layers.

By investigating the respective attention weights, one can see the model's learned soft association in hindsight for each layer. An example for this is pictured in Figure 4.4: One nearby object is colored black, whereas the tracked object of interest is yellow. The attention weights between its track query and the bird's-eye view grid are pictured by coloring each grid cell according to the attention score. Brighter colors denote higher attention. Between the previous and this frame, only the ego-motion has been corrected, whereas the tracked car's own movement has not been compensated for. This causes the model to initially associate the track to



Layer 0. Layer 5.

Figure 4.4: Observable attention shift through decoder layers on an excerpt of the bird's-eye view grid. A tracked object of interest is colored in yellow, along with measured lidar points and another object in black. Each bird's-eye view grid cell is colored according to its attention weight with the track query of interest, where brighter colors denote higher attention [RFGD22a]. © 2022 IEEE

measurements in the wrong place, i.e. high attention weights behind the object to track in the first decoder layer (left image). Notably, the model is able to compensate for this through the layers, causing an attention shift towards the object to track in the final layer (right image).

4.3.2 Access to Latent Features

When following a *tracking-by-detection* paradigm, i.e. when one has a pipeline that comprises of a detector followed by a tracking module, an interface between detector and tracker has to be created. Commonly, low dimensional vectors are used for this that contain information about detected objects, such as bounding boxes with a location, size, orientation and velocity (if applicable) [CPLB20]. To detect an object, however, a deep learning model uses visual and spatial cues from the measured data to form a high-dimensional latent space that encodes such information and allows to deduce where an object is observed and what kind of object it is [LVC+19]. In this regard, a low-dimensional interface to the tracker is a bottleneck, because much of the information that the detector aggregated for each object is lost. This could, e.g.,

be latent information about an object's shape or surrounding environment.

Some *tracking-by-detection* methods have noticed this shortcoming and compensate by enlarging the interface between detector and tracker, e.g. by sampling latent features per object and passing them to the tracker, along with the bounding boxes [CLAB21]. Still, in such a model, the detector is oblivious to what details a tracker may be looking for, since it is a uni-directional interface. Contrary to this, the aim of this work is to maintain a bi-directional connection between detector and tracker, i.e. to allow the tracker full access to the encoded lidar point cloud through cross-attention, alongside with the detector in one common forward pass.

4.3.3 High-Dimensional Track State

Not only the interface between a detector and a tracker is a design choice, but also the interface through time, i.e. for passing track information from one step in time to the next. In a Bayesian method, track state matrices are maintained and predicted until a new measurement is available to perform another track update [CPLB20]. With an attention-based tracking method, *queries* are a natural way to carry information through time [MKLF21]: First, an object query is initialized, it then aggregates information about its respective object through cross-attention. This is illustrated in Figure 4.5 on the left, where object queries are colored in dark gray. Passing through the transformer decoder results in a modified vector of the same length as the original query, which is then denoted a track query (orange) and can be presented to the model again in the next time step (dark orange) to update the respective track.

By design, the tracker state is propagated in a high dimensional feature vector per track, which can aggregate information over the lifetime of its track. While the principle described above was first introduced by [MKLF21], many questions remained open problems of research, some of which this work aims to answer. One of them is how to perform ego motion correction on the track queries, which is a non-trivial task in high-dimensional feature space and detailed in chapter 6. This is necessary in the context of autonomous driving, since an ego-vehicle can move a significant distance between two measured frames. Besides this, methods to perform short-term prediction on a per-agent basis are introduced and compared. Another research question is the initialization and composition of object queries, which is discussed in chapter 5, among others.

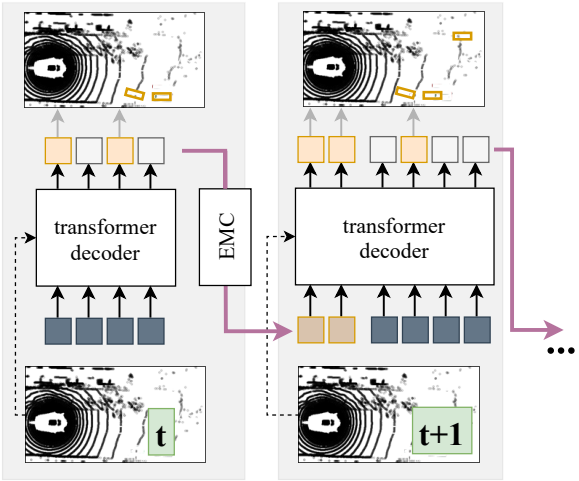


Figure 4.5: Blueprint of the proposed transformer-based detector and tracker. The track states are propagated over time with track queries (orange) and updated in the transformer decoder [RFGD22a]. © 2022 IEEE

4.4 Transformer-based Multi-Object Detection and Tracking

This chapter has provided an introduction into the motivation and concept of TransMOT, a blueprint of which is pictured in Figure 4.5. A main component is the transformer decoder, which fulfills multiple functions at once, i.e. it updates and initializes tracks and associates measurements to them, by allowing both track and object queries access to the encoded measurements. Here, track queries are used to continue existing tracks, whereas object queries serve as placeholders for potential new tracks. Besides this, the queries can interact with one another through self-attention to avoid producing duplicate detections and tracks [MKLF21].

The tracker's state is saved in the form of high dimensional track queries, which travel through multiple decoder layers per forward pass and through multiple forward passes over the course of a sequence, each time collecting more recent information about the respective object. The input frames are given to the tracker sequentially, each at the time they were measured. They are encoded through a backbone. Different variants of such a backbone have been employed, which are introduced in chapter 5. The propagation of track queries through time is another aspect of interest. Denoted

as EMC, an ego-motion correction is performed. Afterwards, a per-agent short-term prediction can be employed.

The following chapters separate the topics object detection and tracking; i.e. chapter 5 provides a more detailed view into the proposed transformer detector, whereas chapter 6 builds upon this knowledge to develop the joined detector and tracker, mainly focusing on the latter task. This work's contributions in both fields are summarized in the following.

4.4.1 Contributions to Object Detection

Environment perception from a moving platform, such as a car, entails certain challenges. In the area of object detection with transformer models, this thesis contributes as follows:

- When this work commenced, transformer based object detection was available for image data [CMS+20], as well as small-scale lidar point clouds [MGJ21], but not for large point clouds, such as lidar data measured from a vehicle. Therefore, this work proposes adaptations to the network architecture to make it suitable in the context of autonomous driving, which has been pre-published in [RFGD22b].

- Queries are placed at anchor locations. Different strategies for initial query placement are introduced and evaluated. Among others, a novel feature-informed initialization is presented [GRDG24].

- A refinement strategy is introduced that allows object queries to travel towards the object they are aiming to detect [RFGD22b]. This is especially useful given a large field of view around the vehicle, and allows for a comparably small necessary number of queries.

- A modification for the detection of multiple different classes is introduced, such as cars, pedestrians and bicycles. This also enables deep tracking on the given classes of objects, as pre-published in [RFGD23].

- The different model variants are thoroughly evaluated to gain insight in the model behavior and performance.

4.4.2 Contributions to Tracking

In this joint approach, object detection and tracking are tightly coupled. Therefore, the tracker benefits from each improvement in the detection domain. Some challenges are unique to tracking, though. In this regard, this work contributes in the following areas:

- Sequential tracking from a moving platform requires an ego-motion compensation (EMC) between measured frames. It is performed on a per-object basis by correcting their states, i.e. each track query is ego-motion corrected. Since the track queries are feature vectors, a method is developed to perform the correction in latent feature space [RFGD22a].

- The joined deep tracker and detector is designed to be suitable for tracking on the surrounding area of a vehicle; a comparably large area of interest. For this, the queries move their anchor position with their tracked object, so that a relatively small number of queries is sufficient.

- A deep tracker needs to be trained on sequential data. Different training strategies are introduced, from pairwise training to longer sequences. Both the tracking and the detection task are trained in a balanced way to achieve competitive performance on both tasks with a single model.

- Multi-class tracking is introduced, both with a unified regression head, as well as dedicated class group regression [RFGD23]. Besides this, to benefit from SOTA detection performance, the tracker is extended with a dedicated track update layer as well as a short-term prediction.

- All proposed model variants are thoroughly analyzed and a comparison to classical tracking methods, i.e. one that is based on a Kalman filter, and one that utilizes random finite sets to model the tracking problem, is drawn.

4.5 Conceptual Comparison to Filter-based Approaches

Both similarities and differences between TransMOT and filter based methods have been discussed in this chapter. In Figure 4.6, the proposed model's components are marked with terms known from Bayesian tracking for a visual comparison. First, the tracker is run on the first frame with object queries only, to initialize its state. The

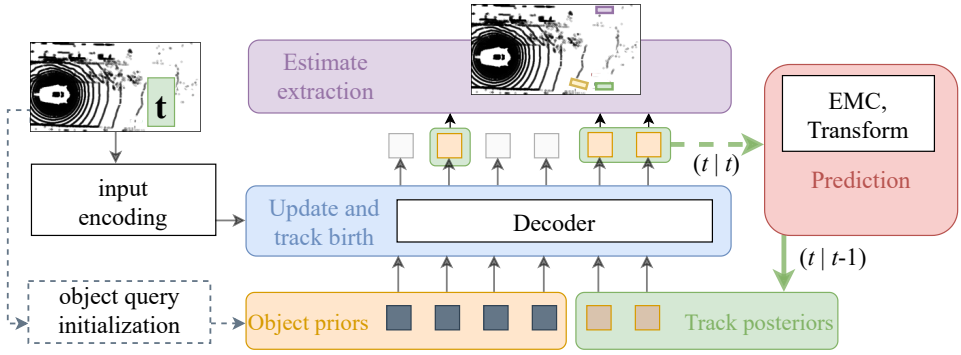


Figure 4.6: Filter interpretation: Components of the deep tracker are marked with terms known from Bayesian tracking.

object queries can be understood as a set of *priors*, each aiming to find an object in their area of interest. The decoder (blue) results in *track birth* for those queries that found an object after interacting with the input data. The respective output queries become track queries, which can be understood as *posteriors* in the sense that they were computed from an object prior, which aggregated given measured information about an object. Note that they do not follow any known parameterization of a posterior distribution, but the model learns a proprietary format in (d-dimensional) latent space. To compute a state estimate from a high dimensional track query, an *extraction* is needed. For this, a trained FFN outputs a bounding box estimate as well as a confidence score per given query. In order to prepare for the next measurement, a *prediction* and ego-motion correction is applied to the posteriors, i.e. the high-dimensional track queries. Then, once a new measurement is available, both the track queries and object priors are presented to the decoder. It now fulfills both a *track update* (for the existing posteriors) as well as *track birth* (based on the priors). The *data association* is solved implicitly by the cross-attention between the queries and the input data.

To summarize, many aspects of the proposed sequential deep tracker remind of components that are well-known from tracking research. Still, neither the posteriors nor the objects' motion are explicitly modeled, but rather learned by the model.

This is both a chance as well as a challenge: It is more difficult to give provable performance guarantees and to provide well-calibrated existence probabilities. However, the Transformer may learn more complex models given the training data, and fewer prior assumptions have to be made. Besides this, it is an active field of research to produce reliable uncertainty estimates with a deep learning model. Chapter 7 investigates the behavior of Transformer attention weights to contribute in this

direction.

Bayesian filters are commonly developed upon a Markov assumption, i.e. that the following state is independent of all past states, given the current one. The proposed method follows a similar notion, since only the most recent track query per track is presented to the model. Though, because a track query is a high-dimensional vector, it can encapsulate information from multiple past time steps, which is also encouraged during sequential training.

Chapter 5

Transformer-based Object Detection for Tracking

5.1 Motivation and Aim

In order to perform joint object detection and tracking, it is indispensable to have a well-performing object detector, which can either be used stand-alone, or function as track initializer. DETR [CMS+20] has been one of the first object detection methods based on a transformer model [VSP+17], as detailed in chapter 2, and achieved SOTA performance. At the time when this work was commenced, however, no solution for transformer-based object detection on large-scale point clouds was available, since DETR's concept could not easily be ported to a different sensor and environment domain. A method to perform this task is therefore developed in this work, which is called transformer-based detector for large point clouds (TransLPC) and enables usability for autonomous driving. Parts of it have been pre-published in the author's own research papers [RFGD22b; RFGD23; GRDG24]. While originally limited to the detection of the 'car' class [RFGD22b], the model is extended to also detect all other classes available in the nuScenes dataset [RFGD23], i.e. car, bus, truck, trailer, construction vehicle, pedestrian, bicycle, motorcycle, traffic cone and barrier. Besides this, attention based object detection has been a very active field of research, leading to some relevant concurrent work [BHZ+22; CZW+23]. The findings from recent research on transformer based perception enable to further improve TransLPC, as well as inspire to improve upon them [GRDG24].

An overview of the transformer detector is pictured in Figure 5.1. Its core structure is based on DETR [CMS+20], i.e. there are object queries that interact with the (encoded) input data through a decoder, and each aim to find at most one object. All objects are detected at the same time in one forward pass. In color, Figure 5.1 illustrates the contributions of this work to enable this concept for autonomous

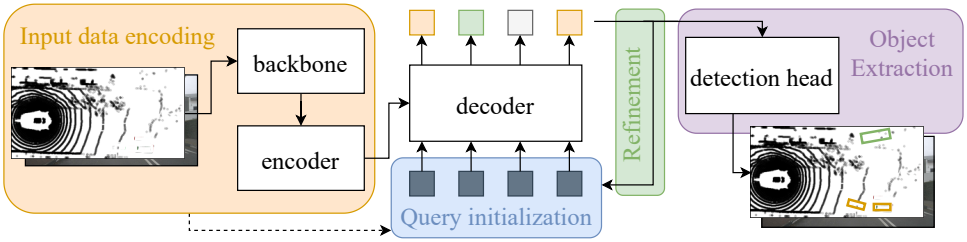


Figure 5.1: Transformer detector overview. Areas that this chapter contributes to are marked in color: the input data encoding, query initialization, positional refinement, as well as object extraction from feature vectors.

driving, as well as for deep tracking, in the following areas: Input data encoding (orange); i.e. how can large point clouds be encoded for a transformer, without consuming a large amount of GPU memory? This topic is discussed in section 5.2. Second, section 5.3 introduces multiple different ways to initialize the object queries (blue) before they are input to the decoder. To enable queries to travel towards their object, which is especially useful for a large measured range, section 5.4 introduces a refinement strategy between the decoder layers (green). Once a query has aggregated information about its object in feature space, this information shall be transformed to a human-readable format, i.e. a bounding box. This extraction (purple) is discussed in section 5.5, especially in the context of multi-class detection. Section 5.6 discusses recent advances from related work and develops a novel query initialization strategy (blue) based on them. Finally, section 5.7 presents a detailed evaluation of the presented contributions for transformer based 3D object detection, both in terms of overall performance as well as to gain insights into the model behavior.

5.2 Input Data Encoding

Before a sensor measurement can interact with the queries in the decoder, it needs to be processed and encoded. In DETR [CMS+20], the captured image is processed through an image backbone first, i.e. a convolutional neural network that encodes context information for each pixel via information aggregation from its vicinity. The image resolution is also reduced to ca. 30×45 pixels in this step, but with a large channel dimension of 256 per cell. This means that information once present in an individual pixel (color, brightness) is abstracted into a feature vector that covers a larger area. In DETR, the feature vectors are then passed to a transformer encoder, which allows for global self-attention. For example, a feature vector located at the

bottom left of the image can (potentially) interact with one at the opposite end. Such a large field of view in the encoding stage may be beneficial when an object covers a significant space of the image or to capture large dependencies such as the course of a street as an indicator for probable locations of road users. Given lidar input data, though, significant changes to DETR's original architecture are necessary, which are described in the following.

5.2.1 Backbone and Voxelization

Similar to the image domain, lidar input data is distributed in small entities, i.e. low dimensional information (intensity, occupancy) is measured at thousands of locations in the sensor's surrounding space. It is infeasible for a transformer model to process such a large number of individual entities [VSP+17]. Therefore, a backbone is necessary to aggregate the given information into more manageable sections, just like in DETR. In its first version, the transformer detector that is presented in this chapter, TransLPC, utilizes a PointPillars [LVC+19] backbone as introduced in section 2.2.2. This means that the point cloud is divided into pillars of arbitrary height that are placed on a regular 2D grid in bird's-eye view. The grid size is set to 128×128, which results in a cell size of 78.1 centimeters with a detection range of ± 50 meters to the front and sides. Compared to others, this is a rather slim backbone with a comparably coarse grid. To maximize detection performance, it is exchanged with a more elaborate one in a later version of TransLPC (cf. section 5.6).

Since a transformer model accepts a set of tokens as input and is unaware of their order or spatial location, the backbone output is augmented with an additive positional encoding, following [VSP+17; CMS+20]:

$$\mathrm{PE}_i(x) = \begin{cases} \sin\left(\frac{x}{10000^{\frac{i}{d/2}}}\right) & \text{if } i \text{ is even} \\ \cos\left(\frac{x}{10000^{\frac{i-1}{d/2}}}\right) & \text{otherwise,} \end{cases} \tag{5.1}$$

where x is the cell's position in x-direction in BEV, d the number of channel dimensions and $i = 0, \ldots, \frac{d}{2} - 1$. Each entry in the positional encoding with index i stems from a sine or cosine with a different wavelength, ranging from 2π ($i = 0$) to $10000 \cdot 2\pi$ ($i = \frac{d}{2}$). The result is visualized in Figure 5.2 for an excerpt of 50 meters range, which shows that cells with closer locations to one another have a more similar positional encoding than those farther away. The choice of a very large maximum wavelength prevents cyclic duplicates, even when a very large number of positions needs to be encoded. The encoding's length is $\frac{d}{2}$ and the same is repeated for the y-direction and concatenated. One may ask: Why is an additive positional encoding

used, instead of simply concatenating a position to each vector? In the attention stage in the transformer, a similarity between the vectors is computed, e.g. via a dot product. In such a measure, vectors with a similar location embedding will have a higher similarity, which is in line with the expectation that nearby locations are more relevant to one another than farther away ones. Besides this, many transformers use multi-head attention, which splits the feature vectors into multiple parts, and processes them separately in the multiple heads. A concatenated location would not be accessible by all heads in this case. Note, however, that the exact type of additive encoding is less important, as many different variants exist to project the low-dimensional location information into high-dimensional feature space [CZW+23; TSM+20].

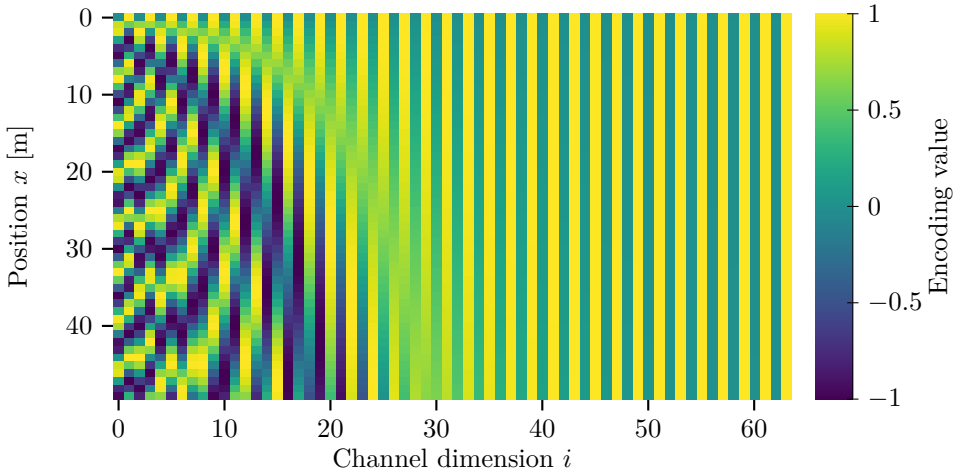


Figure 5.2: Positional encoding with alternating sine and cosine components. Each horizontal line is a positional encoding for a certain x-location and is of length $d/2 = 64$.

5.2.2 Accommodation of Large Input Sequence Length

The transformer encoder allows for a global interaction between its input tokens. While it may be beneficial to understand the global structure of the observed surroundings in a street scene, i.e. self-attention between the full BEV grid cells, it has proven infeasible to pass the 3D backbone's output into a transformer encoder for global context encoding: Global attention comes with a quadratic memory requirement, since all queries can have one-to-one interactions. DETR remedies this

by reducing the image size to a smaller one. Though, with an area of interest of 100×100 meters (or more) in BEV for the driving use-case, such a size reduction leads to too coarse cells with a side length of almost 3 meters. Besides down-sampling, transformer alternatives that reduce the quadratic memory requirement to a linear one have been explored: Linformer [WLK+20] and [TDBM20]. However, such methods only approximate the attention operation while still limiting the possible input sequence length. Experiments within this work have shown that, surprisingly, it is sufficient to *leave out* the transformer encoder, since the backbone already encodes context information. Therefore, the $N = 128 \cdot 128$ backbone-encoded tokens are augmented with the positional encoding, described above, and directly passed to the decoder. In the decoder, global attention is still possible, i.e. every query has access to the full input BEV grid, though this is cross-attention and there are much fewer queries M than grid cells N. Therefore, the largest attention matrix is only of size $N \times M$. Since $N \gg M$, this is a much smaller matrix than what would have been observed in an encoder, i.e. $N \times N$ with self-attention.

5.3 Object Embedding

The placement and initialization of object queries is an active topic of research. While early approaches let the model learn both the feature initialization as well as the region of interest per query [CMS+20], recent developments include a more strategic placement, either through sampling [MGJ21; RFGD22b], with a region proposal network [YALZ21], or through feature-informed initialization [GRDG24]. The first two mentioned concepts have been incorporated into TransLPC and are therefore described in the following, while the latter is subject of a follow-up paper [GRDG24] and detailed in section 5.6.3.

5.3.1 Learned Queries

The first model that performed query based object detection, DETR [CMS+20], initializes the M query vectors $\boldsymbol{y}_i \in \mathbb{R}^d$, $i = 1, \ldots, M$, randomly and treats their values as learnable parameters during training. Queries are not bound to a certain location. Incorporating this principle into TransLPC entails that the final bounding boxes are estimated in global coordinates, i.e. each query can potentially detect a bounding box anywhere in the measured range:

$$\boldsymbol{b}_i = (x, y, z, w, l, h, \gamma, v_x, v_y, \texttt{cls}), \tag{5.2}$$

where x, y, z denotes a location in 3D space, w, l, h is the width, length and height, γ is an orientation in the BEV plane and v_x, v_y denotes a velocity that can be estimated during detection due to the common practice of temporal lidar sweep aggregation as introduced in section 2.1.2. `cls` denotes a class score. During training, the query initialization is learned, whereas it is held constant during inference. This means that the same set of queries interacts with each input measurement without adaptation. Even though queries are not bound to a location, research has shown that the they still tend to find objects in certain regions of interest, which can be observed during inference [CMS+20] and may not always fit to the current scene. Some issues arise with a learned query initialization, which are described in the following.

5.3.2 Sampled Anchor Locations

TransLPC operates on lidar data, which is often sparse, in the sense that a measured sweep can contain areas with no observed points. Such empty spaces can, for example, be caused by a large building that occludes the space behind it. However, when using a fixed, learned distribution of initial query locations, it is not possible to accommodate for sparse spots in the current input frame: There will still be queries that try to find an object there. A second shortcoming of learned queries is that the bounding box locations are estimated in global coordinates, covering the whole measured range, which is commonly large in the context of autonomous driving. Therefore, an *input-dependent* initial query placement is proposed, which has shown to be beneficial for point cloud data [MGJ21; RFGD22b] and makes sure that no query is placed in empty space. For this, a fixed number of anchor locations $\boldsymbol{\rho}_i$, $i = 1, \ldots, M$, is sampled from the input point cloud via farthest point sampling (FPS) [ELPZ97]: $\boldsymbol{\rho}_1$ is sampled randomly, and all following $\boldsymbol{\rho}_j$ are drawn so that their distance to the previously sampled points $\{\boldsymbol{\rho}_1, \boldsymbol{\rho}_2, \ldots, \boldsymbol{\rho}_{j-1}\}$ is maximized. These anchor locations serve as a prior for the object search, and for each query, the final bounding box is estimated relative to its anchor location:

$$\boldsymbol{b}_i = (\Delta x, \Delta y, \Delta z, w, l, h, \gamma, v_x, v_y, \texttt{cls}), \tag{5.3}$$

so that the box location is given by $(x, y, z) = \boldsymbol{\rho}_i + (\Delta x, \Delta y, \Delta z)$. Since the anchor locations are low-dimensional, a high dimensional feature embedding is constructed from each of them, following [TSM+20]:

$$\boldsymbol{y}_i = \mathrm{FFN}\left[\sin(\boldsymbol{B}\tilde{\boldsymbol{\rho}}_i), \cos(\boldsymbol{B}\tilde{\boldsymbol{\rho}}_i)\right], \tag{5.4}$$

where FFN denotes a trainable feed-forward network, $\boldsymbol{B} \in \mathbb{R}^{\frac{d}{2} \times 3}$ is a matrix whose values were randomly sampled from a normal distribution, and the $\tilde{\boldsymbol{\rho}}_i$ is normalized

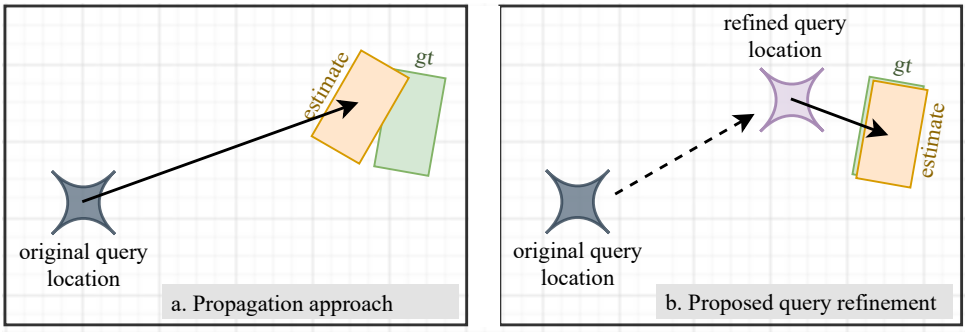


Figure 5.3: Illustration of different detection approaches. *Left:* Propagation approach. The query is fixed at its position. *Right:* Refinement enables the query to travel towards its object. [RFGD22b] © 2022 IEEE

so that $\tilde{\boldsymbol{\rho}}_i \in [0,1)^3$. The 3D locations are transformed to d-dimensional feature space via multiplication with $\boldsymbol{B}$ and concatenation of the sine and cosine of the product. Intuitively, the anchor information $\boldsymbol{\rho}_i$ is smeared into high-dimensional space, so that the model can access it.

5.4 Iterative Positional Refinement

A transformer decoder typically consists of multiple layers, which means that parts of its structure, namely an attention block followed by a feed-forward network, are repeated multiple times after one another. In Figure 3.2 C, this repetition is denoted as "$L\times$". In a *propagation approach*, the output of the previous layer is propagated directly to the following. A positional encoding is added before every layer, which is the same for all layers, since it encodes a fixed anchor position. Even though a query may detect an object at a distance, the object location is estimated relative to its original anchor position. Therefore, the location offset is larger the farther an object is away from the query, which is illustrated in Figure 5.3 (left). To prevent such large offsets, a *refinement approach* is proposed to enable a query to travel towards its estimated bounding box iteratively. As illustrated in Figure 5.3 (right), the query moves towards the object it is trying to detect. To enable this, a location update $(\Delta x, \Delta y, \Delta z)$ is computed from each output token $\boldsymbol{z}_i^{(0)}$ after the first layer, i.e. layer (0). This update is used to reposition the corresponding anchor

$$\boldsymbol{\rho}_i^{(1)} = (\Delta x, \Delta y, \Delta z)^\top + \boldsymbol{\rho}_i^{(0)}. \tag{5.5}$$

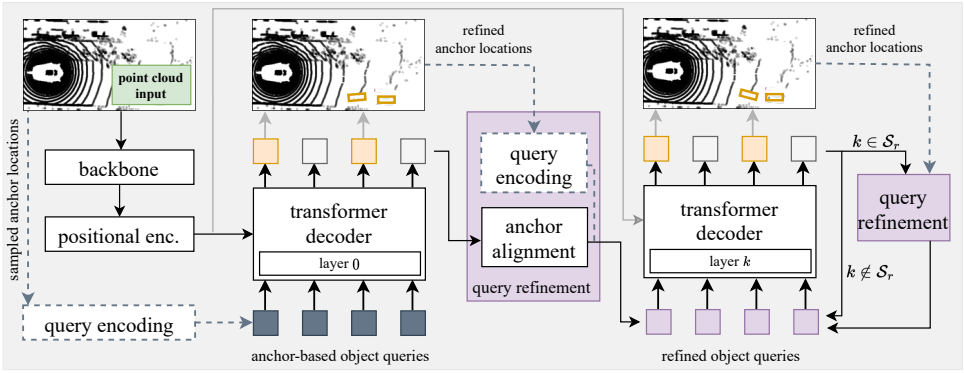


Figure 5.4: TransLPC model overview. After the first layer (layer 0), refined object positions are estimated and each query anchor location is moved to this location (purple box). Through anchor alignment, the query features are aligned with the new anchor. The refinement can be repeated after every following layer. Note that the first layer is pictured explicitly, while the following layers are abstracted as layer k. [RFGD22b] © 2022 IEEE

The new anchor is encoded into a new feature vector, resulting in the refined positional encoding $\boldsymbol{y}_i^{(1)}$ with Eq. (5.4). In the following layer, the bounding box is estimated relative to this new anchor location, which is generally much closer to it than the initial anchor. However, not only the updated anchor location, but also the remaining content of the previous layer's output, $\boldsymbol{z}_i^{(0)}$, shall be propagated to the following layer. To retain the information that is encoded in the vectors $\boldsymbol{z}_i^{(0)}$, they are transformed to align with their new anchor in a proposed anchor alignment module (AAM):

$$\tilde{\boldsymbol{z}}_i^{(0)} = \mathrm{AAM}\left(\boldsymbol{z}_i^{(0)}\right), \tag{5.6}$$

The AAM is a light-weight feed-forward network with a skip connection that is depicted in Figure 5.5. It is trained to a align a feature vector with its new anchor: First, a bounding box $\boldsymbol{b}_i = (\Delta x, \Delta y, \Delta z, w, l, h, \gamma, v_x, v_y, \texttt{cls})$ is computed from $\boldsymbol{z}_i^{(0)}$. Since the offset to the new anchor shall be zero after the AAM, the *target* bounding box of the transformed feature vector shall be $\tilde{\boldsymbol{b}}_i = (0, 0, 0, w, l, h, \gamma, v_x, v_y, \texttt{cls})$, i.e. all parameters except for the location offsets are unchanged, so that $\tilde{\boldsymbol{z}}_i^{(0)}$ encodes an offset of 0 w.r.t. the new anchor. An ℓ_1-loss is computed relative to $\tilde{\boldsymbol{b}}_i$ and back-propagated. The result of the query refinement is therefore an updated anchor embedding $\boldsymbol{y}_i^{(1)}$, as well as an aligned feature vector $\tilde{\boldsymbol{z}}_i^{(0)}$. These are input to the following layer (1), $\mathcal{Y}_1 = \{\tilde{\boldsymbol{z}}_i^{(0)} + \boldsymbol{y}_i^{(1)}\}_{i=1}^{M}$.

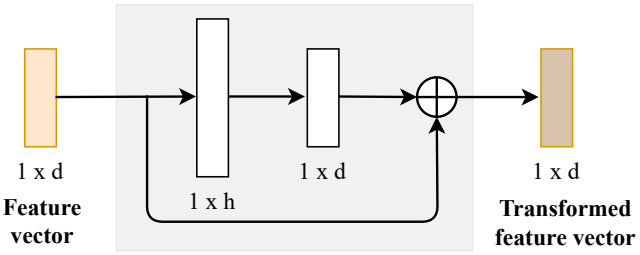


Figure 5.5: Overview of the anchor alignment module (AAM) [RFGD22b]. © 2022 IEEE

The refinement process can be repeated before every subsequent transformer layer or before a selection of layers $l \in \mathcal{S}_r$, so that

$$\mathcal{Y}_l = \begin{cases} \{\tilde{\boldsymbol{z}}_i^{(l-1)} + \boldsymbol{y}_i^{(l)}\}_{i=1}^{M} & \text{for } l \in \mathcal{S}_r \\ \{\boldsymbol{z}_i^{(l-1)} + \boldsymbol{y}_i^{(j)}\}_{i=1}^{M} & \text{for } l \notin \mathcal{S}_r \text{ and } l \neq 0 \\ \{\boldsymbol{y}_i^{(0)}\}_{i=1}^{M} & \text{for } l = 0 \end{cases} \tag{5.7}$$

is the set of object queries that is input to the lth layer. The latest anchor location encoding is always added to the propagated query, which is enforced by

$$j = \max\{k \mid (k < l) \wedge (k \in S_\mathrm{r})\}.$$

An overview of the proposed detection model with query refinement (in purple) is depicted in Figure 5.4.

5.5 Object Extraction and Multi-Class Detection

Originally, TransLPC has been restricted to single-class detection of cars. nuScenes [CBL+20], however, provides annotations for multiple additional classes of interest: *pedestrians*, *trucks*, *construction vehicles*, *buses*, *trailers*, *bicycles*, *motorcycles*, *traffic cones* and *barriers*. This section presents two different approaches to enhance TransLPC with multi-class detection, which is then denoted as TransLPC-M and has been pre-published in [RFGD23]. To provide a basis for this, object extraction is first described in a single-class setting (for class *car*).

5.5.1 Unified Bounding Box Regression

TransLPC uses a detection head to extract bounding box parameters from the feature vectors $\boldsymbol{z}_i$ that were computed by the transformer decoder. It consists of a *regression head*, which computes the following regressed parameters:

$$\mathrm{REG}\left(\boldsymbol{z}_i\right) = \left(\Delta x, \Delta y, \Delta z, w, l, h, \gamma, v_x, v_y\right), \tag{5.8}$$

where the 3D location is estimated relative to the anchor location $\boldsymbol{\rho}_i$. By design, REG is compatible with any intermediate layer's outputs $\boldsymbol{z}_i^{(k)}$ and even with the initial object queries $\boldsymbol{y}_i$. This makes a query interpretable as an object descriptor at any point in the model. Even though the query is a feature vector that is difficult to interpret on its own, it can be probed through the regression head to comprehend its behavior or movement and to supervise the model. The same holds for track queries in a tracking-and-detection setting.

Second, a *class head* CLS outputs a binary classification score, assigning each query's output either to the *car* class or to *background* (bg):

$$\mathrm{CLS}\left(\boldsymbol{z}_i\right) = \mathtt{cls} = \{\mathtt{cls}_{\mathrm{car}}, \mathtt{cls}_{\mathrm{bg}}\}. \tag{5.9}$$

The scores are normalized so that their sum is one. To train TransLPC, a forward pass is performed to obtain the class and regression parameters for each query output, i.e. one classified candidate box per query. To compute a loss, the estimates need to be matched to the given ground truth (GT) objects, similar to what has been illustrated in Figure 2.7. For this, a cost between each estimate and GT object is computed and placed in a cost matrix $\boldsymbol{C}_U^{\mathrm{car}} \in \mathbb{R}^{T_{\mathrm{car}} \times M}$, where T_{car} is the number of GT cars and M the number of object queries. The Hungarian algorithm [Kuh55] is used to find the cost-minimizing assignment, which matches each GT box to one estimate, whereas the remaining unmatched estimates are assigned to the *background* class. A regression and classification loss is computed accordingly. This process is performed for each decoder layer's output to supervise the detections. Note that loss computation and matching is only necessary during training.

In a naive extension to multi-class detection, assume that each initialized query can become any class of object, depending on what it finds in its vicinity. Now, the regression head is left unchanged, while the class head is extended to output a classification score for each of the ten classes of interest, as well as for the *background* class:

$$\mathrm{CLS}\left(\boldsymbol{z}_i\right) = \mathtt{cls} = \{\mathtt{cls}_{\mathrm{car}}, \mathtt{cls}_{\mathrm{truck}}, \ldots, \mathtt{cls}_{\mathrm{bg}}\}. \tag{5.10}$$

The highest score determines which class a query's object belongs to or if it didn't detect anything (i.e., background). The regression head shall output a box that aligns



Multiple traffic cones.



Large bus.

Figure 5.6: Examples of class shape inhomogeneity. © 2020 nuScenes [CBL+20]

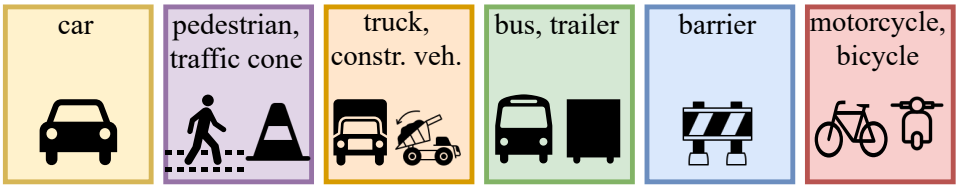


Figure 5.7: Overview of proposed class groups following [ZJZ+19]. [RFGD23] © 2023 IEEE

with the observed object. During training, the assignment strategy is unchanged compared to the single-class case, however, the cost matrix size increases to $\boldsymbol{C}_U \in \mathbb{R}^{T \times M}$, where T is the number of GT objects of all classes.

With this approach, the singular regression head REG needs to estimate bounding boxes of vastly different shapes, depending on which class the respective query is aiming to detect. Examples of the shape inhomogeneity in the nuScenes dataset is pictured in Figure 5.6, i.e. small traffic cones compared to a large bus. Besides this, the regression head is challenged by a second aspect: There exists an imbalance in terms of class prevalence. For example, 40 times more cars are observed than motorcycles in the nuScenes dataset. To solve these problems, class-specific regression heads are developed in the following.

5.5.2 Multi-Group Regression

Motivated by both the observed shape imbalance as well as imbalance of prevalence, Zhu *et al.* [ZJZ+19] propose to train multiple different regression heads to each learn class-specific features. For this, similar classes are combined into groups,

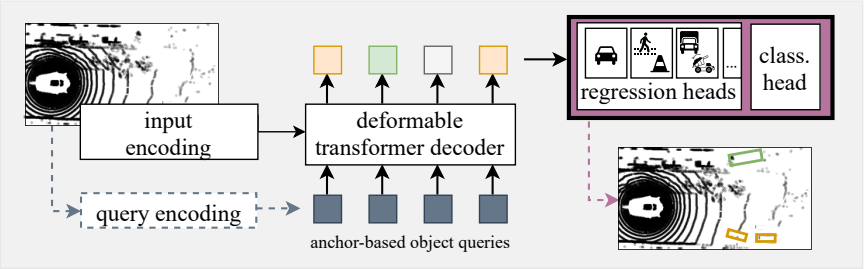


Figure 5.8: Overview of TransLPC-M with group regression heads (cf. [RFGD23]).

while dissimilar ones are separated. Such a grouping is illustrated in Figure 5.7. Contrary to transformer detectors, many commonly used object detectors rely on placing a large amount of anchor boxes on the bird's-eye view grid, as introduced in chapter 2, and they remove duplicates in post-processing. Since such anchor boxes are class-specific, each of them can be decoded with its own regression head. This makes the incorporation of grouped regression convenient and explains why it has been widely used with such methods, cf. [MMD20].

With query based object detection, however, group regression has not been used nor possible to the best of the author's knowledge. To enable its use for transformer object detection, this section presents an approach, which has been pre-published in [RFGD23]. Since the query self-attention in the decoder has quadratic memory requirements, the number of queries needs to be kept to a reasonable amount. This means that it is not feasible to cover the bird's-eye view with a plethora of class-specific queries. Therefore, to keep the number of queries bearable, each query shall be class-agnostic at initialization and be able to become any object class, depending on the measurement. An overview of TransLPC-M is given in Figure 5.8. Instead of one regression and one classification head, it comprises six different regression heads, one for each group pictured in Figure 5.7

$$\mathrm{REG} = \{\mathrm{REG}_{\mathrm{car}}, \mathrm{REG}_{\mathrm{ped./co.}}, \ldots, \mathrm{REG}_{\mathrm{bike}}\}, \tag{5.11}$$

where *ped./co.* refers to pedestrians and traffic cones, and *bike* to motorcycles and bicycles. The classification head remains as defined in Eq. (5.10). Their compatibility with all intermediate outputs $\boldsymbol{z}_i^{(k)}$ remains.

Training The training procedure of TransLPC-M needs to be adapted to multi-group regression: Each query now constitutes six candidate boxes, one through each

regression head. To prevent the matching cost matrix $\boldsymbol{C}_G$ from becoming six times as large as the unified one, $\boldsymbol{C}_U$, the following is proposed: Since the class of each GT object is known, it shall only be compared to one candidate box per query, i.e. the one computed by the corresponding regression head. This means that, for each row t in $\boldsymbol{C}_G$, the associated GT box $\boldsymbol{b}_i^{[t]}$ with class $\texttt{cls}^{[t]}$ is compared to

$$\{\mathrm{REG}_{\mathrm{gr_cls}^{[t]}}(\boldsymbol{z}_i), \mathrm{CLS}(\boldsymbol{z}_i)\}, \tag{5.12}$$

$i = 1, \ldots, M$, where $\mathrm{gr_cls}^{[t]}$ denotes the group that $\texttt{cls}^{[t]}$ belongs to. With this technique, the size of $\boldsymbol{C}_G$ is equal to $\boldsymbol{C}_U$. Besides this, it ensures that each query can be matched at most to one GT box, and that a GT box can only be matched to a fitting group head's output. Since the regression head parameters are only trained with objects from their group, they are able specialize on group-specific characteristics.

Query Refinement Between the decoder layers, queries can travel towards their object, i.e. refine their location. In the multi-group case, each query corresponds to six candidate boxes. Which one shall it travel towards? Two options are pictured in Figure 5.9. The query can either travel towards a weighted mean of candidate boxes (left) or to the most prominent one based on the class score. Since the former allows for smooth gradients during training, it is selected. The sum of the respective group's class scores are used as weights. However, the second refinement alternative may be beneficial during inference, which is analyzed in the results section.

Model Details TransLPC-M utilizes a PointPillars [LVC+19] backbone for encoding the measured lidar point cloud. In the transformer decoder, it features a deformable attention module, following [ZSL+21], which restricts the cross-attention's field of view to a local area to speed up training convergence in the challenging multi-class use case. $M = 200$ queries are used to accommodate for the increased number of observable objects in a multi-class setting. The nuScenes classes are grouped into six sets of classes, as illustrated in Figure 5.7. For training the model, a fade-out augmentation strategy is used, which is introduced in the following section.

Application to Existing Methods The main contribution of TransLPC-M is an adaptation to the regression head(s), as well as the corresponding GT matching and training strategy. Its core idea can be applied to other query-based object detectors from related and concurrent work. To showcase this, the lidar object detector TransFusion-L [BHZ+22] is extended with multi-group regression, as well as with an optional single-class regression, i.e. one individual regression head per class.

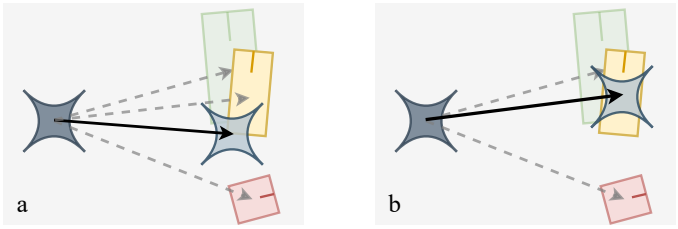


Figure 5.9: Illustration of two different options for query refinement with group regression. *Left:* The refined query position is placed at the weighted mean of object hypotheses. *Right:* The query is moved toward the strongest hypothesis. [RFGD23] © 2023 IEEE

For this, the model's original regression head is replaced and the training strategy is adapted, whereas the rest of the method remains unchanged. Results for this are presented in section 5.7.3.

5.6 Further Advances Towards Competitive Performance

Transformer-based object detection has been an active field of research since the initial concept was proposed by [CMS+20]. While TransLPC has enabled the use of this technology for autonomous driving, it can be improved in some areas to reach SOTA detection performance: Data augmentation increases the data diversity during training by performing operations each loaded frame. The backbone determines the level of detail that the encoded input data contains. Finally, concurrent research has shown that a feature-informed query initialization is beneficial to the 3D object detection performance, which inspired the development of a novel initialization strategy that is introduced in this section.

5.6.1 Data Augmentation

Since the amount of available labeled training data is typically limited, data augmentation is a way to increase diversity at a low cost. In the context of 3D object detection with point clouds, two kinds of augmentations are common [RSS21]: The first type is a transformation applied to the full point cloud. This can include a

random translation, rotation, scale, and/or a flip that mirrors the point cloud. To the model, each modified point cloud looks different, so that it is less prone to overfit on training data that it has seen repeatedly. Another type of augmentation is GT sampling [YML18], which aims to increase the prevalence of objects to detect. Based on the labeled 3D bounding boxes, the measured points from each individual object are cropped from their point clouds and saved in a GT database. Then, during training, additional objects are sampled from this database and placed in the scenes. All mentioned augmentation strategies are used in TransLPC's training.

While the GT sampling improves the model's performance on underrepresented classes [YML18], it presents the model with a data distribution that is different from the real world. Both the prevalence, as well as the placement of objects differs, since the augmentation typically does not reason about a placement that fits to scene, i.e. it does not respect road or building boundaries. This is especially problematic with a transformer-based approach, since it can, potentially, observe the full scene and road structure via cross-attention. Since disabling GT augmentation completely is harmful to the model's performance is, a fade-out strategy is applied to TransLPC-M: The GT sampling is disabled in the last five training epochs only, which aims to provide the model with a more realistic perspective in the last stages of the training, following [WMZY21b]. The effect of this training strategy is evaluated in section 5.7.3.

5.6.2 Enhanced Backbone

Both TransLPC and TransLPC-M utilize a PointPillars backbone to transform the measured lidar point cloud into a feature representation in 2D BEV. For this, the space is divided into a set of pillars that are placed on a 2D grid in BEV, as illustrated in Figure 2.3 (right). While is a comparably slim backbone, its relatively coarse grid and limited resolution in the height dimension limit the detection performance. TransLPC v2 aims to achieve a competitive detection performance to provide a solid basis for tracking. Therefore, it features a Voxelnet [ZT18] backbone. Note that this comes at the cost of increased runtime. At the time of its release, PointPillars achieved a runtime of 62 Hz, which was two to four times faster than voxel-based methods at the time [LVC+19]. With the most recent optimization techniques, the runtime of both backbones may be improved, but a trade-off between runtime and accuracy remains.

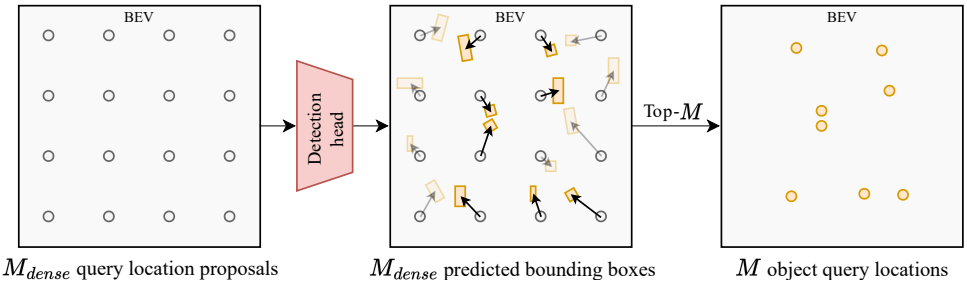


Figure 5.10: Illustration of EfficientQ3M's query initialization. Using the current measurement, a dense grid of queries is generated by sampling features at the respective locations from all modalities. Through a detection head, a location offset and confidence is estimated for each. The most confident locations are selected as initial query locations. [GRDG24] © 2024 IEEE

5.6.3 Feature-Informed Initialization

Taking input-dependent query initialization to the next level, one can think about placing queries already at probable object locations rather than using a simple sampling strategy. In [YALZ21], potential object locations are estimated in an initialization phase and the object queries are placed at these locations, which can reduce the necessary number of decoder layers. Yao *et al.*, however, did not see an improvement in detection performance, in the context of object detection on 2D images. On the contrary, they found that the initial query placement does not affect the final 2D detection score at all, given enough decoder layers to refine the object locations. With lidar data, on the other hand, objects can be very close to one another in relation to the measured range. To detect or track all of them, a sufficient amount of queries needs to be placed in densely populated areas. In this regard, a feature-informed query initialization may be beneficial, not only for efficiency reasons, but to also improve the performance.

This work differentiates two different types of feature-informed query initialization for 3D object detection: The first one has been proposed by [BHZ+22] for their method TransFusion. This method uses the current lidar feature map to estimate bird's-eye view heatmaps of potential object locations, one for each class of interest. The heatmap estimator is supervised with a target heatmap that is created from the GT object locations, following Centerpoint [YZK21]. Queries are placed at the estimated heatmaps' M most prominent peaks. For lidar data, this method has shown to improve both performance as well as efficiency, therefore it is incorporated into TransLPC v2 in an unmodified manner and is denoted *heatmap* in table 5.1.

Even though most experiments in this thesis are performed with lidar data, the work aims to be compatible with a multi-modal setting. However, the described heatmap initialization strategy [BHZ+22], has some shortcomings when extended to a multi-modal setting: In order to estimate initial object locations also from camera data, TransFusion [BHZ+22] follows a lidar-first principle, i.e. the camera input is only used to refine locations obtained with lidar so that the method can not function without the lidar sensor. Besides this, the camera component in the initialization is designed inefficiently, both in terms of latency as well as the number of learnable parameters, since an elaborate transformer network is used (in addition to the detection transformer that is used after the initialization).

Therefore, a novel query initialization strategy is presented in the following, which has been developed jointly with M. van Geerenstein [GRDG24] and is called EfficientQ3M. It aims to be efficient in the multi-modal fusion setting, as well as modular, so that it can be used with any decoder. An overview of the method is depicted in Figure 5.10. The main idea is described as follows: First, the BEV grid is densely covered with M_{dense} query location proposals $\boldsymbol{\rho}_{i,\text{dense}} \in \mathbb{R}^3$, where M_{dense} is much larger than the desired number of queries M. Note that the locations only vary in the x- and y-dimension, and are set at a fixed height z. Then, each query's position is used to determine its location in the respective sensor coordinate systems. For the images, this entails computing view projections, according to the camera calibration parameters. Then, current sensor features are sampled for each proposal, from the respective locations:

$$\boldsymbol{f}_{i,\text{dense}} = \text{FFN}\left(\boldsymbol{f}_{i,\text{lid}} \oplus \boldsymbol{f}_{i,\text{cam}}\right), \tag{5.13}$$

where $\boldsymbol{f}_{i,\text{lid}}$ and $\boldsymbol{f}_{i,\text{cam}}$ are sampled lidar and camera features, respectively. They are concatenated and passed through a feed-forward network to obtain one feature vector per proposal location, $\boldsymbol{f}_{i,\text{dense}}$. The sampling strategy follows [CZW+23]. Note that it mimics a cross-attention, which only has access to one selected reference location. It is light-weight, since it does not provide access to the full BEV grid, no attention scores need to be computed and no self-attention is performed. The resulting feature vector is processed through a detection head, i.e., a regression and a classification head, to obtain a location offset $(\Delta x, \Delta y, \Delta z)$, as well as a classification score for each $\boldsymbol{\rho}_{i,\text{dense}} \in \mathbb{R}^3$. Finally, the M object queries are initialized from the top-M highest scoring proposals, i.e. $i \in \mathcal{S}^M$, at the locations

$$\boldsymbol{\rho}_i = \boldsymbol{\rho}_{i,\text{dense}} + (\Delta x, \Delta y, \Delta z)\,. \tag{5.14}$$

The proposed strategy is evaluated both in a lidar-only setting, as well as for camera-lidar fusion, cf. section 5.7.4.

5.7 Detection Evaluation

The progression of TransLPC and its variants has been introduced in this chapter. To give an overview, their characteristics are listed in tables 5.1 and 5.2. In this section, experiments are presented in chronological order, so that the improvements over time become apparent. It is structured as follows: The single-class model TransLPC [RFGD22b] is evaluated in detail first, both in terms of performance, as well as to investigate the effect of its individual components. Then, the results of TransLPC-M are presented, which allows for multi-class detection [RFGD23]. Finally, TransLPC v2 incorporates a feature-informed query initialization. This concept is further improved with EfficientQ3M, which has been developed in joint work with M. van Geerenstein [GRDG24].

Table 5.1: Overview of proposed detection variants (encoding)

Name	Backbone	Voxel size [m^3]	Encoder	Query init	Reference
TransLPC	PointPillars	$0.2^2 \times 8$	none	sampled	[RFGD22b], [RFGD22c]
TransLPC-M	PointPillars	$0.2^2 \times 8$	deformable	sampled	[RFGD23]
TransLPC v2	VoxelNet	$0.075^2 \times 0.2$	none	heatmap	-
EfficientQ3M	VoxelNet	$0.075^2 \times 0.2$	none	multimodal	[GRDG24]

Table 5.2: Overview of proposed detection variants (decoding)

Name	Decoder	# Queries	Regression head	Reference
TransLPC	transformer	100	unified (single class)	[RFGD22b], [RFGD22c]
TransLPC-M	deformable	200	grouped (multi-class)	[RFGD23]
TransLPC v2	transformer	200	unified (multi-class)	-
EfficientQ3M	deformable	200, 900	unified (multi-class)	[GRDG24]

5.7.1 Quantitative Results of TransLPC

TransLPC has been trained on the nuScenes `train` dataset on an NVIDIA A100 GPU with an initial learning rate of 10^{-4} and a batch size of 30 to detect objects of type *car*. Only lidar data is used. The number of decoder layers, channel size and

number of object queries are set to $L = 6$, $d = 256$ and $M = 100$, respectively. The model is first trained for 300 epochs without query refinement, so that the anchor alignment module can be trained based on that. Then, the training is continued for another 300 epochs with query refinement. The learning rate is decreased every 200 epochs by a factor of 10.

Table 5.3: Object detection results of TransLPC compared to its baseline (class: car) [RFGD22b] © 2022 IEEE

Method	AP↑	ATE (m)↓	ASE (1-IOU) ↓	AOE (rad)↓
PointPillars [LVC+19]	0.684	0.281	0.164	0.204
Propagation approach	0.713	0.310	0.158	0.086
TransLPC	**0.759**	**0.251**	**0.157**	**0.080**

Since TransLPC is a transformer-based detection method with a PointPillars backbone, the original PointPillars model [LVC+19] is a suitable baseline for comparison: It has the same backbone but a different detection method combined with it. In table 5.3, quantitative results on the nuScenes `val` dataset are presented in terms of the nuScenes metrics introduced in section 2.2.3. Here, *Propagation approach* refers to a variant of TransLPC without query refinement. The results show that the proposed method is able to outperform its baseline, both in terms of *average precision*, as well as the true positive metrics average *translation*, *size* and *orientation* error. Besides this, a performance improvement due to the query refinement is apparent. Note that only detections for the *car* class are output and evaluated.

While the PointPillars model requires a non-maximum suppression (NMS), transformer-based methods typically do not require any post-processing. To showcase this, TransLPC is evaluated with and without NMS in table 5.4. Two variants of NMS are tested, one with a minimal overlap for filtering of 10%, one with 20%. While a slight improvement due to NMS is visible, it is not significant enough to justify the post-processing effort, therefore making one advantage of TransLPC apparent: It does not require a non-maximum suppression.

Table 5.4: Non-maximum suppression analysis with TransLPC (class: car) [RFGD22b] © 2022 IEEE

	Overlap	AP↑	ATE (m)↓	ASE (1-IOU) ↓	AOE (rad)↓
No NMS	-	0.759	0.251	0.157	0.080
NMS	0.1	0.759	**0.245**	**0.156**	**0.077**
NMS	0.2	**0.762**	0.247	0.157	0.079

5.7.2 Query Refinement Evaluation

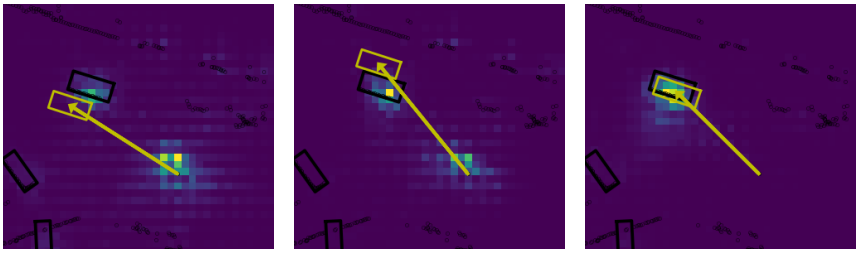
To examine the effect of the query refinement, it is applied to different extends to TransLPC: One model has been trained with no refinement at all, which is referred to as the *propagation approach*. Besides this, models with refinement only after layer 1, after layers $\{1, 3, 5\}$ and after each layer are compared.

Detection Results For all model variants, the detection results on the nuScenes `val` dataset for the *car* class are listed in table 5.5. The results show that the propagation approach is clearly outperformed by the refinement, even if performed only once after the first layer. Applying it again after every second or all layers brings some slight additional improvements, though it is apparent that the very first refinement is the most beneficial one.

Attention Visualization In Figure 5.11, the detection of one specific object is compared for the *propagation* (top) and *refinement* approach (bottom). Both the ground truth objects as well as the measured lidar data is colored black. The original anchor location (denoted by the base of the yellow arrow) does not move in the top row, forcing a box estimation from a distance. With refinement, on the other hand, the location offsets become smaller with every decoder layer, as the query travels towards its object. Intuitively, an accurate detection becomes easier, once smaller offsets need to be estimated. Note that the query travels farthest after the first layer, explaining why the refinement is especially beneficial in this step.

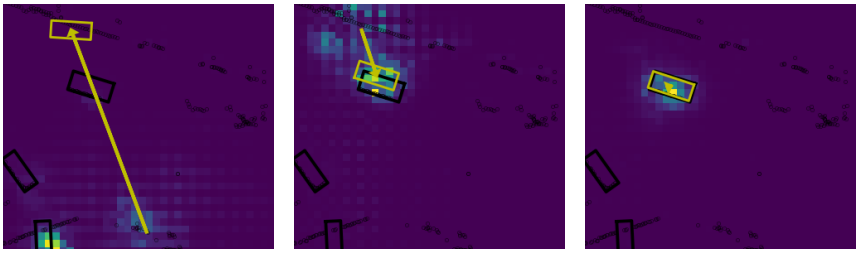
The attention weights between the query and the BEV are depicted as the color of the respective BEV grid cells. For both approaches, the attention shifts towards the detected object through the layers is observable, though some focus remains at the original query location in the *propagation* approach.

Query Travel Length How does the distance a query travels towards an object affect the detection accuracy, quantitatively? In Figure 5.12 (top), the median bounding box location error w.r.t. the query travel length, i.e. the difference between the initial anchor location and the final bounding box, is depicted in green for the *propagation approach*. The error bars denote the 25th and 75th percentiles. One can observe that the error increases with larger travel lengths. This means that the propagation approach is very susceptible to the initial query placement. For reference, the relative frequency of observed travel lengths is plotted in gray, showing that even travel lengths above 7 meters occur relatively frequently.



Layer 0 Layer 1 Layer 5

(a) Propagation approach



Layer 0 Layer 1 Layer 5

(b) Query refinement approach

Figure 5.11: Object detection through multiple decoder layers pictured for one object query on an excerpt of the BEV range. The query refinement method (*bottom*) allows the query to travel towards the object of interest, while its location stays unchanged in the propagation approach (*top*). [RFGD22b] © 2022 IEEE

With the *refinement approach* on the other hand, the dependency of the location error on the distance between the first query location (FQ) and the final box is flattened (orange), since the queries are able to travel closer to the object than with their original placement. The histogram in this plot shows the relative frequency of travel lengths between the last refined query position (LQ) and the final box, showing that almost all final query locations are very close to their bounding box and that only small position refinement is necessary from there.

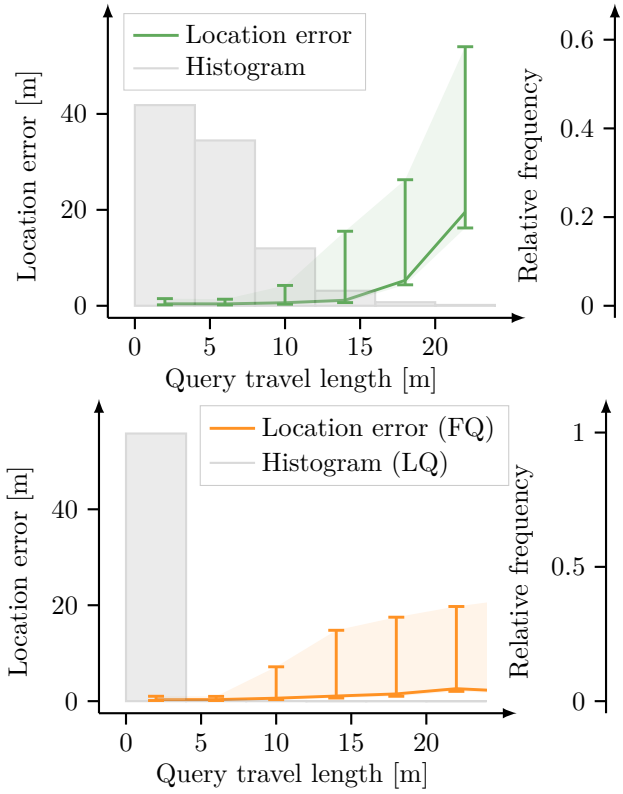


Figure 5.12: In color, median location errors w.r.t. the query travel lengths are displayed, with 25th and 75th percentiles. The gray histograms depict the relative frequency of travel length occurrence. *Top:* Propagation approach. *Bottom:* With query refinement. [RFGD22b] © 2022 IEEE

Table 5.5: Query refinement variants with TransLPC (class: car) [RFGD22b] © 2022 IEEE

Method	S_r	AP↑	ATE (m)↓	ASE (1-IOU) ↓	AOE (rad)↓
Propagation	$\{\varnothing\}$	0.713	0.310	0.158	0.086
Once	$\{1\}$	0.750	0.260	0.157	0.082
Every 2nd	$\{1, 3, 5\}$	0.756	0.255	0.157	0.081
After each	$\{1, 2, 3, 4, 5\}$	**0.759**	**0.251**	**0.157**	**0.080**

5.7.3 Multi-Class Detection

As introduced in this chapter and in [RFGD23], TransLPC-M allows for multi-class detection. This model is evaluated in this section. Furthermore, the group regression concept can be applied to any given transformer detector, which is also demonstrated, as well as its effect on the multi-class detection.

nuScenes Evaluation Table 5.6 presents quantitative results of TransLPC-M: The model is evaluated on the nuScenes `val` dataset, both with and without a grouped regression head as well as with and without the fade-out strategy during training. The average precision metric is listed, both as a mean over all classes (mAP), as well as individually per class. Different to the previous TransLPC, the novel model is based on deformable attention rather than standard attention, and is able to outperform it on the *car* class.

Concerning the model variants, the results show that the fade-out training strategy has a positive effect on the results: In the last epochs, the model is able to observe the true data distribution rather than the augmented one, which would have more objects than usual. Regarding the group regression, a slight benefit is visible compared to unified regression, though the model is able to perform multi-class detection with a single regression head almost equally well. This indicates that, contrary to what one would expect, the nuScenes class shape imbalance can be handled by the model. However, this may be different in a dataset with an even larger imbalance.

Query Refinement Through multi-group regression, six candidate boxes per query can be computed. Which of them should a query travel to during refinement? In training mode, a refinement towards the weighted mean location is performed. During inference, it is also possible to travel towards the best scoring candidate box. Both options are compared in table 5.7, which shows that even during inference, a score-weighted mean location is the better choice.

Table 5.6: Results of different Multi-Class Detection Approaches [RFGD23] © 2023 IEEE

Method	Regr. type	Fade	mAP↑	AP (car)↑	AP (truck)↑	AP (pedestrian)↑	AP (bus)↑	AP (bicycle)↑	AP (trailer)↑
TransLPC	unified	✗	-	0.727	-	-	-	-	-
TransLPC-M	unified	✗	0.329	0.734	0.419	0.498	0.578	0.042	0.271
	group	✗	0.339	0.736	0.436	0.500	0.589	0.046	0.273
	unified	✓	0.380	0.755	0.453	**0.541**	**0.603**	0.116	0.320
	group	✓	**0.384**	**0.758**	**0.455**	0.539	0.590	**0.135**	**0.326**

Table 5.7: Query refinement ablation for grouped multi-class detection [RFGD23] © 2023 IEEE

Method	mAP↑	mATE (m)↓	mASE (1-IOU) ↓
Weighted Mean	**0.384**	**0.534**	0.264
Top Candidate	0.378	0.538	0.264

Group regression head behavior Do the grouped regression heads learn which shape their classes of interest typically have? In Figure 5.13, the shape distribution of all objects from the nuScenes `val` dataset in terms of length is pictured in orange. Next to it, the observed output shapes from the individual regression heads are depicted in green. Notably, the output shape distribution is very similar to the GT one, indicating that the heads indeed specialize on their respective classes. One exemption, though, is the group with *bus* and *trailer*, where the estimated median length is smaller than the ground truth. Since this group includes both long truck trailers and long buses, but also shorter car trailers, its shape distribution may differ significantly between the training and validation split. Given this observation, it may be beneficial to group the shorter trailers with a more similar class.

Another question of interest is how the different regression heads behave for one

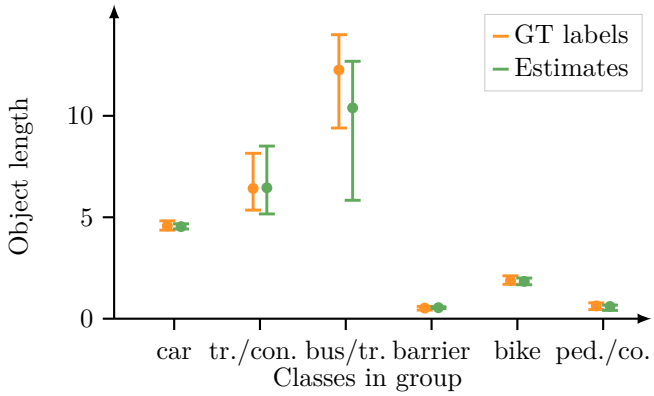


Figure 5.13: *Orange:* Distribution of the GT labels from the nuScenes `val` dataset in terms of the length parameter. *Green:* Observed length distribution from the group regression heads' outputs. Median values are plotted, along with their 25th and 75th percentiles. [RFGD23] © 2023 IEEE
Class names: car, truck/construction vehicle, bus/trailer, barrier, bicycle/motorcycle, pedestrian/traffic cone

given query. Do the object candidates that they output differ significantly, or do they remain close to one another? This is analyzed in Figure 5.14: For each query on the nuScenes `val` dataset, a standard deviation has been computed between its six candidate boxes' locations. Their median is plotted in color. The observations show that, while the regression heads use their ability to output different locations from one another, they stay in a similar region in the bird's-eye view, which can be considered the query's region of interest.

Application to TransFusion-L The presented group regression head can be applied to any transformer-based detection model. To showcase this, it is applied to the SOTA lidar detector TransFusion-L [BHZ+22]. For this, its unified regression head is replaced with grouped regression, while all other parameters remain unchanged. Besides this, a variant with singular regression is tested, which means that one individual regression head per class is created. Compared to TransLPC-M, TransFusion is based on a stronger backbone with smaller voxel size and comprises a feature-informed query initialization, therefore achieves more competitive detection results on the nuScenes benchmark. The results of this experiment in table 5.8 show that, while the grouping is beneficial for some classes, the observed differences are not significant. This shows that group regression can be applied to a given

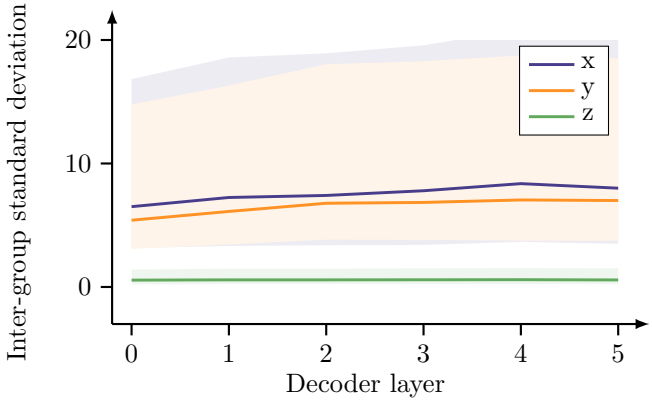


Figure 5.14: Median inter-group standard deviation, along with the 25th and 75th percentiles. For each query on the nuScenes `val` dataset, a standard deviation between all six candidate boxes obtained from the group regression heads in terms of location has been computed. [RFGD23] © 2023 IEEE

transformer detector without harming the performance. This may be beneficial for further research, e.g. when detecting classes with a larger shape imbalance is of interest.

Discussion The results indicate that a transformer detector is able to perform multi-class detection both with a single unified regression head for all classes, as well as with dedicated grouped class heads. In this regard, transformer detectors are more robust than dense anchor-based detection approaches such as PointPillars [LVC+19], where either grouped heads or one individual head per class are commonly used and bring benefit [ZJZ+19]. However, one can think of applications where the grouped regression heads can still be used in combination with a transformer detector: They allow for a variable parameterization of object attributes, e.g. one could add a lines detector that detects line markings as poly-lines rather than bounding boxes. Another use-case may be to understand the output of the lower scoring heads as additional object hypotheses, especially when detecting objects which can not be classified with certainty, such as a large van that could also be a truck. In addition, one could think of detecting classes with a more extreme shape imbalance.

Table 5.8: Results of the group head applied to the TransFusion-L [BHZ+22] detector [RFGD23] © 2023 IEEE

Method	Regr. type	Fade	mAP↑	AP (car)↑	AP (truck)↑
TransFusion-L [BHZ+22]	unified	✓	**0.611**	0.851	0.552
TransFusion-L - grouped	group	✓	0.600	**0.854**	**0.554**
TransFusion-L - single	singular	✓	0.602	0.849	0.538

Continued from above:

AP (pedestrian)↑	AP (bus)↑	AP (bicycle)↑	AP (trailer)↑
0.864	0.692	**0.492**	**0.404**
0.855	0.680	0.457	0.396
0.857	**0.693**	0.470	0.395

5.7.4 Feature-Informed Query Initialization

More recent state-of-the art transformer detectors commonly include a feature-informed query initialization [YALZ21], since it allows for more accurate detections as well as more efficient models. TransLPC v2 utilizes the query initialization strategy from TransFusion [BHZ+22]. This model is not evaluated here, since it does not provide additional contributions in the field of object detection, but rather serves as the basis for the deep tracker TransMOT v2.

EfficientQ3M, on the other hand, contributes to object detection and is evaluated in the following. It has been developed with two goals in mind: Firstly, to improve the efficiency and performance of the query initialization. Second, this method broadens the scope from lidar-only detection to multi-modal sensor fusion and aims to create an initialization that can cope with temporary unavailability of one sensor. Its detection results for lidar-video fusion are reported in table 5.9, both for the nuScenes `val` and `test` dataset in terms of mean average precision as well as the nuScenes detection score [CBL+20]. EfficientQ3M outperforms all related transformer based camera-lidar detectors on the nuScenes `val` dataset and is second best on the `test` dataset. Note that this model is based on FUTR3D [CZW+23], i.e. the only difference between EfficientQ3M and FUTR3D is the query initialization strategy, which causes a significant performance improvement.

In Figure 5.15, a direct comparison between FUTR3D's and EfficientQ3M's performance after each transformer layer is displayed. Since FUTR3D uses a fixed set of initial query positions, it requires more transformer layers to converge to good performance. Besides this, it requires a large amount of queries (900). EfficientQ3M,

Table 5.9: EfficientQ3M results on the nuScenes `val` and `test` dataset. [GRDG24] © 2024 IEEE

Method	Modality	`val`		`test`	
		mAP ↑	NDS ↑	mAP ↑	NDS ↑
PointAugmenting [WMZY21a]	L+C	-	-	0.668	0.710
TransFusion [BHZ+22]	L+C	0.673	0.709	0.689	0.716
DeepInteraction [YCM+22]	L+C	0.699	0.726	**0.708**	**0.734**
FUTR3D [CZW+23]	L+C	0.703	0.731	0.694	0.721
EfficientQ3M	L+C	**0.712**	**0.735**	0.705	0.726

on the other hand, can outperform it with only 200 queries, and converges already after three decoder layers. This holds true both for a lidar-only and lidar-camera fusion setting.

One motivation to develop EfficientQ3M's query initialization strategy has been the need for an efficient query initialization compared to the one introduced as part of TransFusion [BHZ+22]. To quantify whether this is the case, the runtime of both the base variant of FUTR3D [CZW+23], as well as the same model combined with TransFusion's query initialization are measured and compared to EfficientQ3M's runtime. The three models are all based on FUTR3D and only differ in the query initialization strategy. The results in table 5.10 show that EfficientQ3M's query initialization only adds 0.9 milliseconds to FUTR3D's runtime. This is four times faster than TransFusion's initialization strategy and shows that the proposed initialization is indeed more runtime-efficient than its alternative. Note that the absolute runtime values noted in the table are measured from code that has not been optimized for speed, therefore there is potential to decrease them, e.g. to make the model suitable for running in a vehicle.

Table 5.10: Comparison of FUTR3D [CZW+23] combined with different query initialization strategies, based on [GRDG24]

Initialization Strategy	Queries	mAP ↑	Runtime (ms)	Init time (ms)
FUTR3D (input-agnostic)	900	63.7	**208.6**	-
w/ TransFusion	200	64.5	212.7	4.1
EfficientQ3M	200	**64.7**	209.5	**0.9**

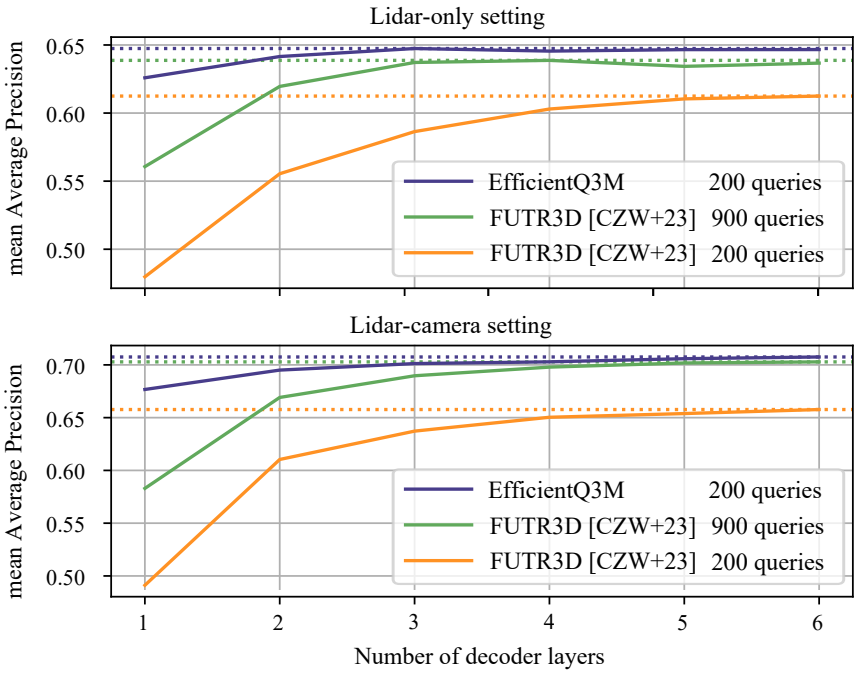


Figure 5.15: Per-layer performance of EfficientQ3M and FUTR3D [CZW+23] with varying number of queries. *Top:* Lidar-only setting. *Bottom:* Lidar-camera fusion. [GRDG24] © 2024 IEEE

5.8 Discussion

This chapter has presented the development of a transformer based 3D object detection method that is applicable to lidar point clouds measured from a vehicle. When no suitable model in this domain was available in the beginning, TransLPC became one of the first transformer detectors that could cope with processing large lidar input point clouds. It outperformed its baseline, PointPillars [LVC+19], and has been extended to a multi-class detector. In the following years, the research field has progressed rapidly, so that TransLPC lacked behind the more current state of the art. However, note that some aspects of TransLPC have become part of these more current SOTA transformer detectors, such as the left-out transformer encoder and the query refinement, e.g. in [CZW+23].

Inspired from the results of concurrent work, two additional models have been

presented in this chapter: TransLPC v2 is the basis for TransMOT v2, but does not contribute to the field of object detection, since it is based on [BHZ+22] with no modification. Still, it is used within TransMOT v2 to provide SOTA detections to the joint detector and tracker and to enable further contributions to the tracking task. Besides this, EfficientQ3M has been presented, which has been pre-published in [GRDG24], and includes a novel query initialization strategy. This method has improved the object detection performance compared to the SOTA methods on the nuScenes `val` dataset, as shown in table 5.9.

The result of this chapter is an object detector in multiple variants that has contributed to the novel field of transformer object detection and can compete with the SOTA in its later iterations. So far, the presented models have not been optimized to run in a vehicle, but have been run locally on a linux machine with a V100 Nvidia GPU on pre-recorded data. Recent developments show, however, that similar models can achieve a runtime below 100 ms when optimized for vehicle deployment [Bru24], which allows to use them in practice.

Chapter 6

Multi-Object Tracking Based on Attention

TransLPC and its versions lay a solid foundation for building a joined tracking and detection model upon it. This chapter includes all aspects that are necessary for this, and that this work contributes to. Parts of it have been pre-published in the author's own publications: TransMOT is a *car*-only tracker for lidar data measured from a moving vehicle [RFGD22a], which follows the joined tracking and detection paradigm. Its core components have been introduced in chapter 4, whereas this chapter provides a more detailed perspective. Subsequently, TransMOT is extended to enable multi-class tracking [RFGD23]. Finally, the latest iteration of the model, TransMOT v2, is presented, which benefits both from recent progress in the detection domain, as well as from including physical constraints into the method. Similar to the previous chapter, the development is described in chronological order.

An overview of TransMOT, the very first tracker iteration, is given in Figure 6.1. On the initial frame of a sequence, a detection step is performed to initialize the tracker (Figure 6.1 a.). The resulting feature vectors are carried over to the next time step and serve as track states (b.) to be updated through the transformer decoder based on new information. An ego-motion correction between measurements is necessary, which is discussed in sections 6.2 and 6.5.2. The tracker runs sequentially until stopped and can therefore operate on sequences of any length.

6.1 Track Life Cycle

A track lives through multiple phases, including birth, death and possibly temporary occlusion. This section describes how a track's life cycle is modeled in TransMOT. While the concept follows [MKLF21], it has been brought to the 3D domain in the

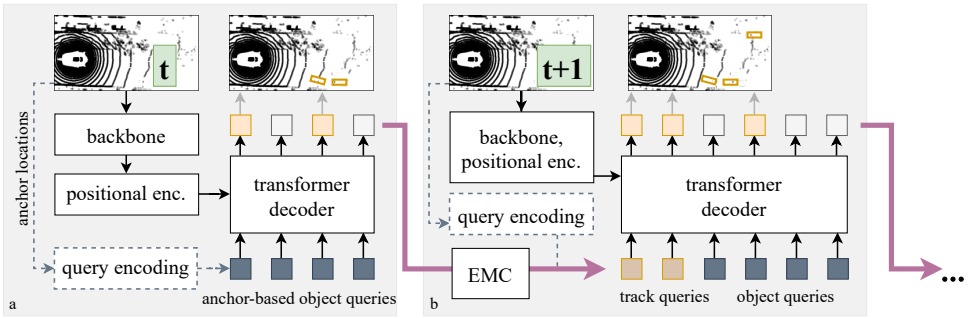


Figure 6.1: TransMOT overview. [RFGD22a] © 2022 IEEE

author's paper [RFGD22a]. The track management depends on whether the model is currently in training mode, i.e. when its weights are trained, or not, during inference.

6.1.1 Training Mode

During training, GT labels are available and therefore the whereabouts of each track are known over time. This information is used to teach the model how to perform tracking.

Initialization This step is depicted in Figure 6.1 a: On the first frame of a sequence, no track queries are available yet, therefore, only the object queries $\mathcal{Y}_t = \{\boldsymbol{y}_{i,t}\}_{i=1}^{M}$, which are depicted as gray boxes at the input of the transformer decoder, are initialized from M sampled anchor locations. With them, a detection step is performed through the transformer decoder to obtain the output feature vectors $\mathcal{Z}_t = \{\boldsymbol{z}_{i,t}\}_{i=1}^{M}$ (light gray and orange boxes). Each of these vectors has been computed from an initial query and is a candidate to become a track query, i.e. to be propagated to the next time step. To decide which of them to keep, the regression and classification heads are applied, extracting a class score and bounding box estimate $\boldsymbol{b}_i$ from each. Since GT information is available during training, a matching between GT and the estimates is performed with the Hungarian method to determine which queries' estimated are closest to GT. Note that the number of GT objects K is smaller than the number of estimates M, so that those queries that constitute to a match become track queries $\{\boldsymbol{z}_{k,t}\}_{k=1}^{K} \subset \mathcal{Z}_t$ (orange), whereas the others are discarded (light gray).

Track update and birth As pictured in Figure 6.1 b, once available, both the track queries and object queries are presented to the decoder, i.e. $\mathcal{Y}_{t+1} = \{\boldsymbol{z}'_{k,t}\}_{k=1}^{K} \cup \{\boldsymbol{y}_{i,t+1}\}_{i=1}^{M}$, where $\boldsymbol{z}'$ denotes ego motion-corrected track queries. The track queries are used to update existing tracks, whereas the object queries are re-initialized at each time step to facilitate the detection of newly spawned tracks.

A forward pass of the model is run to obtain $K + M$ outputs from the queries (cf. Figure 6.1 b). During training, it is known which GT track each of the K track queries belongs to. Therefore, tracks known to be continued are matched accordingly to compute a loss. Occluded tracks can either be interpolated in training, or assigned to *background* during the interruption. Tracks known to have finished at time $t + 1$ are assigned to the *background* class as well, to teach the model to suppress the track in this case.

At any time, a new object to track may appear. For this, the object queries $\{\boldsymbol{y}_{i,t+1}\}_{i=1}^{M}$ serve as slots for new spawns. Similar to the initialization phase, tracks known to start at time $t + 1$ are matched to the M bounding box estimates obtained from the object queries and a successful match becomes a new track query.

Occlusion and termination A track exists as long as its track query is passed to the model at subsequent time steps. As mentioned above, a track query is brought to the frame $t + 1$ if the object exists at time t. If it disappears at time $t + 1$, the model is trained to detect that it is missing, and the respective track query is discarded. Though, if an object is only occluded, this is known from the GT information, and the track query is kept until it reappears.

6.1.2 Inference mode

Once the training has finished, the model weights are frozen and it can be used to function as a tracker. Obviously, no GT data is provided any more. This means that the track life cycle functions differently in this mode.

Initialization In the first frame of a sequence, the object queries $\mathcal{Y}_t = \{\boldsymbol{y}_{i,t}\}_{i=1}^{M}$ are initialized and input to the model, same as during training, and bounding box estimates $\boldsymbol{b}_i$ are computed from the resulting output feature vectors $\mathcal{Z}_t = \{\boldsymbol{z}_{i,t}\}_{i=1}^{M}$. To decide which of them to propagate as track queries, a confidence threshold λ_{det} is applied, so that object queries that led to a detection above this threshold are considered to have found a newly spawned track. This threshold is a manually

tunable parameter. To select it, one can decide on a desired detection probability and select the threshold so that this rate is achieved empirically on a given data sample. Which value has been chosen is mentioned in the experiments section for the respective experiment. The selected queries above the threshold constitute the set of new track queries $\{\boldsymbol{z}_{k,t}\}_{k=1}^{K}$.

Track update and birth Again, both the track queries and object queries are presented to the decoder, i.e. $\mathcal{Y}_{t+1} = \{\boldsymbol{z}'_{k,t}\}_{k=1}^{K} \cup \{\boldsymbol{y}_{i,t+1}\}_{i=1}^{M}$. During inference, a track is continued if its track query leads to a detection above a confidence of λ_{track}, which is a second manually selected parameter. If the confidence is too low, the track is first considered occluded and then terminated after some time. The termination time is a trade-off between the option to continue an occluded track and the risk that a retained query may lead to a false positive detection. Unless mentioned otherwise, queries are kept for 1.5 seconds for possible re-occurrence after an occlusion, since this results in stable results in practice.

A new track is spawned if a *detection query* leads to a detection above λ_{det}, and the respective feature vector is propagated as an additional track query. Note that $\lambda_{\text{det}} > \lambda_{\text{track}}$ to require a large confidence for track birth, whereas an existing track can have lower confidence temporarily.

Occlusion and termination As mentioned above, once a track scores below the confidence threshold λ_{track}, its query is kept for a set amount of time to enable possible re-detection. A larger amount of time allows for longer occlusion times, while it should not be set too large to avoid false re-detections. Once this grace period is over, the track query is discarded and the track is considered terminated.

6.2 Ego-Motion Correction in Feature Space

The ego-vehicle can move a significant distance between measured frames. In the nuScenes dataset, lidar frames are annotated at a rate of 2 Hz, resulting in a movement of 5.5 meters at 40 kph. If one or two frames are dropped, ego pose change magnitudes of up to 18 meters can be observed. The movement of the vehicle makes an ego-motion compensation (EMC) indispensable. In principle, EMC can be performed in two different ways: One can either adjust the measured point clouds from an ego-centered to a global coordinate system (*type I*), or, alternatively, keep the ego-vehicle centered and correct the tracker state on a per-object level accordingly (*type II*).

At first glance, a *type I* correction may seem easier, i.e. operating in a global coordinate frame. However, as described in the previous chapter, a bird's-eye view feature map is encoded from the measured point cloud and input to the transformer decoder. To retain its spatial context, it is enhanced with a positional encoding beforehand. This is necessary, since the original grid structure is not visible to the transformer. The positional encoding covers the area around the ego vehicle, i.e. a space of ca. 100×100 meters in an ego-centered system. To enable a *type I* EMC, the encoding would need to cover the entire global coordinate system, i.e. the full area the car can drive on during a sequence. However, a positional encoding has a limited resolution, and therefore a limitation in area it can cover. Besides this, the boundaries of the driven area would need to be known in advance to define the positional encoding accordingly, which is infeasible in practice. Therefore, this work opts for an object-level EMC (*type II*), which is presented in the following. Since the tracker state is encoded in feature space in the form of track queries, this is a non-trivial task.

6.2.1 Method Overview

The proposed EMC module accepts a feature vector as input, in conjunction with the ego pose change to be corrected. Its output is a feature vector that has been corrected according to the ego motion. The ego pose change between measured frames is assumed to be known in form of a translation and a rotation. This information is provided by the nuScenes dataset. In a real-world application, one could, for instance, measure the ego pose with an IMU and GPS system and refine it with available algorithms [CBL+20].

Since the correction takes place in latent space, a trained module is proposed for this, an overview of which is given in Figure 6.2. The input feature vector of length d is first transformed with a FFN to a smaller channel length k. In parallel, the input pose change is transformed to a larger channel size to enable it to be reshaped into a matrix of size $k \times k$. Inspired from a transformation in 3D space, the dot product between this matrix and the track query is computed to transform it. Then, the result is transformed back to its original channel length and a residual connection is applied, aiming to prevent loss of information. Note that, since this is a learned model, common characteristics of a 3D affine transformation matrix are not guaranteed to hold for the $k \times k$ transformation matrix. Still, it only depends on the pose, therefore the same matrix is applied to all feature vectors of a given frame, allowing for a joined transformation, similar to an affine transformation. With learned queries, the EMC module is formalized as

$$\boldsymbol{z}'_{i,t} = \mathrm{EMC}(\boldsymbol{z}_{i,t}, \boldsymbol{p}), \tag{6.1}$$

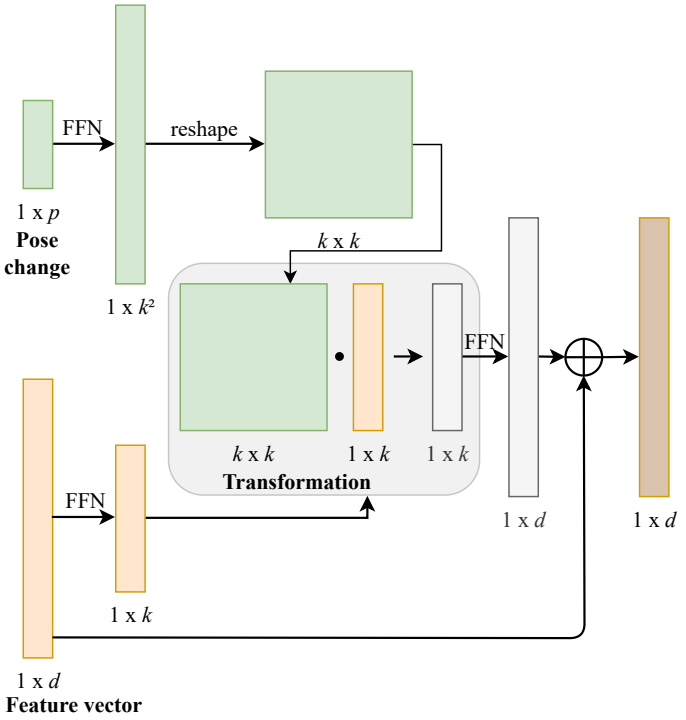


Figure 6.2: EMC module overview. Both the pose change and the feature vector are transformed to a different length to enable a dot-product transformation. [RFGD22a] © 2022 IEEE

where $\boldsymbol{p}$ denotes the ego pose change $\boldsymbol{p} = (t_1, t_2, t_3, q_0, q_1, q_3, q_4)$ in terms of a 3D translation (t_1, t_2, t_3) and a rotation (q_0, q_1, q_3, q_4) in quaternion form [Beu03]. $\boldsymbol{z}_{i,t}$ denotes a decoder output token, selected to be a track query, at time t and $\boldsymbol{z}'_{i,t}$ its ego motion-corrected version. With anchor-based queries, the anchor location can be corrected deterministically, whereas the rest of the parameters in feature space are transformed by the module:

$$\left(\boldsymbol{z}'_{i,t}, \boldsymbol{\rho}'\right)_i = \mathrm{EMC}(\boldsymbol{z}_{i,t}, \boldsymbol{\rho}_i, \boldsymbol{p}). \tag{6.2}$$

To transform the anchor $\boldsymbol{\rho}_i$, it is moved to its expected position in 3D space, $\boldsymbol{\rho}'_i$, based on $\boldsymbol{p}$. A new anchor encoding is computed following Eq. (5.4). In this case, the EMC module also aligns the transformed feature vector with its new anchor.

6.2.2 Training Strategy

The EMC module is trained separately from the rest of the tracking and detection model. For this, the detector is trained first and its weights are frozen. For each frame of the nuScenes `train` dataset, a detection forward pass is performed, obtaining the decoder output vectors for this frame. The detection head is applied to each of them to receive a classified bounding box, $\boldsymbol{b}_i = (x, y, z, w, l, h, \gamma, v_x, v_y, \texttt{cls})$. This allows to compute a GT ego-motion corrected bounding box from the translation and rotation of the known ego pose change applied to $\boldsymbol{b}_i$. The location, orientation and velocities in x- and y-directions change due to the ego-motion correction, whereas the object size and class remain the same. The GT corrected box for $\boldsymbol{b}_i$ is denoted as

$$\boldsymbol{b}'_i = (x', y', z', w, l, h, \gamma', v'_x, v'_y, \texttt{cls}). \tag{6.3}$$

Note that $\boldsymbol{b}'_i$ is only computed during the training of the EMC module, not during inference. In the case of anchor-based queries, one has additional capabilities, since not only the feature vector can be transformed, but also the anchor location that belongs to a query. Then, the anchor is placed directly at the ego-motion corrected location $\boldsymbol{\rho}'_i = (x', y', z')$, while the GT box for the compensation in feature space is set to $\boldsymbol{b}'_i = (0, 0, 0, w, l, h, \gamma', v'_x, v'_y, \texttt{cls})$, where the first three entries denote a distance of zero to the new anchor $\boldsymbol{\rho}'_i$.

To train the EMC module, all decoder output feature vectors of a frame (including those that did not find a confident object) are input to it, along with the pose change to the next frame. To increase robustness, a frame drop is simulated by using either the pose change to the frame after the next, or the one after that. This pose change frame distance is sampled uniformly from $d \in \{1, 2, 3\}$ for each training frame, which serves the goal to both encounter large and small pose changes, which the module shall be able to correct after training.

From the EMC output vectors, a bounding box is extracted again with the detection head, to check the correction result. An ℓ_1-loss is computed between the corrected and the target corrected box and the model weights are updated accordingly. The ℓ_1-loss has been selected because it is often used to train the regression heads in object detectors, and also applied to the regressed bounding boxes in that case. The EMC module is trained for 30 epochs. Note that, depending on the application, any kind of pose changes may be used to train the model. This can enable compatibility to a larger range of pose change magnitudes, if needed.

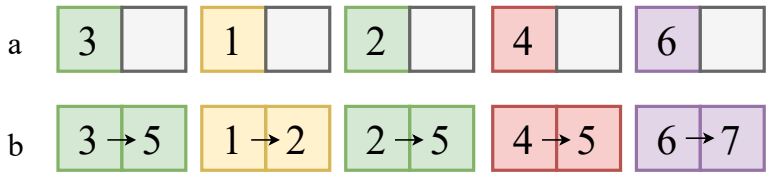


Figure 6.3: Illustration of pair-wise sampling. Each color marks a sequence.

6.3 Tracker Training Procedure

To train TransMOT, pre-trained weights of TransLPC and the EMC module are used. While the EMC module's weights remain fixed, the detector's weights are refined to accommodate for the tracking task. Many deep multi-object trackers are trained on frame pairs, i.e. on sequences of a very short length. One of them is TrackFormer [MKLF21] and TransMOT has also been trained this way in its original form [RFGD22a]. The pair-wise training is described in the following, whereas a technique for training TransMOT v2 on longer sequences is developed afterwards.

6.3.1 Pair-wise Training

The training is performed on the nuScenes `train` dataset. Given a batch size b, b lidar frames are sampled randomly and serve as first participant of the pairs. They do not have to belong to the same scene, on the contrary, a diverse selection is helpful to prevent overfitting. This is illustrated in Figure 6.3 a, with an exemplary batch size of $b = 5$, where each scene is marked with a different color and the numbers denote the frame id within the sequence. Then, the second participants of the pairs are retrieved. They are sampled to either be consecutive to their partner, or two or three frames apart (Figure 6.3 b). This simulates dropped frames to learn a robustness against it.

A forward pass is performed on the first partners of the pairs (with efficient batching). Since there is no track history, this can be considered a detection step. Accordingly, the estimates are matched to GT with the Hungarian method and a detection loss is computed and back propagated, just as during standard detection training. A tracking loss is then computed from the second partners of the pairs: From the GT matching in the detection step, it is decided which track queries to carry over to the second frame. They are ego-motion corrected accordingly and passed to the model,

along with the object queries. Depending on if a track is continued in the second frame or not, the track queries' outputs are either matched to the respective GT box or to *background*. The group of newly spawned GT objects in the second frames are matched with the object queries' outputs, just like during detection training. From [MKLF21], the following track augmentation strategies are employed:

- Removal of track queries. With a probability of p_{rem}, a track query is discarded instead of propagated. The respective track is considered newly spawned in the second frame. This remedies the fact that newly spawned tracks occur less frequently than continued ones.

- Spawn a discarded track query. With a probability p_{sp}, a selected track query can spawn an additional one that has previously been discarded. The spawned query shall not be continued but assigned to the *background* class. It simulates a track query that has been propagated erroneously.

Other common forms of data augmentation, such as random rotation and translation of the point cloud as well as the placement of additional objects in a scene are incompatible with the tracking task since they are not consistent over consecutive frames. They are therefore disabled during tracker training. Given that such techniques have improved 3D detection methods significantly, it may be beneficial to adapt them for tracking as well, however, this would require additional research.

The described pair-wise training has multiple advantages. Firstly, the combination of a detection and a tracking loss in each step ensures that the model retains ability in both domains. Besides this, the pairing allows for efficient batching. Lastly, overfitting on a single scene is prevented by sampling randomly from all scenes. The sampling ensures that each measured frame is selected as first partner exactly once. One disadvantage, on the other hand, is that pair-wise training only allows for a history of one frame. This means that occlusion can not occur during training, since at least three frames are necessary to observe a track, its occlusion and reappearance. Presenting occlusion examples during training may, however, increase the model's performance in such situations during inference. Therefore, an approach to mix sequential with pair-wise training is introduced in the following, which is part of TransMOT v2.

6.3.2 Mixed Sequence- and Pair-wise Training

The development of a mixed sequence and pair-wise training follows these thoughts:

1. Training speed shall not be affected, i.e. batching should still be possible.

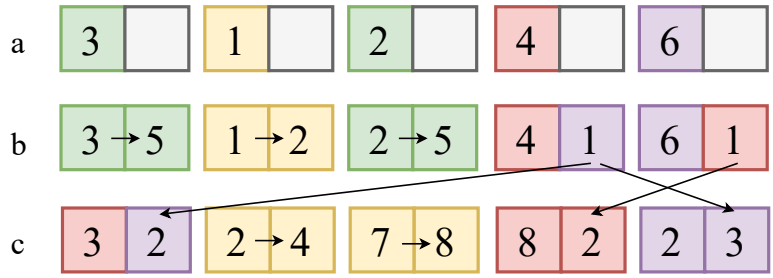


Figure 6.4: A mixture of pair-wise (green, yellow) and sequential training (red, purple). Best viewed in color.

2. Prevent overfitting on one sequence.

3. Retain a balance between detection and tracking loss.

4. Present occlusion cases in training and allow for a longer history.

An overview of the proposed method is given in Figure 6.4. A share of sequential training p_{seq} versus pair-wise training $(1 - p_{\text{seq}})$ can be set. According to it, each scene is assigned either to be presented sequentially (with probability p_{seq}), or pair-wise. This assignment is performed before each epoch and then fixed. In Figure 6.4, the green and yellow sequence are in the pair-wise group, whereas the red and purple ones are sequential. To keep training speed unchanged, the sampling structure is similar to before: In the first line of the figure (a), first partners of each sampling pair are sampled randomly. To them, a detection loss is applied. In line b, the pair-wise samples are assigned their partner, as before. For the others, however, the result from the detection step of the first partners is *discarded*. Instead, the second partner is read from a buffer. For each sequential scene, it contains information about the next consecutive frame and the latest history (i.e. track queries). At the beginning, no history is available yet, so the tracker is run on the first frames of each sequence. The resulting track queries are saved. Next, in line c, the process is continued: First partners have been sampled and the pair-wise ones are paired. The others are continued from the buffer: The second frame of the purple and red sequence with the respective buffered track queries are presented. This can be continued in the following, until the end of a sequence is reached. Note that, in line c, only two sequences are in the buffer, while three sequential spots need to be filled. Each time the buffer is read, the next consecutive frame is increased by one, so that the third slot receives the third frame of the purple sequence. Still, it obtains the track queries from frame 1, which emulates a frame drop and the EMC is performed accordingly.

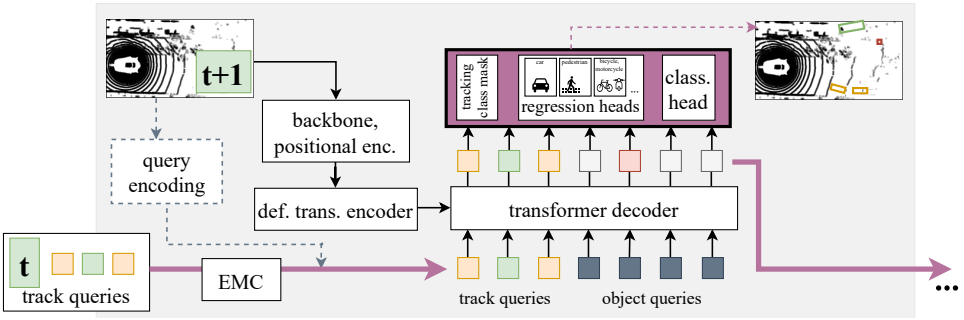


Figure 6.5: Multi-class tracking model overview (TransMOT-M) (cf. [RFGD23]).

Since the sequential sequences are presented in their original order, occlusion cases naturally occur during training. Still, the sequences are partitioned over multiple training steps, so that an overfitting on one sequence is prevented. Furthermore, the detection loss on first participants of all sampling pairs is unchanged, so that the balance between detection and tracking losses is kept the same.

6.4 Multi-Class Tracking

Tracking multiple different types of objects is crucial for autonomous driving, since not only surrounding cars are of interest, but also other road users, such as cyclists, pedestrians and trucks. The nuScenes dataset contains annotations for static and dynamic objects, but only (potentially) dynamic ones are considered for the tracking task, therefore, barriers, traffic cones and stationary construction vehicles, are excluded. Following the development of the detector TransLPC-M, TransMOT is equipped with multi-class regression heads and then called TransMOT-M, indicating multi-class capability. Similar to the detection task, both a multi-group regression as well as a unified one are compared in the evaluation section.

In Figure 6.5, an overview of TransMOT-M is given. During inference, to initialize the tracks in the very first frame, a detection step is performed with the grouped class heads. Those queries that yielded detections above a confidence threshold of $\lambda_{\text{track-m}}$ are ego-motion corrected and brought to the next time step. They are updated through the decoder. Since nuScenes only accepts tracks with a consistent class assignment throughout their existence, class switches within tracks are prevented by masking the outputs from other group heads for continued tracks.

In a less restrictive evaluation setting, however, it may be beneficial to explicitly allow such switches within tracks: The outputs from the different regression heads can be interpreted as multiple hypotheses for the location and shape of a tracked object, similar to multiple model methods [MABD98]. A track is typically initialized when an object is far away, i.e. when first entering the field of view. Such an object is covered by few lidar points, which makes a correct classification at first sight difficult. When multiple hypotheses are maintained, the class assignment is less strict, so that the correct class can become the most probable hypothesis after a few frames of observation, without needing to set up a new track for this.

6.5 Physics-Aware Deep Tracking

In TransMOT and TransMOT-M, the ego motion correction is performed in feature space and instead of an explicit per-agent prediction, the attention shifts towards a tracked object: The transformer learns an implicit predict-and-search strategy, which has been shown in Figure 4.4. Both models demonstrate that deep tracking is possible and has some advantages, such as a learned, soft association, as will be shown in the results section of this chapter. However, when taking a step back, one may wonder if some physics-based constraints may improve the tracking, while keeping its transformer-based core. This motivates the development of TransMOT v2, which is presented in the following. It is build upon a more powerful detector, TransLPC v2 as mentioned in the previous chapter, which uses a Centerpoint backbone and a feature-informed query initialization strategy based on [BHZ+22].

6.5.1 Short-Term Prediction

A transformer is capable of finding a tracked object again for its track update, even if it moved. However, it may need multiple decoder layers to do so. Besides this, a velocity is already estimated with each bounding box, which can easily be used for a prediction. Therefore, a short-term prediction is incorporated into TransMOT v2:

$$(x', y', z') = (x, y, z) + \Delta t \cdot (v_x, v_y, 0), \tag{6.4}$$

where Δt denotes the time difference between measured frames, (x, y, z) and (v_x, v_y) are the estimated location and velocity for one object. The anchor-based approach allows to move the respective anchor to the new, predicted position, in conjunction with the EMC. This is a deterministic operation that enables to give the transformer decoder a hint where it should look for its object. It has two shortcomings: Firstly,

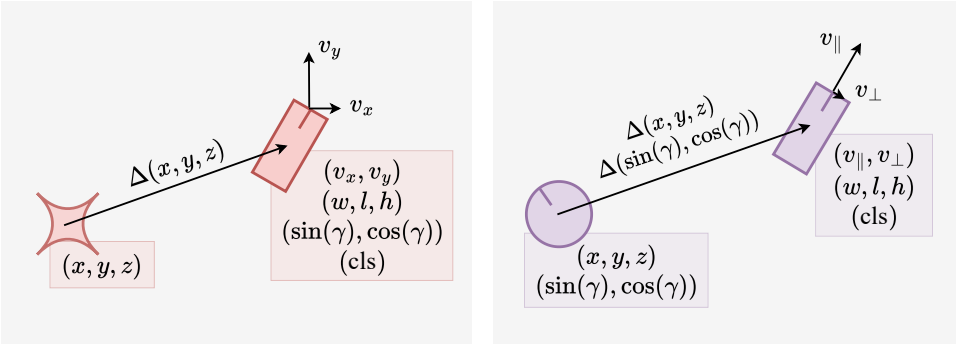


Figure 6.6: Comparison of different anchor approaches. *Left:* The bounding box is estimated relative to an anchor location. *Right:* The anchor also encapsulates an orientation.

it follows a constant velocity model, which may not be accurate. Secondly, only the objects' location is predicted, whereas their orientation and velocity stay unchanged.

Concerning the first issue, note that this prediction aims to avoid introducing additional manually tunable parameters. It can therefore be considered a coarse prediction, whereas the transformer is still required (and able) to find the actual object's location in the new measurement. A solution for the second issue, on the other hand, is presented in the following.

6.5.2 Orientation Queries

The experiments on the presented, trainable EMC module have shown that it is able to perform an accurate ego-motion correction in feature space. Such a module is indispensable when having a track state in feature space. The anchor location, however, opens up new possibilities, as a translation can be applied by moving the anchor location accordingly. Should all ego-motion correction be performed on the anchor only? This could have the advantage that the EMC is performed correctly, even in seldom cases that a trained EMC did not encounter during training. However, two things have to be researched: Can the other box parameters that need to be corrected by EMC be expressed as part of the anchor as well, such as the orientation? How can alignment of the new anchor with the propagated feature vector be ensured?

To incorporate an orientation into the anchors, *orientation queries* are proposed:

Each query is initialized not only with a location, but also an orientation:

$$\boldsymbol{\rho}_i = (x, y, \sin(\gamma), \cos(\gamma)), \tag{6.5}$$

where (x,y) denote a position in the BEV plane and γ is an orientation in terms of an angle that defines a rotation around the vertical z-axis. The estimated object's orientation is then expressed relative to the anchor orientation, just like the location. Note that the z-coordinate (height) is omitted in the anchor information, since it has a smaller value range and can be estimated in absolute terms. To transform the anchor information into an anchor encoding, the Fourier encoding as introduced in Eq. (5.4) is performed with an enlarged matrix, $\boldsymbol{B} \in \mathbb{R}^{\frac{d}{2} \times 4}$

$$\boldsymbol{y}_i = \mathrm{FFN}\left[\sin(\boldsymbol{B}(\tilde{\boldsymbol{\rho}}_i \odot \boldsymbol{n})), \cos(\boldsymbol{B}(\tilde{\boldsymbol{\rho}}_i \odot \boldsymbol{n}))\right]. \tag{6.6}$$

Here, $\odot$ denotes an element-wise multiplication, $\tilde{\boldsymbol{\rho}}_i = \boldsymbol{\rho}_i / ||\boldsymbol{\rho}_i||$ is normalized so that its values range from 0 to 1. $\boldsymbol{n} = 1/(1, 1, n_\gamma, n_\gamma)$ performs an additional normalization, which down-weights the influence of the orientation values on the encoding by dividing them by a positive natural number $n_\gamma \in \mathbb{N}$.

The estimated object velocity (v_x, v_y) is another parameter that needs to be ego-motion corrected, since it is expressed relative to the coordinate axes, which are centered around the ego-vehicle. To keep the anchor information as small as possible, and to decouple the the parameters that are estimated in absolute terms, the following is proposed: As pictured in Figure 6.6 (right), the velocity is expressed relative to the orientation of a detected object, with a component that is parallel to it and one that is perpendicular. This kind of velocity is not affected by the ego-motion and does not need to be corrected. Therefore, a full ego-motion correction can be performed by computing a new anchor position and orientation. For reference, the previous anchor design is pictured in Figure 6.6 on the left.

6.5.3 Deterministic Ego-Motion Correction

In Figure 6.7, an overview of deterministic EMC in conjunction with prediction is given. Both the prediction as well as EMC are performed by modifying the anchor parameters. For the prediction, a velocity estimate is obtained from the feature vector through the regression head. The feature vector itself is only modified to align it with its new anchor, with the light-weight anchor alignment module that has been introduced in the context of the query refinement, cf. Figure 5.5.

During the prediction and deterministic EMC, the anchor is first moved to the location of the estimated bounding box, i.e. $\boldsymbol{\rho}_i + (\Delta x, \Delta y, \Delta \sin(\gamma), \Delta \cos(\gamma))$. Then,

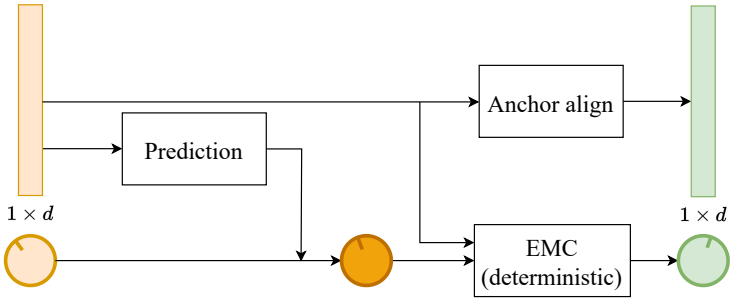


Figure 6.7: Illustration of prediction and EMC with orientation queries. Both transformations can be performed by moving and rotating the anchor. The feature vector is aligned with its new anchor via an anchor alignment module.

a prediction is added and the result is transformed according to the ego-motion pose change. This method enables a correct EMC while limiting disturbance on the feature vector. The final track query consists of the anchor-aligned feature vector, $\tilde{\boldsymbol{z}}_i^{(t)}$, with its new anchor, $\boldsymbol{y}_i^{(t+1)}$, that is computed from the corrected anchor position and orientation.

6.5.4 Track Update Layer

TransMOT v2 builds upon TransLPC v2, therefore features a stronger detection backbone and a feature-informed query initialization strategy. While this initialization improves the detection performance, it poses a challenge for the tracking task: Object queries are placed in locations that probably contain an object, based on the encoded input point cloud. Since their initialization is track-agnostic, some object queries will be placed on objects that belong to a continued track. The respective track query has to compete with them. If the object query prevails, a duplicate track is initialized. However, the track query is at a disadvantage: while the object queries have been initialized based on the newest measurement, the track query has only had access to the previous frame. To strengthen the track queries, a *track update layer* is incorporated into TransMOT v2. It is an additional decoder layer that allows the track queries to access the input data first, and to perform an initial update without disturbance by the object queries, as depicted in Figure 6.8. This pre-update allows the track queries to move towards their object and then compete with the object queries in the following layer under balanced conditions.

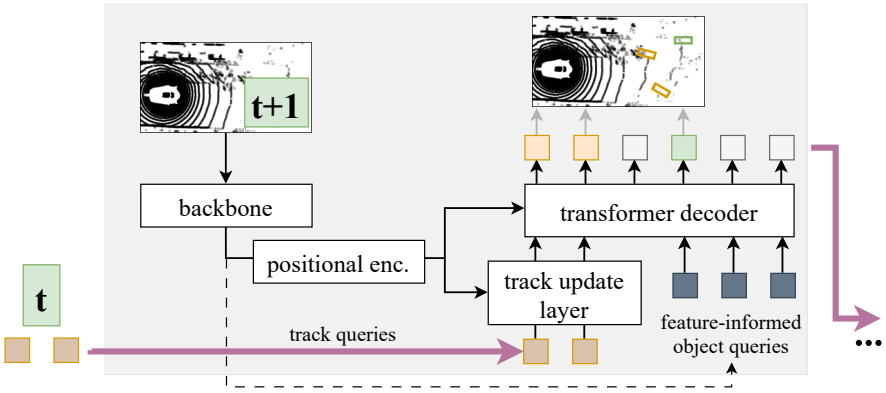


Figure 6.8: Illustration of the tracking step of TransMOT v2 with an additional track update layer that enables track queries to refine their position, given a novel measurement. The remaining structure is unchanged as depicted in Figure 6.1.

6.6 Evaluation

TransMOT has evolved from a proof of concept for the *car* class only, to the multi class detector and tracker TransMOT-M and to a model that enables competitive tracking performance, TransMOT v2. All model variants and their characteristics are listed in table 6.1. This evaluation section follows the chronological order, i.e. it provides an evaluation of TransMOT with EMC in feature space first, as well as an analysis on its susceptibility to dropped frames, compared to a Kalman filter based tracker [CPLB20]. Then, the multi-class tracking capability of TransMOT-M is evaluated, followed by an analysis of TransMOT v2, both in terms of the influence of certain model components, such as the track update layer, as well as general performance. A comparison is drawn to an RFS-based tracking filter, namely a PMBM filter [GWGS18], which has been introduced in section 2.3.1.

6.6.1 Quantitative Results of TransMOT

The tracker TransMOT is evaluated on the nuScenes `val` dataset. For this, the model is first pre-trained for the detection task. Then, the EMC module is trained for 30 epochs, given feature vectors from the transformer and observed pose changes

Table 6.1: Overview of proposed tracking variants

Name	Detector	EMC	Prediction	Training	Track Update	Query Initialization	Reference
TransMOT	TransLPC	learned	none	pair-wise	jointly	sampled	[RFGD22a], [RFGD22c]
TransMOT-M	TransLPC-M	deterministic	none	pair-wise	jointly	sampled	[RFGD23]
TransMOT v2	TransLPC v2	deterministic	simple	seq./pair-wise	track layer, jointly	feature-informed	-

from the nuScenes `train` dataset. Finally, the full model is trained for tracking in a pair-wise manner for 300 epochs with up to 2 intermediate frame skips and a learning rate of $3 \cdot 10^{-5}$.

The thresholds for spawning a new track and continuing an existing one during inference are set to $\lambda_{\text{det}} = 0.5$ and $\lambda_{\text{track}} = 0.3$, respectively. For the evaluation, the tracker is applied to full sequences of the nuScenes `val` dataset, i.e. each sequence is passed to the model sequentially from its start to end. The nuScenes `val` dataset consists of 150 sequences of ca. 20 seconds length, annotated at a rate of 2 Hz, so that a total of about 6000 frames are used.

A comparison to a widely used Kalman filter-based tracker called StanfordIPRL -TRI [CPLB20] is drawn. This tracker has achieved the first place in the nuScenes tracking challenge held at the AI Driving Olympics workshop at NeurIPS 2019 and was highly ranked for a long time. The authors have initialized the Kalman filter's parameters based on statistics from the nuScenes `train` dataset for each object class and utilize a Mahalanobis distance between the detected and predicted objects to associate novel measurements to existing tracks. The linear and angular velocities are modeled as random variables, but assumed constant during the prediction step. Details on the tracker's parameter settings are listed in appendix A. Among those trackers that were applicable to the nuScenes dataset at the time TransMOT was developed, StanfordIPRL -TRI is considered a SOTA filter-based tracking method, which is why it has been chosen for comparison.

To enable a fair comparison, both trackers process the same detections. To obtain

them, TransMOT is run in detection-only mode once (i.e. with no temporal context) and the resulting detection bounding boxes are passed to StanfordIPRL -TRI. This aims to compare the tracking performance disjunct from the detection performance, since both methods rely on the same transformer detections.

Table 6.2 presents the results in terms of the nuScenes metrics, as defined in section 2.3.3. StanfordIPRL-TRI requires a filtering of less confident detection proposals, which is done by removing boxes for which the detector estimated a score value below a certain threshold. Different values for this threshold have been tested, as listed in table 6.2. TransMOT is able to outperform StanfordIPRL-TRI in terms of both average MOT accuracy and precision (AMOTA, AMOTP). Note that the Kalman filter achieves the lowest overall number of ID switches (IDS) with a threshold of 0.9. This, however, comes at the cost of many fewer mostly tracked (MT) tracks, i.e. the track coverage is reduced significantly[1]. TransMOT, on the other hand, achieves the highest value in the mostly tracked metric.

Table 6.2: Tracking performance of StanfordIPRL-TRI [CPLB20] compared to TransMOT (class: *car*) [RFGD22a] © 2022 IEEE

Method	threshold	AMOTA↑	AMOTP↓	MT↑	FP↓	FN↓	IDS↓
StanfordIPRL-TRI [CPLB20]	0.3	0.596	0.880	1497	8018	19853	1677
	0.5	0.645	0.828	1721	6505	16801	542
	0.7	0.650	0.822	1689	6297	16809	559
	0.9	0.614	0.897	1522	**5167**	18216	**502**
TransMOT	-	**0.674**	**0.754**	**2096**	9449	**14071**	1403

6.6.2 Ego-Motion Correction in Feature Space

The EMC module is an essential component of TransMOT. Its correction performance is evaluated in this section. EMC is applied to the nuScenes `val` dataset, whereas the distance between consecutive frames is varied in $d \in \{1, 2, 3\}$, i.e. the time difference between them is 0.5 to 1.5 seconds long. This allows to observe large ego-pose changes of up to 18 meters in translation magnitude. The location of the ego-motion corrected feature vectors is compared to the expected corrected location.

In Figure 6.9, the location, size and velocity errors are evaluated w.r.t. the pose change translation magnitude and rotation angle. For this, the root mean squared

[1] A mostly tracked track is defined as a track which is covered by a true positive track estimate for at least 80% of its life-span [CBL+20].

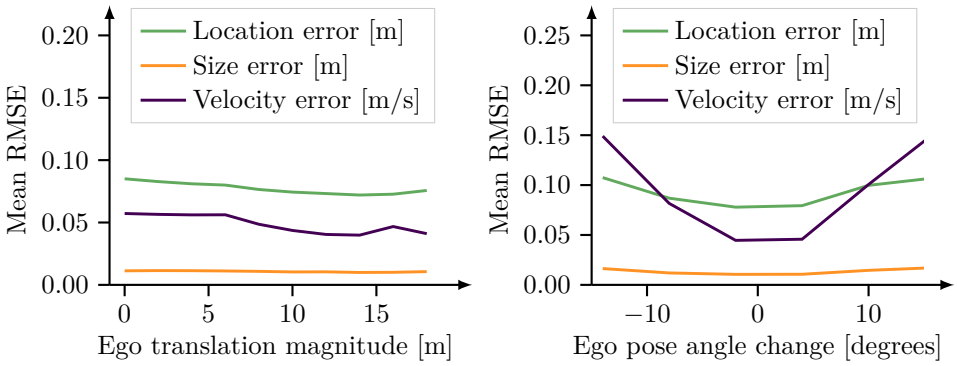


Figure 6.9: EMC module correction performance w.r.t. the ego pose change, i.e. the translation magnitude (*left*) and angle change (*right*). [RFGD22a] © 2022 IEEE

error (RMSE) values are computed for each corrected query, and averaged over all. The results show that the mean RMSE values for object *size* are very low, irrespective of the pose change, which indicates that the model leaves the size parameters unchanged, as intended. Notably, the location and velocity errors are similar for all pose translations of interest as well, i.e. the model can correct even large pose changes reliably, as pictured in Figure 6.9 (left).

Concerning the correction performance w.r.t. the ego pose change *angle*, a dependency of the errors on the angle is visible. This can be explained by the nuScenes data distribution: 95% of observed angles are between -6 and 6 degrees, therefore the EMC module learns to perform well in this range only. If a larger range is desired, one should adapt the training pose change distribution accordingly. To summarize, the results show that the trained EMC module is able to compensate for the ego motion accurately in feature space, given a pose change of up to 18 meters in translation magnitude and ± 6 degrees in rotation.

Table 6.3 presents an ablation study of TransMOT with different ego-motion compensation methods. The model is equipped with either no ego-motion compensation (*without*), the transformation is performed on the anchor locations only (*anchor transform*) or with the full EMC module. Unsurprisingly, the results show that an ego-motion compensation is a necessity for good performance. While the anchor transformation performs well already, the trained EMC module outperforms it slightly. Note that such an anchor transform is only feasible in a model setting with anchor-based queries, while the trained is EMC is more flexible: It can either be combined with a correction of the anchor position, if available, or it can be used

stand-alone in feature space.

Table 6.3: Ablation of TransMOT with and without EMC [RFGD22a] © 2022 IEEE

Method	AMOTA↑	AMOTP↓	MT↑	FP↓	FN↓	IDS↓
Without EMC	0.564	0.994	1703	**8364**	17162	4829
Anchor transform	0.657	0.791	2020	9307	15070	1699
With EMC	**0.674**	**0.754**	**2096**	9449	**14071**	**1403**

6.6.3 Susceptibility to Skipped Frames

An autonomous vehicle is a complex system that needs to be robust towards sensor latency and temporary unavailability of new measurements. In particular, for a tracker, this comprises two challenges: The sensor input may not be available at perfectly spaced time intervals, i.e. one can not expect that the exact same time has passed between different sets of two consecutive frames (small-scale time irregularities). Second, larger-scale irregularities may occur if a frame is skipped completely.

TransMOT contains modular components for the track update and ego-motion compensation. Therefore, the current state can be propagated to any arbitrary point in time at which a new measurement is available, given that the ego pose change is known. To test its ability to compensate for skipped frames, the inference on the nuScenes `val` dataset is performed, whereas each frame is skipped with a probability of p. Evaluation results are reported for varying values of p, both for TransMOT as well as for StanfordIPRL -TRI [CPLB20], for comparison. Both methods can accept measurements at arbitrary points in time. Therefore, both are expected to cope well in this setting. Note that both trackers operate on comparable detections, as described in section 6.6.1. The experiment shows that TransMOT is very robust to missing sensor input and outperforms StanfordIPRL-TRI significantly in this regard.

One reason for this may be that TransMOT has been trained with up to two intermediate frame skips and therefore may have been able to learn search patterns in the attention stage to find an object again, even after some time has passed. Such search patterns are a way to depict different motion models: Since the attention mechanism can access multiple candidate locations at the same time, it can explore different possible motion patterns and then choose the one that is supported by evidence of the object. StanfordIPRL -TRI, on the other hand, is not designed for

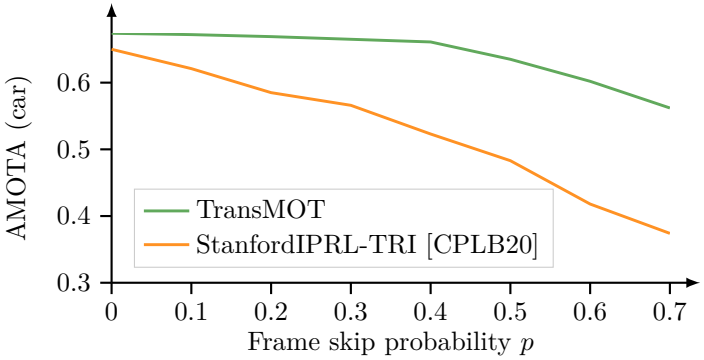


Figure 6.10: Susceptibility to skipped frames. Each frame is dropped with a probability of p. [RFGD22a] © 2022 IEEE

an irregular frame rate by the choice of its parameters. As detailed in appendix A, its process noise covariance parameters are obtained from empirical analysis of the nuScenes `train` dataset dataset, with a focus on a frame distance of 1, i.e. a difference of 0.5 seconds between measurements. As a subject of future research, its parameters could be re-tuned with a different frame distance.

6.6.4 Multi-Class Tracking

While previous experiments were restricted to the *car* class, TransMOT-M [RFGD23] is able to track all classes available in the nuScenes tracking challenge: car, bus, truck, trailer, pedestrian, bicycle and motorcycle. The model is evaluated in two settings: One comprises a unified regression head, meaning one learned head that regresses the box parameters of all classes. The second setting features multiple regression heads, one for each group of classes. The groups are, in parentheses: (car), (bus, trailer), (truck), (motorcycle, bicycle), (pedestrian).

Table 6.4 presents the results of both versions of TransMOT-M, as well as of the car-only tracker TransMOT. *Fade* indicates that the fade-out strategy has been applied, i.e. the GT augmentation was disabled in the last few epochs. Similar to what has been concluded for object detection in the previous chapter, the results show that the tracker is able to track objects of different classes with comparable performance in both regression approaches. This indicates that the unified regression head is able to compensate for the class imbalances in shape and prevalence observed in

Table 6.4: Joint tracking and detection with different multi-class approaches [RFGD23] © 2023 IEEE

Method	Regr. type	Fade	AMOTA↑	AMOTP↓	AMOTA (car)↑
TransMOT [RFGD22a]	unified	✗	-	-	0.674
TransMOT-M	unified	✓	0.378	1.281	0.662
	group	✓	**0.385**	**1.245**	**0.680**

Continued from above:

AMOTA (truck)↑	AMOTA (pedestrian)↑	AMOTA (bus)↑
-	-	-
0.385	0.331	0.623
0.383	**0.333**	0.623

the nuScenes dataset and therefore does not require further specialization. However, using multiple regression heads doesn't harm the performance either, therefore they may be beneficial in the following circumstances:

1. Presence of classes with larger shape difference. This evaluation has been restricted to physical objects of limited size. However, one could also track different things, such as the line markings on the road or occupied area. The former should not be described by a bounding box, but rather as a poly-line. A separate regression head for such a class could accommodate for that.

2. Interest in multiple object hypotheses. An exemplary situation is depicted in Figure 6.11: The model detects and tracks an object, but assigns *car* as the most probable class, although it is actually a motorcycle. Now, having multiple regression heads allows to inspect the second-most probable class, which, in this case, is (motorcycle, bicycle) with a confidence of 0.19 and a bounding box that fits to the object.

To summarize, the results show that the unified regression head does not necessarily require additional specialization in the nuScenes class setting. Since multi-group regression doesn't harm the performance though, it enables the aforementioned potential towards multi-hypotheses detection and tracking and for a larger class shape imbalance.



Figure 6.11: Example of a track, for which the second most probable hypothesis (bike/ motorcycle) is correct. Tracked boxes are pictured in color, while the GT is displayed in dotted gray. *Left:* Only the most confident hypothesis is pictured per query. *Right:* The circled object is replaced by its second best hypothesis with a confidence of 0.19. [RFGD23] © 2023 IEEE

6.6.5 Track Update Layer

TransMOT v2 features an input-informed object query initialization. This means that object queries tend to be initialized close to their object's location already and allow the transformer model to focus on fine-grained localization and orientation. As mentioned before, this method may pose a disadvantage to propagated track queries: Track queries are only aware of the previous frames' input data, whereas the object queries they compete with were computed from the current measurement already. To balance the available prior knowledge between track and object queries, a dedicated track update layer is part of TransMOT v2.

For reference, Figure 6.12 (top) depicts the median movement of track query positions per processing stage *without* a dedicated track update layer: Their global reference location changes most during the ego-motion compensation, a bit during prediction and some more during the joined object and track query update (Layer 1). Note that the 25th and 75th percentiles are denoted with blue error bars and the 5th and 95th percentiles in gray.

In the bottom row, a track update layer has been added, which is denoted as Layer 1, so that the joined update layer becomes Layer 2. One can observe that most of the track query movement now takes place in the secluded track update layer, so that both object and track queries can focus on a more fine-grained object localization in Layer 2. This indicates that the desired effect is achieved, i.e. that both track and

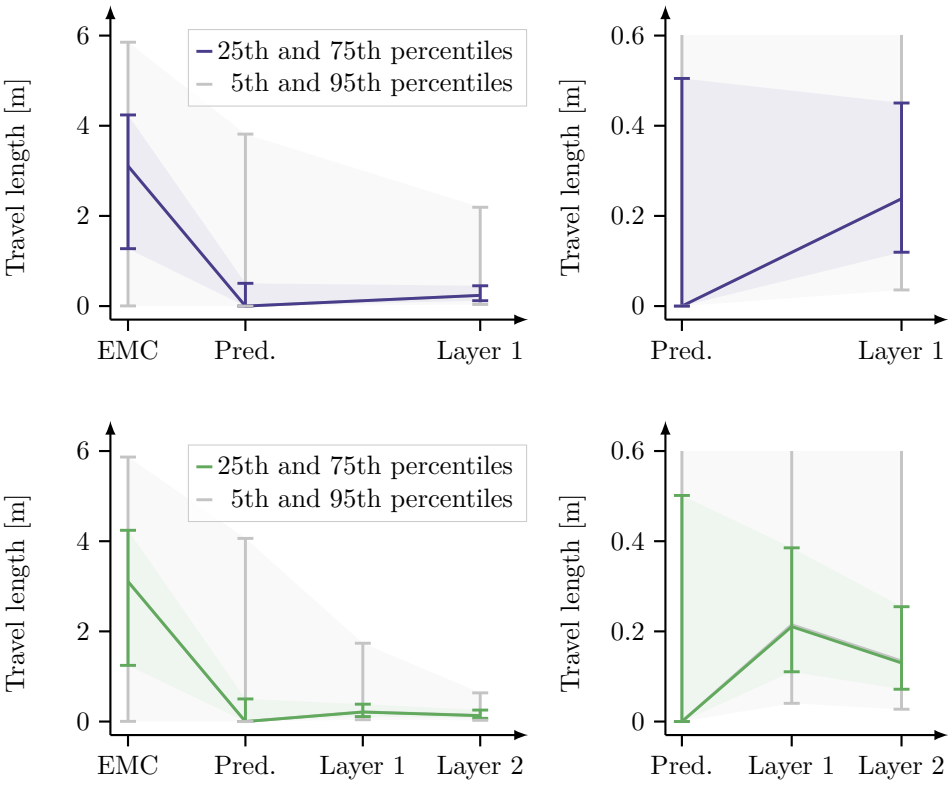


Figure 6.12: Median observed travel lengths per stage of the algorithm: ego-motion correction (EMC), prediction, and the decoder layers. Right-hand plots are zoomed-in on the latest stages for better visibility. *Top:* Model with a single decoder layer. *Bottom:* Model with additional track update layer.

object queries are located similarly close to an object when passed to Layer 2, so that its purpose can be a fine-grained location refinement. Because of this, a tracker performance improvement is expected from the track layer within TransMOT v2, which is evaluated in the following.

Table 6.5: Tracking results of TransMOT v2 compared to a PMBM filter (nuScenes metrics, nuScenes `val` dataset)

Method	TL	OR	AMOTA↑	AMOTP↓	IDS ↓	MT ↑
PMBM [GWGS18]			0.650	0.790	1473	**4124**
TransMOT v2			0.625	0.712	**435**	3596
	✓	✓	0.659	0.705	973	**4124**
	✓		**0.664**	**0.680**	872	4076

Continued from above:

AMOTA (car)↑	AMOTA (truck)↑	AMOTA (pedestrian)↑	AMOTA (bus)↑
0.777	0.641	**0.775**	0.736
0.779	0.604	0.671	0.744
0.819	**0.661**	0.712	0.750
0.819	0.650	0.721	**0.776**

6.6.6 TransMOT v2 Evaluation

TransMOT v2 is evaluated on the nuScenes `val` dataset dataset. It has been trained for 30 epochs on the nuScenes `train` dataset and has a deterministic (anchor-based) ego-motion compensation, a feature-informed query initialization, as well as an initial track update layer and a VoxelNet [ZT18] backbone with a voxel size of 7.5 cm × 7.5 cm × 20 cm. The approach is compared to a SOTA RFS-based Bayesian tracking method, the Poisson multi-Bernoulli mixture (PMBM) tracker [GWGS18]. This filter combines random finite set theory with a Poisson birth process and has a connection to the δ-GLMB filter, i.e. equivalence under some conditions. A publicly available version of PMBM is utilized, which has been tuned for the nuScenes dataset [Dao20]. Its parameters are listed in appendix A. For comparability, the evaluation is performed on detections obtained from the same model, i.e. the detection component of TransMOT v2. For this, TransMOT v2 is run once in detection-only mode to generate object detections for the full nuScenes `val` dataset split, which are input to the PMBM tracker.

In table 6.5, the nuScenes metrics are listed for both methods. Multiple variants of TransMOT v2 were evaluated, where TL denotes the track update layer and OR denotes the orientation anchors, which allow for a fully anchor-based ego-motion compensation. TransMOT v2 with TL and without OR achieves both the best AMOTA and the best AMOTP score in this comparison. A significant positive effect of the track update layer on the performance is apparent.

The PMBM tracker achieves a competitive performance, better than the base variant of TransMOT v2 (without TL). Interestingly, PMBM outperforms the full variant of TransMOT v2 on the *pedestrian* class, whereas it is outperformed on *car*, *truck* and *bus*. This may be an indicator of insufficient training data for the pedestrian class, which can affect a deep learning model. Remedies for this may be to increase the prevalence of pedestrians in the training data, or to increase the impact of existing ones on the model training, e.g. by increasing their loss weights.

Since the AMOTA and AMOTP metrics are not common in Bayesian tracking, a second evaluation is performed in the following. The GOSPA metric [RGS17] is a generalization of the well-known OSPA metric and provides an implementation in python code [Wil19]. The authors introduce a parameter α, so that GOSPA equals OSPA for $\alpha = 1$. With $\alpha = 2$ on the other hand, the metric can be formulated as follows:

$$\text{GOSPA} = \left[\min_{\pi \in \Pi} \left(\sum_{(i,j) \in \pi} d(x_i, y_j)^p + \frac{c^p}{2} \left(|\mathcal{X}| + |\mathcal{Y}| - 2|\pi| \right) \right) \right]^{1/p}, \tag{6.7}$$

where Π contains all possible assignments between GT and the estimates, and π is a specific assignment. A match requires a maximum distance of c, so that missed detections (unassigned GT locations) and false positives (unassigned estimates) can occur. $\mathcal{X}$ is the set of GT locations, $\mathcal{Y}$ the set of estimates and $|\cdot|$ denotes the cardinality of a set. $d(\cdot)$ denotes the euclidean distance between two locations and p is a parameter that influences how outliers are penalized. Different from OSPA, GOSPA with $\alpha = 2$ is not normalized by the cardinality of the larger set. The authors argue that this is beneficial, since false positive detections are otherwise not punished adequately: If none of the targets were found, OSPA equals c, no matter how many false positive detections occurred. GOSPA with $\alpha = 2$, on the other hand, penalizes any false or missed detection with $\frac{c^p}{2}$ [RGS17], while a positive match contributes $d(x_i, y_i)^p$. This metric, however, grows with the number of existing GT targets. Since the nuScenes `val` dataset is very large, GOSPA is normalized by a constant in the following, i.e. the number of GT objects on the validation split, $|\mathcal{X}|$. Besides this, p is set to 1 and a time index t is introduced:

$$\text{GOSPA-n}(\pi) = \frac{1}{|\mathcal{X}|} \sum_{(i,j,t) \in \pi} d(x_i^{(t)}, y_j^{(t)}) + \frac{c}{2} \left(|\mathcal{X}| + |\mathcal{Y}| - 2|\pi| \right). \tag{6.8}$$

By definition, GOSPA quantifies tracking performance momentarily, without taking temporal consistency into account. Similar to the MOTA metric, an identity-switch component is proposed to add a temporal aspect to the metric. Let

$$\mathcal{I}(\pi) = \{ \ [(i,j,t),(i,k,t+1)] \ \mid \ (i,j,t),(i,k,t+1) \in \pi \text{ and } j \neq k \} \tag{6.9}$$

contain pairs of assignments from π, which assign the same GT target x_i to different estimated tracks y_j, y_k in consecutive time steps t and $t+1$. Such identity switches are penalized, weighted by a parameter γ:

$$\text{GOSPA-nt}(\pi) = \frac{1}{|\mathcal{X}|} \sum_{(i,j,t)\in\pi} d(x_i^{(t)}, y_j^{(t)}) + \frac{c}{2} \left(|\mathcal{X}| + |\mathcal{Y}| + \gamma|\mathcal{I}| - 2|\pi| \right). \quad (6.10)$$

Table 6.6 contains both proposed metrics for TransMOT v2 and the PMBM filter evaluated on the nuScenes `val` dataset. For comparability, the metrics use the same assignment π as in table 6.5. $\gamma = 10$ is set and a cut-off distance of $c = 2$m is selected for consistency with table 6.5. False positives, false negatives and identity switches are defined as follows: FP $= |\mathcal{Y}| - |\pi|$, FN $= |\mathcal{X}| - |\pi|$ and IDS $= |\mathcal{I}|$

The results show that the PMBM filter achieves the best GOSPA-n score in this comparison, as well as the lowest number of false positives. While achieving a similar number of false negatives as the Bayesian filter, TransMOT v2 lays more focus on reducing the number of identity switches. According to GOSPA-nt, TransMOT v2 with TL outperforms the other variants. When inspecting the class-wise scores it becomes again apparent that a learned tracker excels at tracking the class it observed most during training (*cars*), while less prevalent classes remain challenging. Notably, which tracker achieves the highest performance depends on which metric is chosen for comparison among AMOTA, GOSPA-n and GOSPA-nt. While the analysis presented here is a first step to bring the fields of classical and learned tracking closer together, each tracker performs best on the respective metric that it was evaluated with during development and has been optimized for. As a direction of future research, it'd be beneficial to optimize both types of models with the same metrics in mind. Still, the differences in performance are currently not very large, so that one can conclude that both types of models are valid solutions for an autonomous driving stack. While a classical method has multiple advantages, such as explainability and adaptability, the presented AI tracker also comes with some advantage, e.g. it is compatible with an end-to-end trainable system, which is a concept that has become relevant in recent years [CWC+24].

Short-Term Prediction TransMOT v2 contains a short-term prediction, which shifts the anchor positions of the existing tracks according to the time between measured frames multiplied by the estimated velocity. Is such a prediction beneficial, or could the model learn the track displacement due to agent motion on its own? To answer this question, both a model variant with and without explicit short-term prediction are evaluated.

In Figure 6.13, the histograms of observed travel lengths between the track queries'

Table 6.6: Tracking results of TransMOT v2 compared to a PMBM filter in terms of GOSPA-n (nuScenes `val` dataset)

Method	TL	OR	GOSPA-n↓	GOSPA-nt↓	FP ↓	FN ↓	IDS ↓
PMBM [GWGS18]			**0.564**	0.711	**9151**	**22009**	1473
TransMOT v2			0.679	0.721	13618	27813	**435**
	✓	✓	0.621	0.718	12817	22088	973
	✓		0.609	**0.695**	12376	22702	872

Continued from above:

GOSPA-nt (car)↓	GOSPA-nt (truck)↓	GOSPA-nt (ped.)↓	GOSPA-nt (bus)↓
0.645	**0.723**	**0.634**	0.634
0.577	0.810	0.754	0.643
0.533	0.738	0.844	0.622
0.532	0.763	0.811	**0.618**

locations and their estimated bounding box are depicted in gray. While the first six histogram bins are equally spaced at a width of 0.25 meters, the last bin encompasses all travel lengths above 1.5 meters. Unsurprisingly, without explicit prediction (top), larger travel lengths occur than with prediction (bottom). Though, the question is whether this harms the performance. For this, the observed tracking scores from the classification head are gathered, based on the distance between the original query position (before prediction, if any) and the estimated box. One can observe that the model without prediction outputs *less confident* tracking scores with increasing distance, while the explicit prediction remedies this. Therefore, one can conclude that a short-term prediction achieves a desired effect for TransMOT.

6.7 Discussion

This chapter has presented TransMOT, a joined object detector and tracker, compatible with sensor data measured from a moving vehicle, as well as its development and improvement over time. Main contributions have been the ego-motion compensation module, a temporal training strategy and the inclusion of physical constraints, as well as a dedicated track update layer. Each proposed model component has been evaluated and shown to serve its purpose. Besides this, the well-known GOSPA metric has been modified to enable an evaluation of a deep learning model with it, as well as to draw a comparison to an RFS-based model. In this regard, the results show

that the proposed model can compete with classical trackers and outperform them in some regards, though further research potential lays in improving the training strategy for less prevalent classes than *car*. As mentioned above, TransMOT is fully differentiable and provides the option to propagate track queries to downstream components, i.e. feature vectors that contain information about their respective object. Therefore, as a direction of future research, TransMOT v2 can be integrated into a fully (or partly) end-to-end trainable system [CWC+24]. While TransMOT v2 has not been optimized to run online in the vehicle yet, its core concepts have become part of a perception model that has been deployed to a research test vehicle with a runtime of less than 100 milliseconds [Bru24].

An example for a tracked scene by TransMOT v2 is pictured in Figure 6.14. Three frames are pictured, which have been measured 2.5 seconds apart. The colors refer to the track id. In the lidar BEV, GT objects are displayed in dark green, whereas they are not shown in the images. Note that the images were not input to the model, but are shown for illustration purposes only.

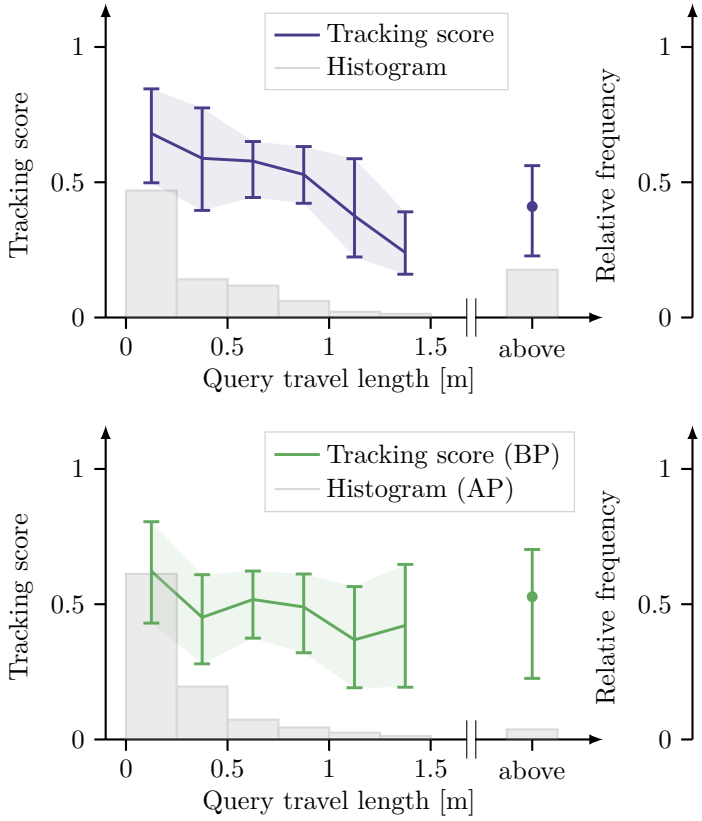


Figure 6.13: Median tracking scores w.r.t. the distance from track anchors (after EMC) to their final location are displayed, along with 25th and 75th percentiles, in color. The histogram in gray depicts the relative frequency of observed lengths. *Top:* Tracker without prediction. *Bottom:* Tracker with simple short-term prediction. BP refers to the distance from the anchor before prediction, AP after prediction.

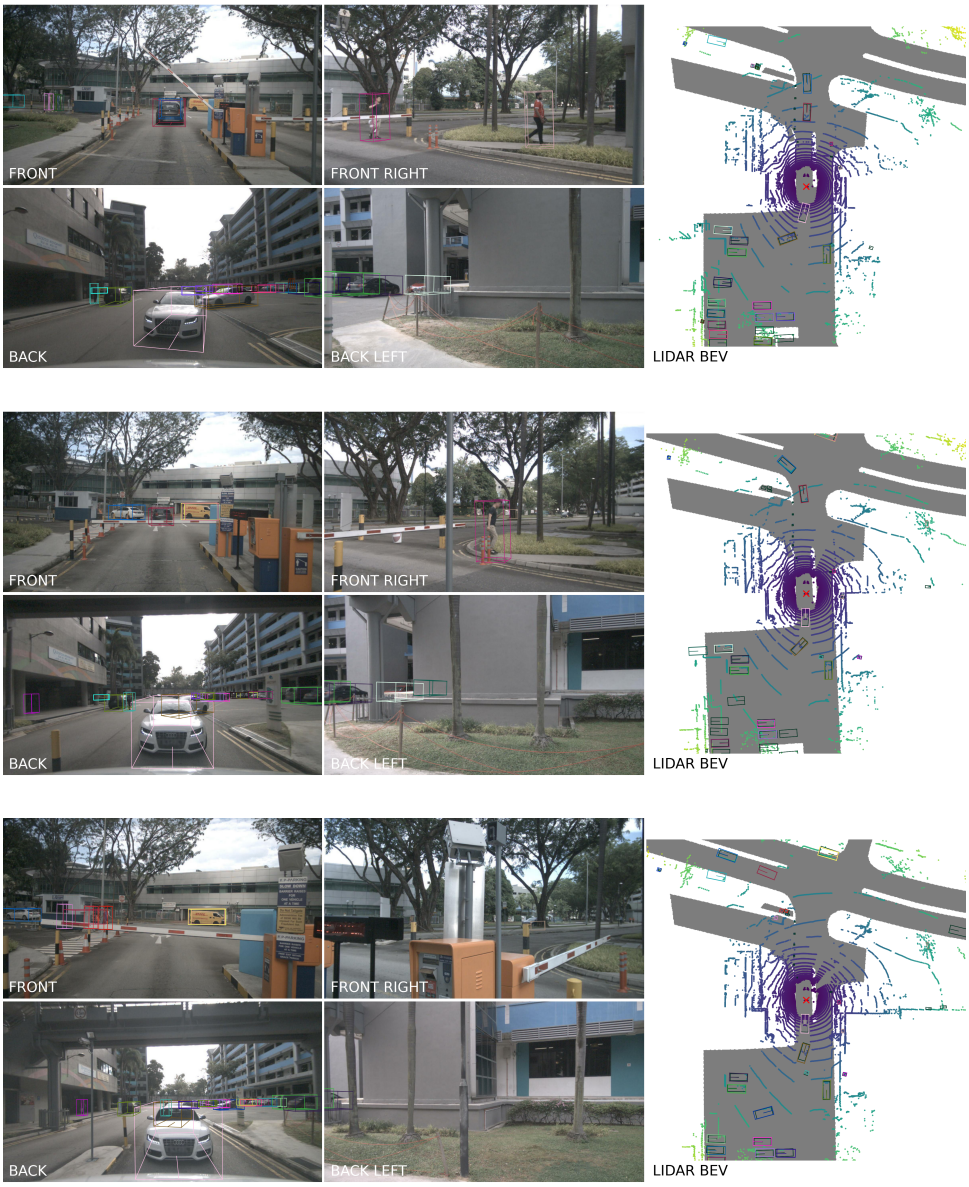


Figure 6.14: Every fifth frame from an exemplary tracked scene (2.5 seconds apart) on the © 2020 nuScenes [CBL+20] dataset. Color denotes track ids, and black boxes in BEV are GT. The cameras are only displayed for visualization and not used for tracking. Best viewed zoomed-in and in color.

Chapter 7

Uncertainty Estimation from Attention Weights

7.1 Introduction

A fundamental challenge with deep learning methods is to quantify the uncertainty of predictions. Object detection and tracking uncertainty can stem from two different sources: One can distinguish between aleatoric and epistemic uncertainty. Aleatoric uncertainty is also known as stochastic uncertainty, and describes stochastic variations in the input data, such as sensor measurement noise. This kind of uncertainty can not be reduced by training a model with additional data. Epistemic uncertainty, on the other hand, is caused by shortcomings of the model itself. For example, a model may not be able to detect a type of vehicle that it did not encounter during the training [FRD18].

While object detectors commonly output *confidence scores* for each detected object, they can not function as a reliable uncertainty measure: Their values do not behave like a probability distribution, but tend to be overly confident [GPSW17]. Some authors therefore propose to replace the output layers of networks with a probabilistic counterpart and to estimate distribution parameters rather than point estimates [GR18]. Other methods involve more computational overhead by running inference multiple times with Monte Carlo dropout [Gal16] or with ensemble models [LPB17].

The aim of this chapter is to gain insights into uncertainty without changing the model architecture at all, which has been pre-published in [RFGD22c]: A unique characteristic of attention based methods, such as transformers, is that the attention weights can be visualized and analyzed. In the context of the proposed joint detector and tracker, one can investigate *how* a certain object of interest was detected or tracked, i.e. which part of the input feature map was accessed to obtain the result.

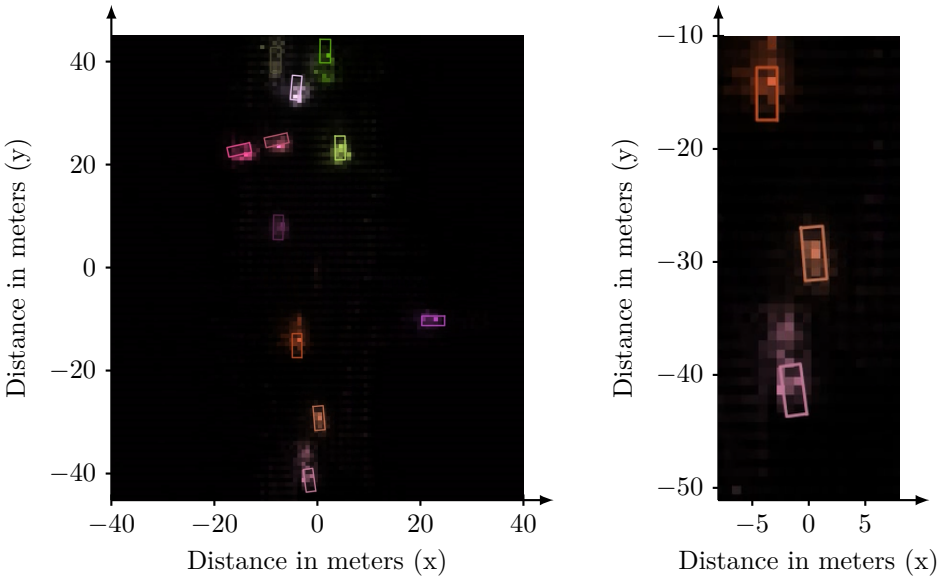


Figure 7.1: Visualization of detected boxes with their respective attention weights in the same color in bird's-eye view in an urban scene. The ego-vehicle is positioned at the coordinate center. Ground truth bounding boxes are not pictured. *Left:* Full bird's-eye view. *Right:* Zoomed-in excerpt.

For each individual object or track query, its respective cross-attention weights with the feature map are accessible (in every decoder layer). An example for this is pictured in Figure 7.1: Multiple objects were detected in this frame, each pictured in a different color. The feature map grid cells are colored according to their attention score with the query that detected the object, where brighter colors denote higher scores. Note that only the largest 100 attention scores per query are pictured. The scores were obtained in the last layer of the transformer decoder. It can be observed that the attention tends to be localized close to the respective object, but it is more focused for some than others, which is visible in the zoomed-in bird's-eye view excerpt on the right. In the following, two questions shall be answered: How can the attention behavior of a detected or tracked object be quantified? Does it give insights into how well the object was detected or tracked, i.e., could it be used to indicate uncertainty?

7.2 Method Overview

The attention score distribution is analyzed in combination with the detection and tracking model TransMOT, which was first introduced in [RFGD22a] and has been described in detail in chapter 6. This model is illustrated in Figure 7.2, along with the subject of interest: The attention matrices in the decoder. One attention matrix per query is available (per layer), illustrated as 3×3 matrices of colored squares, where brighter color intensity denotes a higher attention score. Note that, for illustration purposes, the pictured matrices are much smaller than the actual ones, which cover the entire BEV space and have, e.g. $\sqrt{N} \times \sqrt{N} = 128 \times 128$ cells.

As introduced in chapter 3 with Figure 3.2 B and C, the transformer decoder consists of L transformer layers, each with a self-attention stage among the queries, as well as a cross-attention between the queries and the input feature map of size $\sqrt{N} \times \sqrt{N}$. For this analysis, the focus is laid on cross-attention weights only. Each cross-attention is computed with h attention heads. Unless specified otherwise, the attention weights from the h heads are averaged and the weights are obtained from the last decoder layer. Therefore, the full cross-attention matrix between the M queries and N feature map grid cells is of size $N \times M$. Per query, the N attention weights can be reshaped to $\sqrt{N} \times \sqrt{N}$, so that each cell corresponds to a 2D location in BEV. This is the attention matrix of interest, available for each query, as mentioned above.

The analysis is performed once for the model in detection-only mode, as pictured in Figure 7.2 a, i.e., only with object queries and no temporal propagation. Second, it is performed in detection-and-tracking mode, where each sequence of the nuScenes `val` dataset is processed sequentially, the track queries are propagated in time and ego-motion corrected (Figure 7.2 a and b).

7.3 Definition of Attention Spread

A matrix of cross-attention weights of size $\sqrt{N} \times \sqrt{N}$ is obtained for each query in each forward pass of the model. Since its entries w_{pq}, $1 \leq p, q \leq \sqrt{N}$ each correspond to a cell of the BEV grid, it can be interpreted as a spatial distribution. Note that the attention weights are learned and do not oblige any parameterized distribution rules by design. Besides being scaled to sum up to one, their distribution could take any form. From observation, however, as shown in the introduction of this chapter, it was found that attention weights tend to be centered around the center position of detected objects, making it meaningful to compute a location

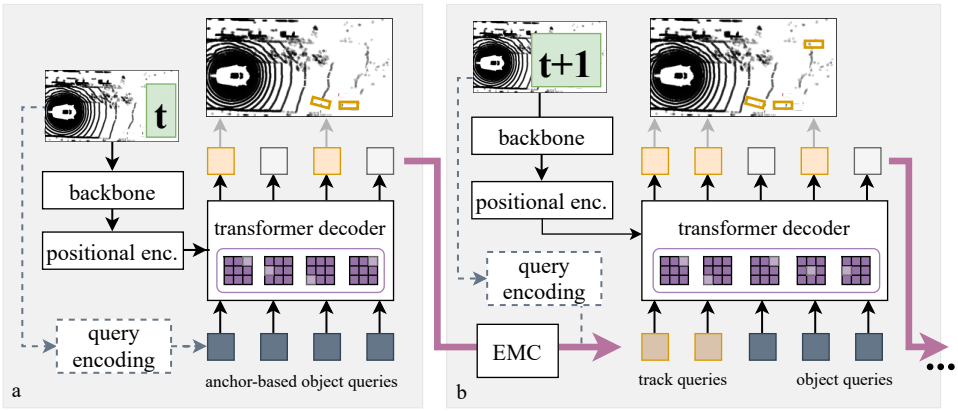


Figure 7.2: Illustration of TransMOT with cross-attention in the transformer decoder. Each query can attend to the full bird's-eye view grid individually. (cf. [RFGD22c])

and dispersion measure for them. Especially the dispersion is of interest: It is to be investigated if a larger dispersion, i.e. a less focused attention, corresponds to a more uncertain detection. Since an optimal estimator can not be derived from the limited distribution information available, well-known estimators for variance and mean derived for Gaussian distributions are utilized.

Only the largest K weights are taken into account for this, denoted by $\{w_{pq} \mid p, q \in \mathcal{S}^K\}$. The distance to the ego vehicle of a grid cell with indices p, q is denoted as x_q, y_p. The mean of top-K attention then is

$$\boldsymbol{\mu}_K = \frac{1}{W} \sum_{p,q \in \mathcal{S}^K} w_{pq} \begin{pmatrix} x_q \\ y_p \end{pmatrix}, \tag{7.1}$$

where $W = \sum_{p,q \in \mathcal{S}^K} w_{pq}$ is a normalization term. Aiming to quantify dispersion of the spatial distribution, an attention covariance is proposed to be obtained as follows, as pre-published in [RFGD22c]:

$$\boldsymbol{Q}_K = \frac{1}{W} \sum_{p,q \in \mathcal{S}^K} w_{pq} \left[\begin{pmatrix} x_q \\ y_p \end{pmatrix} - \boldsymbol{\mu}_K \right] \left[\begin{pmatrix} x_q \\ y_p \end{pmatrix} - \boldsymbol{\mu}_K \right]^\top. \tag{7.2}$$

Following the concept of generalized variance [Wil32], this covariance matrix is reduced to a single value, namely the proposed attention spread measure (ASM)

$$\text{ASM} = \det \boldsymbol{Q}_K. \tag{7.3}$$

Generalized variance has been described as a measure for dispersion of multidimensional scatter data and, for normally distributed data, it is proportional to the ellipsoid area occupied by the data above a certain probability threshold [And03]. This motivates its use to quantify attention spread, even though the distribution of the given attention weight data is not well defined. Note that this reduction removes direction information from the attention covariance, which might be of interest and is lost in the attention spread measure. It may be of interest for future research to analyze all components of the attention covariance individually, whereas the following evaluation focuses only on the attention spread.

How does attention spread correlate with detection and tracking uncertainty? This question is non-trivial to answer, since the true uncertainty is usually unknown. It has, however, been shown that the detection (and tracking) accuracy in terms of intersection-over-union (IoU) between estimates and GT correlates with epistemic uncertainty [FRD18]. Therefore, IoU is used as a proxy for this kind of uncertainty and its behavior is compared to that of attention spread. Aleatoric uncertainty on the other hand is known to be caused by low visibility, among other causes, which often occurs with larger distance to an object [WRLS19]. This means that, in order to behave similarly to aleatoric uncertainty, attention spread should become larger with greater distance of an object to the ego vehicle. Concerning tracking, Bayesian approaches are able to maintain an uncertainty per object at each time step [CPLB20] whereas many deep tracking methods are not. The more consistent measurements of a tracked object have been observed, the larger its confidence will be. If attention spread can serve as an estimator for uncertainty, it could enhance TransMOT [RFGD22a] with this capability.

7.4 Evaluation

The evaluation has been carried out on the nuScenes `val` dataset with a pre-trained version of TransMOT [RFGD22a] with $M = 100$ object queries, a bird's-eye view grid side length of $\sqrt{N} = 128$, $L = 6$ decoder layers, $h = 8$ attention heads and considering the top $K = 100$ attention weights per query. For the analysis, only the class *car* is considered. Since the number of queries is larger than the expected number of observable cars, many queries are idle for a given frame. To exclude them, only detections above a confidence threshold of 0.8 are taken into account.

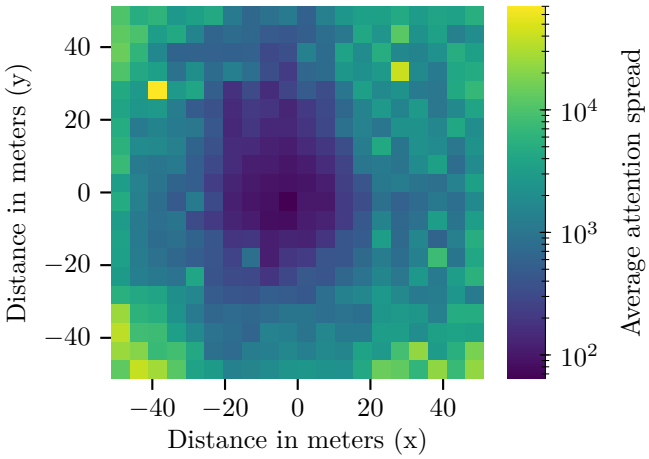


Figure 7.3: Average attention spread obtained from spatial grouping. For this, the bird's-eye view grid was divided into 20×20 cells with a width and length of 5 meters. (cf. [RFGD22c])

7.4.1 Spatial Distribution

The spatial distribution of attention spread is investigated. For the visualization in Figure 7.3, the BEV grid is divided into bins of 20×20 cells of size 5×5 meters. Each detection on the nuScenes `val` dataset is assigned to one of the cells, depending on its location, and the observed attention spread values per cell are aggregated. The average ASM per cell is pictured in Figure 7.3. Note that the coordinate system in BEV is defined as follows: The ego vehicle is placed at the center, and facing up, i.e. in positive y-direction. It can be observed that the average attention spread increases with distance to the ego vehicle. Besides this, attention spread tends to be smaller in an oval shape around the ego vehicle that is pointed towards the direction of travel. A larger distance to the ego vehicle typically coincides with a lower number of observed lidar points per object, which is a cause for aleatoric uncertainty. In this regard, attention spread behaves similarly to this kind of uncertainty, as its mean increases with distance.

7.4.2 Relation to Detection Accuracy

Figure 7.4 displays the attention spread in terms of intersection-over-union (IoU). For each detected (or tracked) bounding box, its closest GT match is determined

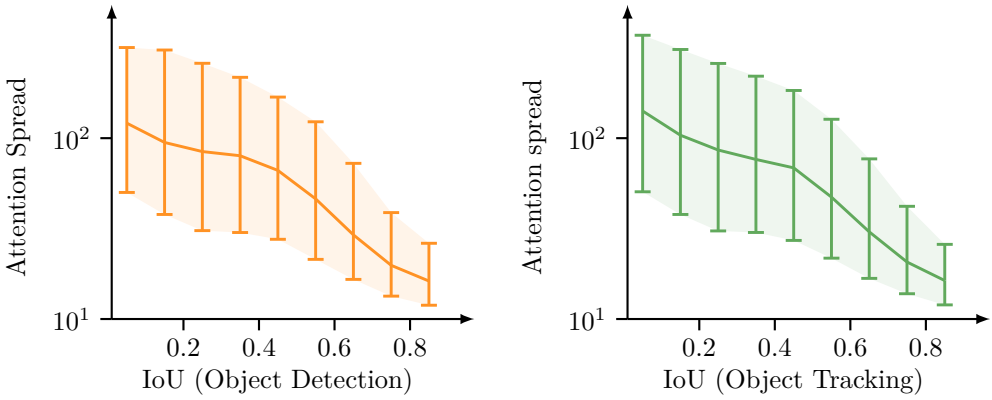


Figure 7.4: Median attention spread in terms of IoU for object detection (left) and joint detection and tracking (right). Error bars denote the 25th and 75th percentiles, respectively. (cf. [RFGD22c])

and their three-dimensional intersection volume is divided by their union, to obtain the IoU. Matches with no overlap, i.e. IoU = 0, are removed. The observed IoU values have been grouped into ten bins of width 0.1. For each bin, the median of corresponding attention spread observations is plotted, as well as their 25th and 75th percentiles. As mentioned above, only the cross-attention weights from the last decoder layer are considered. Both for object detection (left) and joint tracking and detection (right), the plots look very similar. It is observable that the median attention spread decreases with growing IoU. This shows that high attention spread can be an indicator for low IoU and vice-versa, which, on the other hand correlates with epistemic uncertainty [FRD18]. Note, however, that the 25th and 75th percentiles span quite large ranges, so that it is difficult to provide an expected IoU based on a given attention spread value for a single object.

Refinement Through the Decoder Layers TransMOT consists of six decoder layers that the queries are passed through. Before the first one, their locations are randomly sampled from the input point cloud. Through the layers, they iteratively refine their location and travel towards an observed object. The median attention spread per decoder layer, observed from all queries on the nuScenes `val` dataset is depicted in Figure 7.5. The largest median ASM occurs in the first layer and the smallest in the last one. This implies that the search space per query is larger at the beginning, when it aims to find an object, whereas the attention becomes more focused towards the end, when refinement is performed. In the layers in between, however, median ASM does not decrease strictly, but rather fluctuates. This could be caused

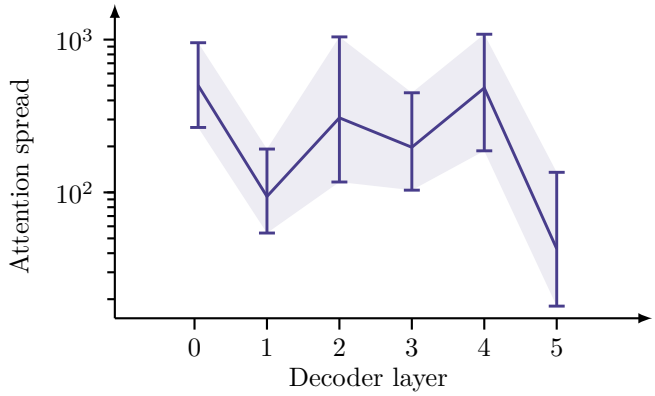


Figure 7.5: Median attention spread observed in the six decoder layers. The error bars denote 25th and 75th percentiles, respectively. (cf. [RFGD22c])

by an alternation between object searching and object refinement. Alternatively, it may imply that not all decoder layers are equally important for object detection, i.e. it may be possible to remove one or two of the middle layers.

7.4.3 Behavior during Tracking

Attention spread is analyzed over the course of a track's lifetime. For this evaluation, only tracks that last longer than 7 seconds are considered. Figure 7.6 (left) displays their median ASM in the initialization phase, during the first 3.5 seconds. The same is pictured on the right side for the final 3.5 seconds of the tracks, which is their finalization phase. Errors bars denote the 25th and 75th percentiles. Similar to the behavior of uncertainty estimated with Bayesian trackers, the median ASM decreases after track initialization. This can be caused by two different aspects: Firstly, tracks often start at a distance, and become easier to detect over time if they move towards the ego vehicle. Second, the model may gain confidence through observing the same object over multiple frames. In the finalization phase, tracks display an inverse behavior: The ASM rises towards the end. This could, again, be explained with tracks commonly concluding in the distance, where object localization is more difficult.

Figure 7.7 shows excerpts of the BEV grid at different distances to the ego vehicle. One object is visible in all three of them: The bright green colored car appears at 50

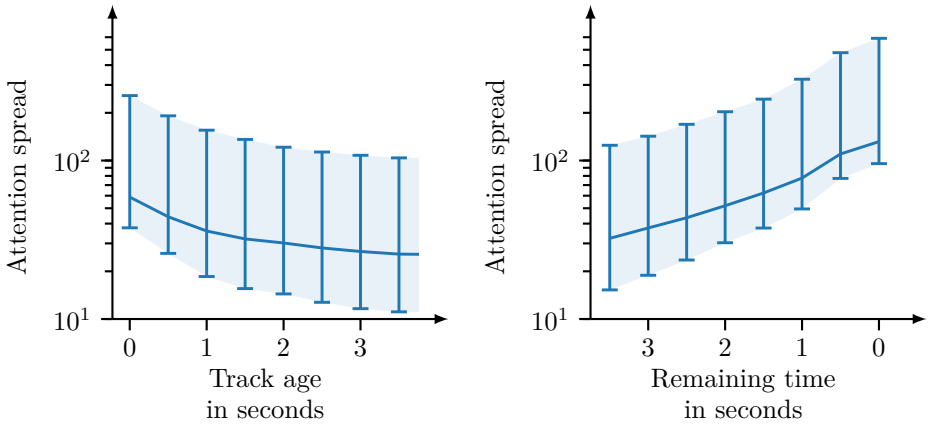


Figure 7.6: Median attention spread for the track initialization (*left*) and finalization (*right*) phase, obtained from all tracks with a duration longer than 7 seconds. The error bars denote 25th and 75th percentiles, respectively. (cf. [RFGD22c])

meters distance and is observed again at 30 and 10 meters away. From the attention covariance matrices as defined in Eq. (7.2), an ellipse is computed. If the attention weights were to be normally distributed, these ellipses would be isometric lines at 1σ. The lime green track of interest is first detected when its front half enters the field of view. At this distance of 50 meters, a larger ellipse is observable, which then gets smaller and rounder as the car moves closer to the ego vehicle.

7.5 Conclusion

This chapter introduced a new measure that quantifies the attention weight behavior in the transformer decoder, namely attention spread. Attention spread behavior has been analyzed in terms of detection accuracy, distance to the ego vehicle and tracking over time. The results show that the attention spread behaves similarly to both aleatoric and epistemic uncertainty. This means that it tends to increase with distance, decrease with higher detection accuracy and first decrease then increase again over the lifetime of a track. Therefore, this measure can be useful as an indicator for both kinds of uncertainty, it does, however, not differentiate between them. While a trend is clearly observable, the 25th and 75th percentile span large ranges in the experiments, so that ASM can, in its current state, not be used as the only uncertainty measure if downstream components in a productive system



50 meters, $t = 0$s | 30 meters, $t = 1.5$s | 10 meters, $t = 2.5$s

Figure 7.7: Excerpts from the bird's-eye view grid at different distances to the ego vehicle observed at $t = 0, 1.5$ and 2.5 seconds. The same lime green track is visible in all images. The attention covariance matrices as defined in Eq. (7.2) are depicted as 1σ-ellipses. (cf. [RFGD22c])

rely on it. Nonetheless, the results show that it may be useful as an indicator in combination with additional, existing uncertainty quantifiers. To find the best way to combine them is an interesting direction for future research, as well as possible improvements to the ASM to reduce its range of fluctuation.

Chapter 8

Summary and Outlook

This work has investigated the potential of deep multi-object tracking in the context of autonomous driving. A transformer architecture was leveraged to develop TransMOT, which propagates track queries over time, each of which represents a track state in feature space. The track-to-data association problem is solved by the transformer decoder with a learned soft association via attention weights. The structure of the model consists of an update and a prediction step, inspired by Bayesian methods. To bring this model to life, contributions both in the area of object detection as well as object tracking were necessary. They are summarized in the following.

Regarding object detection, a major challenge were the specific requirements of the driving use-case: A very large field of view of at least 100 times 100 meters, a large measured lidar point cloud, as well as multiple different relevant object classes. To accommodate for the large field of view, the presented object detector TransLPC features anchor-based queries, as well as query refinement strategy, which allows each query to travel towards its object rather than estimating at a distance. This technique also enables the use of a relatively small number of queries, since they cover a larger area, which makes the model resource-efficient. To enable the processing of large point clouds, the transformer *encoder* was removed from the model, so that only the backbone encodes the measurements and their surrounding context. In order to detect objects of many different relevant classes, a novel grouped regression head was presented. Finally, inspired from recent progress in related work, a feature-informed query initialization was included into the model, and modified to allow for a balanced initialization with multiple available sensor modalities. All model variants were evaluated on the challenging nuScenes dataset, both in comparison to other object detection methods, as well as with a focus on certain design choices and their impact on the performance. The final detector was able to outperform the state of the art in transformer-based object detection.

The contributions to object detection built the basis for the joined detector and

tracker, TransMOT. Again, it was faced with driving-specific challenges: An ego-vehicle can move significantly between measured frames, which needs to be compensated for. Besides this, all surrounding road users may also move during that time. To compensate, an ego-motion correction in feature space was developed, which transforms each track's feature vector according to the measured ego pose change. In addition, a training strategy that incorporates sequences of frames was presented, to teach the model to track over time. Besides this, the tracker was extended from car-only to multi-class tracking.

While deep learning methods have many advantages and are very flexible, it has been shown that they can benefit from prior knowledge or physical constraints, especially if limited training data is available. In this regard, TransMOT was enhanced with a target motion prediction based on the estimated agent velocities and the time difference between measurements. In addition to this, a deterministic ego-motion compensation was presented, which performs the correction on the anchors directly, both in terms of the position and the orientation. For this, orientation queries were developed, which estimate both the orientation and location relative to a reference anchor.

The tracking performance of the proposed model was evaluated on the nuScenes tracking challenge benchmark. The results of the initial deep tracker were compared with those of the Kalman filter-based tracker StanfordIPRL-TRI [CPLB20], both in terms of general performance, as well as with regard to the ability to cope with skipped measurements. This tracker was ranked at the top of the nuScenes benchmark for a long time and was considered a SOTA filter-based approach among those that were available for the nuScenes dataset at the time. To determine the tracking performance separately from the object detection task, the Kalman filter was supplied with transformer detections that were produced by the deep tracker, when run in detection-only mode. In a similar fashion, the later tracker iteration TransMOT v2 was compared to a PMBM filter, which has become available with publicly available code and tuned parameters for nuScenes recently [Dao20]. This tracker utilizes random finite set theory. Since different metrics are common in the research fields they originate from, both the nuScenes metrics as well as a variant of GOSPA were computed. The results showed that the learned tracker is able to outperform classical methods in some aspects. For example, it can cope very well with missing measured data or long inter-frame distances and does not require a confidence score threshold. Still, there is potential for an improved performance, especially for some classes that are not very prevalent in the training data. In addition to the performance evaluation, experiments were presented that allow for an understanding of TransMOT's behavior, e.g. regarding the ego-motion compensation module, which is able to compensate for large ego pose changes.

Since one drawback of learned methods is the difficulty to produce reliable uncertainty

measures, the final chapter focused on this topic. A measure to describe the observed attention weight distribution was introduced, denoted as *attention spread*. The experiments have shown that a more focused attention, i.e. a smaller attention spread, is associated with more accurate detections in terms of IoU. Additionally, similar to the uncertainty in Bayesian methods, an increased attention spread can be observed in the initialization phase of a track, which decreases once the track has been observed for some time. Attention spread can be an indicator for the total uncertainty, but it does not distinguish between aleatoric and epistemic uncertainty. Note that, while an indication of uncertainty has been observed, attention spread is not accurate enough as an uncertainty measure (yet) to be used in practice as the only source of truth. Nonetheless, its range of fluctuation may be reduced in a future iteration, or it may be used in combination with additional uncertainty estimators.

To conclude, this work has presented a solution for deep tracking that is applicable for autonomous driving. While it does not outperform classical methods by a very large margin, it has proven to be a suitable approach to the given task and has many beneficial properties. Besides this, the transformer architecture is very flexible and can be developed further in multiple different directions. Some of them are described in the following, as an outlook for further research.

One research area is *sensor fusion*: The modality-balanced query initialization in chapter 5 has provided the first step towards a multi-modal detector and tracker, i.e. one that can utilize input data from any combination of available sensor measurements. Since the track state is propagated over time, it can be updated at any time through the transformer decoder with newly available information, even if one sensor may not be available or reliable. The sensor fusion can be performed by learned attention weights that determine the relevance of each measurement for the given task. Notably, this can be performed in an *asynchronous* manner, whereas current machine learning models typically rely on a sensor synchronization, which provides all measurements at the same point in time. This synchronization involves a waiting period until all modalities are available, which may be prevented in a multi-model transformer system.

Another research direction concerns the downstream tasks in an autonomous driving system. Since the track queries contain information about their respective objects in high-dimensional feature space, they can be passed to downstream modules to widen the interface from low-dimensional bounding boxes to additional information. Besides this, the downstream prediction or planning components could be trained with the tracker in an end-to-end manner. Still, the intermediate interface of the tracker would be accessible (i.e., estimated bounding boxes and track ids), both to supervise the training, as well as to observe the end-to-end model's behavior.

In this regard, Yinzhe Shen performed initial research in his master thesis [She23],

which built upon TransMOT and was supervised jointly by the author of this work and Faris Janjoš: Following the structure of [GHZ+23], TransMOT was connected to a trainable prediction model [JDZ21]. For this, each observed track was input to a *past states encoder* as a sequence of high-dimensional track queries and then passed to the predictor to compute possible trajectories, forming a model called Tradictor.

Besides expanding to further tasks, some aspects regarding multi-object tracking could also be investigated further. One of them is the comparability between Bayesian trackers and deep learning methods. While continuously improved methods are presented in both areas, very few comparisons exist, because, typically, different data sets and different performance metrics are used, which makes a fair comparison difficult. While this work presents a comparison, both in terms of the nuScenes metrics as well as a modified GOSPA metric, the lack of an established metric and data source that suits both approaches became apparent. Finding such a common ground would enable both fields to benefit from each other's progress.

TransMOT has provided a versatile platform for joined object detection and tracking that can be further extended in multiple different directions. The transformer architecture is able to adapt to different use cases and sensor modalities. It will be exciting to see how this technology is developed further and how it can benefit from additional training data.

Appendix A

Additional Parameters

This chapter contains details about the parameter settings for both the StanfordIPRL -TRI tracker [CPLB20] and the PMBM tracker [GWGS18] in the implementation from [Dao20], which have been used as reference methods in the evaluation of this work.

In listing A.1, the covariance parameters of StanfordIPRL -TRI, obtained from the publicly available source code[2], are listed. $\boldsymbol{P}$ denotes the initial state covariance, which is called $\boldsymbol{\Sigma}_0$ in their paper [CPLB20], and $\boldsymbol{Q}$ and $\boldsymbol{R}$ are the process and observation noise covariance matrices, respectively. To determine the process noise matrices $\boldsymbol{Q}$ for each object class, the authors have computed the empirical variance of the respective parameters' change between two measurements in the nuScenes `train` dataset. Since the observation noise matrix, on the other hand, models errors in the object detector, its values are obtained from the empirical variance of the differences between GT and the detector's estimates on the nuScenes `train` dataset. The initial state covariance is set equal to $\boldsymbol{R}$. For additional details, the reader is referred to [CPLB20].

Listing A.1: Covariance parameters of StanfordIPRL-TRI [CPLB20], cited from their code repository[2]

```
# Kalman Filter state: [x, y, z, rot_z, l, w, h, x_dot,
    ↪ y_dot, z_dot, rot_z_dot], where the last four denote
    ↪ linear and angular velocities

P = {
        'bicycle':      [0.05390982, 0.05039431, 0.01863044,
            ↪ 1.29464435, 0.02713823, 0.01169572,
            ↪ 0.01295084, 0.04560422, 0.04097244,
            ↪ 0.01725477, 1.21635902],
```

[2] https://github.com/eddyhkchiu/mahalanobis_3d_multi_object_tracking/blob/master/covariance.py

```
        'bus':        [0.17546469, 0.13818929, 0.05947248,
           ↪ 0.1979503 , 0.78867322, 0.05507407,
           ↪ 0.06684149, 0.13263319, 0.11508148,
           ↪ 0.05033665, 0.22529652],
        'car':        [0.08900372, 0.09412005, 0.03265469,
           ↪ 1.00535696, 0.10912802, 0.02359175,
           ↪ 0.02455134, 0.08120681, 0.08224643,
           ↪ 0.02266425, 0.99492726],
        'motorcycle': [0.04052819, 0.0398904 , 0.01511711,
           ↪ 1.06442726, 0.03291016, 0.00957574, 0.0111605
           ↪ , 0.0437039 , 0.04327734, 0.01465631,
           ↪ 1.30414345],
        'pedestrian': [0.03855275, 0.0377111 , 0.02482115,
           ↪ 2.0751833 , 0.02286483, 0.0136347 , 0.0203149
           ↪ , 0.04237008, 0.04092393, 0.01482923,
           ↪ 2.0059979 ],
        'trailer':    [0.23228021, 0.22229261, 0.07006275,
           ↪ 1.05163481, 1.37451601, 0.06354783,
           ↪ 0.10500918, 0.2138643 , 0.19625241,
           ↪ 0.05231335, 0.97082174],
        'truck':      [0.14862173, 0.1444596 , 0.05417157,
           ↪ 0.73122169, 0.69387238, 0.05484365,
           ↪ 0.07748085, 0.10683797, 0.10248689, 0.0378078
           ↪ , 0.76188901]
}

Q = {
        'bicycle':    [1.98881347e-02, 1.36552276e-02,
           ↪ 5.10175742e-03, 1.33430252e-01, 0, 0, 0,
           ↪ 1.98881347e-02, 1.36552276e-02, 5.10175742e
           ↪ -03, 1.33430252e-01],
        'bus':        [1.17729925e-01, 8.84659079e-02,
           ↪ 1.17616440e-02, 2.09050032e-01, 0, 0, 0,
           ↪ 1.17729925e-01, 8.84659079e-02, 1.17616440e
           ↪ -02, 2.09050032e-01],
        'car':        [1.58918523e-01, 1.24935318e-01,
           ↪ 5.35573165e-03, 9.22800791e-02, 0, 0, 0,
           ↪ 1.58918523e-01, 1.24935318e-01, 5.35573165e
           ↪ -03, 9.22800791e-02],
        'motorcycle': [3.23647590e-02, 3.86650974e-02,
           ↪ 5.47421635e-03, 2.34967407e-01, 0, 0, 0,
           ↪ 3.23647590e-02, 3.86650974e-02, 5.47421635e
           ↪ -03, 2.34967407e-01],
```

```
        'pedestrian': [3.34814566e-02, 2.47354921e-02,
            ↪ 5.94592529e-03, 4.24962535e-01, 0, 0, 0,
            ↪ 3.34814566e-02, 2.47354921e-02, 5.94592529e
            ↪ -03, 4.24962535e-01],
        'trailer':    [4.19985099e-02, 3.68661552e-02,
            ↪ 1.19415050e-02, 5.63166240e-02, 0, 0, 0,
            ↪ 4.19985099e-02, 3.68661552e-02, 1.19415050e
            ↪ -02, 5.63166240e-02],
        'truck':      [9.45275998e-02, 9.45620374e-02,
            ↪ 8.38061721e-03, 1.41680460e-01, 0, 0, 0,
            ↪ 9.45275998e-02, 9.45620374e-02, 8.38061721e
            ↪ -03, 1.41680460e-01]
}

R = {
        'bicycle':    [0.05390982, 0.05039431, 0.01863044,
            ↪ 1.29464435, 0.02713823, 0.01169572,
            ↪ 0.01295084],
        'bus':        [0.17546469, 0.13818929, 0.05947248,
            ↪ 0.1979503 , 0.78867322, 0.05507407,
            ↪ 0.06684149],
        'car':        [0.08900372, 0.09412005, 0.03265469,
            ↪ 1.00535696, 0.10912802, 0.02359175,
            ↪ 0.02455134],
        'motorcycle': [0.04052819, 0.0398904 , 0.01511711,
            ↪ 1.06442726, 0.03291016, 0.00957574, 0.0111605
            ↪ ],
        'pedestrian': [0.03855275, 0.0377111 , 0.02482115,
            ↪ 2.0751833 , 0.02286483, 0.0136347 , 0.0203149
            ↪ ],
        'trailer':    [0.23228021, 0.22229261, 0.07006275,
            ↪ 1.05163481, 1.37451601, 0.06354783,
            ↪ 0.10500918],
        'truck':      [0.14862173, 0.1444596 , 0.05417157,
            ↪ 0.73122169, 0.69387238, 0.05484365,
            ↪ 0.07748085]
}
self.P = {tracking_name: np.diag(P[tracking_name]) for
    ↪ tracking_name in P}
self.Q = {tracking_name: np.diag(Q[tracking_name]) for
    ↪ tracking_name in Q}
self.R = {tracking_name: np.diag(R[tracking_name]) for
    ↪ tracking_name in R}
```

In listing A.2, the parameters of the PMBM filter [Dao20; GWGS18] are listed, which are publicly available in the source code file[3]. Note that the authors have used the same values for the process and observation noise covariance matrices as StanfordIPRL -TRI, which are therefore not listed. The following parameters for the PMBM tracker are defined in listing A.2: A state and measurement dimension, a probability of survival and detection, a Poisson birth weight and gating size, as well as a pruning and merge threshold and a clutter intensity. Besides this, a desired number of global hypotheses is set, as well as an existence probability threshold for adding a hypothesis of an object that is undetected and a weight threshold for global hypothesis pruning. Further details about the implementation in code and the derivation of the filter equations can be found in [Dao20] and [GWGS18], respectively.

Listing A.2: Parameters of the PMBM filter, cited from the code[3]

```
state_dim: int = 6,
measurement_dim: int = 3,
prob_survival: float = 0.85,
prob_detection: float = 0.9,
poisson_birth_weight: float = np.log(1),
poisson_birth_gating_size: float = 11.0,
poisson_prune_threshold: float = -5,
poisson_merge_threshold: float = 5.0,
poisson_clutter_intensity: float = 1e-4,
pmbm_desired_num_global_hypotheses: int = 5,
pmbm_prune_single_hypothesis_existence: float = 1e-3,
pmbm_prune_global_hypothesis_log_weight: float = -5.0
```

[3] https://github.com/quan-dao/pmbm-filter/blob/master/filter_config.py

Appendix B

List of Publications

Parts of this thesis have been published in the author's own research papers:

Ruppel, Felicia; Faion, Florian; Gläser, Claudius; and Dietmayer, Klaus: *Transformers for multi-object tracking on point clouds.* In: *IEEE Intelligent Vehicles Symposium (IV)*, pages 852–859, 2022.

Ruppel, Felicia; Faion, Florian; Gläser, Claudius; and Dietmayer, Klaus: *Transformers for object detection in large point clouds.* In: *IEEE International Conference on Intelligent Transportation Systems (ITSC)*, 2022.

Ruppel, Felicia; Faion, Florian; Gläser, Claudius; and Dietmayer, Klaus: *Can transformer attention spread give insights into uncertainty of detected and tracked objects?* In: *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) PNARUDE Workshop*, 2022.

Ruppel, Felicia; Faion, Florian; Gläser, Claudius; and Dietmayer, Klaus: *Group regression for query based object detection and tracking.* In: *IEEE International Conference on Intelligent Transportation Systems (ITSC)*, 2023.

The author shares the following publication with Mathijs van Geerenstein as *first author with equal contribution*:

van Geerenstein*, Mathijs; Ruppel*, Felicia; Dietmayer, Klaus; and Gavrila, Dariu: *Multimodal object query initialization for 3D object detection.* In: *2024 IEEE international conference on robotics and automation (ICRA)*. IEEE, 2024. **equal contribution.*

Appendix C

Supervised Theses and Projects

van Geerenstein, Mathijs: *Generating Synthetic Data for 3D Object Detection with CARLA*. Internship Project, 2022.

Shen, Yinzhe: *End-to-end Perception and Prediction in Autonomous Driving*. Master Thesis, Stuttgart University, (Co-Supervision with Faris Janjoš), 2023.

Wagner, Nick: *Object Tracking with a Temporal Transformer Using Low-Dimensional Input Data*. Master Thesis, Stuttgart University, (Co-Supervision with Marcel de Jong), 2023.

van Geerenstein, Mathijs: *3D Object Detection: A Review of Transformer-Based Deep Learning Models*. Literature Research Report, TU Delft, 2023.

van Geerenstein, Mathijs: *Input-Dependent Query Initialization with a Modality-Balanced Transformer Decoder for LiDAR-Camera 3D Object Detection*. Master Thesis, TU Delft, 2023.

Acronyms

2D	two-dimensional	ix
3D	three-dimensional	1
AMOTA	average multi-object tracking accuracy	18
AMOTP	average multi-object tracking precision	18
AP	average precision	12
ASM	attention spread measure	112
BEV	bird's-eye view	11
CNN	convolutional neural network	9
DETR	detection transformer	10
DL	deep learning	ix
FFN	feed-forward network	26
FoV	field of view	2
FPS	farthest point sampling	52
GLMB	generalized labeled multi-Bernoulli	15
GOSPA	generalized optimal sub-pattern assignment	19
GPS	global positioning system	8
GT	ground truth	12
HOTA	higher order tracking accuracy	18
IDS	identity switch	17
IMU	inertial measurement unit	8
IoU	intersection-over-union	22
lidar	light detection and ranging	6
LMB	labeled multi-Bernoulli	15

LSTM	long short-term memory	28
mAP	mean average precision	13
ML	machine learning	23
MOT	multi-object tracking	14
MOTA	multi-object tracking accuracy	17
MOTP	multi-object tracking precision	17
NDS	nuScenes detection score	13
NLP	natural language processing	28
NMS	non-maximum suppression	10
OSPA	optimal sub-pattern assignment	18
PMBM	Poisson multi-Bernoulli mixture	3
PPP	Poisson point process	16
radar	radio detection and ranging	6
ReLU	rectified linear unit	26
RFS	random finite set	15
RMSE	root mean squared error	94
RNN	recurrent neural network	27
SOTA	state-of-the-art	3
TransLPC	transformer-based detector for large point clouds	47
TransMOT	transformer-based multi-object detector and tracker	xv

List of Symbols

General Notation

x	Scalar
$\boldsymbol{x}$	Column vector
$\boldsymbol{x}^\top$	Row vector
$\|\|\boldsymbol{x}\|\|_p$	p-norm (vector norm)
$\boldsymbol{x}', \tilde{\boldsymbol{x}}$	Transformed vector
$\boldsymbol{X}$	Matrix
x_{ij}	(i,j)th matrix entry
$\det \boldsymbol{X}$	Determinant of a matrix
$\mathcal{X}$	Set
$\oplus$	Concatenation operator

Physical Quantities

x, y, z	3D location
$\Delta x, \Delta y, \Delta z$	3D location offset
γ	Orientation (angle)
w, l, h	Width, length and height
v_x, v_y	Velocity in x- and y-direction
`cls`	Class identifier
t	Timestamp
$\boldsymbol{d}_i$	ith measured lidar point
$\boldsymbol{p}$	Pose change
t_1, t_2, t_3	Translation
q_0, q_1, q_3, q_4	Rotation in quaternion format

Statistical Variables

$\boldsymbol{Q}$	Sample covariance matrix
$\boldsymbol{\mu}$	Arithmetic mean
p	Probability value

Model Components

EMC	Ego-motion compensation
AAM	Anchor alignment module
REG	Regression head
CLS	Class head

Signals

$\boldsymbol{y}_i$	ith object query
$\mathcal{Y}$	Set of object and track queries
$\boldsymbol{z}_i$	ith decoder output token
$\boldsymbol{\rho}_i$	ith query anchor location
$\boldsymbol{b}_i$	ith estimated bounding box
$\boldsymbol{t}_k$	kth track
$\boldsymbol{B}$	Matrix with entries drawn from a normal distribution
$\boldsymbol{T}$	Multiplicative transformation matrix
$\boldsymbol{C}$	Cost matrix
$\boldsymbol{f}_i$	ith sampled sensor feature

Model Parameters

N	Input sequence length
M	Number of object queries
L	Number of decoder layers
d	Length of feature vectors
D	Number of lidar points
K	Length of sampled subset
h	Number of attention heads
l_r	Learning rate
λ	Confidence threshold

Deep Learning Components and Symbols

FFN	Feed-forward network
$\boldsymbol{W}$	Learned transformation matrix in a feed-forward layer
$\boldsymbol{q}, \boldsymbol{k}, \boldsymbol{v}$	Query, key and value vectors
$\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}$	Query, key and value matrices
ℓ_1	Mean absolute deviation, ℓ_1-loss
b	Batch size

Bibliography

[ALS19] Allen-Zhu, Zeyuan; Li, Yuanzhi; and Song, Zhao: *A convergence theory for deep learning via over-parameterization.* In: *International Conference on Machine Learning*, pages 242–252. PMLR, 2019.

[And03] Anderson, T. W.: *An introduction to multivariate statistical analysis.* From the series *Wiley Series in Probability and Statistics.* 3rd ed., Hoboken, NJ: Wiley, 2003.

[Beu03] Beutelspacher, Albrecht: *Lineare Algebra.* In: Springer, 2003, page 30.

[BHZ+22] Bai, Xuyang; Hu, Zeyu; Zhu, Xinge; et al.: *TransFusion: robust lidar-camera fusion for 3D object detection with transformers.* In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1090–1099, 2022.

[Bru24] Brunhöber, Eike: *Startklar für das autonome Fahren: Boschs Technologen sind schon ziemlich weit.* In: *Ludwigsburger Kreizeitung*, 2024.

[BS08] Bernardin, Keni and Stiefelhagen, Rainer: *Evaluating multiple object tracking performance: the clearMOT metrics.* In: *EURASIP Journal on Image and Video Processing*, volume 2008, pages 1–10, 2008.

[BVV17] Beard, Michael; Vo, Ba Tuong; and Vo, Ba-Ngu: *OSPA (2): Using the OSPA metric to evaluate multi-target tracking performance.* In: *2017 International Conference on Control, Automation and Information Sciences (ICCAIS)*, pages 86–91. IEEE, 2017.

[CBL+20] Caesar, Holger; Bankiti, Varun; Lang, Alex H; et al.: *nuScenes: A multimodal dataset for autonomous driving.* In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 11621–11631, 2020.

[CBP21] Canatar, Abdulkadir; Bordelon, Blake; and Pehlevan, Cengiz: *Spectral bias and task-model alignment explain generalization in kernel regression and infinitely wide neural networks.* In: *Nature communications*, volume 12, no. 1, page 2914, 2021.

[CLAB21] Chiu, Hsu-Kuang; Li, Jie; Ambrus, Rares; and Bohg, Jeannette: *Probabilistic 3D multi-modal, multi-object tracking for autonomous driving.* In: *2021 IEEE international conference on robotics and automation (ICRA)*, pages 14227–14233, 2021.

[CMS+20] Carion, Nicolas; Massa, Francisco; Synnaeve, Gabriel; et al.: *End-to-end object detection with transformers.* In: *European Conference on Computer Vision (ECCV)*, pages 213–229. Springer, 2020.

[CMW+17] Chen, Xiaozhi; Ma, Huimin; Wan, Ji; Li, Bo; and Xia, Tian: *Multi-view 3d object detection network for autonomous driving.* In: *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pages 1907–1915, 2017.

[CPLB20] Chiu, Hsu-Kuang; Prioletti, Antonio; Li, Jie; and Bohg, Jeannette: *Probabilistic 3D multi-object tracking for autonomous driving.* In: *arXiv:2001.05673*, 2020.

[CWC+24] Chen, Li; Wu, Penghao; Chitta, Kashyap; et al.: *End-to-end autonomous driving: Challenges and frontiers.* In: *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2024.

[Cyb89] Cybenko, George: *Approximation by superpositions of a sigmoidal function.* In: *Mathematics of control, signals and systems*, volume 2, no. 4, pages 303–314, 1989.

[CZW+23] Chen, Xuanyao; Zhang, Tianyuan; Wang, Yue; Wang, Yilun; and Zhao, Hang: *FUTR3D: a unified sensor fusion framework for 3D detection.* In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, pages 172–181, 2023.

[Dao20] Dao, Minh-Quan: *Multi-object tracking with Poisson multi-Bernoulli mixture (PMBM) filter.* `https://github.com/quan-dao/pmbm-filter`. 2020.

[DDS+09] Deng, Jia; Dong, Wei; Socher, Richard; et al.: *Imagenet: A large-scale hierarchical image database.* In: *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. IEEE, 2009.

[Elm90] Elman, Jeffrey L: *Finding structure in time.* In: *Cognitive science*, volume 14, no. 2, pages 179–211, 1990.

[ELPZ97] Eldar, Yuval; Lindenbaum, Michael; Porat, Moshe; and Zeevi, Yehoshua Y: *The Farthest Point Strategy for Progressive Image Sampling.* In: *IEEE Transactions on Image Processing*, volume 6, no. 9, pages 1305–1315, 1997.

[FRD18] Feng, Di; Rosenbaum, Lars; and Dietmayer, Klaus: *Towards safe autonomous driving: capture uncertainty in the deep neural network for lidar 3D vehicle detection.* In: *IEEE Int. Conf. Intell. Transp. Syst.* Pages 3266–3273, 2018.

[Gal16] Gal, Yarin: *Uncertainty in deep learning.* PhD thesis, University of Cambridge, 2016.

[GBB11] Glorot, Xavier; Bordes, Antoine; and Bengio, Yoshua: *Deep sparse rectifier neural networks.* In: *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pages 315–323. JMLR Workshop and Conference Proceedings, 2011.

[GBC16] Goodfellow, Ian; Bengio, Yoshua; and Courville, Aaron: *Deep learning.* `http://www.deeplearningbook.org`. MIT Press, 2016.

[GDDM14] Girshick, Ross; Donahue, Jeff; Darrell, Trevor; and Malik, Jitendra: *Rich feature hierarchies for accurate object detection and semantic segmentation.* In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 580–587, 2014.

[GHZ+23] Gu, Junru; Hu, Chenxu; Zhang, Tianyuan; et al.: *Vip3d: End-to-end visual trajectory prediction via 3D agent queries.* In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5496–5506, 2023.

[Gir15] Girshick, Ross: *Fast R-CNN.* In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015.

[GLSU13] Geiger, A; Lenz, P; Stiller, C; and Urtasun, R: *Vision meets robotics: The KITTI dataset.* In: *The International Journal of Robotics Research*, volume 32, no. 11, pages 1231–1237, 2013.

[GPSW17] Guo, Chuan; Pleiss, Geoff; Sun, Yu; and Weinberger, Kilian Q: *On calibration of modern neural networks.* In: *International conference on machine learning*, pages 1321–1330. PMLR, 2017.

[GR18] Gast, Jochen and Roth, Stefan: *Lightweight probabilistic deep networks.* In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3369–3378, 2018.

[GRDG24] van Geerenstein*, Mathijs; Ruppel*, Felicia; Dietmayer, Klaus; and Gavrila, Dariu: *Multimodal object query initialization for 3D object detection.* In: *2024 IEEE international conference on robotics and automation (ICRA)*. IEEE, 2024. **equal contribution.*

[GWGS18] García-Fernández, Ángel F; Williams, Jason L; Granström, Karl; and Svensson, Lennart: *Poisson multi-Bernoulli mixture filter: Direct derivation and implementation.* In: *IEEE Transactions on Aerospace and Electronic Systems*, volume 54, no. 4, pages 1883–1901, 2018.

[Heb22] Hebermehl, G: *Mercedes mit Level-3-Zulassung: Bei Unfall mit "Drive Pilot" haftet Mercedes.* In: *Auto Motor und Sport*, 2022.

[HS97] Hochreiter, Sepp and Schmidhuber, Jürgen: *Long short-term memory.* In: *Neural computation*, volume 9, no. 8, pages 1735–1780, 1997.

[Hub92] Huber, Peter J: *Robust estimation of a location parameter.* In: *Breakthroughs in statistics: Methodology and distribution*, pages 492–518, Springer, 1992.

[JDZ21] Janjoš, Faris; Dolgov, Maxim; and Zöllner, J Marius: *Self-supervised action-space prediction for automated driving.* In: *2021 IEEE Intelligent Vehicles Symposium (IV)*, pages 200–207. IEEE, 2021.

[Kal60] Kalman, Rudolph Emil: *A New Approach to Linear Filtering and Prediction Problems.* In: *Transactions of the ASME Journal of Basic Engineering*, 1960.

[KJD18] Kocić, Jelena; Jovičić, Nenad; and Drndarević, Vujo: *Sensors and Sensor Fusion in Autonomous Vehicles.* In: *2018 26th Telecommunications Forum (TELFOR)*, pages 420–425, 2018.

[KSH12] Krizhevsky, Alex; Sutskever, Ilya; and Hinton, Geoffrey E: *ImageNet classification with deep convolutional neural networks.* In: *Advances in Neural Information Processing Systems.* Volume 25, Curran Associates, Inc., 2012.

[Kuh55] Kuhn, Harold W: *The Hungarian method for the assignment problem.* In: *Naval research logistics quarterly*, volume 2, no. 1-2, pages 83–97, 1955.

[LBBH98] LeCun, Yann; Bottou, Léon; Bengio, Yoshua; and Haffner, Patrick: *Gradient-based learning applied to document recognition.* In: *Proceedings of the IEEE*, volume 86, no. 11, pages 2278–2324, 1998.

[LBH15] LeCun, Yann; Bengio, Yoshua; and Hinton, Geoffrey: *Deep learning.* In: *nature*, volume 521, no. 7553, pages 436–444, 2015.

[LCW+23] Liao, Bencheng; Chen, Shaoyu; Wang, Xinggang; et al.: *MapTR: Structured modeling and learning for online vectorized HD map construction.* In: *The Eleventh International Conference on Learning Representations*, 2023.

[LI20] Li, You and Ibanez-Guzman, Javier: *Lidar for autonomous driving: the principles, challenges, and trends for automotive lidar and perception systems.* In: *IEEE Signal Processing Magazine*, volume 37, no. 4, pages 50–61, 2020.

[LLG+20] Liu, Liyuan; Liu, Xiaodong; Gao, Jianfeng; Chen, Weizhu; and Han, Jiawei: *Understanding the difficulty of training transformers.* In: pages 5747–5763, 2020.

[LOD+21] Luiten, Jonathon; Osep, Aljosa; Dendorfer, Patrick; et al.: *Hota: A higher order metric for evaluating multi-object tracking.* In: *International journal of computer vision*, volume 129, pages 548–578, 2021.

[LPB17] Lakshminarayanan, Balaji; Pritzel, Alexander; and Blundell, Charles: *Simple and scalable predictive uncertainty estimation using deep ensembles.* In: *Advances in neural information processing systems*, volume 30, 2017.

[LVC+19] Lang, Alex H; Vora, Sourabh; Caesar, Holger; et al.: *Pointpillars: Fast Encoders for Object Detection from Point Clouds.* In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 12697–12705, 2019.

[LYMP21] Liu, Shiwei; Yin, Lu; Mocanu, Decebal Constantin; and Pechenizkiy, Mykola: *Do we actually need dense over-parameterization? in-time over-parameterization in sparse training.* In: *International Conference on Machine Learning*, pages 6989–7000. PMLR, 2021.

[LYU18] Luo, Wenjie; Yang, Bin; and Urtasun, Raquel: *Fast and furious: Real Time End-to-End 3D Detection, Tracking and Motion Forecasting.* In: *Proc. IEEE Conf. Comput. Vis. Pattern Recog.* Pages 3569–3577, 2018.

[MABD98] Mazor, E.; Averbuch, A.; Bar-Shalom, Y.; and Dayan, J.: *Interacting Multiple Model Methods in Target Tracking: A Survey.* In: *IEEE Transactions on Aerospace and Electronic Systems*, volume 34, no. 1, pages 103–123, 1998.

[Mah14] Mahler, Ronald PS: *Statistical Multisource-Multitarget Information Fusion.* Artech House, Inc., 2014.

[MGJ21] Misra, Ishan; Girdhar, Rohit; and Joulin, Armand: *An End-to-End Transformer Model for 3D Object Detection.* In: *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 2906–2917, 2021.

[MKLF21] Meinhardt, Tim; Kirillov, Alexander; Leal-Taixe, Laura; and Feichtenhofer, Christoph: *TrackFormer: Multi-Object Tracking with Transformers.* In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 8844–8854, 2021.

[MMD20] MMDetection3D-Contributors: *MMDetection3D: OpenMMLab next-generation platform for general 3D object detection.* `https://github.com/open-mmlab/mmdetection3d`. 2020.

[Moz13] Mozer, Michael C: *A focused backpropagation algorithm for temporal pattern recognition.* In: *Backpropagation*, pages 137–169, Psychology Press, 2013.

[Nie15] Nielsen, Michael: *How the backpropagation algorithm works.* In: *Neural networks and deep learning. Determination Press*, 2015.

[NKB+21] Nakkiran, Preetum; Kaplun, Gal; Bansal, Yamini; et al.: *Deep double descent: Where bigger models and more data hurt.* In: *Journal of Statistical Mechanics: Theory and Experiment*, volume 2021, no. 12, page 124003, 2021.

[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*, volume 32, 2019.

[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: *Adv. Neural Inf. Process. Syst.* Volume 30, 2017.

[Ray23] Ray, Partha Pratim: *ChatGPT: A comprehensive review on background, applications, key challenges, bias, ethics, limitations and future scope.* In: *Internet of Things and Cyber-Physical Systems*, 2023.

[RDGF16] Redmon, Joseph; Divvala, Santosh; Girshick, Ross; and Farhadi, Ali: *You only look once: Unified, real-time object detection.* In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 779–788, 2016.

[RFGD22a] Ruppel, Felicia; Faion, Florian; Gläser, Claudius; and Dietmayer, Klaus: *Transformers for multi-object tracking on point clouds.* In: *IEEE Intelligent Vehicles Symposium (IV)*, pages 852–859, 2022.

[RFGD22b] Ruppel, Felicia; Faion, Florian; Gläser, Claudius; and Dietmayer, Klaus: *Transformers for object detection in large point clouds.* In: *IEEE International Conference on Intelligent Transportation Systems (ITSC)*, 2022.

[RFGD22c] Ruppel, Felicia; Faion, Florian; Gläser, Claudius; and Dietmayer, Klaus: *Can transformer attention spread give insights into uncertainty of detected and tracked objects?* In: *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) PNARUDE Workshop*, 2022.

[RFGD23] Ruppel, Felicia; Faion, Florian; Gläser, Claudius; and Dietmayer, Klaus: *Group regression for query based object detection and tracking.* In: *IEEE International Conference on Intelligent Transportation Systems (ITSC)*, 2023.

[RGS17] Rahmathullah, Abu Sajana; García-Fernández, Ángel F; and Svensson, Lennart: *Generalized optimal sub-pattern assignment metric.* In: *2017 20th International Conference on Information Fusion (Fusion)*, pages 1–8. IEEE, 2017.

[RHGS15] Ren, Shaoqing; He, Kaiming; Girshick, Ross; and Sun, Jian: *Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks*. In: *Advances in Neural Information Processing Systems*. Volume 28, 2015.

[RSS21] Reuse, Matthias; Simon, Martin; and Sick, Bernhard: *About the ambiguity of data augmentation for 3d object detection in autonomous driving*. In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 979–987, 2021.

[RVC10] Ristic, Branko; Vo, Ba-Ngu; and Clark, Daniel: *Performance evaluation of multi-target tracking using the OSPA metric*. In: *2010 13th International Conference on Information Fusion*, pages 1–7. IEEE, 2010.

[RVVD14] Reuter, Stephan; Vo, Ba-Tuong; Vo, Ba-Ngu; and Dietmayer, Klaus: *The labeled multi-Bernoulli filter*. In: *IEEE Transactions on Signal Processing*, volume 62, no. 12, pages 3246–3260, 2014.

[She23] Shen, Yinzhe: *End-to-end Perception and Prediction in Autonomous Driving*. Master Thesis, Stuttgart University, 2023.

[SKD+20] Sun, Pei; Kretzschmar, Henrik; Dotiwalla, Xerxes; et al.: *Scalability in Perception for Autonomous Driving: Waymo Open Dataset*. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020.

[SLX15] Song, Shuran; Lichtenberg, Samuel P.; and Xiao, Jianxiong: *SUN RGB-D: A RGB-D scene understanding benchmark suite*. In: *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 567–576, 2015.

[SVV08] Schuhmacher, Dominic; Vo, Ba-Tuong; and Vo, Ba-Ngu: *A consistent metric for performance evaluation of multi-object filters*. In: *IEEE transactions on signal processing*, volume 56, no. 8, pages 3447–3457, 2008.

[TDBM20] Tay, Yi; Dehghani, Mostafa; Bahri, Dara; and Metzler, Donald: *Efficient Transformers: A Survey*. In: *ACM Computing Surveys*, 2020.

[TLI+23] Touvron, Hugo; Lavril, Thibaut; Izacard, Gautier; et al.: *Llama: Open and efficient foundation language models*. In: *arXiv:2302.13971*, 2023.

[TSM+20] Tancik, Matthew; Srinivasan, Pratul; Mildenhall, Ben; et al.: *Fourier features let networks learn high frequency functions in low dimensional domains*. In: *Advances in neural information processing systems*, volume 33, pages 7537–7547, 2020.

[Uni19] Union, Generalized Intersection Over: *A metric and a loss for bounding box regression*. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 658–666, 2019.

[VS11] Von Luxburg, Ulrike and Schölkopf, Bernhard: *Statistical learning theory: Models, concepts, and results*. In: *Handbook of the History of Logic*, volume 10, pages 651–706, Elsevier, 2011.

[VSP+17] Vaswani, Ashish; Shazeer, Noam; Parmar, Niki; et al.: *Attention is all you need*. In: *Advances in neural information processing systems*, pages 5998–6008, 2017.

[VV13] Vo, Ba-Tuong and Vo, Ba-Ngu: *Labeled random finite sets and multi-object conjugate priors*. In: *IEEE Transactions on Signal Processing*, volume 61, no. 13, pages 3460–3475, 2013.

[WDS+19] Wolf, Thomas; Debut, Lysandre; Sanh, Victor; et al.: *Huggingface's transformers: State-of-the-art natural language processing*. In: *arXiv: 1910.03771*, 2019.

[Wil19] Wilthil, Erik: *GOSPApy*. `https://github.com/ewilthil/gospapy`. 2019.

[Wil32] Wilks, Samuel S: *Certain Generalizations in the Analysis of Variance*. In: *Biometrika*, pages 471–494, 1932.

[WLK+20] Wang, Sinong; Li, Belinda Z.; Khabsa, Madian; Fang, Han; and Ma, Hao: *Linformer: Self-Attention with Linear Complexity*. In: *arXiv: 2006.04768*, 2020.

[WMZY21a] Wang, Chunwei; Ma, Chao; Zhu, Ming; and Yang, Xiaokang: *PointAugmenting: Cross-Modal Augmentation for 3D Object Detection*. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 11794–11803, 2021.

[WMZY21b] Wang, Chunwei; Ma, Chao; Zhu, Ming; and Yang, Xiaokang: *Pointaugmenting: Cross-modal augmentation for 3d object detection*. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11794–11803, 2021.

[Woo14] Woodward, Philip M: *Probability and information theory, with applications to radar: international series of monographs on electronics and instrumentation*. Volume 3. Elsevier, 2014.

[WRLS19] Wirges, Sascha; Reith-Braun, Marcel; Lauer, Martin; and Stiller, Christoph: *Capturing Object Detection Uncertainty in Multi-Layer Grid Maps*. In: *IEEE Intelligent Vehicles Symposium (IV)*, pages 1520–1526, 2019.

[WWHK20] Weng, Xinshuo; Wang, Jianren; Held, David; and Kitani, Kris: *3D Multi-Object Tracking: A Baseline and New Evaluation Metrics*. In: *Proc. Int. Conf. Intell. Robots Syst.* Pages 10359–10366, 2020.

[YALZ21] Yao, Zhuyu; Ai, Jiangbo; Li, Boxun; and Zhang, Chi: *Efficient DETR: improving end-to-end object detector with dense prior*. In: *arXiv:2104.01318*, 2021.

[YCM+22] Yang, Zeyu; Chen, Jiaqi; Miao, Zhenwei; et al.: *Deepinteraction: 3d object detection via modality interaction*. In: *Advances in Neural Information Processing Systems*, volume 35, pages 1992–2005, 2022.

[YML18] Yan, Yan; Mao, Yuxing; and Li, Bo: *Second: Sparsely embedded convolutional detection*. In: *Sensors*, volume 18, no. 10, page 3337, 2018.

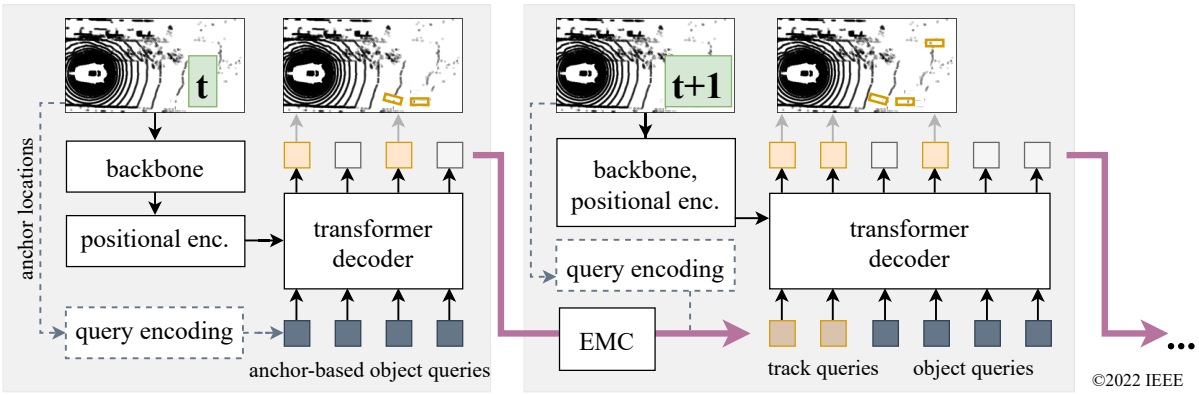
[YZK21] Yin, Tianwei; Zhou, Xingyi; and Krähenbuhl, Philipp: *Center-Based 3D Object Detection and Tracking*. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 11784–11793, 2021.

[ZJZ+19] Zhu, Benjin; Jiang, Zhengkai; Zhou, Xiangxin; Li, Zeming; and Yu, Gang: *Class-Balanced Grouping and Sampling for Point Cloud 3D Object Detection*. In: *arXiv:1908.09492*, 2019.

[ZSL+21] Zhu, Xizhou; Su, Weijie; Lu, Lewei; et al.: *Deformable DETR: Deformable Transformers for End-to-End Object Detection*. In: *International Conference on Learning Representations (ICLR)*, 2021.

[ZT18] Zhou, Yin and Tuzel, Oncel: *VoxelNet: End-to-End Learning for Point Cloud Based 3D Object Detection*. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[ZWK19] Zhou, Xingyi; Wang, Dequan; and Krähenbühl, Philipp: *Objects as points*. In: *arXiv:1904.07850*, 2019.



©2022 IEEE

ISBN 978-3-941543-99-7

ISBN 978-3-911223-00-3 (E-Book)