Christian Fruhwirth-Reisinger

Multiple Object Tracking in the context of Autonomous Driving

MASTER'S THESIS

to achieve the university degree of
Diplom-Ingenieur

Master's degree programme
Information and Computer Engineering

submitted to

Graz University of Technology

Supervisor

Univ.-Prof. Dipl.-Ing. Dr.techn. Horst Bischof
Institute of Computer Graphics and Vision

Advisor

Dipl.-Ing. Georg Krispel
Institute of Computer Graphics and Vision

Graz, Austria, Oct. 2019

Abstract

Autonomous driving has gained a lot attention over the last years. A huge contribution to that has been provided by recent progress in the fields of computer vision and machine learning. Nowadays, self-driving vehicles find their way on public roads which raises the needs for reliable visual perception systems. The main tasks of such systems are the detection and tracking of objects from uncertain measurements.

In this thesis we present an online modular Multiple Object Tracking (MOT) framework which makes use of spatial information, gathered from a 3D laser scanner and cameras. Within this framework we implemented multiple variants of Bayesian filters. We investigate the influence of various motion models and different association strategies which are responsible for the assignment of detections to tracked targets.

Furthermore, we exchange the motion model of these trackers with an implemented and trained data-driven Recurrent Neural Network (RNN). Additionally, we replace the used data association algorithm with an encoder-decoder structured neural network which is able to deal with a variable number of tracked targets and detections.

In our extensive evaluation on a publicly available autonomous driving dataset, we reach state-of-the-art performance with the best performing model. In addition, we study the influence of different object detectors, as well as the behavior of all implemented trackers when we drop detections of subsequent time steps, *i.e.* simulating a significantly unreliable object detector. While the exchange of equally performing object detectors do not have a huge impact on the overall performance, the subsequently dropped detections are compensated best by trackers with data-driven model components.

Kurzfassung

Autonomes Fahren hat in den letzten Jahren viel Aufmerksamkeit auf sich gezogen. Einen enormen Beitrag dazu leistet der Fortschritt in den Bereichen Maschinelles Sehens und Maschinelles Lernen. Aktuell gibt es einige Testversuche, diese selbstfahrenden Fahrzeuge in den öffentlichen Verkehr zu integrieren, was zu einer erhöhten Notwendigkeit an verlässlichen Systemen zur visuellen Wahrnehmung führt. Die Hauptaufgaben solcher Systeme sind die Detektion und das Tracken bzw. Verfolgen von Objekten mit Hilfe von ungenauen Messwerten.

In dieser Arbeit präsentieren wir ein modulares online Multiple Object Tracking (MOT) System, das räumliche Information von 3D Laser Sensoren und Kameras nutzt und unterschiedliche Bayessche Filter implementiert. Dabei untersuchen wir den Einfluss von Bewegungsmodellen und Assoziationsstrategien, die eine Zuordnung von Detektionen zu verfolgten Objekten sicherstellt.

Zusätzlich ersetzen wir eben diese Bewegungsmodelle mit eignes entwickelten und gelernten Neuronalen Netzwerken. In weiterer Folge ersetzen wir bisher verwendete Assoziationsalgorithmen durch ein Neuronales Netzwerk mit Encoder-Decoder Struktur. Dieses ist in der Lage mit einer variierenden Anzahl an Detektionen und verfolgten Objekten umzugehen.

In unserer ausführlichen Evaluierung auf einem öffentlich verfügbaren Datensatz für autonomes Fahren, erreichen wir mit dem besten Modell Ergebnisse, die am aktuellen Stand der Technik sind. Zusätzlich untersuchen wir den Einfluss von verschiedenen Detektoren und das Verhalten unserer Modelle beim Verlust mehrerer aufeinanderfolgender Detektionen. Dies simuliert zum Beispiel einen sehr unzuverlässigen Detektor oder andere, in der Praxis zu erwartende Detektionsfehler. Während das Austauschen von ungefähr gleich gut funktionierenden Detektoren kaum Einfluss auf das Gesamtresultat hat, kann der Ausfall von Detektionen am besten von Modellen kompensiert werden, bei denen einzelne Komponenten von einem Neuronalen Netz ersetzt wurden.

Affidavit

I declare that I have authored this thesis independently, that I have not used other than the declared sources/resources, and that I have explicitly indicated all material which has been quoted either literally or by content from the sources used.

The text document uploaded to TUGRAZonline is identical to the present master's thesis.

Date	Signature

Eidesstattliche Erklärung

Ich erkläre an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst, andere als die angegebenen Quellen/Hilfsmittel nicht benutzt, und die den benutzten Quellen wörtlich und inhaltlich entnommene Stellen als solche kenntlich gemacht habe.

Das in TUGRAZonline hochgeladene Textdokument ist mit der vorliegenden Masterarbeit identisch.

Ort	Datum	Unterschrift

Acknowledgments

First and foremost, I would like to thank my supervisor, Prof. Horst Bischof, for his guidance and encouragement, as well as for providing me the facilities to work at the Institute for Computer Graphics and Vision. Additionally, I am grateful to my advisor Georg Krispel, for his technical support and guidance. His door was always open for me. Furthermore, I would like to express my gratitude to Horst Possegger, for proofreading my thesis and giving me a lot of helpful comments and hints.

At this point, I also want tho thank my parents, Annemarie and Christian, for their love, for supporting me my whole life and for giving me the opportunity to follow my own path. Furthermore, I am deeply grateful to my wife's parents, Veronika and Alfred, for their love and their support over the last years.

Last but not least, I would like to express my deepest gratitude to my beloved wife Veronika, for supporting me all the time and for putting some pressure on me at the right moments. You make my life complete.

This work was partially supported by the Austrian Research Promotion Agency (FFG) under the project *DGT - Dynamic Ground Truth* (860820).

Contents

List of Figures

List of Tables

1 Introduction

Contents

In recent years, the interest in autonomous driving and especially in key technologies to enable self-driving vehicles has grown rapidly. The reasons for that are manifold. First of all, there are still approximately 1.35 million people dying each year as a result of traffic accidents, as published in the latest WHO status report on road safety in 2018 [111]. Most of them are induced by human errors or inattention. Furthermore, it enables mobility to disabled persons which are, for example, blind, have low vision or are otherwise incapable of steering a car. Apart from that, with an average driving time of 307.8 hours per year[1] and person, it wastes a lot of resources which could be better used otherwise. Hence, significant research has been done in the field of autonomous driving.

According to Janai et al. [67], the exploration of Intelligent Transportation Systems (ITS) started in 1986 in Europe with the EUREKA PROMETHEUS project[2] which was headed by car manufacturers and composed of several research units from governments and universities. Within this project the first autonomous long-distance drive from Munich (Germany) to Odense (Denmark) was performed with about 95% autonomy in 1995 [33, 34, 45]. A similar venture started in the United States with the Navlab [135] project by the Carnegie Mellon University in 1988. Later, the first real-time vision system [44] for autonomous driving in complex urban traffic situations was inspired by the successful PROMETHEUS project. Furthermore, many other projects in public environments like PROUD [20] with the adapted BRAiVE [57] prototype and challenges in controlled environments like the Defense Advanced Research Projects Agency (DARPA) Grand Challenge in 2004 occurred.

[1] `https://aaafoundation.org/wp-content/uploads/2018/02/18-0019_AAAFTS-ADS-Research-Brief.pdf` (accessed October 8, 2019)

[2] `http://www.eurekanetwork.org/project/id/45` (accessed November 30, 2018)

The first *DARPA* competition with an offered prize money of $1 million had no finishing team on a 140 miles long course through the desert. Another Grand Challenge was held one year later with five teams finishing the 132 miles long twisting and unpaved course [21]. The Stanford Racing Team won the competition with their robot car Stanley [137] and two cars from Carnegie Mellon University [140] finished in second and third place. The most recent event was the *DARPA* Urban Challenge [22] with the winning car Boss [139] and the runner-up Junior [104]. That competition consisted of a 60 miles long urban course including other traffic participants and real traffic regulations, which came closer to real driving conditions than previous challenges.

Nowadays, many commercial autonomous driving projects are known. Google started their self-driving car project in 2009 and founded an independent company called Waymo in 2016. Until October 2018, their cars covered more than 10 million miles[3] autonomously. Tesla has an advanced driver assistance system called autopilot[4] since 2015 and Uber currently tests their self-driving taxis in Pittsburgh (Pennsylvania)[5].

Within all these projects and prototypes, robust and reliable visual perception including object detection and tracking is a key issue. It allows reactive navigation, motion planning and is therefore crucial to avoid crashes and dangerous situations. Hence, significant research has been done in the field of Multiple Object Tracking (MOT) from 2D video sequences or 3D data recorded for example by Light Detection and Ranging (LIDAR) sensors. The latter is heavily used in the autonomous driving domain because it delivers spatial information which is essential for environmental perception. Today, requirements for autonomous vehicles – also called self-driving vehicles or driverless vehicles – are very strict because they find their way on public roads and therefore, a reliable perception system is crucial.

1.1 Multi-Object Tracking Challenges

The purpose of multi-object tracking algorithms is to jointly estimate the number of objects and their current states (*i.e.* position, velocity, orientation) from sensor data. *MOT* has been used in areas, like sports analysis (*e.g.* [15, 92, 110, 156]), biology (*e.g.* ants [76], bats [13], birds [93], cells [86, 101], fish [40, 131, 132]), robot navigation (*e.g.* [36, 37]) and autonomous driving (*e.g.* [42, 95, 109, 114, 117, 118, 126, 154]).

The main challenges in *MOT* are the assignment of detections to tracks and the determination whether a track exists or not. The former strongly depends on the detection quality. Many false or missed detections require additional knowledge of the tracked targets, *e.g.* their motion behavior, to produce a reasonable trajectory. The question whether a track exists or not, on the other hand, is much harder to answer. For example, tracks which are occluded by surrounding objects or other tracked targets could be wrongly clas-

[3] `https://www.waymo.com/ontheroad/` (accessed November 30, 2018)
[4] `https://www.tesla.com/autopilot` (accessed November 30, 2018)
[5] `https://www.uber.com/cities/pittsburgh/self-driving-ubers/` (accessed November 30, 2018)

sified as out of sight, while false detections may initiate a nonexistent track. However, the basis for a solid tracker are reliable detections.

Nowadays, object recognition in monocular images or 3D point clouds is dominated by deep neural networks, *e.g.* [52, 91, 129]. These models can be trained easily end-to-end to recognize objects such as cars, bicycles, pedestrians, traffic signs and many more. In contrast to object recognition, however, *MOT* is much more difficult to solve in an end-to-end fashion. The reasons for this are manifold. First, to train the huge amount of parameters within a deep model, we need large datasets of annotated training data which is not publicly available yet for *MOT*. Second, the input and output space can be different in every time step just like the sequence length for every training run or the number of detections and tracked targets. Moreover, continuous and discrete variables are used within the tracking process. While estimated internal states, like the position, are continuous, the termination or initialization of tracks must be discrete. This poses an additional challenge for potential end-to-end trained models.

1.2 Contribution and Outline

The aim of this thesis is to implement a modular *MOT* framework which contains various online models following the tracking-by-detection paradigm. These models are based on a combination of Bayesian filters. The modularity of our framework allows to easily exchange single components by data-driven Recurrent Neural Networks (RNNs), which we also implemented.

The input is restricted to geometrical and positional properties. Therefore, we do not use appearance features for the assignment of detections and tracks, because the computation of such features is mostly expensive regarding processing power and time. The reason for this limitation is twofold. First, we aim to achieve real-time behavior within our framework. Second, we want to make use of different object detectors and therefore require that they can be exchanged easily. Within our detailed evaluation, we also analyze the effect of exchanging single parts with *RNNs* to observe potential advantages and disadvantages. Finally, we investigate the robustness of all implemented trackers by dropping detections of certain frames.

The remainder of this thesis is structured as follows: First, in Chapter 2, we formulate the *MOT* problem and review related work along with some exemplary applications. In addition, we discuss datasets and explain common measures. Second, we derive Single Object Tracking (SOT) with Bayesian filters and summarize data association concepts for *MOT* in Chapter 3. Afterwards, we discuss preliminaries of *RNNs* and their usage for tracking in Chapter 4. In Chapter 5, we present the implemented modular tracking framework in detail – containing a specific composition of the introduced concepts – followed by the evaluation results in Chapter 6. Finally, we conclude this thesis and give an outlook on future work in Chapter 7.

2 Multiple Object Tracking

Contents

We now take a close look at Multiple Object Tracking (*MOT*). In particular, we give a formal representation of the problem and explain the tracking-by-detection paradigm and its categories. Furthermore, we discuss seminal work related to autonomous driving. Finally, this chapter ends with the description of typically used evaluation measures and common datasets.

2.1 Problem Formulation

Multiple Object Tracking (MOT) is the process of jointly estimating the number of targets and their states from noisy measurements of different sensor types, *e.g.* Radio Detection and Ranging (RADAR), Sound Navigation and Ranging (SONAR), camera (monocular, stereo) or Light Detection and Ranging (LIDAR). Thus, it can be seen as an estimation problem of multiple variables, where a track is defined as a state trajectory estimated from a set of measurements, associated with the same target [8]. The states are mostly position, orientation, velocity and acceleration of the tracked targets (*e.g.* [10, 78, 117]) but could also include the bounding box dimensions (*e.g.* [112, 114, 149]) or object class as it is useful for autonomous driving applications.

Because of its simplicity and universal form, we use an adapted version of the *MOT* formulation inspired by Luo et al. [94]. Formally, given a sequence of measurements $\mathbf{Z}$, the

state of the i^{th} object at measurement time t is denoted $\mathbf{x}_t^i$. Then, $\mathbf{X}_t = \{\mathbf{x}_t^1, \mathbf{x}_t^2, \ldots, \mathbf{x}_t^{M_t}\}$ defines the states of all M_t active objects at time t. Furthermore, we use $\mathbf{x}_{t_s:t_e}^i = \left\{\mathbf{x}_{t_s}^i, \ldots, \mathbf{x}_{t_e}^i\right\}$ to denote sequences of states regarding the i^{th} object, existing from time t_s to t_e. $\mathbf{X}_{0:t} = \{\mathbf{X}_0, \mathbf{X}_1, \ldots, \mathbf{X}_t\}$ represents then all the state sequences of all objects from the beginning until time t.

Following this formulation, we assume the dynamic system to be a Markov process, where we are not able to observe the true object states $\mathbf{X}_{0:t}$ directly. Instead, we observe noisy measurements from the dynamic system, modeled as random variables, to infer the true state. Such observations made from the measurement sequence at time t are denoted $\mathbf{z}_t^i$ for the i^{th} object. Similar to the definition of states, all M_t observations at time t can be written as $\mathbf{Z}_t = \{\mathbf{z}_t^1, \mathbf{z}_t^2, \ldots, \mathbf{z}_t^{M_t}\}$. Hence, all collected observation sequences made from the beginning until time t are defined as $\mathbf{Z}_{1:t} = \{\mathbf{Z}_1, \mathbf{Z}_2, \ldots, \mathbf{Z}_t\}$. Note, we assume measurements are taken after state initialization, which follows the common literature, *e.g.* [136]. Thus, we denote the first state of a sequence with index zero, whereas the first measurement of a sequence is denoted with index one.

The problem of finding the optimal state sequence $\widehat{\mathbf{X}}_{0:t}$ for all objects can be modeled as the Maximum a Posteriori (MAP) problem

$$\widehat{\mathbf{X}}_{0:t} = \arg\max_{\mathbf{X}_{0:t}} P(\mathbf{X}_{0:t}|\mathbf{Z}_{1:t}), \tag{2.1}$$

where $P(\mathbf{X}_{0:t}|\mathbf{Z}_{1:t})$ denotes the posterior distribution which describes the probability of the state sequence $\mathbf{X}_{0:t}$ given all observations $\mathbf{Z}_{1:t}$. Within this thesis we focus on statistical and probabilistic methods, respectively. These algorithms operate in two steps containing prediction and update which are formulated in a recursive form:

$$\textbf{Prediction:}\quad P(\mathbf{X}_t|\mathbf{Z}_{1:t-1}) = \int P(\mathbf{X}_t|\mathbf{X}_{t-1})\, P(\mathbf{X}_{t-1}|\mathbf{Z}_{1:t-1}) d\,\mathbf{X}_{t-1} \tag{2.2a}$$

$$\textbf{Update:}\quad P(\mathbf{X}_t|\mathbf{Z}_{1:t}) \propto P(\mathbf{Z}_t|\mathbf{X}_t)\, P(\mathbf{X}_t|\mathbf{Z}_{1:t-1}). \tag{2.2b}$$

This representation describes the general form of the majority of Bayesian filters [26]. The prediction step is the so called Chapman-Kolmogorov equation and represents a prior state Probability Density Function (PDF) at time step t. Here, $P(\mathbf{X}_t|\mathbf{X}_{t-1})$ represents the dynamic model. Furthermore, the update step uses the measurement likelihood function to correct the prior state and contains the observation model $P(\mathbf{Z}_t|\mathbf{X}_t)$. Note that the expression in Equation (2.2b) follows the most widely used notation, which omits the constant normalization factor. For each iteration, before the update can be done, it is necessary to solve the Data Association (DA) problem between measurements and objects of active tracks.

2.2 Tracking-by-Detection

The *MOT* formulation in Section 2.1 leads us to the frequently used tracking-by-detection paradigm which requires detected objects in each time step. These detections are the mentioned observations. As a consequence, only relevant objects are used for tracking which saves a lot of processing power and time. This property enables real-time behavior. Additionally, tracking-by-detection approaches are able to handle a variable number of targets over time. Both properties are crucial for online tracking in autonomous driving applications. Drawbacks, however, arise from detection errors, which are for example missing objects or False Negatives (FNs) and false detections or False Positives (FPs), because of clutter. This in turn means, that the performance of tracking-by-detection applications significantly depends on the detection quality.

Typically, tracking-by-detection consists of four main steps. First, the object detector tries to identify all targets at the current time step. Second, based on the previous states, the model predicts a new state for each object. Afterwards, all predicted states get associated to available measurements. Finally, the model corrects each target state with its assigned detection.

However, not every implementation follows this scheme. There are for example differences w.r.t. track handling. Some implementations need an explicit track management to take care of new tracks or tracks which should be terminated [81]. Monitoring of some physical constraints (*e.g.* overlapping trajectories) is an additional issue.

Another possibility is to follow a detection-free paradigm which does not rely on object detection. This kind of algorithms require manual initialization at the beginning and are not able to handle a variable number of objects over time. Because of these limitations, detection-free tracking algorithms are not part of this thesis.

2.3 Categorization of Tracking-by-Detection Approaches

In general, *MOT* implementations can be classified in different ways. In the following, we summarize two major categorization schemes, online or offline and deterministic or probabilistic.

2.3.1 Online vs. Offline

Online or causal tracking uses past information only and does not depend on future measurements. It processes data step-by-step. Hence, this type of algorithms is used for real-time applications. In contrast, offline approaches use batches of measurements to process estimates. One batch can contain all available data or specific time periods. Occasionally, if only soft real-time systems are needed, the model is allowed to look a few time steps into the future. In reality, the algorithm makes use of future measurements by running a predefined number of time steps behind.

2.3.2 Deterministic vs. Probabilistic

This distinction is based on the behavior regarding the system's output. On the one hand, if the model is deterministic, the output stays the same for every run, if the input data does not change. On the other hand, probabilistic models generate different output every time because of additive noise within the filtering process. Deterministic approaches are usually optimization methods which try to minimize some energy function and can be solved step-by-step or offline.

2.4 Related Work

Because of the large diversity of tracking methodologies, we focus mainly on online filtering-based (Section 2.4.1) and Deep Learning (DL) (Section 2.4.2) approaches which have been applied to traffic scenes. For more information, Yilmaz et al. [159] presented a detailed overview of fundamental work in tracking, which covers object detection and representation along with feature selection. More recently, Vo et al. [143] reviewed Joint Probabilistic Data Association (JPDA) filters, Multiple Hypothesis Tracking (MHT) filters and Random Finite Set (RFS) methods. Within their surveys, Cannons [24], Yang et al. [158] and Fan et al. [38] focused especially on visual object tracking, whereas Li et al. [87] reviewed and compared state-of-the-art methods based on *DL*. Krebs et al. [81] provided another detailed summary of traditional tracking methods and also examined tracking with *DL*. Sivaraman and Trivedi [130], Zhu et al. [164] and Janai et al. [67] summed up relevant work regarding autonomous driving.

Within this thesis we consider the tracking process itself and do not take object detection or appearance models into account. Thus, we assume preprocessed detections, *e.g.* points with bounding boxes in 2D (*e.g.* [65]) or 3D (*e.g.* [116, 129, 157]). An exception are *DL* models which usually incorporate some kind of object detector, but do not necessarily yield a bounding box. Instead they produce some high dimensional feature vector containing, for example, appearance features or motion features (*e.g.* [39, 148]). For detailed informations regarding 2D object detection we refer the interested reader to [65, 162].

2.4.1 Bayesian Filtering

Most *MOT* algorithms which follow the tracking-by-detection paradigm are modeled as parallel Single Object Tracking (SOT) approaches joined by some Data Association (*DA*) procedure. However, according to Vo et al. [143], even for the simple case of a single target, well-known filtering approaches, *e.g.* Kalman Filter (KF) [72], Extended Kalman Filter (EKF) [73] or Unscented Kalman Filter (UKF) [70, 147], can not be applied directly. The reason for this are measurement origin uncertainties, False Negatives (*FNs*) and False Positives (*FPs*).

A simple solution to this problem is the Nearest Neighbour (NN) filter [8, 26]. Thereby, the closest measurement in terms of spatial distance to each predicted measurement gets used for the target state update. Asvadi et al. [6] proposed a method for detection and tracking of moving objects based on *NN* data assignment and a linear *KF* for state estimation. Such a setup is prone to lose tracks because of wrong measurement-to-track associations due to clutter.

An improved version is the Probabilistic Data Association (PDA) filter [7, 9]. It uses association probabilities of certain measurements at each time step and applies the state estimation filter including the weighted measurements to all targets individually [8, 143]. Schreier [126] used *PDA* in a combined filter approach for vehicle tracking in a cluttered environment. However, both solutions, the *NN* filter and the *PDA* filter are usually designed for *SOT* and are not the best choice for *MOT* problems.

In contrast to local data assignment, there exist global strategies which take care of all measurements and tracked objects in every time step. Beginning with the simplest one, Global Nearest Neighbour (GNN) approaches try to solve the global *DA* problem by minimizing/maximizing some total cost/value w.r.t. distance or likelihood. For tracking multiple detected objects in autonomous vehicles out of 3D *LIDAR* scans, Choi et al. [30] combined *GNN* association with a linear *KF*. Their association criterion is based on a weighted sum of distance and hypothesis size. A more sophisticated approach which does sensor fusion and provides a 360° observation around the vehicle has been proposed by Rangesh and Trivedi [118]. Within their work they used the popular Hungarian algorithm [106] which yields the optimal target-to-measurement assignment if every target produces exactly one measurement. The same *DA* method has been used by Sharma et al. [128], but with different cues for the similarity scores. Here, object similarities are modeled by costs derived from 3D cues, which are directly learned from monocular images.

Two of the most popular algorithms for *MOT* are Joint Probabilistic Data Association (*JPDA*) [7, 43] and Multiple Hypothesis Tracking (*MHT*) [17, 120]. Furthermore, the *JPDA* filter is an extension to the *PDA* filter for a fixed and known number of targets which requires track existence. It performs a weighted update including all measurements within a certain gating region simultaneously regarding to all tracks. Rachman [117] used *JPDA* within a combination of Bayesian filters – designed for position tracking of vehicles and pedestrians – including an *UKF* for non-linear state estimation. Additionally, after object detection from a 3D *LIDAR* point cloud, a separate logic-based mechanism takes care of the bounding box size and heading. Another framework relying on *JPDA* was used by Ćesić et al. [25]. It performs object detection and tracking from *LIDAR* data, recorded with a sensor mounted on a mobile platform.

Contrary to *JPDA* filters, *MHT* provides an optimal solution to the assignment problem in cluttered environments. Moreover, a varying number of targets over time is handled implicitly. In general, *MHT* builds hypotheses for all possible target-to-measurement assignments. Hypothetical new tracks get considered, as well as the termination of current ones. The main drawback is the exponentially growing complexity with every time step

and new target or measurement, respectively. This leads to some implementation strategies which try to reduce the complexity *e.g.* hypothesis pruning which has been proposed by Ess et al. [37]. Within this process the unlikely hypotheses are pruned and only the most promising remain.

Nevertheless, since *MHT* is very expensive and the *JPDA* filter is not able to deal with a varying number of targets over time, the Joint Integrated Probabilistic Data Association (JIPDA) [107, 108] filter can be seen as a promising alternative. It additionally obtains track existence probabilities for an individual and limited number of tracks. Otto et al. [113] used a *JIPDA* filter for pedestrian tracking with monocular cameras and *RADAR* sensors in the frontal field of view and the vehicle's blind spot. For a more detailed overview, we refer to Cox [32] who explains several statistical *DA* techniques for motion correspondences.

Another family of *MOT* algorithms are Random Finite Set (*RFS*) methods which make use of Finite Set Statistics (FISST), where both, the cardinality of the set as well as its members, are random variables. Therein, the Probability Hypothesis Density (PHD) [97, 144] filter and the Cardinalized Probability Hypothesis Density (CPHD) [98] filter are two popular approaches which perform state and target existence estimation implicitly without the need of an additional *DA* procedure.

Nevertheless, these methods do not provide target identification mechanisms. Therefore, some track management is needed to extract target tracks with assigned IDs. Approaches without such an additional extraction process are the Generalized Labeled Multi-Bernoulli (GLMB) [121, 145] filter and the Poisson Multi-Bernoulli Mixture (PMBM) [47, 153] filter. For all the *RFS*-based algorithms there exists no closed-form solution and therefore Particle Filter (PF) [2, 5, 59] implementations are usually applied, *e.g.* [124]. Scheidegger et al. [125] proposed a method to estimate the position of other vehicles on the road in world coordinates form monocular images only. Within their work, a *PMBM* filter is used for *MOT*. Instead of using a *PF* for motion estimation, they leverage a Gaussian assumption for their measurement and motion model to apply an *UKF*.

Sequential Monte Carlo (SMC) methods like the *PF*, also known as bootstrap filtering [54] or the condensation algorithm [96], can also be directly used for state estimation in tracking-by-detection approaches followed by *DA*. It performs better in non-linear/non-Gaussian environments because it approximates the posterior *PDF* by a finite set of particles. *PF*-based approaches for autonomous driving applications have been proposed, e.g., by Fortin et al. [41, 42], Morales et al. [105] and Niknejad et al. [109]. Additionally, tracking targets with *PF* can be performed by extending the state space of the filter to contain multiple targets (*e.g.* [66, 75, 96]). This leads to a fixed number of tracks and can be inefficient by calculating the posterior *PDF* for such high-dimensional spaces. Another drawback is the difficult *DA* within the state space which leads to many ID switches.

To cope with these issues, Markov Chain Monte Carlo (MCMC) sampling has been applied to *MOT* problems which reduces complexity and solves the association problem. Reversible Jump MCMC (RJMCMC) additionally handles varying numbers of objects,

(*e.g.* [31, 76, 85]). Moreover, Doucet et al. [35] have shown that in some cases, parts of the model could be solved analytically, *e.g.* by *KF*. Hence, not all state variables have to be sampled by the *PF* which results in better performance regarding time and accuracy. The method is called Rao-Blackwellized Particle Filter (RBPF) and has been used for *MOT*, *e.g.* by Petrovskaya and Thrun [114], Vatavu et al. [142] and Wojke and Häselich [154].

All these algorithms need an appropriate model of the object dynamics, *i.e.* motion model. Typically, for maneuvering targets such as vehicles, Constant Velocity (CV), Constant Turn-Rate Velocity (CTRV) or Constant Acceleration (CA) are used frequently. For the interested reader, Li and Jilkov [88] give a good overview of dynamic models for maneuvering and non-maneuvering targets. Because the behavior of vehicles, especially in urban environments, can not be modeled by a single motion pattern, multiple model approaches are common solutions to this issue. Thus, Weiss et al. [149] used an approach to switch between different models based on statistical tests of the fused data of their sensors.

This simple approach could rise problems regarding delayed or oscillating model switches. Hence, a more stable and more precise solution is the Interacting Multiple Model (IMM) [18] filter. It uses a weighted sum of the individual filter estimates. They result from various dynamic models which can additionally have different filter types. Within their work, Barth and Franke [10], Kaempchen and Dietmayer [71], Kim and Hong [79], Rachman [117] and Schreier [126] used such an *IMM* filter for smooth model switching.

2.4.2 Deep Learning

A more recent research direction is to leverage Deep Neural Networks (DNNs) for *MOT*, which are architectures with many hidden layers between the input and output neurons. They have gained a lot of attention within the last years because of their impressive performance in object detection and classification (*e.g.* [83]) tasks as well as other computer vision challenges such as semantic (instance) segmentation (*e.g.* [27, 28, 60, 90]) or motion (*e.g.* [134]) and pose estimation (*e.g.* [155]).

Because tracking-by-detection approaches crucially depend on good detections, such networks are commonly used to find good appearance features (*e.g.* [58]) or detect objects directly (*e.g.* [14, 85]). The best public available solution on the KITTI Vision Benchmark [49] leaderboard for tracking[1] uses a *DNN* and has been proposed by Sharma et al. [128]. Their pipeline finds appearance features in the form of 2D and 3D cues out of monocular images. Afterwards, they solved the Data Association (*DA*) with different cost functions depending on these cues.

Another interesting application which uses detected features is the work form Kim et al. [77]. They used a Convolutional Neural Network (CNN) to find features for a *MHT* filter. Scheidegger et al. [125] used monocular images as an input for a deep neural network

[1] `http://www.cvlibs.net/datasets/kitti/eval_tracking.php` (accessed January 22, 2019)

to estimate the position of multiple objects in 3D world coordinates relative to the ego perspective of the car. Afterwards, they performed tracking with a *PMBM* filter within this coordinate system. The available 3D information from the point cloud was used only for training the network, but not for inference.

However, a lot of research deals with end-to-end training of deep neural networks which are able to learn *MOT* from raw input. Thus, monocular images, depth maps or point clouds are used separately or in combination as an input to detect objects and track them over the whole sequence. A Recurrent Neural Network (*RNN*) approach with simple point detections as an input was proposed by Milan et al. [103] . The network represents equal structures like well-known Bayesian filters. It performs a prediction step followed by an update step which uses the associated measurement values determined by a Long Short-Term Memory (*LSTM*) network. Implicitly, they handle the initialization and termination of tracks by some estimated probability for both cases. Except for the object detection, their work can be seen as an end-to-end approach.

Another solution to that problem has been proposed by Frossard and Urtasun [46]. Within their work, 3D *LIDAR* data and monocular images were used as an input to train an object detector followed by a Siamese network to exploit matching costs of objects. Inference has been done by feed forward passes followed by solving a linear program. Their model can be trained end-to-end and has been applied on tracking vehicles, cyclists and pedestrians. Luo et al. [95] proposed another network structure which detects and tracks objects from 3D *LIDAR* data only, represented in a bird's eye view. Moreover, they used motion forecasting as an additional contribution against loosing tracks during occlusions. For evaluation, an own not yet published large scale dataset was recorded to train and test the network which complicates reproducibility.

2.5 Evaluation Measures

Within this section we discuss standardized methods for evaluating Multiple Object Tracking (*MOT*) approaches. A comparable performance measure for such algorithms is given by the Classification of Events, Activities and Relationships (CLEAR) [12] measures, which are also used by both benchmarks we train and test our models on. In addition to the *CLEAR* measures, the MOT16 Benchmark [102], as well as the KITTI Vision Benchmark Suite [49] are using quality measures regarding to trajectory coverage, as proposed by Li et al. [89].

2.5.1 CLEAR MOT Measures

Bernardin and Stiefelhagen [12] defined two independent and intuitive measures which are called Multiple Object Tracking Precision (MOTP) and Multiple Object Tracking Accurency (MOTA). They are designed to reflect the tracker's precision in terms of exact object locations and to represent the overall tracking ability, which can be described

as the consistent labeling of objects over time. The reliability of correct *DA* between existing tracks and detected objects is expressed by *FP* and *FN* assignments per frame. Another figure of merit is the number of ID switches (IDS), which describes the mismatch of tracks. It is influenced by switching identities between different trajectories and the fragmentation of tracks. The latter occurs, when a trajectory is interrupted (*e.g.* due to missing observations during occlusions) and a new ID is assigned after it resumes.

The overall tracking performance score *MOTA* combines the three explained error parameters *FP*, *FN* and *IDS*

$$\text{MOTA} = 1 - \frac{\sum_t (\text{FP}_t + \text{FN}_t + \text{IDSW}_t)}{\sum_t GT_t}, \tag{2.3}$$

with the sum over time steps t, containing all incorrect events within a sequence, in the numerator and the total number of Ground Truth (GT) objects in the denominator. This equation leads to a maximum value of 1, describing 100% accuracy and on the other hand to a negative value if there are more false events than ground truth objects. According to Bernardin and Stiefelhagen [12], *MOTA* can be seen as a derivation of three individual error ratios, which are all averaged by the total number of present GT objects. The three error measures are given as the ratio of missed object associations (FN) over time

$$\overline{\text{FN}} = \frac{\sum_t \text{FN}_t}{\sum_t GT_t}, \tag{2.4}$$

the *FP* ratio

$$\overline{\text{FP}} = \frac{\sum_t \text{FP}_t}{\sum_t GT_t}, \tag{2.5}$$

and the ratio of *IDS*

$$\overline{\text{IDSW}} = \frac{\sum_t \text{IDSW}_t}{\sum_t GT_t}. \tag{2.6}$$

Combining these error scores as $E_{tot} = \overline{\text{FN}} + \overline{\text{FP}} + \overline{\text{IDSW}}$, we obtain the *MOTA* score as in Equation (2.3) by $1 - E_{tot}$, which provides a very intuitive performance measure.

Besides the overall tracking performance, *MOTP* represents the precision of a tracker. It evaluates the average deviation between True Positive (TP) and GT tracks w.r.t. some similarity measurement as

$$\text{MOTP} = \frac{\sum_{i,t} d_t^i}{\sum_t c_t}, \tag{2.7}$$

where d_t^i and c_t denote the distance error of the i^{th} match and the number of matches at time t, respectively. Mostly the Euclidean distance is used to get an average deviation in meters or pixels. For bounding boxes in 2D and 3D space, as used in both related benchmarks MOT16 and KITTI, the overlap or Intersection over Union (IoU) is used instead of the Euclidean distance. This yields an average overlap percentage of bounding

boxes. For distance measurements, lower values indicate better results, whereas for IoU, a score of 1 is desirable, which denotes 100% precision.

While state variables like speed or turn rate are not considered, the orientation implicitly influences the evaluation by using the IoU criterion. Furthermore, for autonomous driving applications in the real-world coordinate system, the overlap calculation can be reduced to the ground plane of objects, because the height of vehicles or pedestrians does not change. In addition, there is almost no vertical movement.

2.5.2 Quality Measures

The *CLEAR* measures are helpful to compare precision and accuracy, but they are not informative regarding the coverage of individual tracks. Therefore, we discuss some quality measures [89] to classify trajectories into Mostly Tracked (MT), Partly Tracked (PT) and Mostly Lost (ML). These classes describe the coverage of single tracks and do not take ID switches into account. Objects which are tracked for at least 80% of the time – compared to the ground truth – are considered as MT. On the other hand, targets with a coverage of less than 20% are classified as ML. Everything in between is said to be $PT = 1 - MT - ML$. Thus, these values can be written as absolute numbers or as a ratio to the total number of ground truth trajectories.

Another helpful metric is the number of track Fragmentations (FM) which describes how often a track gets interrupted. An interruption happens if the track gets lost and resumes later on. This should not be confused with the similar concept of IDS. If we think of two tracks changing its ID, the track does not necessarily get interrupted. The value is important for tasks where we are interested in long persistent tracks without interruptions.

2.6 Datasets

To compare state-of-the-art approaches, two of the most common datasets for MOT are used in this thesis. The MOT16 Dataset [102], which also provides an online benchmark and a corresponding leaderboard[2], is an established standard for this sort of algorithms. It consists of monocular image sequences containing persons and vehicles from both, static and moving cameras. Another publicly available dataset is the KITTI Dataset [48] provided by the Karlsruhe Institute of Technology and the Toyota Technological Institute of Chicago. Contrary to MOT16, the KITTI Dataset includes not only monocular images but stereo images, high-precision GPS measurements, IMU accelerations and *LIDAR* point clouds, which enables tracking in real-world coordinates. It has also an online benchmark completed by a leaderboard for MOT. The application area is limited to autonomous driving and therefore, the sensor setup stays the same for all recordings.

[2] `https://motchallenge.net/results/MOT16/` (accessed January 25, 2019)

2.6.1 MOT16 Dataset

With the aim of a new standardized benchmark for *MOT* methods, Leal-Taixé et al. [84] proposed the first version of the MOT Challenge. It contains a collection of various publicly available datasets, a centralized evaluation method and an infrastructure for other researchers in this field to share data, new evaluation methods and annotations. This resulted in the MOT15 Dataset and Benchmark with corresponding online leaderboard[3].

Because of some shortcomings, a new version of the MOT Challenge has been released by Milan et al. [102] with the MOT16 Benchmark. This benchmark contains 14 sequences of different crowded scenarios including various viewpoints, camera motions and weather conditions. The sequences are equally divided into a training set, which contains also ground truth annotations, and a test set used for verification. Another important improvement was a standardized ground truth labeling process.

The main focus of this dataset and benchmark lies on video surveillance tasks containing crowded scenes with people. Therefore, annotations are split into three main classes. The first one contains moving or standing pedestrians and people on bikes or skateboards. The second class includes artificial persons, *e.g.* reflections, people behind glass or mannequins, and static pedestrians in a not upright position. Tracking of detections within this class gets neither rewarded nor penalized while testing. The last group gets not considered within the benchmark test. It contains vehicles and objects which occlude pedestrians.

Recorded sequences have different image resolutions and vary in length and frame rate (Frames Per Second (FPS)). The dataset contains 11286 frames in total including 1342 trajectories. Figure 2.1 shows an example frame from the MOT16 dataset. We refer the interested reader to the work of Milan et al. [102] for further details.

This dataset is not the perfect choice for testing algorithms which are designed for solving the *MOT* problem within an autonomous driving scenario. Such trackers are built to work in a real-world coordinate system and not on the image plane. Because the motion behavior in real-world coordinates relative to the ego-vehicle differs in contrast to movements on the 2D image plane, also trackers have different requirements. Notwithstanding, we use the dataset to test the basic functionality of our models.

2.6.2 KITTI Dataset

The most important benchmark w.r.t. autonomous driving is the KITTI Vision Benchmark Suite [49] and the corresponding KITTI Dataset [48], because of its opportunity to test algorithms on real-world situations. It contains sequences of different traffic scenes recorded in Karlsruhe (Germany) with a modified VW Passat shown in Figure 2.2. The six hour long recordings capture freeways, rural areas and urban environments, whereby, only 25% of them are publicly available. They consist of high resolution color and grayscale stereo images synchronized with 3D *LIDAR* scans, taken by a Velodyne HDL-64E sensor.

[3] `https://motchallenge.net/results/2D_MOT_2015/` (accessed January 25, 2019)

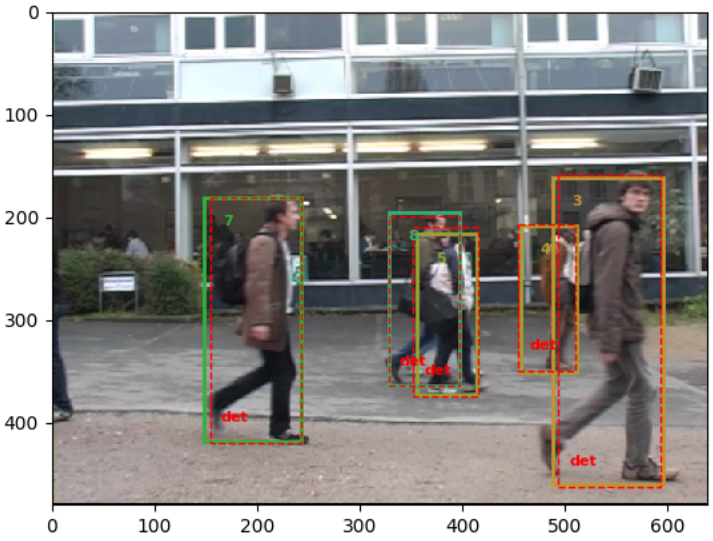


Figure 2.1: MOT16 [102] sample image including provided detections and tracks. The image originates from the sequence TUD-Campus [4] and shows frame number 51. Ground truth detections are marked with red dotted boxes and tracks are solid boxes.

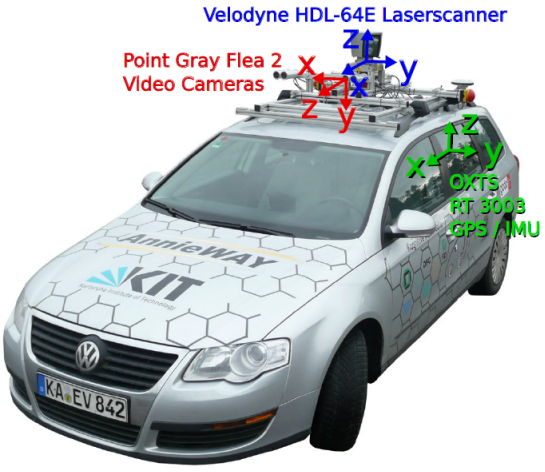


Figure 2.2: VW Passat equipped with video cameras and 3D *LIDAR* scanner used by the KITTI team. Image taken from [48]

Furthermore, the dataset includes hand-labeled 3D tracklets of static and dynamic objects representing the ground truth. All raw recordings are classified into: *Residential*, *City*, *Road*, *Campus* and *Person*. A labeled 3D point cloud example and the corresponding image frame can be seen in Figure 2.3.

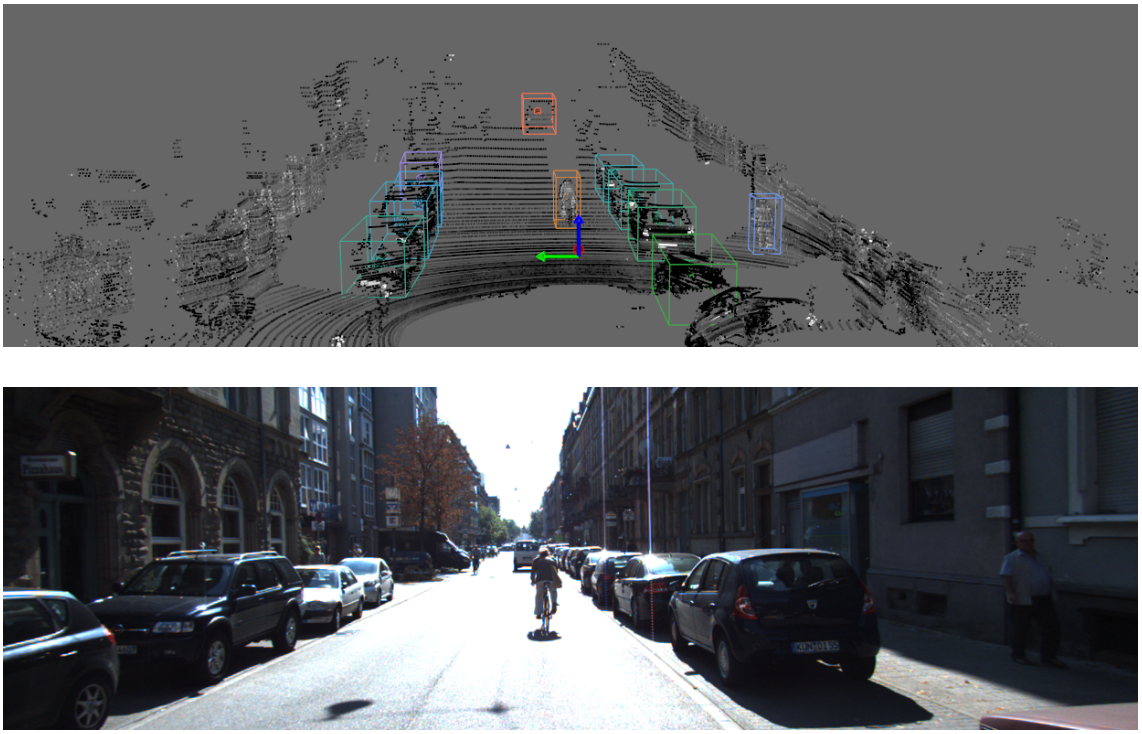


Figure 2.3: KITTI dataset example with annotated point cloud and corresponding image frame. It originates from sequence *2011_09_26_drive_0005* and shows frame number 151. Within the point cloud snapshot, an axis defines the ego-car position, where arrows in red, green and blue point along the x, y and z axes, respectively. Note that the view point in both images is not exactly the same.

The focus of our work is on tracking multiple objects in real-world coordinates and therefore, detections in 3D space are needed, where detectors are allowed to make use of all available data. The KITTI Vision Benchmark Suite evaluates tracking algorithms – more or less the same as the MOT16 benchmark – on the 2D image plane. Therefore, targets are only labeled if they are in the cameras' fields-of-view. Hence, the evaluation of tracks in real-world coordinates is only possible offline with sequences where ground truth is provided. Detailed information concerning evaluation with the KITTI Dataset can be found in Chapter 6.

3 Multi-object Tracking with Bayesian Filters

Contents

The main focus of this chapter is on the fundamental theory of Bayesian filters for tracking multiple targets. After a short introduction to the optimal Bayesian filter and its derivation, we discuss different solutions for state estimation and model switching, primarily following the literature [8, 26, 82, 126, 136]. Finally, we explain probabilistic Data Association (DA) concepts in form of the Probabilistic Data Association (PDA) filter and the Joint Probabilistic Data Association (JPDA) filter, both designed to enable tracking in clutter for Bayes filters.

3.1 Recursive Bayesian Filter for Object Tracking

Online multi-object tracking can be seen as a dynamic state estimation problem under uncertainty with changing states over time, driven by sequential noisy measurements. When new observations are available, the next state or belief of all active objects gets predicted for each time step based on a dynamic model. Afterwards, assigned measurements are used to update the belief with a corresponding measurement model. For such problems with noisy measurements, all quantities are modeled as random variables. Because of that and the sequential structure, a recursive method for reasoning under uncertainty is needed. A well-developed framework which fulfills this requirements and allows some computationally traceable solutions is the Bayesian filter. It represents the state of an object as a posterior Probability Density Function (PDF) over all parameters based on the previous state, observations and control actions.

3.1.1 Optimal Filter Derivation

We use and extend the definition in Section 2.1 for the optimal Bayesian filter derivation, but reduce the problem to a single object to avoid cluttering the notation. Thus, hidden target states at time t are denoted $\mathbf{x}_t$ and a sequence of all states, starting with the initial state $\mathbf{x}_0$ up to the current time t, as $\mathbf{x}_{0:t} = \{\mathbf{x}_0, \dots, \mathbf{x}_{t-1}, \mathbf{x}_t\}$ accordingly. Similarly, the observed measurement sequence up to time t is $\mathbf{z}_{1:t} = \{\mathbf{z}_1, \dots, \mathbf{z}_{t-1}, \mathbf{z}_t\}$. Since both, the states as well as the measurements, are affected by uncertainties, they are modeled as random variables. For the filter derivation we assume to know the measurement-to-target correspondences. Additionally, for each time-step there is exactly one measurement available.

The posterior *PDF* or belief $p(\mathbf{x}_{0:t}|\mathbf{z}_{1:t})$, representing the complete probabilistic knowledge of the state, can be calculated with the measurement sequence $\mathbf{z}_{1:t}$ and a prior distribution $p(\mathbf{x}_{0:t})$ applying Bayes' rule:

$$p(\mathbf{x}_{0:t}|\mathbf{z}_{1:t}) = \frac{\overbrace{p(\mathbf{z}_{1:t}|\mathbf{x}_{0:t})}^{\text{Likelihood}} \overbrace{p(\mathbf{x}_{0:t})}^{\text{Prior}}}{\underbrace{p(\mathbf{z}_{1:t})}_{\text{Normalization}}}. \tag{3.1}$$

The conditional probability distribution $p(\mathbf{x}_{0:t}|\mathbf{z}_{1:t})$ denotes the posterior *PDF* of the object state sequence. The numerator contains a multiplication of the state's prior with the likelihood. The latter provides a probability, that given the hidden state sequence is $\mathbf{x}_{0:t}$, the observed measurement sequence is $\mathbf{z}_{1:t}$. The normalization factor within the denominator ensures that the posterior *PDF* integrates to one and thus, is valid. It can be calculated by marginalizing out the numerator:

$$p(\mathbf{z}_{1:t}) = \int_{\mathbf{x}_t} \dots \int_{\mathbf{x}_0} p(\mathbf{z}_{1:t}|\mathbf{x}_{0:t})\, p(\mathbf{x}_{0:t})\, d\,\mathbf{x}_t \dots d\,\mathbf{x}_0. \tag{3.2}$$

Here, $\int_{\mathbf{x}_t}$ denotes integration over the whole range of $\mathbf{x}_t$. It represents all operations of summation and integration, *e.g.* states containing a mixture of continuous and discrete values.

To obtain recursive behavior where the time dependent measurement sequence can be processed step-by-step over time, the terms in Equation (3.1) can be factorized into

$$p(\mathbf{z}_{1:t}|\mathbf{x}_{0:t}) = p(\mathbf{z}_t|\mathbf{z}_{1:t-1}, \mathbf{x}_{0:t})\, p(\mathbf{z}_{1:t-1}|\mathbf{x}_{0:t}), \tag{3.3a}$$

$$p(\mathbf{x}_{0:t}) = p(\mathbf{x}_t|\mathbf{x}_{0:t-1})\, p(\mathbf{x}_{0:t-1}) \quad \text{and} \tag{3.3b}$$

$$p(\mathbf{z}_{1:t}) = p(\mathbf{z}_t|\mathbf{z}_{1:t-1})\, p(\mathbf{z}_{1:t-1}). \tag{3.3c}$$

The factorized likelihood in Equation (3.3a) can further be simplified by taking the causality principle into account [26]. This means that measurements do not depend on future

states:

$$p(\mathbf{z}_{1:t}|\mathbf{x}_{0:t}) = p(\mathbf{z}_t|\mathbf{z}_{1:t-1}, \mathbf{x}_{0:t})\, p(\mathbf{z}_{1:t-1}|\mathbf{x}_{0:t-1}). \tag{3.4}$$

From the substitution of Equation (3.1) with the factorized elements in Equation (3.3b), (3.3c) and (3.4) along with reordering of the terms follows:

$$p(\mathbf{x}_{0:t}|\mathbf{z}_{1:t}) = \frac{p(\mathbf{z}_t|\mathbf{z}_{1:t-1}, \mathbf{x}_{0:t})\, p(\mathbf{x}_t|\mathbf{x}_{0:t-1})}{p(\mathbf{z}_t|\mathbf{z}_{1:t-1})} \left(\frac{p(\mathbf{z}_{1:t-1}|\mathbf{x}_{0:t-1})\, p(\mathbf{x}_{0:t-1})}{p(\mathbf{z}_{1:t-1})} \right). \tag{3.5}$$

The second term, can then be rewritten as the prior joint *PDF*

$$p(\mathbf{x}_{0:t-1}|\mathbf{z}_{1:t-1}) = \frac{p(\mathbf{z}_{1:t-1}|\mathbf{x}_{0:t-1})\, p(\mathbf{x}_{0:t-1})}{p(\mathbf{z}_{1:t-1})}, \tag{3.6}$$

which leads to the recursive form of the Bayesian solution:

$$p(\mathbf{x}_{0:t}|\mathbf{z}_{1:t}) = \frac{p(\mathbf{z}_t|\mathbf{z}_{1:t-1}, \mathbf{x}_{0:t})\, p(\mathbf{x}_t|\mathbf{x}_{0:t-1})}{p(\mathbf{z}_t|\mathbf{z}_{1:t-1})}\, p(\mathbf{x}_{0:t-1}|\mathbf{z}_{1:t-1}). \tag{3.7}$$

Furthermore, in real-world applications two simplifying assumptions are made. First, measurements are generated from states at the current time only and thus, are conditionally independent from previous measurements. Hence, the expression $p(\mathbf{z}_t|\mathbf{z}_{1:t-1}, \mathbf{x}_{0:t})$ leads to the simpler form $p(\mathbf{z}_t|\mathbf{x}_t)$ which points out, that the state $\mathbf{x}_t$ is sufficient to predict the measurement $\mathbf{z}_t$ [136]. Therefore, the state has to be considered as complete. Second, the object state has to meet the Markov assumption. This requires conditional independence between the current state $\mathbf{x}_t$ given the previous state $\mathbf{x}_{t-1}$ and the older state sequence $\mathbf{x}_{0:t-2}$. Formally, from $p(\mathbf{x}_t|\mathbf{x}_{0:t-1})$ follows $p(\mathbf{x}_t|\mathbf{x}_{t-1})$. According to Thrun et al. [136], these conditional independence assumptions for states and measurements are the primary reason why most of the algorithms derived from this formulation are computationally traceable.

Applying both assumptions to the original formulation in Equation (3.7), we get another simplified representation:

$$p(\mathbf{x}_{0:t}|\mathbf{z}_{1:t}) = \frac{p(\mathbf{z}_t|\mathbf{x}_t)}{p(\mathbf{z}_t|\mathbf{z}_{1:t-1})}\, p(\mathbf{x}_t|\mathbf{x}_{t-1})\, p(\mathbf{x}_{0:t-1}|\mathbf{z}_{1:t-1}). \tag{3.8}$$

Attentive readers may have noticed, that the result of Equation (3.8) still represents the full joint posterior density $p(\mathbf{x}_{0:t}|\mathbf{z}_{1:t})$ after receiving the last measurement. However, for object tracking we are interested in the sequence of objects and their states at time t which is given by the marginal posterior state density $p(\mathbf{x}_t|\mathbf{z}_{1:t})$. This can be achieved by integrating over all previous states

$$p(\mathbf{x}_t|\mathbf{z}_{1:t}) = \int_{\mathbf{x}_{t-1}} \dots \int_{\mathbf{x}_0} p(\mathbf{x}_{0:t}|\mathbf{z}_{1:t})\, d\,\mathbf{x}_0 \dots d\,\mathbf{x}_{t-1}, \tag{3.9}$$

leading to the final recursive Bayesian filtering solution

$$p(\mathbf{x}_t|\mathbf{z}_{1:t}) = \frac{p(\mathbf{z}_t|\mathbf{x}_t)}{p(\mathbf{z}_t|\mathbf{z}_{1:t-1})} \underbrace{\int_{\mathbf{x}_{t-1}} p(\mathbf{x}_t|\mathbf{x}_{t-1})\, p(\mathbf{x}_{t-1}|\mathbf{z}_{1:t-1})\, d\,\mathbf{x}_{t-1}}_{\text{Chapman-Kolmogorov}} . \tag{3.10}$$

The detailed integration steps can be looked up in related literature, *e.g.*, [26]. This final formula provides the fundamental framework of Bayesian filtering containing a prediction step, represented by the Chapman-Kolmogorov equation, and an update step consisting of a measurement likelihood and a normalization factor.

Within the prediction step, the product between posterior state *PDF* $p(\mathbf{x}_{t-1}|\mathbf{z}_{1:t-1})$ of time $t-1$ and transition density $p(\mathbf{x}_t|\mathbf{x}_{t-1})$ gets integrated over all states $\mathbf{x}_{t-1}$ of the previous stage. This results in a predicted prior state *PDF* $p(\mathbf{x}_t|\mathbf{z}_{1:t-1})$ including the current knowledge of $\mathbf{x}_t$ and all previous measurements up to time $t-1$. The remaining part of the formula describes the update or correction step. Here, the measurement $\mathbf{z}_t$ at time t is used within the measurement likelihood function $p(\mathbf{z}_t|\mathbf{x}_t)$ to correct the prior. Normalization with $p(\mathbf{z}_t|\mathbf{z}_{1:t-1})$ then yields the posterior state *PDF* $p(\mathbf{x}_t|\mathbf{z}_{1:t})$. Finally, the mean values and their corresponding covariance matrices – describing the estimation accuracy – are computed from these densities.

3.1.2 Stochastic State Space Representation

Before the theory can be applied to object tracking, it is necessary to model the system dynamics. This includes a specification for the transition density $p(\mathbf{x}_t|\mathbf{x}_{t-1})$, the measurement likelihood $p(\mathbf{z}_t|\mathbf{x}_t)$ and the initial prior state $p(\mathbf{x}_0)$. An elegant way to do so is the definition of a discrete-time stochastic state space model with an additive noise assumption represented by the dynamic equation

$$\mathbf{x}_t = f(\mathbf{x}_{t-1}, \mathbf{u}_{t-1}) + \mathbf{w}_t, \tag{3.11}$$

and the measurement equation

$$\mathbf{z}_t = h(\mathbf{x}_t) + \mathbf{v}_t, \tag{3.12}$$

with system function $f(\cdot)$ and measurement function $h(\cdot)$. Both functions are in general non-linear. Furthermore, the input vector, measurement vector and state vector are denoted $\mathbf{u}_t \in \mathbb{R}^p$, $\mathbf{z}_t \in \mathbb{R}^q$ and $\mathbf{x}_t \in \mathbb{R}^n$, respectively. Noise sequences are represented by the process noise vector $\mathbf{w}_t \in \mathbb{R}^n$ with its corresponding covariance matrix $\mathbf{Q}_t \in \mathbb{R}^{n\times n}$ and the measurement noise vector $\mathbf{v}_t \in \mathbb{R}^q$ with its covariance matrix $\mathbf{R}_t \in \mathbb{R}^{q\times q}$. They are considered as identically independently distributed (i.i.d.), zero-mean, white Gaussian noise sequences.

Under the additive noise assumption, the system dynamics equation (3.11) yields the transition density function by transforming random variables [26]

$$p(\mathbf{x}_t|\mathbf{x}_{t-1}) = p_{w_t}(\mathbf{x}_t - f(\mathbf{x}_{t-1}, \mathbf{u}_{t-1})), \tag{3.13}$$

where $p_{w_t}(\cdot)$ is the *PDF* of the process noise. Similarly, the measurement Equation (3.12) leads to the likelihood function

$$p(\mathbf{z}_t|\mathbf{x}_t) = p_{v_t}(\mathbf{z}_t - h(\mathbf{x}_t)), \tag{3.14}$$

with $p_{v_t}(\cdot)$ denoting the *PDF* of the measurement noise.

The discrete-time stochastic state space model in Equations (3.11) and (3.12) can be expressed in a linear time-invariant form

$$\mathbf{x}_t = \mathbf{A}\mathbf{x}_{t-1} + \mathbf{B}\mathbf{u}_{t-1} + \mathbf{w}_t, \tag{3.15a}$$
$$\mathbf{z}_t = \mathbf{H}\mathbf{x}_t + \mathbf{v}_t, \tag{3.15b}$$

where $\mathbf{A} \in \mathbb{R}^{n\times n}$ denotes the system matrix, $\mathbf{B} \in \mathbb{R}^{n\times p}$ denotes the input matrix and the measurement matrix is given by $\mathbf{H} \in \mathbb{R}^{q\times n}$. With the already mentioned Gaussian noise assumption and the system dynamics equation (3.15a), the transition density is expressed by

$$\begin{aligned} p(\mathbf{x}_t|\mathbf{x}_{t-1}) &= p(\mathbf{w}_t) \\ &= p(\mathbf{x}_t - (\mathbf{A}\mathbf{x}_{t-1} + \mathbf{B}\mathbf{u}_{t-1})) \\ &= \mathcal{N}(\mathbf{x}_t - (\mathbf{A}\mathbf{x}_{t-1} + \mathbf{B}\mathbf{u}_{t-1});\, 0, \mathbf{Q}_t) \\ &= \mathcal{N}(\mathbf{x}_t;\, \mathbf{A}\mathbf{x}_{t-1} + \mathbf{B}\mathbf{u}_{t-1}, \mathbf{Q}_t), \end{aligned} \tag{3.16}$$

while the measurement Equation (3.15b) is used to represent the measurement likelihood

$$\begin{aligned} p(\mathbf{z}_t|\mathbf{x}_t) &= p(\mathbf{v}_t) \\ &= p(\mathbf{z}_t - \mathbf{H}\mathbf{x}_t) \\ &= \mathcal{N}(\mathbf{z}_t - \mathbf{H}\mathbf{x}_t;\, 0, \mathbf{R}_t) \\ &= \mathcal{N}(\mathbf{z}_t;\, \mathbf{H}\mathbf{x}_t, \mathbf{R}_t), \end{aligned} \tag{3.17}$$

where Gaussian densities are expressed in the short notation

$$\mathcal{N}(\mathbf{z};\, \boldsymbol{\mu}, \boldsymbol{\Sigma}) = \frac{1}{\sqrt{\det(2\pi\boldsymbol{\Sigma})}} \exp\left(-\frac{1}{2}(\mathbf{z}-\boldsymbol{\mu})^T\boldsymbol{\Sigma}^{-1}(\mathbf{z}-\boldsymbol{\mu})\right), \tag{3.18}$$

parametrized with mean vector $\boldsymbol{\mu}$ and covariance matrix $\boldsymbol{\Sigma}$.

3.2 Filter Solutions for the Recursive Problem

Within this thesis we make use of two different implementations which provide a recursive solution to Bayesian filtering (recall Equation (3.10)), the Kalman Filter (KF) [72] and the Unscented Kalman Filter (UKF) [69]. While the former is designed to solve linear Gaussian problems exact in a closed form, the latter tries to approximate non-linear behavior. An extension to the optimal Bayesian filter is given by the Interacting Multiple Model (IMM) [18] filter, which approximates the posterior distribution by a Gaussian mixture and therefore, allows to use different motion models. The next sections explain the main idea behind these filters and present the final filter equations.

3.2.1 Kalman Filter

The Kalman Filter (*KF*) [72] solves the recursive filtering problem in Equation (3.10) exactly for linear Gaussian systems. In such systems, the object dynamics and measurement equations have to be linear. Furthermore, white, uncorrelated Gaussian process and measurement noise with zero mean is required. Finally, the object state posterior density for the last time step $t-1$ has to be Gaussian. If these assumptions are fulfilled, all resulting state densities are also Gaussian and can therefore be described by their mean and covariance.

Because most tracking systems do not have knowledge about control inputs for tracked objects, we can simplify Equation (3.15) by setting the control parameter $\mathbf{u}_{t-1}$ to zero. Additionally, we rename the system matrix according to common literature in control theory and tracking as

$$\begin{aligned} \mathbf{x}_t &= \mathbf{F}\mathbf{x}_{t-1} + \mathbf{w}_t, \\ \mathbf{z}_t &= \mathbf{H}\mathbf{x}_t + \mathbf{v}_t, \end{aligned} \tag{3.19}$$

with the system matrix $\mathbf{F} \in \mathbb{R}^{n \times n}$, the measurement matrix $\mathbf{H} \in \mathbb{R}^{q \times n}$, the state vector $\mathbf{x}_t \in \mathbb{R}^n$, the measurement vector $\mathbf{z}_t \in \mathbb{R}^q$, the process noise vector $\mathbf{w}_t \in \mathbb{R}^n$ with corresponding covariance matrix $\mathbf{Q}_t \in \mathbb{R}^{n \times n}$ and measurement noise vector $\mathbf{v}_t \in \mathbb{R}^q$ with corresponding covariance matrix $\mathbf{R}_t \in \mathbb{R}^{q \times q}$.

According to Challa et al. [26], we write the Kalman filter equations using the linear system in Equation (3.19) beginning with the prediction step

$$\begin{aligned} \hat{\mathbf{x}}_{t|t-1} &= \mathbf{F}\hat{\mathbf{x}}_{t-1|t-1}, \\ \mathbf{P}_{t|t-1} &= \mathbf{F}\mathbf{P}_{t-1|t-1}\mathbf{F}^T + \mathbf{Q}_t, \end{aligned} \tag{3.20}$$

where the predicted mean $\hat{\mathbf{x}}_{t|t-1}$ and covariance matrix $\mathbf{P}_{t|t-1}$ are computed using the posterior state estimate $\hat{\mathbf{x}}_{t-1|t-1}$ and covariance matrix $\mathbf{P}_{t-1|t-1}$ of the last time-step.

Afterwards the first part of the update step (measurement prediction)

$$\begin{aligned} \hat{\mathbf{z}}_{t|t-1} &= \mathbf{H}\mathbf{x}_{t|t-1}, \\ \mathbf{S}_t &= \mathbf{H}\mathbf{P}_{t|t-1}\mathbf{H}^T + \mathbf{R}_t, \\ \mathbf{K}_t &= \mathbf{P}_{t|t-1}\mathbf{H}^T\mathbf{S}_t^{-1}, \end{aligned} \tag{3.21}$$

calculates the predicted measurement $\hat{\mathbf{z}}_{t|t-1}$, innovation covariance matrix $\mathbf{S}_t \in \mathbb{R}^{q\times q}$ and the Kalman gain $\mathbf{K}_t \in \mathbb{R}^{n\times q}$. Finally, these values are used to evaluate the posterior mean $\hat{\mathbf{x}}_{t|t}$ and the corresponding covariance matrix $\mathbf{P}_{t|t}$ with

$$\begin{aligned} \hat{\mathbf{x}}_{t|t} &= \hat{\mathbf{x}}_{t|t-1} + \mathbf{K}_t(\mathbf{z}_t - \hat{\mathbf{z}}_{t|t-1}), \\ \mathbf{P}_{t|t} &= \mathbf{P}_{t|t-1} - \mathbf{K}_t\mathbf{H}_t\mathbf{P}_{t|t-1}, \\ &= \mathbf{P}_{t|t-1} - \mathbf{K}_t\mathbf{S}_t\mathbf{K}_t^T. \end{aligned} \tag{3.22}$$

Basically, Equation (3.22) shows the main behavior of the *KF*. Predictions and measurements are balanced by the Kalman gain. If the prediction is good compared to the measurement uncertainty, the Kalman gain is small and thus, takes the prediction more into account and vice versa. The uncertainties of both, predictions and measurements, are influenced by noise, *i.e.* the measurement noise covariance matrix $\mathbf{R}_t$ or process noise covariance matrix $\mathbf{Q}_t$.

We explain the *KF* equations by already derived formulas and assumptions made at the beginning of this section. Equation (3.20) exactly solves the prediction part of the final Bayesian filtering solution in Equation (3.10), representing the Chapman-Kolmogorov equation. The predicted prior state *PDF* $p(\mathbf{x}_t|\mathbf{z}_{1:t-1})$ is given by

$$p(\mathbf{x}_t|\mathbf{z}_{1:t-1}) = \int_{\mathbf{x}_{t-1}} p(\mathbf{x}_t|\mathbf{x}_{t-1})\, p(\mathbf{x}_{t-1}|\mathbf{z}_{1:t-1})\, d\,\mathbf{x}_{t-1}, \tag{3.23}$$

with transition density $p(\mathbf{x}_t|\mathbf{x}_{t-1})$ as in Equation (3.16), but without control input

$$p(\mathbf{x}_t|\mathbf{x}_{t-1}) = \mathcal{N}(\mathbf{x}_t;\, \mathbf{F}\mathbf{x}_{t-1|t-1}, \mathbf{Q}_t), \tag{3.24}$$

and posterior state *PDF* $p(\mathbf{x}_{t-1}|\mathbf{z}_{1:t-1})$ of the previous time step

$$p(\mathbf{x}_{t-1}|\mathbf{z}_{1:t-1}) = \mathcal{N}(\mathbf{x}_{t-1};\, \hat{\mathbf{x}}_{t-1|t-1}, \mathbf{P}_{t-1|t-1}), \tag{3.25}$$

resulting in

$$p(\mathbf{x}_t|\mathbf{z}_{1:t-1}) = \mathcal{N}(\mathbf{x}_t;\, \hat{\mathbf{x}}_{t|t-1}, \mathbf{P}_{t|t-1}). \tag{3.26}$$

The theorem used to get Equation (3.26) can be looked up in [26] and was proven first by Ho and Lee [61]. As required for linear systems, the predicted density is Gaussian, as well as the posterior density of the previous state.

The measurement prediction step, solved by Equation (3.21), uses the same theorem for the normalization factor $p(\mathbf{z}_t|\mathbf{z}_{1:t-1})$ in Equation (3.10) written as

$$p(\mathbf{z}_t|\mathbf{z}_{1:t-1}) = \int_{\mathbf{x}_t} p(\mathbf{z}_t|\mathbf{x}_t)\, p(\mathbf{x}_t|\mathbf{z}_{1:t-1})\, d\,\mathbf{x}_t, \tag{3.27}$$

with the likelihood $p(\mathbf{z}_t|\mathbf{x}_t)$ formulated as Gaussian density

$$p(\mathbf{z}_t|\mathbf{x}_t) = \mathcal{N}(\mathbf{z}_t;\, \mathbf{H}\mathbf{x}_t, \mathbf{R}_t), \tag{3.28}$$

and the predicted state density in Equation (3.26), resulting in

$$p(\mathbf{z}_t|\mathbf{z}_{1:t-1}) = \mathcal{N}(\mathbf{z}_t;\, \hat{\mathbf{z}}_{t|t-1}, \mathbf{S}_t), \tag{3.29}$$

which represents the predicted measurement density.

Finally, we are able to substitute all the densities into Equation (3.10) and get the final update equation

$$p(\mathbf{x}_t|\mathbf{z}_{1:t}) = \frac{\mathcal{N}(\mathbf{z}_t;\, \mathbf{H}\mathbf{x}_t, \mathbf{R}_t)\, \mathcal{N}(\mathbf{x}_t;\, \hat{\mathbf{x}}_{t|t-1}, \mathbf{P}_{t|t-1})}{\mathcal{N}(\mathbf{z}_t;\, \mathbf{H}\hat{\mathbf{x}}_{t|t-1}, \mathbf{S}_t)}, \tag{3.30}$$

yielding the posterior state *PDF*

$$p(\mathbf{x}_t|\mathbf{z}_{1:t}) = \mathcal{N}(\mathbf{x}_t;\, \hat{\mathbf{x}}_{t|t}, \mathbf{P}_{t|t}), \tag{3.31}$$

by using the same theorem as in both steps before.

3.2.2 Unscented Kalman Filter

A recursive filtering solution for systems with non-linear dynamics, as it is crucial for most tracking applications, is the Unscented Kalman Filter (*UKF*) [69]. It was designed to overcome some problems of the widely used and well-known Extended Kalman Filter (EKF), which linearizes all non-linear models with a first-order Taylor expansion and applies the standard *KF* afterwards. According to Julier and Uhlmann [69], linearizion can produce highly unstable filters under special conditions. Furthermore, linearizion often introduces large errors in the posterior mean and covariance of the transfered state density [141, 147]. Another issue is the derivation of Jacobian matrices for complex systems, which is mostly nontrivial and thus, results in implementation difficulties. Because we do not cover the *EKF* within this thesis, we refer the interested reader to [26, 136] for further details.

The main idea of the *UKF* is to propagate a fixed number of appropriately chosen weighted sample points – also called sigma points – through a non-linear function by using the Unscented Transformation (UT). This process does not need an analytical derivation of dynamic and measurement equations. Except these functions, the filter does not need to know anything about the non-liner model, which therefore can be treated as a "black

box". After passing the transformation, the sample points include the posterior mean and covariance exactly to the third-order Taylor series expansion for any nonlinearity within the same order of computational complexity as the *EKF* [147]. An accuracy comparison between *UKF* and *EKF* is shown in Figure 3.1, where the drawbacks of linearizion are clearly visible in terms of a huge bias. If some weights inside the *UT* are smaller than zero, which can happen in special cases, the covariance matrix after transformation is possibly negative semidefinite. This leads to problems, *e.g.* while calculating sigma points. To ensure a positive semidefinite covariance matrix, the scaled *UT* [68] is commonly used.

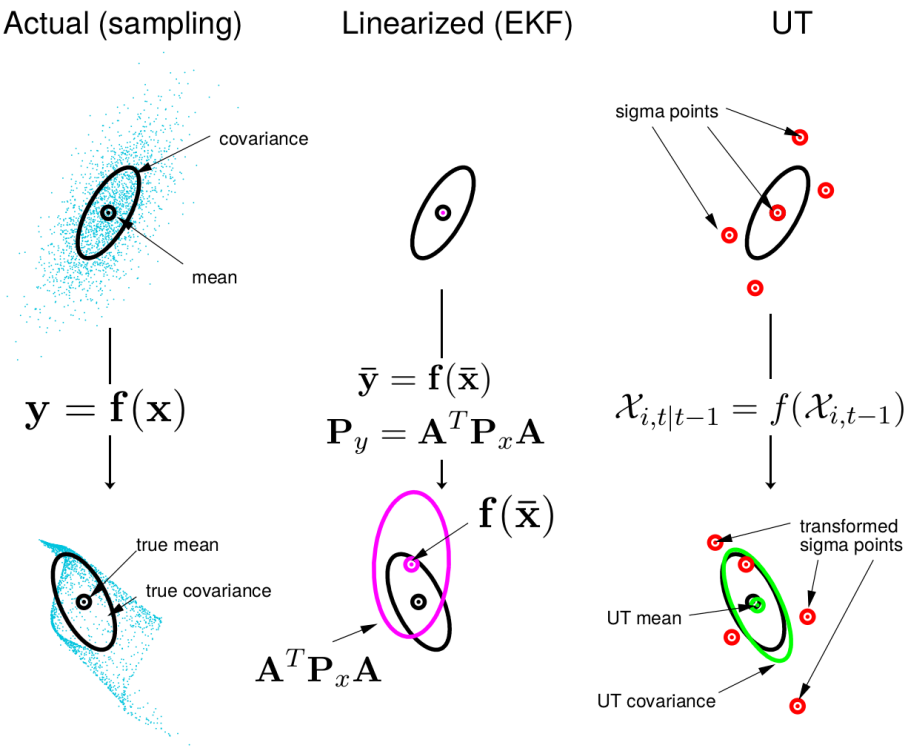


Figure 3.1: Visualization of mean and covariance for linearized function (*EKF*) and scaled *UT* (*UKF*) compared to the true representation. A cloud of 5000 samples drawn from a Gaussian distribution gets propagated through an arbitrary nonlinear function. Left, the true value calculated by a Monte Carlo approach is shown. The middle path shows the same non-linear function approximated by an *EKF* with first-order Taylor expansion. The sigma point propagation through the scaled *UT* is shown on the right. Image adapted from [147].

For convenience and simpler representation of the filter equations, we make the additive noise assumption at this point. However, in general it is not necessary for the *UKF*. For the scaled *UT*, we first need to determine $2n+1$ sigma points $\mathcal{X}_{i,t-1} \in \mathbb{R}^n$ with $i \in \{0, \ldots, 2n\}$ for n state variables, weights $\mathbf{w}^{(m)} \in \mathbb{R}^n$ for mean $\hat{\mathbf{x}}_{t-1|t-1}$, and weights $\mathbf{w}^{(c)} \in \mathbb{R}^n$ for covariance matrix $\hat{\mathbf{P}}_{t-1|t-1}$. The weights for mean $\mathbf{w}^{(m)}$ and covariance matrix $\mathbf{w}^{(c)}$ of the

last time step are calculated as

$$\lambda = \alpha^2(n+\kappa) - n, \tag{3.32a}$$

$$w_0^{(m)} = \frac{\lambda}{n+\lambda}, \tag{3.32b}$$

$$w_0^{(c)} = \frac{\lambda}{n+\lambda} + (1-\alpha^2+\beta), \tag{3.32c}$$

$$w_i^{(m)} = w_i^{(c)} = \frac{1}{2(n+\lambda)}, \quad \forall i \in \{1,\dots,2n\}, \tag{3.32d}$$

where n denotes the state space dimension and λ is a scaling parameter. Here, α determines the spread of the sigma points around the mean, κ is a secondary scaling parameter and β contains prior knowledge of the state distribution. To avoid sampling non-local effects under strong nonlinearities, the parameter α should be a small number in $0 \leq \alpha \leq 1$. Furthermore, positive semi-definiteness can be guaranteed by choosing the parameter $\kappa \geq 0$. A good choice for state estimation problems is $\kappa = 0$. Finally, for Gaussian distributions $\beta = 2$ is optimal, otherwise it should be non-negative. Following Julier [68], scaling parameters are used to sample sigma points

$$\mathcal{X}_{0,t-1} = \hat{\mathbf{x}}_{t-1|t-1}, \tag{3.33a}$$

$$\mathcal{X}_{i,t-1} = \hat{\mathbf{x}}_{t-1|t-1} + \left(\sqrt{(n+\lambda)\mathbf{P}_{t-1|t-1}}\right)_i, \qquad \forall i \in \{1,\dots,n\}, \tag{3.33b}$$

$$\mathcal{X}_{i,t-1} = \hat{\mathbf{x}}_{t-1|t-1} - \left(\sqrt{(n+\lambda)\mathbf{P}_{t-1|t-1}}\right)_{i-n}, \qquad \forall i \in \{n+1,\dots,2n\}, \tag{3.33c}$$

with $\left(\sqrt{\dots}\right)_i$ denoting the i^{th} column of the matrix square root of $(n+\lambda)\mathbf{P}_{t-1|t-1}$, from the previous belief containing mean $\hat{\mathbf{x}}_{t-1|t-1}$ and covariance $\mathbf{P}_{t-1|t-1}$.

The prediction step is then performed by propagating the sampled sigma points $\mathcal{X}_{i,t-1}$ calculated in Equation (3.33) through the non-linear function $f(\cdot)$ of the dynamic model

$$\mathcal{X}_{i,t|t-1} = f(\mathcal{X}_{i,t-1}) \quad \forall i \in \{0,\dots,2n\}, \tag{3.34}$$

followed by using these transformed sigma points $\mathcal{X}_{i,t|t-1}$ to calculate the predicted mean $\hat{\mathbf{x}}_{t|t-1}$ and corresponding covariance $\mathbf{P}_{t|t-1}$ as

$$\hat{\mathbf{x}}_{t|t-1} = \sum_{i=0}^{2n} \mathbf{w}_i^{(m)} \mathcal{X}_{i,t|t-1}, \tag{3.35a}$$

$$\mathbf{P}_{t|t-1} = \sum_{i=0}^{2n} \mathbf{w}_i^{(c)} \left(\mathcal{X}_{i,t|t-1} - \hat{\mathbf{x}}_{t|t-1}\right)\left(\mathcal{X}_{i,t|t-1} - \hat{\mathbf{x}}_{t|t-1}\right)^T + \mathbf{Q}_t, \tag{3.35b}$$

with process noise covariance matrix $\mathbf{Q}_t \in \mathbb{R}^{n\times n}$ and weights for mean and covariance denoted $\mathbf{w}_i^{(m)}$ and $\mathbf{w}_i^{(c)}$, respectively.

The update step requires a new set of $2n+1$ sigma points $\mathcal{X}_{i,t} \in \mathbb{R}^n$ with $i \in \{0, \ldots, 2n\}$ for n state variables and weights as in Equation (3.32) for predicted mean $\hat{\mathbf{x}}_{t|t-1}$ and covariance matrix $\hat{\mathbf{P}}_{t|t-1}$. Again following Julier [68], scaling parameters are used to sample sigma points

$$\mathcal{X}_{0,t} = \hat{\mathbf{x}}_{t|t-1}, \tag{3.36a}$$

$$\mathcal{X}_{i,t} = \hat{\mathbf{x}}_{t|t-1} + \left(\sqrt{(n+\lambda)\mathbf{P}_{t|t-1}}\right)_i, \qquad \forall i \in \{1, \ldots, n\}, \tag{3.36b}$$

$$\mathcal{X}_{i,t} = \hat{\mathbf{x}}_{t|t-1} - \left(\sqrt{(n+\lambda)\mathbf{P}_{t|t-1}}\right)_{i-n}, \qquad \forall i \in \{n+1, \ldots, 2n\}, \tag{3.36c}$$

with $\left(\sqrt{\ldots}\right)_i$ denoting the i^{th} column of the matrix square root of $(n+\lambda)\mathbf{P}_{t|t-1}$, from the predicted mean $\hat{\mathbf{x}}_{t|t-1}$ and covariance matrix $\mathbf{P}_{t|t-1}$.

The measurement prediction step is then performed by propagating the sampled sigma points $\mathcal{X}_{i,t}$ calculated in Equation (3.36) through the probably non-linear function $h(\cdot)$ of the measurement model

$$\mathcal{Z}_{i,t|t-1} = h(\mathcal{X}_{i,t}) \quad \forall i \in \{0, \ldots, 2n\}, \tag{3.37}$$

followed by using these sigma points $\mathcal{Z}_{i,t|t-1}$ to calculate the predicted a priori measurement $\hat{\mathbf{z}}_{t|t-1}$ with corresponding innovation covariance matrix $\mathbf{S}_t$ and cross covariance $\mathbf{C}_t$ as

$$\hat{\mathbf{z}}_{t|t-1} = \sum_{i=0}^{2n} \mathbf{w}_i^{(m)} \mathcal{Z}_{i,t|t-1}, \tag{3.38a}$$

$$\mathbf{S}_t = \sum_{i=0}^{2n} \mathbf{w}_i^{(c)} \left(\mathcal{Z}_{i,t|t-1} - \hat{\mathbf{z}}_{t|t-1}\right)\left(\mathcal{Z}_{i,t|t-1} - \hat{\mathbf{z}}_{t|t-1}\right)^T + \mathbf{R}_t, \tag{3.38b}$$

$$\mathbf{C}_t = \sum_{i=0}^{2n} \mathbf{w}_i^{(c)} \left(\mathcal{X}_{i,t|t-1} - \hat{\mathbf{x}}_{t|t-1}\right)\left(\mathcal{Z}_{i,t|t-1} - \hat{\mathbf{z}}_{t|t-1}\right)^T, \tag{3.38c}$$

with measurement noise covariance matrix $\mathbf{R}_t \in \mathbb{R}^{q \times q}$ and weights for predicted measurements and innovation covariance matrix denoted $\mathbf{w}_i^{(m)}$ and $\mathbf{w}_i^{(c)}$, respectively.

Finally, we compute the posterior mean $\hat{\mathbf{x}}_{t|t}$ and covariance matrix $\mathbf{P}_{t|t}$ as

$$\hat{\mathbf{x}}_{t|t} = \hat{\mathbf{x}}_{t|t-1} + \mathbf{C}_t \mathbf{S}_t^{-1}(\mathbf{z}_t - \hat{\mathbf{z}}_{t|t-1}), \tag{3.39a}$$

$$\mathbf{P}_{t|t} = \mathbf{P}_{t|t-1} - \mathbf{C}_t \mathbf{S}_t^{-1} \mathbf{C}_t^T, \tag{3.39b}$$

given the current measurement $\mathbf{z}_t$. The correction step is equal to that of the linear *KF* in Equation (3.22) by formulating the Kalman gain $\mathbf{K}_t$ as

$$\mathbf{K}_t = \mathbf{C}_t \mathbf{S}_t^{-1}. \tag{3.40}$$

Note that for linear Gaussian systems the *UKF* delivers the same result as the standard *KF*. For further details, *e.g.* the filter derivation, we refer the interested reader to [26, 69, 141].

3.2.3 Interacting Multiple Model (IMM)

Tracking multiple maneuvering targets in clutter is a challenging task for various reasons. Especially, when the objects of interest do not follow the same motion pattern over the whole time, as it is mostly the case, *e.g.* for cars, cyclists or pedestrians. Traffic participants on a road crossing are a good example. They are able to move straight or perform a coordinated turn, independent from their previous behavior. State estimation filters like the *KF* or *UKF* use a linear/non-linear system function, also called motion model in terms of object tracking. Such motion models, *e.g.* Constant Velocity (CV), Constant Acceleration (CA) or Constant Turn-Rate Velocity (CTRV), aim to predict future states which are used in a following correction step inside the filter. However, predictions are wrong if the motion model does not fit the real behavior of the targets. Thus, also measurement-to-target association might fail, if False Positive (FP) detections caused by clutter are present or trajectories are crossing.

The Interacting Multiple Model (*IMM*) [18] filter provides a solution to these issues. It is a traceable approximation to the intractable multiple model optimal Bayes filter [126], which is modeled as jump Markov non-linear system. Besides the states of a system, such a filter estimates mode probabilities, which defines how likely a motion model matches the system's behavior. Furthermore, the *IMM* filter is an excellent compromise in regards to computational complexity and performance [99].

The multiple model optimal Bayes filter and its approximation assumes a fixed set of r models $\mathcal{M} = \{M_j\}_{j=1}^{r}$. Similar to single model filters as in Equation (3.11) and (3.12), a non-linear stochastic state space model, again without control input, represents each model of the set

$$\mathbf{x}_t = f_j(\mathbf{x}_{t-1}) + \mathbf{w}_{j,t}, \tag{3.41a}$$

$$\mathbf{z}_t = h_j(\mathbf{x}_t) + \mathbf{v}_{j,t}, \tag{3.41b}$$

where $f(\cdot)_j$ denotes the non-linear system function and $h(\cdot)_j$ denotes the probably non-linear measurement function. Furthermore, the measurement vector and state vector are denoted $\mathbf{z}_t \in \mathbb{R}^q$ and $\mathbf{x}_t \in \mathbb{R}^n$, respectively. Noise sequences are represented by the process noise vector $\mathbf{w}_{j,t} \in \mathbb{R}^n$ with its corresponding covariance matrix $\mathbf{Q}_{j,t} \in \mathbb{R}^{n \times n}$ and the measurement noise vector $\mathbf{v}_{j,t} \in \mathbb{R}^q$ with its covariance matrix $\mathbf{R}_{j,t} \in \mathbb{R}^{q \times q}$. They are considered as *i.i.d.*, zero-mean, white Gaussian noise sequences.

Model state transitions within these filters are modeled by a first-order Markov chain represented by a state transition probability matrix

$$\Pi = \begin{bmatrix} p_{1,1} & \cdots & p_{r,1} \\ \vdots & \ddots & \vdots \\ p_{1,r} & \cdots & p_{r,r} \end{bmatrix} \in \mathbb{R}^{r \times r}, \tag{3.42}$$

where $p_{i,j}$ denotes the probability of a state transition from model i to model j. Hence, the main diagonal $p_{i,i}$ contains the probabilities to stay in the same state.

In comparison with the recursive formulation of the optimal Bayesian filter in Equation (3.10), the posterior state *PDF* for multiple models needs an additional estimate for the motion mode. Therefore we infer the joint *PDF* as

$$p(\mathbf{x}_t, M_t | \mathbf{z}_{1:t}), \tag{3.43}$$

where all measurements $\mathbf{z}_{1:t}$ up to time t are given and the target state $\mathbf{x}_t$ holds continuous random variables in contrast to the motion mode M_t, which holds discrete values. Hence, a factorization of Equation (3.43) results in

$$p(\mathbf{x}_t, M_t | \mathbf{z}_{1:t}) = p(\mathbf{x}_t | M_t, \mathbf{z}_{1:t})\, p(M_t | \mathbf{z}_{1:t}), \tag{3.44}$$

with state inference step $p(\mathbf{x}_t | M_t, \mathbf{z}_{1:t})$ and mode inference step $p(M_t | \mathbf{z}_{1:t})$. We marginalize out M_t in Equation (3.44) to get

$$\begin{aligned} p(\mathbf{x}_t | \mathbf{z}_{1:t}) &= \sum_{j=1}^{r} p(\mathbf{x}_t, M_{j,t} | \mathbf{z}_{1:t}) \\ &= \sum_{j=1}^{r} p(\mathbf{x}_t | M_{j,t}, \mathbf{z}_{1:t}) \underbrace{p(M_{j,t} | \mathbf{z}_{1:t})}_{\mu_{j,t}}, \end{aligned} \tag{3.45}$$

where $p(\mathbf{x}_t | \mathbf{z}_{1:t})$ denotes the conditional state density, $M_{j,t}$ denotes the motion mode with $p(M_{j,t}) = p(M_t = j)$ and $\mu_{j,t}$ is the posterior mode probability.

Similar to the single model Equation (3.10), the recursive solution for the multiple model optimal Bayes filter can be obtained by applying Bayes' rule to get

$$p(\mathbf{x}_t | \mathbf{z}_{1:t}) = \sum_{j=1}^{r} \mu_{j,t} \frac{p(\mathbf{z}_t | \mathbf{x}_t, M_{j,t})}{p(\mathbf{z}_t | M_{j,t}, \mathbf{z}_{1:t-1})} \underbrace{\int_{\mathbf{x}_{t-1}} p(\mathbf{x}_t | \mathbf{x}_{t-1}, M_{j,t})\, p(\mathbf{x}_{t-1} | M_{j,t}, \mathbf{z}_{1:t-1})\, d\,\mathbf{x}_{t-1}}_{\text{Chapman-Kolmogorov}}, \tag{3.46}$$

with mode probability $\mu_{j,t}$ for the model j, the measurement likelihood $p(\mathbf{z}_t | \mathbf{x}_t, M_{j,t})$, the normalization $p(\mathbf{z}_t | M_{j,t}, \mathbf{z}_{1:t-1})$ and the prediction defined by the Chapman-Kolmogorov equation, which can be written as prior state *PDF* $p(\mathbf{x}_t | M_{j,t}, \mathbf{z}_{1:t-1})$.

Basically, a full cycle of the recursive *IMM* filter contains four steps: interaction, prediction, update and combination. In the following we explain the four steps along with theoretical details, but without full derivations. For a detailed derivation of the optimal multiple model Bayesian filter and missing steps leading to the *IMM* approximation, we refer the interested reader to [26, 126].

Interaction

The second term inside the integral of Equation (3.46), multiplied by the state transition density, can be factorized as

$$p(\mathbf{x}_{t-1}|M_{j,t}, \mathbf{z}_{1:t-1}) = \sum_{i=1}^{r} \mu_{i|j,t-1}\, p(\mathbf{x}_{t-1}|M_{i,t-1}, \mathbf{z}_{1:t-1}), \tag{3.47}$$

where both sides are approximated by a Gaussian *PDF*

$$p(\mathbf{x}_{t-1}|M_{j,t}, \mathbf{z}_{1:t-1}) \approx \mathcal{N}(\mathbf{x}_{t-1};\, \hat{\mathbf{x}}^{\star}_{j,t-1|t-1}, \mathbf{P}^{\star}_{j,t-1|t-1}), \tag{3.48a}$$

$$p(\mathbf{x}_{t-1}|M_{i,t-1}, \mathbf{z}_{1:t-1}) \approx \mathcal{N}(\mathbf{x}_{t-1};\, \hat{\mathbf{x}}_{i,t-1|t-1}, \mathbf{P}_{i,t-1|t-1}), \tag{3.48b}$$

where $\hat{\mathbf{x}}_{i,t-1|t-1}$ denotes the posterior state estimate and $\mathbf{P}_{i,t-1|t-1}$ denotes the covariance estimate of the previous stage for each filter i. The interaction step performs a probabilistic mixing with these filter states, namely

$$\hat{\mathbf{x}}^{\star}_{j,t-1|t-1} = \sum_{i=1}^{r} \mu_{i|j,t-1}\, \hat{\mathbf{x}}_{i,t-1|t-1}, \tag{3.49a}$$

$$\mathbf{P}^{\star}_{j,t-1|t-1} = \sum_{i=1}^{r} \mu_{i|j,t-1} \left(\mathbf{P}_{i,t-1|t-1} + (\hat{\mathbf{x}}_{i,t-1|t-1} - \hat{\mathbf{x}}^{\star}_{j,t-1|t-1})(\hat{\mathbf{x}}_{i,t-1|t-1} - \hat{\mathbf{x}}^{\star}_{j,t-1|t-1})^{T} \right), \tag{3.49b}$$

resulting in a single initial state $\hat{\mathbf{x}}^{\star}_{j,t-1|t-1}$ and covariance $\mathbf{P}^{\star}_{j,t-1|t-1}$ for each filter j [26, 126]. The mixing probabilities are denoted $\mu_{i|j,t-1}$ and calculated as

$$\mu_{i|j,t-1} = \frac{p_{ij}\, \mu_{i,t-1}}{\mu^{-}_{j,t}}, \tag{3.50}$$

where p_{ij} is the transition probability taken from Equation (3.42) at row i and column j denoting the probability for a model switch from i to j. Furthermore, $\mu^{-}_{j,t}$ is the predicted mode probability for filter j at the current time step

$$\mu^{-}_{j,t} = \sum_{i=1}^{r} p_{ij} \mu_{i,t-1}, \tag{3.51}$$

with p_{ij} from Equation (3.42) and the mode probability $\mu_{i,t-1}$ of the last time step. In summary, the previous filters with their mode probability and the transition probability directly influence the initial state of each filter.

Prediction and Update

Both, the state transition $p(\mathbf{x}_t|\mathbf{x}_{t-1}, M_{j,t})$ and the measurement likelihood $p(\mathbf{z}_t|\mathbf{x}_t, M_{j,t})$ in Equation (3.46), can be represented by the j^{th} stochastic state space model in Equation (3.41), similarly to that of a single model, *e.g.* for the *KF* in Equation (3.24) and (3.28). Therefore, we use the initial states $\hat{\mathbf{x}}^{\star}_{j,t-1|t-1}$ and covariances $\mathbf{P}^{\star}_{j,t-1|t-1}$ to perform the prediction step for each individual filter j with the *KF* in Section 3.2.1 or the *UKF* in Section 3.2.2. Hence, we obtain predicted states $\hat{\mathbf{x}}_{j,t|t-1}$ and corresponding covariance matrices $\mathbf{P}_{j,t|t-1}$. Additionally, this step yields predicted measurements $\hat{\mathbf{z}}_{j,t|t-1}$ and corresponding innovation covariance matrices $\mathbf{S}_{j,t}$. Finally, the filter-specific update performs a correction with the current measurement $\mathbf{z}_t$ and yields the posterior state $\hat{\mathbf{x}}_{j,t|t}$ and the covariance $\mathbf{P}_{j,t|t}$.

Another update is needed for the mode probabilities. Thus, we use the model likelihood $\mathcal{L}_{j,t}$ similar to Equation (3.29), representing how well the measurements $\mathbf{z}_t$ fits the model

$$\begin{aligned}\mathcal{L}_{j,t} &= \mathcal{N}(\mathbf{z}_t;\, \hat{\mathbf{z}}_{j,t|t-1}, \mathbf{S}_{j,t}) \\ &= \frac{1}{\sqrt{\det(2\pi\, \mathbf{S}_{j,t})}} \exp\left(-\frac{1}{2}(\mathbf{z}_t - \hat{\mathbf{z}}_{j,t|t-1})^T \mathbf{S}_{j,t}^{-1}(\mathbf{z}_t - \hat{\mathbf{z}}_{j,t|t-1})\right),\end{aligned} \tag{3.52}$$

to update the mode probabilities

$$\mu_{j,t} = \frac{\mathcal{L}_{j,t}\, \mu^{-}_{j,t}}{\sum_{i=1}^{r} \mathcal{L}_{i,t}\, \mu^{-}_{i,t}}, \tag{3.53}$$

where the likelihood of a model fitting the measurement influences the mode probability $\mu_{j,t}$ of the current time-step.

Combination

Finally, we obtain the posterior state $\hat{\mathbf{x}}_{t|t}$ and its covariance $\mathbf{P}_{t|t}$ by combining the output of each filter, weighted by the mode probability

$$\hat{\mathbf{x}}_t = \sum_{j=1}^{r} \mu_{j,t}\, \hat{\mathbf{x}}_{j,t|t}, \tag{3.54a}$$

$$\mathbf{P}_t = \sum_{j=1}^{r} \mu_{j,t|t} \left(\mathbf{P}_{j,t|t} + (\hat{\mathbf{x}}_{j,t|t} - \hat{\mathbf{x}}_{t|t})(\hat{\mathbf{x}}_{j,t|t} - \hat{\mathbf{x}}_{t|t})^T\right), \tag{3.54b}$$

which represents the final result, but is not part of the filter recursion itself.

3.3 Data Association

The *KF* in Section 3.2.1 for linear and the *UKF* in Section 3.2.2 for non-linear systems, are designed to estimate the object states based on noisy observations. However, they assume single tracks and one measurement at a time. In addition to measurement uncertainties caused by inaccurate sensors, Multiple Object Tracking (MOT) has to deal with measurement origin uncertainties due to False Positives (*FPs*) and multiple simultaneous tracks. Therefore, it is not clear from which track a measurement emerges.

Hence, the aim of Data Association (*DA*) is to correctly assign noisy measurements to active tracks in the presence of clutter, which is the most important and difficult problem to solve in *MOT* [161]. Figure 3.2 shows a basic recursive tracking structure combining state estimation and *DA* within one cycle. The system's motion model is responsible for the state prediction, whereas its measurement model yields predicted measurements for this predicted state. This makes both, observed and predicted measurements, comparable for the used *DA* method. Afterwards, assigned measurements are used for the state update. Besides state estimation and *DA*, the tracking filter needs a track management component to deal with measurements which are not assigned to existing trajectories and thus, can be used to start new tracks. Furthermore, it terminates old tracks without assigned observations over a certain time period.

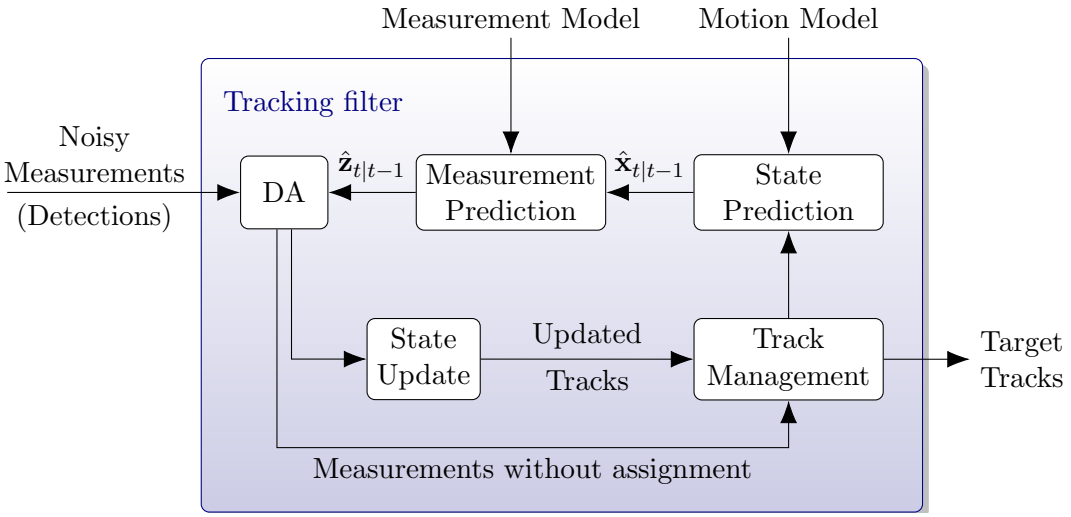


Figure 3.2: Tracking architecture. Image adapted from [126].

DA algorithms can be separated into probabilistic and deterministic methods. On the one hand, probabilistic *DA* approaches perform so-called soft assignments, where weighted measurements are used to update the predicted state. This leads to robust behavior, but the update is never completely correct. On the other hand, deterministic approaches are following a hard assignment strategy. In contrast to probabilistic methods, they assign

only a single measurement to each track. Thus, false assignments result in completely wrong updates.

Usually, a tracking algorithm is named after the *DA* method used in combination with the state estimation filter. An example for Single Object Tracking (SOT) is the Nearest Neighbour (NN) filter, *e.g.*, [6]. It assigns measurements and targets with the smallest Mahalanobis distance and uses a Bayesian filter for state estimation. The counterpart for *MOT* is the Global Nearest Neighbour (GNN) filter, using for example the standard *KF* in combination with the Hungarian method [106], *e.g.*, [30]. Taking all measurements and targets into account, it globally solves a linear assignment problem. Both, the *NN* filter and the *GNN* filter, are deterministic methods which work well in environments with nearly no clutter and require an accurate system model. Additionally, *NN* filters used within a *MOT* scenario require well-separated tracks. Nevertheless, most hard assignment methods are easy to implement and work reasonably well in many applications.

An appropriate choice to cope with heavily cluttered environments are probabilistic *DA* methods. The commonly used Probabilistic Data Association (*PDA*) filter [7, 9] performs a weighted update using all measurements in a certain region (validation gate) independently for all tracks. It can be seen as a soft assignment version of the *NN* filter. The Joint Probabilistic Data Association (*JPDA*) filter [7, 43] extends the *PDA* filter to deal with multiple objects. Hence, it performs a weighted update using all measurements in a certain region, taking all targets into account. Summarizing, it provides a global soft assignment between multiple targets and measurements.

Because of their robust behavior in cluttered environments, which is beneficial for tracking in autonomous driving scenarios, we further discuss filters using *PDA* and *JPDA*. Though *PDA* is normally used for *SOT*, we explain it first, because the implemented *JPDA* filter – designed for multiple interacting targets – differs only in the calculation of data association probabilities. However, we begin with the important concept of gating, which is necessary for both filters.

3.3.1 Gating

Gating is the process of determining which measurements are possible candidates for the state update of a target track. Therefore, a so called validation gate is defined, which contains these measurements. The reason for that is to avoid searching the whole measurement space. According to Bar-Shalom and Li [8], such validation gates mostly have an elliptical shape and are validated by some threshold

$$\mathcal{V}_t(\gamma) = \{\mathbf{z}_t : \left(\mathbf{z}_t - \hat{\mathbf{z}}_{t|t-1}\right)^T \mathbf{S}_t^{-1} \left(\mathbf{z}_t - \hat{\mathbf{z}}_{t|t-1}\right) \leq \gamma\}, \tag{3.55}$$

with the current measurement $\mathbf{z}_t$, the predicted measurement $\hat{\mathbf{z}}_{t|t-1}$, innovation covariance matrix $\mathbf{S}_t$ and the gate threshold γ. All measurements inside the validation region $\mathcal{V}_t$, with a normalized distance smaller than or equal the gating threshold, are later used for data association and subsequently for updating the state.

The gating threshold γ is calculated by using the inverse-χ^2 distribution Inv-$\mathcal{X}^2(P_G, q)$ with gating probability P_G and the measurement dimension $q = \dim(\mathbf{z}_t)$ [8]. Hence, P_G denotes the probability that a true measurement lies within the gate, if detected. The volume of the validation region in Equation (3.55) can be expressed by

$$V_t = V(q)\,|\gamma S_t|^{\frac{1}{2}}\,, \tag{3.56}$$

where

$$V(q) = \frac{\pi^{\frac{1}{2}q}}{\Gamma\left(\frac{1}{2}q + 1\right)}, \tag{3.57}$$

$V(q)$ denotes a hypersphere with dimension q and gamma function $\Gamma(\cdot)$. The resulting gated measurement set

$$\mathbf{Z}_{v,t} = \{\mathbf{z}_{m,t}\}_{m=1}^{N_v}, \tag{3.58}$$

contains all N_v valid measurements. All measurements which are not part of the gating region can be used to initiate potential new tracks.

3.3.2 Probabilistic Data Association Filter

The Probabilistic Data Association (*PDA*) filter [7, 9] performs a weighted update using association probabilities originated from assigning all measurements inside the validation gate to the tracked target. Hence, it works well in cluttered environments with single independent tracks only. To obtain a simple state-estimation scheme like in the *KF*, according to Bar-Shalom and Li [8], the following assumptions are made:

- There is only one target of interest present inside the validation gate, which is modeled by the dynamic Equation (3.11) and measurement Equation (3.12).
- The track is already initialized.
- The past information about the target is summarized approximately by a sufficient statistic in the form of the Gaussian prior

$$p(\mathbf{x}_t|\mathbf{z}_{1:t-1}) = \mathcal{N}(\mathbf{x}_t;\, \hat{\mathbf{x}}_{t|t-1}, \mathbf{P}_{t|t-1}). \tag{3.59}$$

- For each time step a validation region as in Section 3.3.1 is set up around the predicted measurement.
- Target detections occur independently over time with known probability P_D.
- At most one of the valid measurements originates from the target of interest, provided that the true target was detected and is valid. Measurements not originating from the target are assumed to be *FP* detections. They are modeled as *i.i.d.* variables with uniform spatial distribution in measurement space.

With these assumptions, the *PDA* filter can be described with four steps. First, we perform the prediction step like in the original *KF* (recall Section 3.2.1) or the *UKF* (recall Section 3.2.2). Second, the measurement validation is done with the gating procedure (as in Section 3.3.1). Third, we assume a diffuse clutter model which is suitable for heterogeneous clutter environments and results in the non-parametric *PDA* algorithm [7, 9, 26]. It is used to calculate the association probabilities. Finally, we use these association probabilities within the update step of the underlying filter.

In the following, we describe the last two steps in detail. According to [8], the association probabilities $\beta_{m,t}$ for the non-parametric *PDF* filter can be calculated by

$$\beta_{m,t} = \begin{cases} \dfrac{b}{b + \sum\limits_{i=1}^{N_v} e_i}, & m = 0 \\ \dfrac{e_m}{b + \sum\limits_{i=1}^{N_v} e_i}, & m \in \{1, \ldots, N_v\}, \end{cases} \tag{3.60}$$

with the non-normalized likelihood e_m of measurement $\mathbf{z}_{m,t}$ fitting the current predicted measurement

$$\begin{aligned} e_m &= \exp\left(-\frac{1}{2}\left(\mathbf{z}_{m,t} - \hat{\mathbf{z}}_{t|t-1}\right)^T \mathbf{S}_t^{-1} \left(\mathbf{z}_{m,t} - \hat{\mathbf{z}}_{t|t-1}\right)\right) \\ &= \exp\left(-\frac{1}{2}\boldsymbol{v}_{m,t}^T \, \mathbf{S}_t^{-1} \, \boldsymbol{v}_{m,t}\right), \end{aligned} \tag{3.61}$$

where $\boldsymbol{v}_{m,t}$ denotes the innovation of measurement m at time t. The constant parameter b is calculated with the gating parameters from Section 3.3.1 as

$$b = \left(\frac{2\pi}{\gamma}\right)^{\frac{q}{2}} \frac{N_v(1 - P_D P_G)}{V(q) P_D}. \tag{3.62}$$

With the obtained association probabilities $\beta_{m,t}$, we are able to perform the weighted state update of the *PDA* filter as

$$\hat{\mathbf{x}}_{t|t} = \hat{\mathbf{x}}_{t|t-1} + \mathbf{K}_t \boldsymbol{v}_t = \hat{\mathbf{x}}_{t|t-1} + \mathbf{K}_t \underbrace{\sum_{m=1}^{N_v} \beta_{m,t} \left(\mathbf{z}_{m,t} - \hat{\mathbf{z}}_{t|t-1}\right)}_{\boldsymbol{v}_t}, \tag{3.63}$$

where $\boldsymbol{v}_t$ denotes the combined innovation. The corresponding covariance update results in

$$\mathbf{P}_t = \beta_{0,t} \mathbf{P}_{t|t-1} + (1 - \beta_{0,t}) \, \mathbf{P}_{t|t}^c + \widetilde{\mathbf{P}}_t, \tag{3.64}$$

where $\beta_{0,t}$ denotes the case that no update happens at all because none of the measurements belong to the target. The correct measurement update within the covariance matrix

happens with probability $(1 - \beta_{0,t})$ and is defined by

$$\mathbf{P}_{t|t}^{c} = \left(\mathbf{P}_{t|t-1} - \mathbf{K}_t \mathbf{S}_t \mathbf{K}_t^T\right). \tag{3.65}$$

The spread of innovations term [8]

$$\widetilde{\mathbf{P}}_t = \mathbf{K}_t \left(\sum_{m=1}^{N_\mathrm{v}} \beta_{m,t} \boldsymbol{v}_{m,t} \boldsymbol{v}_{m,t}^T - \boldsymbol{v}_t \boldsymbol{v}_t^T \right) \mathbf{K}_t^T, \tag{3.66}$$

increases the innovation covariance of the update state, because it is not known which measurement is correct. $\tilde{\mathbf{P}}$ is a positive semidefinite matrix.

3.3.3 Joint Probabilistic Data Association Filter

In contrast to the *PDA* filter, the Joint Probabilistic Data Association (*JPDA*) filter [7, 43] for *MOT* problems considers all targets simultaneously within one time step. Thus, it provides a better solution to multiple interacting targets in a cluttered environment, as it is common in the domain of autonomous driving. According to Bar-Shalom and Li [8], the *JPDA* algorithm assumes that:

- The number of targets is known and they are already established.
- Measurements from one target are allowed to fall into another target's validation region.
- The past of each target is summarized by an approximate sufficient statistic in the form of state estimates. They are given by approximate conditional mean and corresponding covariance, which are assumed to be Gaussian distributed.
- Each target has a possibly different dynamical system, modeled by its system dynamics equation (3.11) and measurement equation (3.12).

If validation regions of simultaneous tracks do not intersect, the *DA* problem can be simplified and solved by the *PDA* filter. However, overlapping gating regions are common in cluttered environments with several maneuvering objects. Hence, considering the *JPDA* assumptions above, shared measurements between target gates are allowed. Such overlapping regions of multiple targets are called cluster, though also a single target can be seen as a cluster. Compared to the *PDA* filter discussed in Section 3.3.2, the *JPDA* filter differs only by calculating the association probabilities. For this, we have to evaluate the conditional probabilities of the joint events [43]

$$\theta^i = \bigcap_{j=1}^{N_v} \theta_{j,t_j}^i, \tag{3.67}$$

where θ^i_{j,t_j} denotes the event that measurement j originated from target t_j. Furthermore, t_j is the target's index to which measurement j is associated. Joint events where each measurement belongs to a different target are called *feasible events*. For the sake of readability, we drop the time index for the joint event and assume the event to take place at the current time step.

Following You et al. [161] and Fortmann et al. [43], we explain the modified probability calculation starting with the definition of a *validation matrix*. Such a matrix contains all possible associations of valid measurements to targets with overlapping gating regions or *FP* detections and is defined as

$$\mathbf{\Omega} = [\omega_{j,t}] = \begin{bmatrix} \omega_{1,0} & \cdots & \omega_{1,N_t} \\ \vdots & \ddots & \vdots \\ \omega_{N_v,0} & \cdots & \omega_{N_v,N_t} \end{bmatrix}, \tag{3.68}$$

where $\omega_{j,t}$ is a binary variable denoting the target-to-measurement relationship. All measurements $j \in \{1, \ldots, N_v\}$ within the validation gate of target $t \in \{0, \ldots, N_t\}$ are defined by $\omega_{j,t} = 1$. Otherwise, if measurement j is not in the validation gate of target t, $\omega_{j,t} = 0$. The first column of $\mathbf{\Omega}$ represents a dummy target. It models the possibility that measurements originate from *FP* detections and thus, all elements of the first column $\omega_{j,0}$ are set to 1.

The next step contains the derivation of association hypotheses by splitting up the validation matrix following two assumptions:

- Each measurement must have originated from a target or from *FP* detections. Hence, only one source per measurement is allowed.
- Only a single measurement can originate from each target.

Each hypothesis is represented by an association matrix $\hat{\mathbf{\Omega}}$

$$\hat{\mathbf{\Omega}}(\theta^i) = [\hat{\omega}^i_{j,t}(\theta^i)] = \begin{bmatrix} \hat{\omega}^i_{1,0} & \cdots & \hat{\omega}^i_{1,N_t} \\ \vdots & \ddots & \vdots \\ \hat{\omega}^i_{N_v,0} & \cdots & \hat{\omega}^i_{N_v,N_t} \end{bmatrix}, \tag{3.69}$$

with $\hat{\omega}^i_{j,t}(\theta^i)$ indicating whether measurement j originates from target t in the i^{th} feasible joint event θ^i. Thus, the binary indicator

$$\hat{\omega}^i_{j,t}\left(\theta^i\right) = \begin{cases} 1 & \text{if } \theta^i_{j,t_j} \subset \theta^i \\ 0 & \text{otherwise,} \end{cases} \tag{3.70}$$

is 1, if the measurement j belongs to target t and 0 if not. Finally, the previously made

assumptions require the association matrix to satisfy

$$\begin{aligned} &\sum_{t=0}^{N_t} \hat{\omega}_{j,t}^{i}(\theta^i) = 1, \quad j \in \{1, \dots N_v\} \quad \text{and} \\ &\sum_{j=0}^{N_v} \hat{\omega}_{j,t}^{i}(\theta^i) \leq 1, \quad t \in \{1, \dots N_t\}. \end{aligned} \tag{3.71}$$

Following this procedure, we obtain i feasible events with the corresponding association matrices $\hat{\mathbf{\Omega}}_i$. Figure 3.3 shows an example configuration of two clusters with predicted

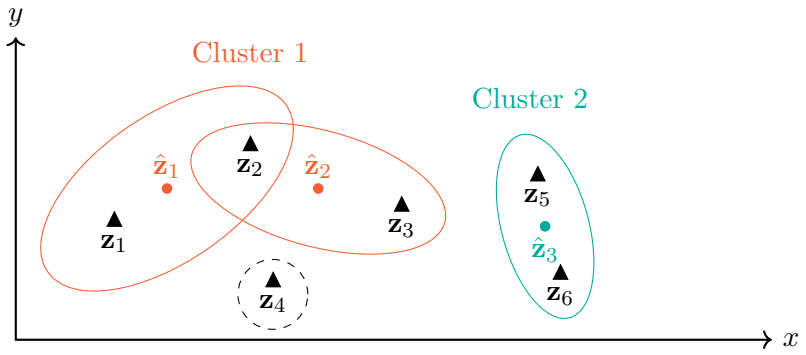


Figure 3.3: Example targets with corresponding gating regions for validation matrix generation. Cluster 1 consists of two targets (t_1 and t_2) with intersecting validation gates, whereas cluster 2 includes only target t_3 and thus, can be handled separately. Image adapted from [161].

measurements $\hat{\mathbf{z}}_{1,t|t-1}$, $\hat{\mathbf{z}}_{2,t|t-1}$ and $\hat{\mathbf{z}}_{3,t|t-1}$ of active targets and their overlapping validation gates including valid measurements. Note, also the non-overlapping gating region of target t_3 is considered to be a cluster. Additionally, there is one measurement – inside the dashed circle – which lies outside the validation gates of all currently known targets. Thus, it originates from clutter or is a new track which needs to be initialized within the next time step. Considering the first cluster, the resulting validation matrix

$$\mathbf{\Omega} = [\omega_{j,t}] = \begin{array}{c} \\ z_1 \\ z_2 \\ z_3 \end{array} \begin{array}{c} \begin{array}{ccc} t_0 & t_1 & t_2 \end{array} \\ \begin{bmatrix} 1 & 1 & 0 \\ 1 & 1 & 1 \\ 1 & 0 & 1 \end{bmatrix} \end{array}, \tag{3.72}$$

shows all possible target-to-measurement associations containing the validation gates of targets t_1 and t_2. Applying the splitting assumptions, we obtain the joint events and

corresponding association matrices:

$$\hat{\mathbf{\Omega}}\left[\theta_1\right] = \hat{\mathbf{\Omega}}_1 = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \end{bmatrix}, \quad \theta^1 = \theta^1_{1,0} \cap \theta^1_{2,0} \cap \theta^1_{3,0}, \tag{3.73}$$

$$\hat{\mathbf{\Omega}}\left[\theta_4\right] = \hat{\mathbf{\Omega}}_4 = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \quad \theta^4 = \theta^4_{1,1} \cap \theta^4_{2,0} \cap \theta^4_{3,2}. \tag{3.74}$$

In summary, we get eight hypotheses for the first cluster shown in the example configuration of Figure 3.3. The association matrices in Equations (3.73) and (3.74) are only two representative examples, starting with the joint event where all measurements originate from clutter. Association matrix $\hat{\mathbf{\Omega}}_4$ represents the case where both targets, t_1 and t_2 have associated measurements and only $\mathbf{z}_2$ results from clutter *i.e.* $\hat{\mathbf{\Omega}}_\mathbf{1}$.

Another way to obtain all possible combinations of measurements and targets is to visualize the hypotheses graph shown in Figure 3.4. We start a new tree for all possible

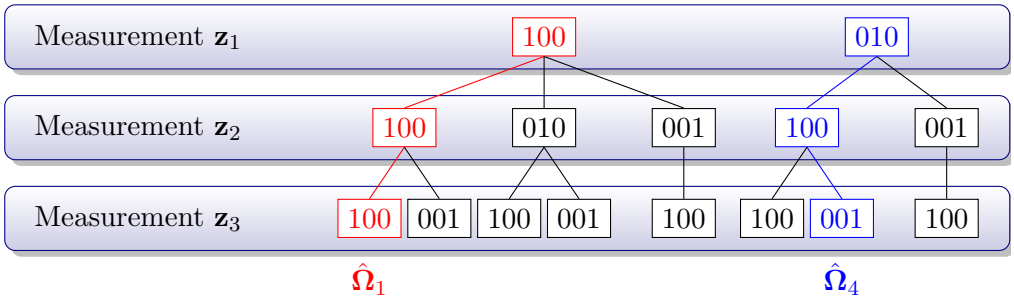


Figure 3.4: Hypothesis graph of cluster 1 depicted in Figure 3.3. Hypotheses represented by the association matrices $\hat{\mathbf{\Omega}}_1$ and $\hat{\mathbf{\Omega}}_4$ as noted in Equations (3.73) and (3.74) are marked in red and blue, respectively. The red hypothesis assumes for example that measurement $\mathbf{z}_1$ is assigned to target t_0 ([100]) which means clutter, whereas the blue hypothesis assigns the same measurement to target t_1 ([010]). Image adapted from [161].

association permutations of measurement $\mathbf{z}_1$, which in our case either originates from clutter or from target t_1. The second row or level two of our trees now contains all allowed allocations for measurement $\mathbf{z}_2$ following the splitting assumptions. Note that for the second tree, starting with the root node [010], there is no further association to target t_1 allowed, because it is already linked to measurement $\mathbf{z}_1$. Double assignments are only allowed between measurements and the dummy target t_0. Finally, the last row delivers all possible hypotheses for the observed cluster. Following the tree from the bottom to the top, we can stack together the complete association matrix for each hypothesis.

With this knowledge we are able to calculate the joint association probabilities. Therefore, two indicators in the form of binary variables are defined as

$$\tau_j\left(\theta^i\right) = \sum_{t=1}^{N_t} \hat{\omega}_{j,t}^i\left(\theta^i\right) = \begin{cases} 1 \\ 0, \end{cases} \tag{3.75}$$

where τ_j is the measurement association indicator and describes whether measurement j is associated with an active target and

$$\delta_t\left(\theta^i\right) = \sum_{j=1}^{N_v} \hat{\omega}_{j,t}^i\left(\theta^i\right) = \begin{cases} 1 \\ 0, \end{cases} \tag{3.76}$$

where δ_t is the target detection indicator and describes whether a target t has been detected or not. Hence, the number of false measurements $\phi(\theta^i)$, for the joint event θ^i results from Equation (3.75) as

$$\phi(\theta^i) = \sum_{j=1}^{N_v} \left(1 - \tau_j(\theta^i)\right). \tag{3.77}$$

Following the derivation in [161], we assume a diffuse prior for the number of false measurements and get the feasible association probabilities for the non-parametric *JPDA* [7] as

$$p(\theta^i|\mathbf{Z}) = \frac{1}{c}\frac{\phi(\theta^i)!}{V^{\phi(\theta^i)}} \prod_j (A_{j,t_j})^{\tau_j(\theta^i)} \prod_t (P_D)^{\delta_t(\theta^i)} (1 - P_D)^{1-\delta_t(\theta^i)}, \tag{3.78}$$

with the number of false measurements denoted $\phi(\theta^i)$, the volume of the validation region as in Equation (3.56) denoted $V^{\phi(\theta_i)}$, the detection probability P_D, the normalization constant c and both indicators for measurement association and target detection denoted $\tau_j(\theta_i)$ as in Equation (3.75) and $\delta_t(\theta_i)$ as in Equation (3.76), respectively. Association probabilities A_{j,t_j} of the joint events θ^i are given by Gaussian likelihoods

$$\begin{aligned} A_{j,t_j} &= \mathcal{N}(\mathbf{z}_j;\, \hat{\mathbf{z}}_{t_j,t|t-1}, \mathbf{S}_{t_j,t}) \\ &= \frac{1}{\sqrt{2\pi\ \det(\mathbf{S}_{t_j,j})}} \exp\left(-\frac{1}{2}(\mathbf{z}_j - \hat{\mathbf{z}}_{t_j,t|t-1})\, \mathbf{S}_{t_j,j}^{-1}\, (\mathbf{z}_j - \hat{\mathbf{z}}_{t_j,t|t-1})\right), \end{aligned} \tag{3.79}$$

where measurement j is associated to target t_j and the subscript t denotes the time. Considering the feasible joint event θ^4 with the association matrix of our example in Equation (3.74), the resulting association probability is calculated as

$$p(\theta^4|\mathbf{Z}_t) = \frac{1}{V} P_D^2 A_{1,1} A_{3,2}\,, \tag{3.80}$$

where $\delta_1(\theta_4) = \delta_2(\theta_4) = 1$ and $\tau_1(\theta_4) = \tau_3(\theta_4) = 1$, whereas $\tau_2(\theta_4) = 0$.

The state update is then performed in the same way as for the *PDA* filter in Equations (3.63) and (3.64), but with the *JPDA* probabilities. Assuming the target states are mutually independent regarding their past, we get the *JPDA* probabilities by marginalizing out all probabilities of feasible joint events as

$$\beta_{j,t} = \sum_{\theta:\theta_{j,t}\in\theta} p(\theta|\mathbf{Z}). \tag{3.81}$$

4 Multi-object Tracking with Recurrent Neural Networks

Contents

The aim of this chapter is to provide an introduction to Artificial Neural Networks (ANNs) and how they are used for target tracking. We start with an explanation of Feedforward Neural Networks (FNNs), including some principle concepts and the meaning of supervised learning. Afterwards, we discuss Recurrent Neural Networks (RNNs) which are an extension to the *FNN* for sequential input data. Furthermore, we describe a special type of *RNN* cell which is called Long Short-Term Memory (LSTM) and takes long-time dependencies into account, which can be useful in tracking applications. Another section of this chapter deals with the training of such networks. Finally, we explain how these concepts can be used for Multiple Object Tracking (MOT).

4.1 Feedforward Neural Networks

Artificial Neural Networks (*ANNs*) are engineered systems inspired by the biological learning processes in the brain [53]. While such networks were originally developed to model the behavior of biological brains, nowadays they are designed to master specific machine learning tasks. The basic concept behind *ANNs* is to solve complex problems with a collection of small and simple, connected processing units. These units, also called artificial neurons or nodes, were first proposed by McCulloch and Pitts [100] and later extended by Rosenblatt [122] which resulted in the *Perceptron*. Both models calculate a weighted sum of inputs followed by a threshold function. While the former has binary inputs and outputs with manually selected parameters, the latter uses floating point numbers and

is able to learn the parameters with an iterative learning algorithm. These models are the basis of recent *ANNs*. Figure 4.1 shows such a Perceptron with one neuron receiving vector $\mathbf{x} = (x_0, x_1, \ldots, x_n)$ as an input and trainable weights $\mathbf{w} = (w_0, w_1, \ldots, w_n)$ influencing the connection strength. Each neuron usually has an additional bias input with a connection weight which is always set to one. This can be simplified by adding an extra input and setting x_0 to one, which then acts like a bias unit. The activation or transfer function, denoted $g(\cdot)$, takes the weighted input sum z, written by the dot product $z = \mathbf{w}^T\mathbf{x} = w_0x_0 + w_1x_1 + \ldots + w_nx_n$, as an argument and produces the output value y.

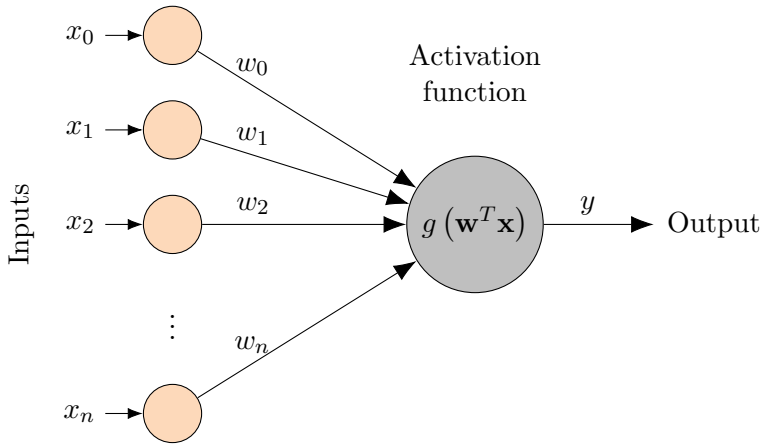


Figure 4.1: A Perceptron example with input values $x_0, x_1, \ldots, x_n$, learnable connection weights $w_0, w_1, \ldots w_n$ and the activation or transfer function $g(\cdot)$, originally modeled by the *Heaviside step* function.

Early models of artificial neurons, *e.g.* ADALINE proposed by Widrow and Hoff [152], used linear activation functions, *i.e.* the identity function (f(x)=x) and thus, were limited to linear problems. Stacking these linear units to a multilayer system, again results in a linear model which can always be replaced by a single linear layer. The power of recent neural networks therefore comes from non-linear activation functions, discussed later in this chapter.

The goal of a Perceptron and *ANNs* in general is to approximate some function f^*, which typically solves a classification or regression task $y = f^*(\mathbf{x})$ by mapping an input value $\mathbf{x}$ to a given target value or label y^*. *FNNs*, for example, define a mapping $y = f(\mathbf{x}, \boldsymbol{\theta})$ where $\boldsymbol{\theta}$ contains the learnable parameters [53]. Such machine learning problems where pairs of input x and corresponding output label y are given for training, are called supervised learning tasks [55]. All problems occurring in this thesis are falling within this class of tasks. In contrast, unsupervised learning deals with unlabeled data and is for example used to explore structures in data or to build clusters of similar examples. Reinforcement learning is another category and describes tasks where, instead of ground truth labels y, only scalar reward values are available at the end of a task. Thus, the aim of such tasks it the reward maximization.

In order to solve more complex problems, we discuss the Feedforward Neural Network (*FNN*) or Multilayer Perceptron (MLP), which is used in a lot of Deep Learning (DL) models [16, 53] and is built by stacking layers of neurons on top of each other. An example of a feed forward network is illustrated in Figure 4.2. It has three input units expecting

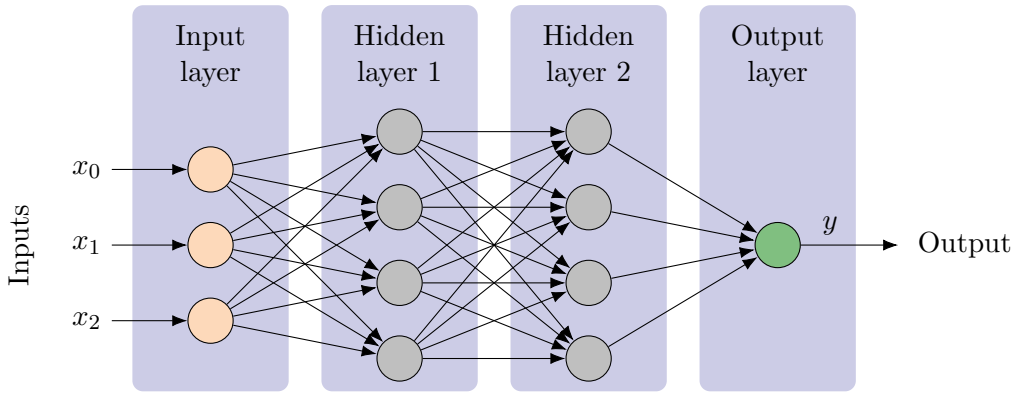


Figure 4.2: Example for a Feedforward Neural Network (*FNN*) with an input layer, two hidden layers and an output layer. All nodes are fully connected and build an acyclic graph.

the same number of input values. These units are *fully connected* with the first of two hidden layers. This means that each unit of one layer is connected with every unit of another layer. The same structure can be found between both hidden layers or the hidden layer and the following output layer, respectively. Connections between neurons within the same layer or to itself are not allowed. Hence, the network can be seen as a directed acyclic graph with nodes and weighted edges. The *FNN* propagates input values in one direction through the network starting at the input nodes. After passing the hidden layers, the propagated values end in the output layer. Such a process of propagating values through the network is known as the *forward pass* [55].

Models with more than one or a high number of hidden layers are usually called Deep Neural Networks (DNNs). *FNNs* with a deep structure are able to approximate nearly every arbitrary non-linear function and can therefore be seen as non-linear function approximators [53]. The learning process of such networks consists of adapting the network weights to optimize the matching of output values and assigned labels by minimizing some error function. Therefore, we feed the network with different input-output pairs and adjust the weights depending on the error every node produces. This error can be figured out by applying *gradient descent* (see Section 4.3) to the error function w.r.t. each of the network's weights. An efficient calculation of these gradients can be achieved by applying the well-known back-propagation algorithm [123]. The process of calculating gradients and applying them to the network weights is also known as *backward pass*.

4.1.1 Forward Pass

The forward pass of a *FNN* can be processed in a very efficient way by using matrix multiplications. To this end, we make use of the fully connected structure between L layers by writing the weights between two layers l and $l+1$ in a matrix $\mathbf{W}^{(l)} \in \mathbb{R}^{M \times N}$, where N denotes the number of units for the previous layer l and M denotes the number of units for the next layer $l+1$. If we do not integrate the bias into the input vector, we have to define it as $\mathbf{b} \in \mathbb{R}^M$. Hence, we obtain the output of the next layer $\mathbf{h}^{(l+1)} \in \mathbb{R}^M$ by applying the activation function $g(\cdot)$ element-wise to the weighted sums of the previous layer $\mathbf{z}^{(l+1)} = \mathbf{W}^{(l)}\mathbf{h}^{(l)} + \mathbf{b}^{(l)}$, where $\mathbf{h}^{(l)} \in \mathbb{R}^N$ defines the output vector of the previous layer. The forward pass of one layer is then given as:

$$\mathbf{h}^{(l+1)} = g\left(\mathbf{W}^{(l)}\mathbf{h}^{(l)} + \mathbf{b}^{(l)}\right) = g\left(\mathbf{z}^{(l+1)}\right). \tag{4.1}$$

Considering the network illustrated in Figure 4.2, we define $\mathbf{h}^{(0)} = \mathbf{x},\ \mathbf{x} \in \mathbb{R}^N$ which results in the following calculation chain:

$$\begin{aligned} \mathbf{h}^{(1)} &= g\left(\mathbf{W}^{(0)}\mathbf{x} + \mathbf{b}^{(0)}\right), \\ \mathbf{h}^{(2)} &= g\left(\mathbf{W}^{(1)}\mathbf{h}^{(1)} + \mathbf{b}^{(1)}\right), \\ \mathbf{h}^{(3)} &= y = g\left(\mathbf{W}^{(2)}\mathbf{h}^{(2)} + \mathbf{b}^{(2)}\right). \end{aligned} \tag{4.2}$$

The final output y for this example network is then a scalar because of the single output node. However, if we want to pass a whole batch of inputs through the network at once, we have to model the input vector $\mathbf{x}$ as a matrix $\mathbf{X} \in \mathbb{R}^{N \times D}$, where D denotes the batch size, which results in an output vector $\mathbf{y} \in \mathbb{R}^D$ containing all output values.

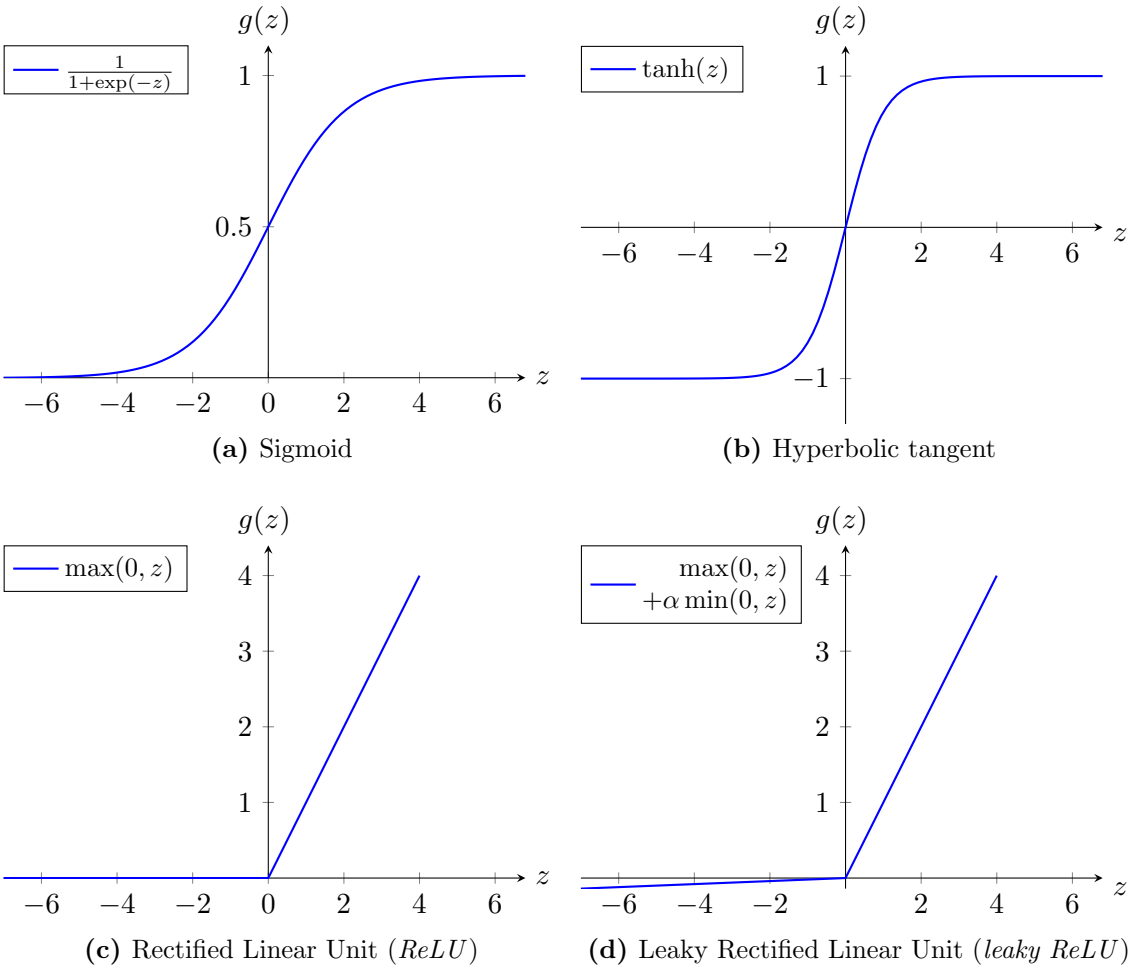
The single output node in our example is usually used for regression tasks activated by the identity function or for binary classification in combination with a sigmoid activation function. The output layer design, regarding number of nodes and activation function, depends on the task we want to solve. Within the next paragraphs we discuss popular activation functions and how they are related to the number of output units and the chosen loss function used for network training.

Activation Functions

One reason for the success of *DNNs* is the usage of non-linear activation functions within the hidden layers. They allow the network to model nearly any arbitrary target function and can for example find non-linear classification boundaries [55]. To overcome the limitations of early *ANNs*, which used only linear activation functions, *i.e.* the identity function, the sigmoid function

$$g(z) = \sigma(z) = \frac{1}{1 + \exp(-z)}, \tag{4.3}$$

as illustrated in Figure 4.3a, was used. Because it yields values between 0 and 1, it is a good choice as an activation for the output layer to model probabilities. As we will see later, in such a case it is important to use an appropriate cost function which eliminates the saturation effect while training. However, used within hidden layers, the function saturates in both directions very quickly and therefore suffers from the vanishing gradient problem. This means, that during back-prop the gradients get very small for a big range of the function, which stops the network from learning.



(a) Sigmoid

(b) Hyperbolic tangent

(c) Rectified Linear Unit (*ReLU*)

(d) Leaky Rectified Linear Unit (*leaky ReLU*)

Figure 4.3: Popular activation functions used in recent *DNNs*. The sigmoid function (a) produces output values between 0 and 1. The same characteristic s-shape can be observed for the hyperbolic tangent (b), yielding values between -1 and 1. The Rectified Linear Unit (ReLU) function (c) and the Leaky Rectified Linear Unit (leaky ReLU) function (d) are both equal to the identity function within the positive domain. The *leaky ReLU* function additionally ensures a gradient not equal to zero for negative activations.

Another activation function heavily used is the hyperbolic tangent given by

$$g(z) = \tanh(z) = \frac{\exp(2z) - 1}{\exp(2z) + 1}. \tag{4.4}$$

It is a zero centered version of the sigmoid function which outputs values between -1 and 1 as shown in Figure 4.3b. They are closely related by the linear transformation

$$\tanh(z) = 2\sigma(2z) - 1, \tag{4.5}$$

which means, that every network with sigmoid activation functions in the hidden layers can be replaced by another network with hyperbolic tangent activation functions [55].

Nowadays, Rectified Linear Units (*ReLUs*) defined as

$$g(z) = \max(0, z), \tag{4.6}$$

are commonly used activation functions in hidden layers of *DNNs* because they act like the identity function in the positive domain and therefore, they are easy to optimize [53]. The plot in Figure 4.3c shows the identity function for positive values and zero for negative ones. This leads to large gradients in the positive domain, when the unit is active. However, a problem occurs when there exist examples which cause zero activations while training, because gradient-based learning algorithms are not able to learn from such examples. Hence, there exist some *ReLU* generalizations like the *leaky ReLU* illustrated in Figure 4.3d and defined as

$$g(z) = \max(0, z) + \alpha \min(0, z), \tag{4.7}$$

ensuring gradients also for the negative domain. The parameter α is normally fixed to a small value.

Loss Functions

The back-propagation algorithm needs a loss or error function to evaluate the difference between predicted output y and target value t while training. In order to obtain useful results, the output layer design, as well as an appropriate choice of the loss function is important. Regression tasks, for example, require the identity function in the output layer to get floating point numbers as a result. The appropriate loss function for such a task is the Mean Squared Error (MSE) function

$$E(y, t) = \frac{1}{2}(y - t)^2, \tag{4.8}$$

with network output y and assigned target t. If we consider a batch of D input values, we get

$$E(y,t) = \frac{1}{2D}\sum_{i=1}^{D}(y_i - t_i)^2, \tag{4.9}$$

which is simply an averaged version as the name suggests.

Another type of task is the classification of inputs, which has other requirements. The number of different classes in the training set defines the output layer dimension. Starting with a binary classification task which has two different classes C_1 and C_2, a single neuron with a sigmoid activation function is needed. It can infer the probability of belonging to a defined class C_1 or not:

$$\begin{aligned} p(C_1|x) &= y = \sigma(\mathbf{w}^T\mathbf{x} + \mathbf{b}), \\ p(C_2|x) &= 1 - y. \end{aligned} \tag{4.10}$$

Following the loss function derivation using maximum likelihood, explained in [53, 55], we get the *cross-entropy error* function

$$E(y,t) = (t-1)\ln(1-y) - t\ln(y), \tag{4.11}$$

where t is the binary target label (*e.g.* $t = 1$ if C_1, else $t = 0$) and y is the output probability. Note that the logarithm within the error function equalizes the exponential of the sigmoid function, which prevents the back-propagation algorithm from saturation.

For classifiers with more than two classes, the number of output units has to match the number of classes appearing in the data. These units have the same activation function which is known as *softmax* and yields a probability for each class. It is defined as

$$g(z_j) = \frac{\exp(z_j)}{\sum_{i=1}^{C}\exp(z_i)}, \tag{4.12}$$

where j denotes different output nodes. Because the softmax function responds to the differences between its inputs, it is invariant of adding a scalar value to all inputs [53]. Therefore, we can derive a more stable variant:

$$\text{softmax}(z) = \text{softmax}(z - \max_i(z_i)). \tag{4.13}$$

The corresponding loss function for multiple classes

$$E(y,t) = -\sum_{i=1}^{C} t_i \ln(y_i), \tag{4.14}$$

is the multi-class *cross-entropy error* function and contains, equal to the case C=2, the logarithm which is undoing the exponential of the loss function and therefore ensures better learning.

4.1.2 Backward Pass

The backward pass of a network in general contains the calculation of gradients from the loss function w.r.t. the network weights. These gradients are later used to minimize the loss while training by applying gradient descent (see Section 4.3). An efficient way to calculate gradients of a *FNN* is the back-propagation algorithm [123], short *backprop*, which applies the chain rule of calculus in a repetitive manner.

We start with the last layer of the network and derive the loss function w.r.t. to the weights

$$\frac{\partial \mathbf{E}(y,t)}{\partial \mathbf{W}^{(L-1)}} = \underbrace{\frac{\partial \mathbf{E}(y,t)}{\partial \mathbf{h}^{(L)}} \cdot \frac{\partial \mathbf{h}^{(L)}}{\partial \mathbf{z}^{(L)}}}_{\boldsymbol{\delta}^{(L)}} \cdot \frac{\partial \mathbf{z}^{(L)}}{\partial \mathbf{W}^{(L-1)}}, \tag{4.15}$$

with $\mathbf{h}^{(L)} = \mathbf{y}$ denoting the last layer's output and $\mathbf{W}^{(L-1)}$ is the corresponding weight matrix. Note, we assume here a scalar target value t and a scalar predicted output y representing a network with a single output neuron. Considering the *FNN* in Figure 4.2, the weight calculation between hidden layer 2 and the output layer follows

$$\frac{\partial \mathbf{E}(y,t)}{\partial \mathbf{W}^{(2)}} = \underbrace{\frac{\partial \mathbf{E}(y,t)}{\partial \mathbf{h}^{(3)}} \cdot \frac{\partial \mathbf{h}^{(3)}}{\partial \mathbf{z}^{(3)}}}_{\boldsymbol{\delta}^{(3)}} \cdot \frac{\partial \mathbf{z}^{(3)}}{\partial \mathbf{W}^{(2)}}, \tag{4.16}$$

where $\boldsymbol{\delta}^{(3)}$ can also be used to calculate the bias

$$\frac{\partial \mathbf{E}(y,t)}{\partial \mathbf{b}^{(2)}} = \boldsymbol{\delta}^{(3)} \cdot \frac{\partial \mathbf{z}^{(3)}}{\partial \mathbf{b}^{(2)}}, \tag{4.17}$$

if explicitly denoted. Hence, we can write down the calculation in a general form for the remaining weights of our network:

$$\frac{\partial \mathbf{E}(y,t)}{\partial \mathbf{W}^{(L)}} = \overbrace{\underbrace{\frac{\partial \mathbf{E}(y,t)}{\partial \mathbf{h}^{(L)}} \cdot \frac{\partial \mathbf{h}^{(L)}}{\partial \mathbf{z}^{(L)}}}_{\boldsymbol{\delta}^{(L)}} \cdot \frac{\partial \mathbf{z}^{(L)}}{\partial \mathbf{h}^{(L-1)}} \cdot \frac{\partial \mathbf{h}^{(L-1)}}{\partial \mathbf{z}^{(L-1)}}}^{\boldsymbol{\delta}^{(L-1)}} \cdot \ldots \cdot \frac{\partial \mathbf{z}^{(1)}}{\partial \mathbf{W}^{(0)}}. \tag{4.18}$$

We can derive a universal formulation for the weight and bias calculation by examining the examples in Equations (4.16), (4.17) and the general expression in Equation (4.18). We observe that the calculation of the last term for the weights can be simplified to

$$\frac{\partial \mathbf{z}^{(3)}}{\partial \mathbf{W}^{(2)}} = \frac{\partial}{\partial \mathbf{W}^{(2)}} \mathbf{W}^{(2)} \mathbf{h}^{(2)} + \mathbf{b}^{(2)} = \left(\mathbf{h}^{(2)}\right)^T, \tag{4.19}$$

which results in the universal form for network weight of layer l

$$\frac{\partial \mathbf{E}(y,t)}{\partial \mathbf{W}^{(l)}} = \boldsymbol{\delta}^{(l+1)} \cdot \left(\mathbf{h}^{(l)}\right)^T. \tag{4.20}$$

Similar to the weights, we can observe for the bias

$$\frac{\partial \mathbf{z}^{(3)}}{\partial \mathbf{b}^{(2)}} = \frac{\partial}{\partial \mathbf{b}^{(2)}} \mathbf{W}^{(2)} \mathbf{h}^{(2)} + \mathbf{b}^{(2)} = 1, \tag{4.21}$$

which results in the universal form for the network bias of layer l

$$\frac{\partial \mathbf{E}(y,t)}{\partial \mathbf{b}^{(l)}} = \boldsymbol{\delta}^{(l+1)}. \tag{4.22}$$

4.2 Recurrent Neural Networks

Recurrent Neural Networks (*RNNs*) can be seen as an extension to *FNNs* and are able to process sequential input data $\mathbf{x}_1, \ldots, \mathbf{x}_T$. In addition to the input and output, the used artificial neurons have a feedback connection which provides the current state of the neuron as an input for the next time step. This means that the current state within a neuron affects the future state like in a dynamic system [53], modeled as

$$\mathbf{h}_t = f(\mathbf{h}_{t-1}, \mathbf{x}_t; \boldsymbol{\theta}), \tag{4.23}$$

where $\mathbf{h}_t$ denotes the systems state and $\mathbf{x}_t$ is the input vector at time t. However, t has not necessarily to be a time index. It could also be some spatial information like the index of a pixel. Only the sequence order is important to apply *RNNs*.

Considering a *RNN*, Equation (4.23) defines the hidden layer of a network. It acts like a memory containing a lossy summary about the past, remembering only relevant aspects of past sequences. This recursive structure allows the network to deal with sequences of different length. Figure 4.4 shows the basic structure of a *RNN* with a hidden layer containing a single recurrent neuron. Another core concept of *RNNs* is parameter sharing

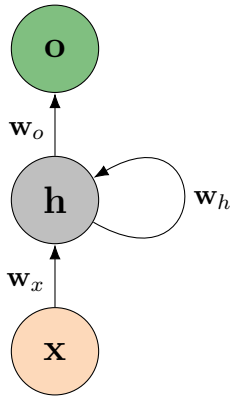


Figure 4.4: Illustration of a basic Recurrent Neural Network (*RNN*) with a hidden layer containing one unit. The recurrent connection provides the future state with a weighted version of the current state.

across different parts of the network [53]. Hence, it enables the model to generalize across

sequences of different length, even when sequences of a certain length are not part of the training set. Furthermore, it shares statistical knowledge of inputs with different positions inside the input sequence.

4.2.1 Unfolding

For a better understanding of the network illustrated in Figure 4.4, we can unfold the recursive structure in time. What we get is a computational graph with repetitive structures and shared weights. Important is the fact, that such a representation does not have

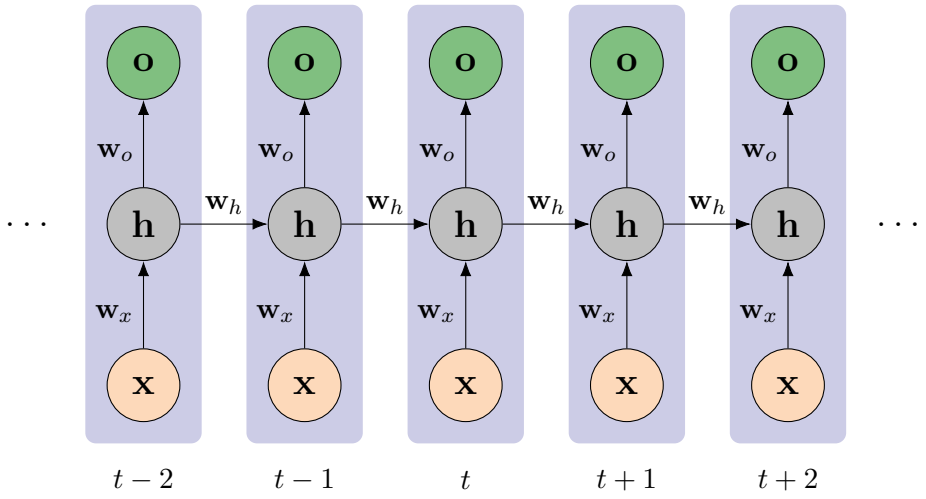


Figure 4.5: Unfolded Recurrent Neural Network (*RNN*) in time with shared weights. Inputs, states and outputs use the same weights for each time instance.

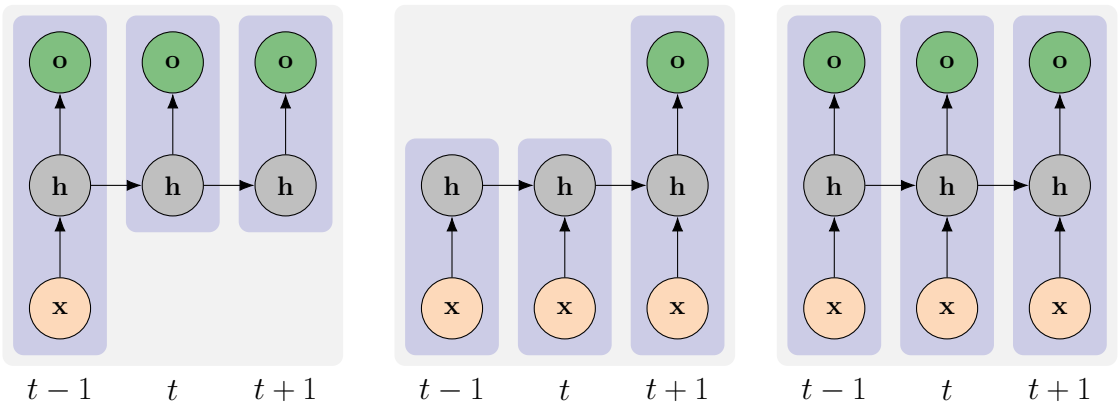
recurrent connections anymore which allows a well defined forward and backward pass. This unfolded representation helps to understand the calculation of both directions.

Because of the flexible structure, *RNNs* are usable for different types of applications. Figure 4.6 shows possible configurations depending on the number of input and output neurons. Popular configurations [74] are: *One-to-many*, with one input initiating a sequential output, *many-to-one* which uses sequential input data to produce one output, read out at the last time step and *many-to-many* where input sequences lead to output sequences. All three models can result in different architectures for a variety of applications.

In order to achieve state predictions based on previous states, we use the many-to-many structure for tracking applications. Every time step we predict the next state based on the current knowledge which is represented by the internal state of the network. The same can be done for state updates as we will discuss in Section 4.4.

4.2.2 Forward Pass

RNNs have a similar forward pass as *FNNs* with one hidden layer. The main difference is the recurrent connection, which leads to an additional input to the activation of the



(a) one-to-many **(b)** many-to-one **(c)** many-to-many

Figure 4.6: Illustration of different *RNN* structures, where weights are omitted because of readability. One-to-many networks (a) take a single input vector while they produce a sequence output. An exemplary task is image captioning, where the input is an image and the output is a description of the image. Networks with a sequence as an input and a single output (b) are used *e.g.* for sentiment analysis or fraud detection. (c) shows a many-to-many architecture for applications which need a prediction/classification for every time instance, *e.g.* image classification in video sequences or state prediction in target tracking. Image adapted from [74].

hidden layer. This loop connection enables the network to keep an internal state of the past, containing relevant information of past inputs.

Considering a *RNN* with multiple units inside the hidden layer and a sequence-to-sequence structure, we can again write the forward pass in matrix notation. To this end, we define three weight matrices $\mathbf{W}_x \in \mathbb{R}^{M\times N}$, $\mathbf{W}_h \in \mathbb{R}^{M\times H}$ and $\mathbf{W}_o \in \mathbb{R}^{O\times M}$ denoting the input weights, the transition weights of the network states and the output weights, respectively. Additionally, we need the bias vectors $\mathbf{b}_{\text{in}} \in \mathbb{R}^M$ and $\mathbf{b}_{\text{out}} \in \mathbb{R}^O$. With the current input vector $\mathbf{x}^{(t)} \in \mathbb{R}^N$ and the state vector $\mathbf{h}^{(t-1)} \in \mathbb{R}^H$ of the last time step, we are able to do a time update as:

$$\mathbf{z}^{(t)} = \mathbf{W}_h\mathbf{h}^{(t-1)} + \mathbf{W}_x\mathbf{x}^{(t)} + \mathbf{b}_{in}, \tag{4.24a}$$

$$\mathbf{h}^{(t)} = \tanh\left(\mathbf{z}^{(t)}\right) \quad \text{and} \tag{4.24b}$$

$$\mathbf{o}^{(t)} = \mathbf{W}_o\mathbf{h}^{(t)} + \mathbf{b}_{out}. \tag{4.24c}$$

This time update has to be done for each element in the sequence by recursively calling the same procedure and incrementing the time index. Depending on the application, one has to apply an activation function, discussed in Section 4.1.1, to the output layer. As an example, a network for multi-class classification would need a softmax activation

$$\mathbf{y}^{(t)} = \text{softmax}(\mathbf{o}^{(t)}), \tag{4.25}$$

applied to every output in $\mathbf{o}^{(t)}$. The required error function is then the cross-entropy loss

$$\mathbf{E}^{(t)}(\mathbf{y}^{*(t)}, \mathbf{y}^{(t)}) = -\mathbf{y}^{*(t)} \log\left(\mathbf{y}^{(t)}\right), \tag{4.26}$$

with true value or label $\mathbf{y}^{*(t)}$ at time t, summed over the whole sequence

$$\mathbf{E}(\mathbf{y}^{*}, \mathbf{y}) = \sum_{t} -\mathbf{y}^{*(t)} \log\left(\mathbf{y}^{(t)}\right). \tag{4.27}$$

4.2.3 Backward Pass

The backward pass for *RNNs* also contains the calculation of gradients regarding some loss function w.r.t. the network weights like in *FNNs*. The main difference is the time dependent recurrence within the hidden layer which we have to take care of while applying the chain rule of calculus. Hence, an efficient way to calculate the gradients provides the Backpropagation Through Time (BPTT) [151].

The *BPTT* algorithm uses the same technique to obtain gradients as for calculating the error. It sums up all gradients over the whole time sequence

$$\frac{\partial \mathbf{E}(\mathbf{y}^{*}, \mathbf{y})}{\partial \mathbf{W}} = \sum_{t} \frac{\partial \mathbf{E}^{(t)}(\mathbf{y}^{*(t)}, \mathbf{y}^{(t)})}{\partial \mathbf{W}}, \tag{4.28}$$

which results from the fact that we use shared weights $\mathbf{W}$. For further equations, we omit the parameters of $\mathbf{E}^{(t)}$ to retain readability and assume target and prediction vectors $\mathbf{t}$ and $\mathbf{y}$, respectively. Thus, we can easily calculate the partial derivative of the loss function w.r.t. $\mathbf{W}_o$ as

$$\frac{\partial \mathbf{E}^{(t)}}{\partial \mathbf{W}_o} = \frac{\partial \mathbf{E}^{(t)}}{\partial \mathbf{y}^{(t)}} \frac{\partial \mathbf{y}^{(t)}}{\partial \mathbf{o}^{(t)}} \frac{\partial \mathbf{o}^{(t)}}{\partial \mathbf{W}_o} \quad \text{and} \tag{4.29a}$$

$$\frac{\partial \mathbf{E}}{\partial \mathbf{W}_o} = \sum_{t} \frac{\partial E^{(t)}}{\partial \mathbf{y}^{(t)}} \frac{\partial \mathbf{y}^{(t)}}{\partial \mathbf{o}^{(t)}} \frac{\partial \mathbf{o}^{(t)}}{\partial \mathbf{W}_o}, \tag{4.29b}$$

starting at the last time step propagating back through time and summing up the results to get the total gradient.

In order to obtain the gradients for $\mathbf{W}_x$ and $\mathbf{W}_h$ we have to deal with some additional difficulties. In general, we define the gradient calculation as

$$\frac{\partial \mathbf{E}^{(t)}}{\partial \mathbf{W}} = \frac{\partial \mathbf{E}^{(t)}}{\partial \mathbf{y}^{(t)}} \frac{\partial \mathbf{y}^{(t)}}{\partial \mathbf{o}^{(t)}} \frac{\partial \mathbf{o}^{(t)}}{\partial \mathbf{h}^{(t)}} \frac{\partial \mathbf{h}^{(t)}}{\partial \mathbf{z}^{(t)}} \frac{\partial \mathbf{z}^{(t)}}{\partial \mathbf{W}}, \tag{4.30}$$

where $\mathbf{W}$ denotes $\mathbf{W}_x$ and $\mathbf{W}_h$, because they can be calculated in the same way. Hence, the term $\mathbf{h}^{(t)} = \tanh\left(\mathbf{W}_h \mathbf{h}^{(t-1)} + \mathbf{W}_x x^{(t)} + b_{in}\right)$ depends on $\mathbf{h}^{(t-1)}$ and, therefore, we can not treat $\mathbf{h}^{(t-1)}$ as a constant. Thus, we need to apply the chain rule again for the

recursive structure [19]

$$\frac{\partial \mathbf{E}^{(t)}}{\partial \mathbf{W}} = \sum_{k=0}^{t} \frac{\partial \mathbf{E}^{(t)}}{\partial \mathbf{y}^{(t)}} \frac{\partial \mathbf{y}^{(t)}}{\partial \mathbf{o}^{(t)}} \frac{\partial \mathbf{o}^{(t)}}{\partial \mathbf{h}^{(t)}} \left(\prod_{j=k+1}^{t} \frac{\partial \mathbf{h}^{(j)}}{\partial \mathbf{z}^{(j)}} \frac{\partial \mathbf{z}^{(j)}}{\partial \mathbf{h}^{(j-1)}} \right) \frac{\partial \mathbf{h}^{(k)}}{\partial \mathbf{z}^{(k)}} \frac{\partial \mathbf{z}^{(k)}}{\partial \mathbf{W}}. \tag{4.31}$$

Finally, to obtain the total gradient, we have to sum again all gradients of each time step similar to Equation (4.29b). An example derivation for a specific configuration can be found in [53].

4.2.4 Bidirectional Structure

A network designed to make use of future information is the Bidirectional Recurrent Neural Network (BRNN) [127]. An additional layer, as illustrated in Figure 4.7, receives the reversed input sequence. Both layers, forward and backward, are connected to the

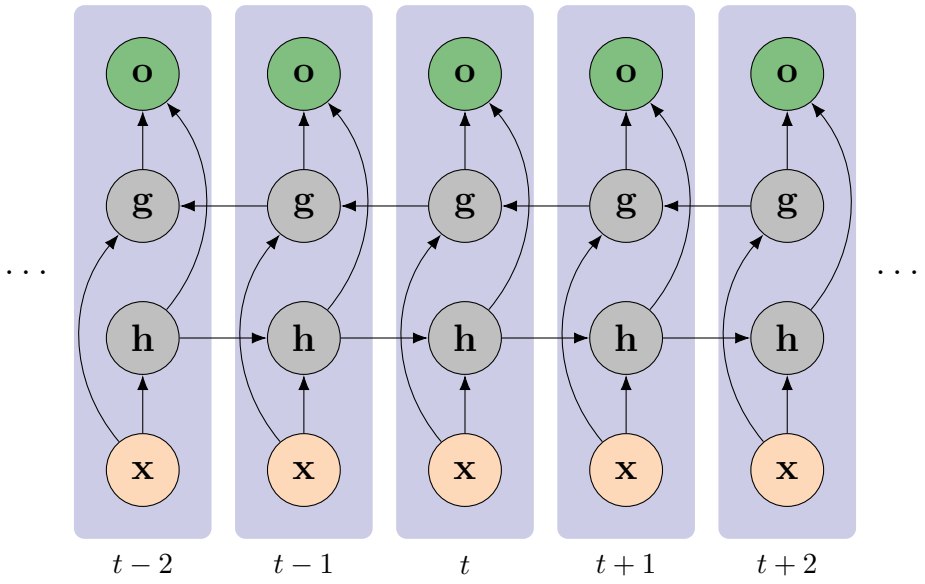


Figure 4.7: Unfolded Bidirectional Recurrent Neural Network (*BRNN*) in time. In addition to basic *RNNs*, the bidirectional version has an extra backward direction, represented by the layer **g**, directly connected with inputs and outputs from the standard layer **h**. For readability reasons we dropped the connection weights. Logically, the new connections require further weights. Image adapted from [55].

input and output units, respectively. This enables the network to observe and process the input in both directions. Hence, the *BRNN* is able to compute a representation depending on the future and on the past, most sensible around the current input time t [53].

Such a network structure is often used in applications, where the output depends strongly on the whole input sequence, *e.g.* speech recognition or handwriting recognition. It is also applicable for spatial data where the sequence has no time-dependence, but relies

on the input order. *MOT* is in general a causal task and thus, is not able to benefit from this kind of network. However, Data Association (DA) within a tracking procedure can be modeled such that a *BRNN* improves the performance [160].

4.2.5 Long-Short Term Memory

The main drawback of *RNNs* with simple recurrent connections is the *vanishing gradient problem* [11, 62]. Because of the recurrent connection in the hidden unit, the gradients get propagated through the network all the time. Thus, small gradients tend to vanish and large gradients may "explode" over time.

There exist a lot of different approaches addressing this problem. Within this thesis we focus on the widely used Long Short-Term Memory (*LSTM*) [63] unit including some extensions, *e.g.* Gers et al. [50, 51]. The *LSTM* architecture is similar to that used in basic *RNNs*. However, instead of a single recurrent connection, the *LSTM* contains a set of recurrently connected subnets known as memory cells [55]. Figure 4.8 illustrates

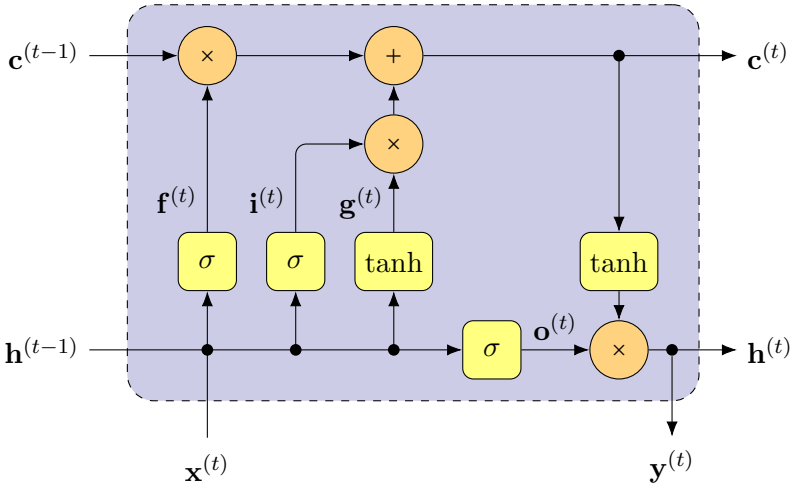


Figure 4.8: Long Short-Term Memory (*LSTM*) architecture. The standard path $\mathbf{g}^{(t)}$ is equal to a basic *RNN* unit which takes $\mathbf{x}^{(t)}$ and the last state $\mathbf{h}^{(t-1)}$ as an input activated by the hyperbolic tangent. An additional forget gate $\mathbf{f}^{(t)}$ decides which information gets removed from the long-term memory $\mathbf{c}^{(t-1)}$, whereas an input gate $\mathbf{i}^{(t)}$ learns which information from the past and new input gets stored. Finally, an output gate $\mathbf{o}^{(t)}$ decides what information gets read out from the long-term memory to obtain a new short-term state $\mathbf{h}^{(t)}$ and output $\mathbf{y}^{(t)}$. Notice, $\mathbf{y}^{(t)}$ denotes the output equal to $\mathbf{o}$ in the basic *RNN* unit. Image adapted from [53].

the architecture of a single *LSTM*. Compared to the basic *RNN* unit, the *LSTM* has an additional state vector $\mathbf{c}^{(t)}$, called *cell*. It can be seen as the long-term memory, whereas $\mathbf{h}^{(t)}$ represents the short-term state. Hence, the basic idea of such an architecture is to train so-called *gates* to learn which information should be kept, removed and read out from the memory cell.

We begin with the previous long-term memory cell $\mathbf{c}^{(t-1)}$ which gets manipulated by the *forget gate*. The sigmoid activation, requiring the input vector and the previous short-term state, decides whether certain regions inside the memory get erased or not. Thus, the sigmoid activation yields values between zero and one for each entry, where zero means nothing is kept in the memory and one that everything is kept inside. The result gets multiplied element-wise with the current long-term state. The forget gate can be formulated as

$$\mathbf{f}^{(t)} = \sigma\left(\mathbf{W}_f[\mathbf{h}^{(t-1)}, \mathbf{x}^{(t)}] + \mathbf{b}_f\right), \tag{4.32}$$

where $\mathbf{W}_f$ denotes the weight matrix for the previous short-term state $\mathbf{h}^{(t-1)}$ and the input vector $\mathbf{x}^{(t)}$ and $\mathbf{b}_f$ denotes the bias. Note that we use a single weight matrix for $\mathbf{h}^{(t-1)}$ and $\mathbf{x}^{(t)}$ and thus, concatenate both inputs expressed by $[\cdot, \cdot]$.

In the central path, the weighted input $\mathbf{x}^{(t)}$ and previous short-time state $\mathbf{h}^{(t-1)}$ get activated by the hyperbolic tangent function which outputs the short-term behavior. This calculation is equal to a basic *RNN* unit, but the result does not get used as an output directly. Instead, the *input gate* decides how much of this knowledge should be stored within the long-term memory. Similar to the forget gate, we can formulate the input gate as

$$\mathbf{i}^{(t)} = \sigma\left(\mathbf{W}_i[\mathbf{h}^{(t-1)}, \mathbf{x}^{(t)}] + \mathbf{b}_i\right), \tag{4.33}$$

with weight matrix $\mathbf{W}_i$ and bias $\mathbf{b}_i$. The short-term path is defined as in Equation (4.24) for the basic *RNN* cell

$$\mathbf{g}^{(t)} = \tanh\left(\mathbf{W}_g[\mathbf{h}^{(t-1)}, \mathbf{x}^{(t)}] + \mathbf{b}_g\right), \tag{4.34}$$

with weight matrix $\mathbf{W}_g$ and bias $\mathbf{b}_g$.

The element-wise multiplication of $\mathbf{g}^{(t)}$ and $\mathbf{i}^{(t)}$ is added to the long-term memory $\mathbf{c}$ which represents the store procedure. In order to obtain the new cell state $\mathbf{c}^{(t)}$, we have to apply both, the forget and input gate as

$$\mathbf{c}^{(\mathbf{t})} = \mathbf{f}^{(t)} \cdot \mathbf{c}^{(t-1)} + \mathbf{i}^{(t)} \cdot \mathbf{g}^{(t)}, \tag{4.35}$$

where the dot represents element-wise multiplication.

Finally, the *output gate* decides what information is read out and passed to the output as

$$\mathbf{o}^{(t)} = \tanh\left(\mathbf{W}_o[\mathbf{h}^{(t-1)}, \mathbf{x}^{(t)}] + \mathbf{b}_o\right), \tag{4.36}$$

with weight matrix $\mathbf{W}_o$ and bias $\mathbf{b}_o$. The output gate result is element-wise multiplied with the activated long-term memory state to produce the current short-term state $\mathbf{h}^{(t)}$ as

$$\mathbf{h}^{(\mathbf{t})} = \mathbf{o}^{(t)} \cdot \tanh\left(\mathbf{c}^{(t)}\right), \tag{4.37}$$

which we use as an output by multiplying it with another weight matrix $\mathbf{W}_y$ to get the required output dimension w.r.t. the task we want to solve.

The backward pass of the network works similar to that of standard *RNNs*, but is out of this thesis' scope. We refer the interested reader to [53, 55, 56]. Furthermore, there exist also other types of gated neurons similar to the *LSTM* architecture. One prominent example with a simpler structure is the Gated Recurrent Unit (GRU) proposed by Cho et al. [29] which is a popular choice for language processing tasks. The idea is to combine the input and forget gate into a single update gate. Finally, all mentioned architectures can additionally be used in a bidirectional way as discussed for simple *RNNs*.

4.3 Network Training

So far we explained *FNNs* and *RNNs* in detail, including the forward and backward pass of different network structures. Now we discuss how these concepts contribute to the learning process of such models using *gradient descent*.

4.3.1 Gradient Descent

We start with the forward pass, where the entire training set is propagated through the network. Afterwards, we calculate the loss caused by the processed training samples. Hence, the goal is now to minimize the loss, which can be achieved by updating the weights repeatedly with a small step into the negative direction of the gradient:

$$W_{t+1} = W_t - \alpha \nabla_W E(W_t). \tag{4.38}$$

While $\nabla_W E(W_t)$ denotes the gradients of the error function w.r.t. parameter W_t, $\alpha \in [0, 1]$ is the step size or *learning rate*. Network training with weight updates after passing the whole dataset through the network is known as *batch learning*. This causes small weight updates and is very inefficient in deep networks, especially for huge datasets.

To overcome this problem, a commonly used method where only small batches of training examples are used to calculate the gradients, is *stochastic gradient descent*. We refer to the number of training examples used as *batch size*. An extreme form is *online learning*, where the gradient update is performed with a batch size of one. The main advantage of this method is the reduced computational cost for weight updates. Furthermore, it is common to use randomly selected input batches from the entire dataset in combination with stochastic gradient descent.

Another problem which occurs is that gradient descent easily gets stuck in local minima, caused by the usually non-convex shape of the loss function. Plaut et al. [115] proposed to use a *momentum* term, which helps to escape from local minima and speed

up convergence, as

$$\begin{aligned} \Delta W_{t+1} &= \mu \Delta W_t - \alpha \nabla_W E(W_t) \quad \text{and} \\ W_{t+1} &= W_t + \Delta W_{t+1}, \end{aligned} \tag{4.39}$$

where $\mu \in [0, 1]$ denotes the momentum parameter. Moreover, the learning rate significantly influences the success of learning. The consequences of an inappropriate learning rate are slow convergence for too small gradients on the one hand, which increases the risk of getting stuck in a local minimum, and divergent behavior or fluctuations around the optimum for too large gradients on the other hand. In the literature there exist a lot of different approaches trying to mitigate these problems. Within this thesis we make use of different optimizers based on gradient descent. We use for example RMSProp [138] which adapts the learning rate based on the moving average of squared gradients. Furthermore, we use the Adam [80] optimizer, which is an extension to RMSProp which also takes the average moments into account.

4.3.2 Regularization

Another challenge in Deep Learning (*DL*) and machine learning in general is generalization. This means that we want a trained network to perform well on previously unseen test data. However, the performance depends strongly on the network complexity and the amount and distribution of data available for training. If we have to deal with a limited number of training samples, the network tends to learn and remember the data which leads to *overfitting*. This can be observed when the gap between training and test or validation loss gets bigger and bigger.

One way to deal with this issue is to gather more training data which is difficult in most cases. Another possibility is dataset augmentation, where we aim to generate new realistic training examples from existing ones. If we imagine an image classification task, we can, for example, generate new training samples by shifting and/or rotating the image.

Furthermore, we can use regularization techniques, which prevents the network from overfitting. The simplest form of regularization is *early stopping*. To this end, we split the training set into two subsets, one used for training and the other one for validation. While training, the network gets tested against the validation set instead of the test set. Finally, if the stopping criteria is reached or the network stops improving on the validation set, we use the best model to measure the error on the test set. This avoids the indirect training on the test set. However, one drawback of this method is that we have less data for training.

Another regularization method is called *weight decay*. Here, we add a regularization term to our loss function which depends on the ℓ_1 or ℓ_2 norm of our network weights

$$E_{\ell_2} = E + \lambda \cdot ||W||_2, \tag{4.40}$$

where λ defines the strength of the regularization.

Srivastava et al. [133] proposed *dropout*, one of the most popular regularization methods used in *DL*. The algorithm drops units of all layers except the output layer with probability p. This means that the network is not able to use these units within one training epoch. This leads to less sensitive units regarding the input because they can not rely on neighboring units and thus, have to learn a more general representation. Dropout is performed for training only and does not affect inference directly.

4.4 Multi-object Tracking Architectures

In general, a tracking architecture consists of four steps: State prediction, data association, state update and track management. Milan et al. [103] proposed a simple end-to-end model which fulfills these steps with the help of *RNNs*. Their network is on the one hand very simple, but on the other hand restricted to a fixed number of measurements per frame and hence does not fit for most *MOT* tasks. To solve this issue at least for assigning a variable number of measurements to tracks, Yoon et al. [160] proposed an encoder-decoder structured model applied to tracking tasks. Within the next two sections we take a closer look at both models.

4.4.1 End-to-End Model

When we talk about end-to-end models, we mean neural networks which take some inputs and produce the expected output within one forward path. There are no hand-crafted intermediate heuristics or calculations necessary.

The end-to-end model proposed by Milan et al. [103] performs at first a prediction step using a simple *RNN* layer as illustrated in Figure 4.9 (blue box). The following update step (green box) requires some modifications of the basic *RNN* cell. Its internal cell state $\mathbf{h}^{(t)}$ is initialized with the new internal state $\mathbf{h}^{(t+1)}$ of the prediction cell, instead of calculating a new one based on the own internal state of the previous time step. Finally, the end-to-end model is completed by the track management component (purple box). It ensures the initiation and termination of tracks. Data association is done by a simple *LSTM* layer (see Section 4.2.5). Note that for tracking with *RNNs* the term "state" has several different meanings. The internal state of an *RNN* is denoted $\mathbf{h}^{(t)}$, whereas the tracking state is $\mathbf{x}^{(t)}$.

The network input at time t is the tracking state $\mathbf{x}^{(t)}$ combined with the previous network state $\mathbf{h}^{(t)}$ of the prediction step. Additionally, the association matrix $\mathbf{A}^{(t+1)}$ containing probabilities for each measurement-track pair, the measurements $\mathbf{z}^{(t+1)}$ of the next time step as well as the track existence probability $\boldsymbol{\varepsilon}^{(t)}$ is needed. In this representation we use a concatenated input vector $[\mathbf{x}^{(t)}; \mathbf{h}^{(t)}]$ and a combined weight matrix $\mathbf{W}_{xh}$. This does not influence the result, but requires only a single matrix multiplication in the forward pass and, thus, can be computed more efficiently in practice.

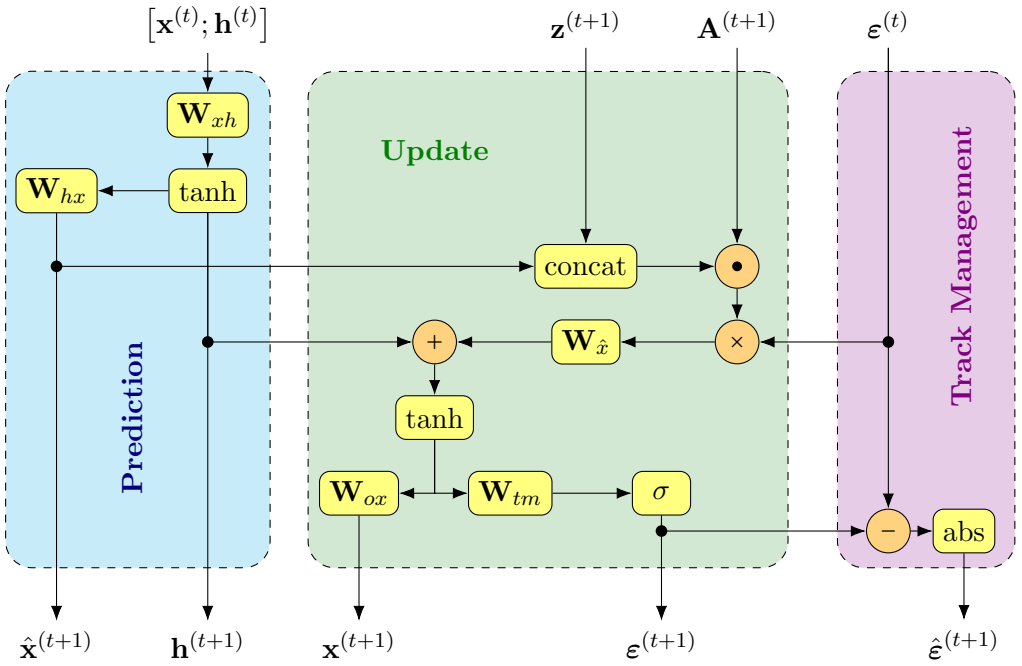


Figure 4.9: Recurrent Neural Network (*RNN*) tracking architecture. The prediction part is a basic *RNN* unit explained in Section 4.2.2 taking the previous state $\mathbf{h}^{(t)}$ and a vector $\mathbf{x}^{(t)}$ as an input. The track update depends on the association matrix $\mathbf{A}^{(t+1)}$ and the measurements $\mathbf{z}^{(t+1)}$ of the next time step, as well as on the track existence probability $\boldsymbol{\varepsilon}^{(t)}$. For calculation paths containing a weight matrix (W), we assume the dot product. This is also true for the (•) operator. All other operations are functions or element-wise operators. Image adapted from [103].

The whole process is straightforward. After tracking-state prediction, the internal state $\mathbf{h}^{(t)}$ of the simple *RNN* is added to the output of a fully-connected layer $\mathbf{W}_{\hat{x}}$ with linear activation. This operation, extended by the following tanh activation, represents another simple *RNN* cell, which shares the internal state with the prediction cell instead of calculating a new one.

The input of the fully-connected layer $\mathbf{W}_{\hat{x}}$ composes of the prediction output mixed with measurements observed at time $t+1$, association probabilities and track existence probabilities. Within this mixing process, measurements are concatenated to the predicted tracking-state and weighted by the association matrix, containing the information of which measurement belongs to which track. Additionally, the track management path multiplies the result with the current track probabilities.

Finally, we obtain the new tracking state $\mathbf{x}^{(t+1)}$ and track existence probabilities $\boldsymbol{\varepsilon}^{(t)}$ for the next time step, as well as a smoothness prior $\hat{\boldsymbol{\varepsilon}}^{(t+1)}$ for the existence probability. Both outputs, $\mathbf{x}^{(t+1)}$ and $\boldsymbol{\varepsilon}^{(t)}$, are calculated by fully-connected layers $\mathbf{W}_{ox}$ and $\mathbf{W}_{tm}$, respectively. While the activation for the tracking state is linear, the track existence probabilities are obtained by a sigmoid activation.

In order to train the entire network end-to-end, we need an appropriate loss function. Milan et al. [103] proposed the loss

$$E(\mathbf{x}, \hat{\mathbf{x}}, \boldsymbol{\varepsilon}, \mathbf{x}^*, \boldsymbol{\varepsilon}^*) = \overbrace{\frac{1}{2\ N\ D} \sum ||x^* - \hat{x}||^2}^{\text{prediction}} + \overbrace{\frac{1}{2\ N\ D} \sum ||x^* - x||^2}^{\text{update}} + E_\varepsilon + \hat{\varepsilon}, \tag{4.41a}$$

$$E_\varepsilon = \varepsilon^* \log(\varepsilon) + (1 - \varepsilon^*) \log(1 - \varepsilon), \tag{4.41b}$$

with x^* and ε^* denoting the true values. While the first two terms (prediction and update) use a sum of *MSE* functions, the third term E_ε is a binary cross entropy loss plus a regularization term $\hat{\varepsilon}$ which should avoid hard decisions when there are no measurements available.

The missing data association part is done with a *LSTM* network as discussed in Section 4.2.5. It takes the difference between the predicted state of a track and all the measurements as an input and outputs association probabilities in the form of a column vector $\mathbf{A}_i^{(t+1)}$, which is part of the entire association matrix $\mathbf{A}^{(t+1)}$. Therefore, we apply a softmax activation at the networks output. Note that because there is no dependency regarding time, we can also use the bidirectional *LSTM* version for this task. The data association path has a separate error function which is the cross entropy loss calculated by the negative log-likelihood. In fact, it is processed like the multi-class classification loss.

However, with such an architecture we are limited to a fixed number of measurements because the input vector has to have a predefined size. Another issue is the independent association of measurements to tracks. This means that the network does not take care of the fact that a measurement should belong to only one or no track. However, multiple associations to no track or the dummy track for clutter is allowed. Thus, one solution to this problem is the Hungarian algorithm which can be applied to the negative association matrix, the dummy track column excluded, to satisfy the one-to-one constraint.

4.4.2 Variable Data Association Model

Because most *MOT* tasks have no fixed number of measurements in each time step, it is necessary to build a network which is able to deal with this issue. For that reason we discuss the approach of Yoon et al. [160]. Their network consists of two parts, namely encoder and decoder. While the first part of the network - it contains only fully-connected layers - tries to encode the input into an internal representation, the second one was designed to learn the assignment for these encoded vectors. The decoder is a Bidirectional Long Short-Term Memory (BLSTM) network with fully-connected projection layers on top, transforming the high dimensional output to the requested size. The reshaped output results again in a score matrix between tracks and measurements which is later used in combination with the Hungarian algorithm.

The advantage of dealing with a variable number of measurements does not necessarily

come from the chosen network structure alone. Another important factor is the way how the input is preprocessed. Thus, the network takes the last T states of N tracks, concatenated with each detection. In this case, the states are only a history of previous detections. Additionally, they use a dummy track containing zeros for all T time steps to represent clutter. This is necessary to deal with measurements which do not emerge from a real track. Hence, we get $(N+1) \cdot M$ inputs for M measurements, which represents all permutations of tracks and measurements.

Each detection $\mathbf{d}$ and state $\mathbf{x}$ then contains the Cartesian coordinates for the upper left (l, u) and bottom right (r, b) corner of the two dimensional bounding box, the dimensions width w and height h, as well as the detection confidence c. This results in a vector $\mathbf{d}_{(t,m)} = \left[l_{(t,m)}, u_{(t,m)}, r_{(t,m)}, b_{(t,m)}, w_{(t,m)}, h_{(t,m)}, c_{(t,m)}\right]$ for detection $\mathbf{d}_m$ and a vector $\mathbf{x}_{(t,n)} = \left[l_{(t,n)}, u_{(t,n)}, r_{(t,n)}, b_{(t,n)}, w_{(t,n)}, h_{(t,n)}, c_{(t,n)}\right]$ for track $\mathbf{x}_n$, both at time t.

Finally, the input is an array of track-measurement pairs which are concatenations of the last T time-steps of all tracks with each measurement

$$\mathbf{X}_t = \begin{bmatrix} \mathbf{x}_{(-4,0)} & \mathbf{x}_{(-3,0)} & \mathbf{x}_{(-2,0)} & \mathbf{x}_{(-1,0)} & \mathbf{x}_{(0,0)} & \mathbf{d}_{(0,0)} \\ & & \vdots & & & \\ \mathbf{x}_{(-4,n)} & \mathbf{x}_{(-3,n)} & \mathbf{x}_{(-2,n)} & \mathbf{x}_{(-1,n)} & \mathbf{x}_{(0,n)} & \mathbf{d}_{(0,m)} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{d}_{(0,m)} \end{bmatrix}, \tag{4.42}$$

with $T = 5$, $\mathbf{0}$ denoting a zero vector representing the clutter track and $\mathbf{X}_t$ denoting the whole input batch at time t. All the permutations are fed into the network as a batch and can therefore be processed very fast.

The network structure is illustrated in Figure 4.10 and represents the encoder which is fed with the input data and the decoder receiving encoded vectors denoted e. Each decoder-block produces a value between -1 and $+1$, which represents a score of how good the track-measurement pair can be assigned. Hence, $+1$ implies that the pair fits very good and -1 vice versa. Finally, the output is reshaped to match the score matrix size with M rows and $N+1$ columns.

A detailed layer description of the encoder can be found in Table 4.1. As mentioned before, the encoder contains only fully-connected layers and therefore, requires a predefined input size. This is ensured by the concatenation scheme of track history and detection. The input size is then calculated by the length of a detection vector $|\mathbf{d}_m|$ multiplied by the number of time-steps T plus one for the detection itself. As a result, the encoder processes

Layer	Type	Input	Output	Activation
1	fully-connected	$\lvert\mathbf{d}_m\rvert \cdot (T+1)$	128	ReLU
2	fully-connected	128	128	ReLU
3	fully-connected	128	64	tanh

Table 4.1: Encoder network architecture. Table taken from [160].

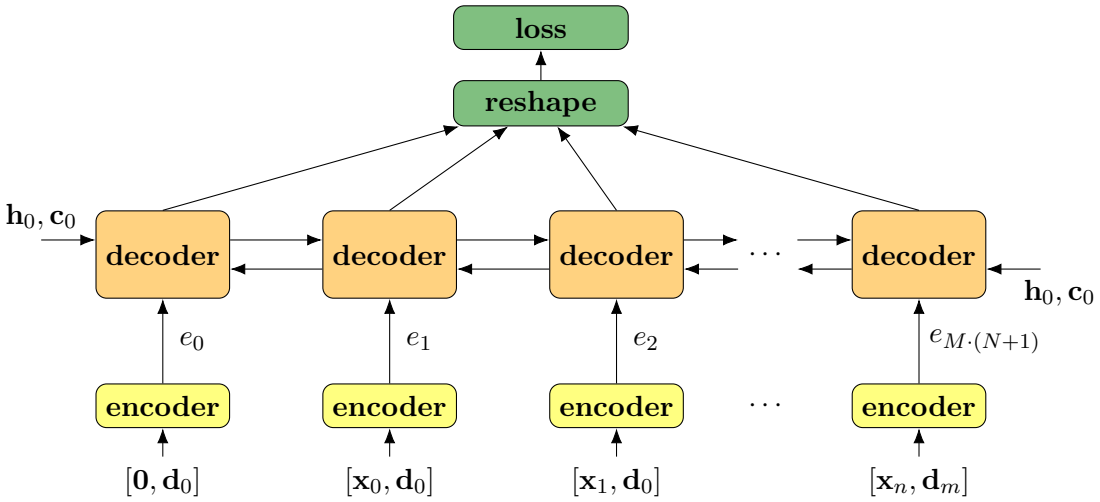


Figure 4.10: Data association model for a variable number of tracks and measurements. All the track-measurement permutations get encoded to a fixed size vector e and the decoder produces a score for each of these encoded combinations. The output is reshaped to a valid score matrix. For training, a loss function is applied to the score matrix. The initial internal state and the cell state of the bidirectional *LSTM* are denoted $\mathbf{h}_0$ and $\mathbf{c}_0$, respectively. Image adapted from [160].

an encoded vector of size 64 which should represent all the information needed to produce an assignment score for each vector.

Table 4.2 shows the detailed layer architecture of the decoder. Because we allow a variable number of measurements and tracks, the first layer of the decoder has to be recurrent. Yoon et al. [160] have chosen a *LSTM* cell in a bidirectional layer. This means that we additionally feed the recurrent layer with a reversed sequence. This allows the network to inspect the data in both directions and, thus, it may be easier to correctly assign detections to tracks.

Layer	Type	Input	Output	Activation
4,5	BLSTM	64	128	-
6	fully-connected	128	64	ReLU
7	fully-connected	64	1	tanh

Table 4.2: Decoder network architecture. Table taken from [160].

The bidirectional structure yields two output states. One for the forward direction and the other for the backward direction. Both output states are concatenated without an activation, followed by two projection layers on top, reducing it to a single value. Afterwards, the reshape function is applied. If we want to train the network, we use the resulting matrix as an input for the loss function. Otherwise, in the case of inference, we remove the loss function and apply the Hungarian algorithm on the inverse of this score matrix (cost matrix).

In order to obtain an appropriate loss, the encoder-decoder network uses a weighted *MSE* function. The weighting is necessary because of the score range between -1 and 1. Another concept applied by Yoon et al. [160] is the learning of initial weights inside a recurrent cell. The internal state $\mathbf{h}$ and the internal cell state $\mathbf{c}$ of a *LSTM* cell are learned based on the mean value of the encoded vectors. For more details, we refer the interested reader to [160].

5 Modular Multi-object Tracking

Contents

In this chapter we present our modular tracking framework, which combines several of the previously presented approaches. We provide information about the sensor model, the architecture and design decisions we made to combine all used trackers and datasets within one framework. Furthermore, we shortly present two used object detectors and a publicly available 2D target tracker, adapted to fit into our 3D object tracking framework.

The first implemented approach we explain is the combination of Bayes filters completed by a heuristic track management. Afterwards, we discuss various neural network structures modeled for different purposes within the tracking framework. Moreover, we discuss an end-to-end trained neural network, designed to solve all Multiple Object Tracking (MOT) subtasks within one single forward path. Finally, we close this chapter with details about the training of mentioned networks.

5.1 Implementation Details

For the implementation we use Python 3 because it is an easy-to-use, platform independent programming language. In addition to that, it covers all required functions and there exist a lot of helpful packages we can rely on, *e.g. NumPy*[1], *scikit-learn*[2] or *SciPy*[3]. Another useful package is TensorFlow [1], a graph based calculation framework, which we use to

[1] `https://numpy.org/` (accessed August 5, 2019)
[2] `https://scikit-learn.org/stable/` (accessed August 8, 2019)
[3] `https://www.scipy.org/` (accessed August 5, 2019)

create neural network models. It provides a fast and easy calculation of forward paths and gradients. Furthermore, it offers the possibility to do calculations either on CPU or GPU.

A main goal is the modeling of a tracking framework which is able to use various object detectors, delivering a bounding box for each detected target. This means that the tracker is completely independent from the object detection mechanism. As a consequence, we designed the state and measurement vector based on the KITTI dataset [48] sensor model, to meet our requirements.

5.1.1 Sensor Model

The KITTI Dataset has been recorded with high resolution stereo cameras, yielding gray scale and color images and a Velodyne HDL-64E rotating 3D laser scanner producing a point cloud. In addition a combined module, containing an Inertial Measurement Unit (IMU) and a Global Positioning System (GPS), are responsible for acceleration, velocity and absolute position, respectively. All sensors are mounted on the ego-vehicle and are synchronized. This can more or less be considered as a standard setup for an autonomous vehicle.

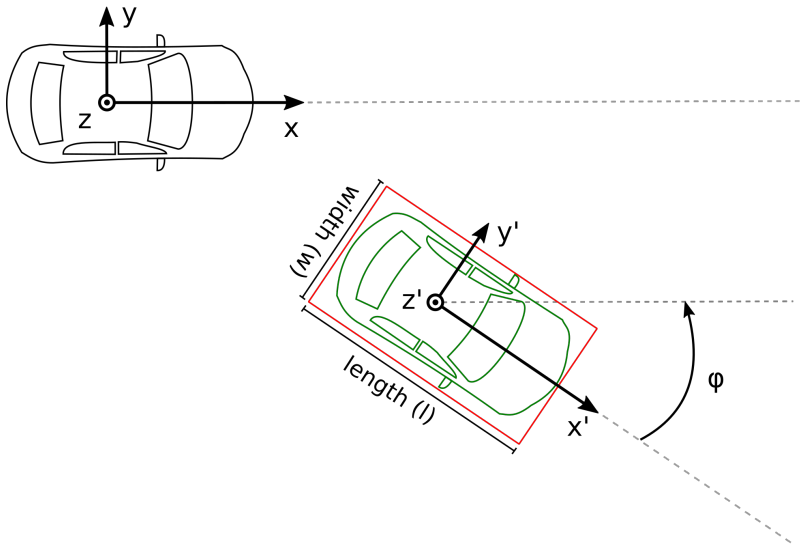


Figure 5.1: KITTI dataset sensor configuration. The ego-vehicle in black defines the coordinate system's origin. The x-axis points towards driving direction and shows a steering angle of 0 degrees. The green car represents a target vehicle with a red bounding box defined by width and length. Image adapted from [48].

The coordinate systems origin is defined by the axes x, y, and z, lying in the center of the ego-vehicle, whereas the coordinate system of each target is represented by x', y' and z' as illustrated in Figure 5.1. Hence, target bounding boxes in the raw KITTI dataset are

provided by relative center coordinates x', y' and z' w.r.t. the ego-vehicle. Additionally, we get the width, length and height of each target, as well as the relative yaw angle φ. The roll and pitch angles are close to zero and, therefore, ignored.

We use the same representation within our tracking framework and thus, require the object detector to provide 3D bounding boxes following this definition. Considering the KITTI dataset representation, we define the state and measurement vector. Because we additionally did some tests on the 2D image based MOT16 dataset [102], we convert 2D detections, to meet our definitions. Therefore, we extend the 2D bounding box by a z-dimension with zero height and move its origin into the origin of the image coordinate system. For such detections, the yaw angle is always zero. Notice that we used the MOT16 dataset only for proof-of-concept tests in the initial project phase. Thus, these qualitative results are not part of the evaluation section.

5.1.2 State and Measurement Representation

The internal representation of an object tracking framework depends mostly on the used tracker. Because we want to make use of different approaches, we need more than one representation for the current state. Bayesian filters can be described by a stochastic state space model (recall Section 3.1.2). One property of such models is the complete state assumption, which assumes that the current state includes the whole information.

In comparison, Recurrent Neural Networks (RNNs) may or may not use information of the past to reach good results. Depending on the whole network architecture, it is possible to feed *RNNs* with the current state while the internal state of the recurrent cells represent the complete state. Another architecture may need the whole history or at least a short period of the states as an input.

Therefore, our tracking framework has two representations. On the one hand, the state vector $\mathbf{x}_t$ contains the track state of time step t. On the other hand, a state sequence $\mathbf{X}_{seq,t}$ contains reduced track states $\mathbf{X}_{seq,t} = (\mathbf{x'}_{t-T+1},\ \ldots,\ \mathbf{x'}_t)$ of the last T time steps. This is the same as a short history of previous states, containing only parameters needed for inference and training.

Within our framework we want to keep track of the vehicle positions as well as the bounding box dimensions and headings. Hence, we define the state vector $\mathbf{x}_t$ as

$$\mathbf{x}_t = \begin{bmatrix} x_t & y_t & z_t & \omega_t & v_t & \dot{\omega}_t & w_t & l_t & h_t & \varphi_t \end{bmatrix}^T, \tag{5.1}$$

with center position coordinates x_t, y_t, z_t, steering angle ω_t, linear velocity v_t, angular velocity $\dot{\omega}_t$, bounding box dimensions w_t, l_t, h_t and bounding box heading φ_t. Notice, because of the relative motion, the steering angle and the bounding box heading does not need to have the same direction. Therefore, we handle both parameters separately. However, both angles are in the range of $[-\pi, \pi)$ and get normalized to this range after each step which manipulates them.

We neither calculate the steering angle, linear and angular velocity nor make use of these parameters within the *RNN* models. However, we keep them in the state vector and set them to zero because of compatibility reasons.

The measurement vector $\mathbf{z}_t$ contains observable parameters which can be obtained from the sensor model. We decided to track not only the center position of each target, but also the bounding box. Therefore, we define the measurement vector $\mathbf{z}_t$ as

$$\mathbf{z}_t = \begin{bmatrix} x_t & y_t & z_t & w_t & l_t & h_t & \varphi_t \end{bmatrix}^T, \tag{5.2}$$

with center position coordinates x_t, y_t, z_t, bounding box dimensions w_t, l_t, h_t and bounding box heading φ_t.

Although there should not be vertical motion or a change of the targets height, the vertical position z_t and bounding box height h_t are both part of the state and measurement vector. The reason why we included both parameters is that we do not have an additional bounding box tracker as, for example, in [117]. As mentioned earlier, for the MOT dataset, the height and z-position as well as the yaw angle are set to zero. This does not influence the evaluation which is done on the ground area of each object.

5.1.3 Framework Architecture

In order to obtain an online *MOT* framework which is able to use different object detectors, trackers and data association mechanisms, we propose a modular architecture consisting of independent blocks. The connection between these blocks are well defined interfaces to ensure flexibility.

Figure 5.2 illustrates the architecture overview. As an entry point, it contains a detection module, which could be any object detector delivering a 3D bounding box. Because the detection of targets is not part of this thesis, we make use of publicly available state-of-the-art detectors. However, it should be mentioned because it is a necessary part of the framework. Hence, Section 5.1.4 gives a brief summary of both detectors used for evaluation.

The central point of our design is the track management module. It takes care of incoming detections, tracks and holds the current frame number and the history of closed tracks. With the first received list of detections at time $t = 0$, the management module gets initialized and adds new track instances. Notice, these instances are not initialized yet. Afterwards, each new list of detections causes another tracking cycle.

Before we loop through one complete state prediction and update cycle, we take a look at the tracker. This module is responsible for holding the whole information of a track/trajectory and for doing the prediction and update step. While the track information, listed in Table 5.1, is equal for all trackers, the underlying prediction and update procedure depend on the method under consideration. The management state of a track is initialized with zero and gets increased by one, if the hit-streak counter reaches a certain

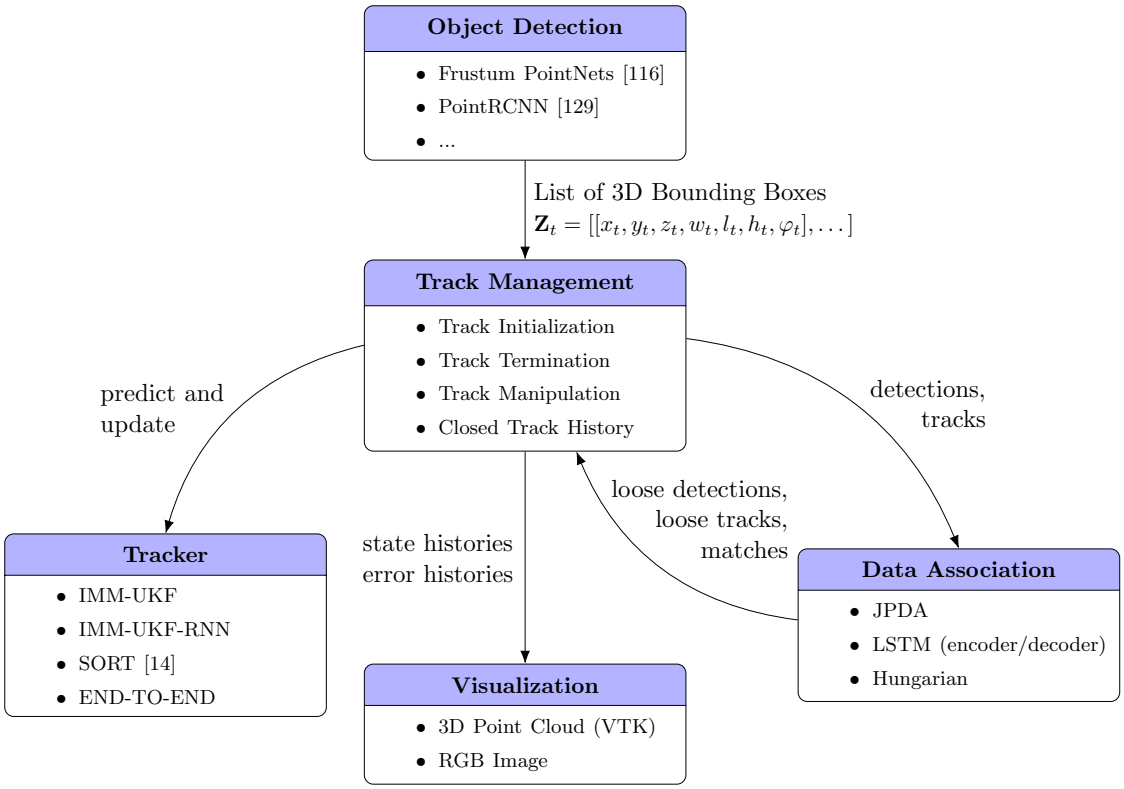


Figure 5.2: Tracking framework overview. See text for details.

Parameter	Description
state history	holds all preceding state vectors of the track
error history	holds all preceding covariance matrices of the track, if available
track state	current track state vector
management state	current management state (active or not)
hit-streak	counts the consecutive assignments of detections
time-since-update	counts the time-steps without assigned detections
predicted track state	state vector containing the last prediction
suspend counter	used to count the number of temporal terminations

Table 5.1: Track management parameters with description.

threshold. Then, the track is active and both history lists, track and error, are updated with the current state and the covariance matrix (only available in Bayesian filter methods), respectively. Otherwise, we add empty entries to the histories. Notice, because we are interested in online *MOT*, we take only active tracks into account and do not manipulate them after termination.

Starting a new cycle, we perform a prediction step for each track instance. In addition to that, the time-since-update parameter gets increased by one. Afterwards, we check if there is a collision between predictions of non-initialized and already active tracks and

remove the non-initialized ones, if this is the case. Therefore, we check the Intersection over Union (IoU) between them. This prevents us from initializing new tracks, emerging from clutter detections. Within each track instance we additionally store the predicted state in a separate history. This helps us to visualize the interaction of prediction and update steps.

Based on precise predictions we are able to perform the association of current detections with existing tracks. Within our framework we provide three different approaches: Joint Probabilistic Data Association (JPDA), Long Short-Term Memory (LSTM) (encoder/decoder network) and the Hungarian algorithm[4]. All three implementations require detections and tracks as an input and return the matched pairs, unmatched tracks and unmatched detections. The *JPDA* result has a slightly different representation. It returns association probabilities which could also be split up into the three mentioned lists. However, the update cycle of an *JPDA* filter makes direct use of these probability values.

While the track manager uses unassigned detections to add new track instances, unmatched tracks are simply ignored in the update step. Matched track and measurement pairs are used to update the current state. Again, the track instances are responsible for this procedure. They receive the assigned measurement and perform a correction step, save the new state and update their internal parameters, *i.e.* hit-streak plus one, time-since-update set to zero. Hence, track instances missing the correction step have now a time-since-update value which is not zero anymore.

Finally, the track manager updates the track management state and the track history of each track instance. Depending on the hit-streak counter, a track is set active or stays in the uninitialized (zero) state. Because we claim online behavior from our framework, only active tracks get a non-empty history entry. The termination of a track is triggered, if the time-since-update value exceeds a predefined threshold.

To overcome the problem of temporary occlusions, we added another parameter which is called *suspend counter*. With this variable we are able to control how often a track is allowed to change back in an uninitialized state before it gets finally terminated. This means, that an already active track with an expired time-since-update value is allowed to further exist, but has to change back its management state to zero. Thus, it exists in the background for a fixed number of attempts where it tries to get initialized again. The reason to use such a heuristic is that temporarily occluded objects may get reinitialized with the same track and therefore we do not have a switching track identifier. Another advantage is that wrong track initializations can be reduced.

5.1.4 Object Detection

Following the tracking-by-detection paradigm, a reliable and precise object detector is required. By taking a look at the KITTI leader board for 3D object detection, we found two state-of-the-art approaches with publicly available code and satisfying results. Because we

[4]Implementation from the *scikit-learn* package.

do not restrict the detection algorithms to specific input data, both methods are relevant for us. While Frustum PointNets [116] make use of RGB images and 3D point clouds, PointRCNN [129] uses 3D point clouds only.

In order to reduce the search space of 3D point clouds, the Frustum PointNets use matured 2D object detectors, *e.g.* [52, 119] on RGB images. The aim is to find objects in these images and extrude 2D bounding boxes to a 3D viewing frustum. Afterwards, all 3D points outside the found frustums are thrown away. Finally, two variants of PointNets perform 3D object instance segmentation and amodal 3D bounding box regression. The final result is an oriented 3D bounding box for each object of the classes *Car*, *Pedestrian* and *Cyclist*. Performing well on all three classes, Frustum PointNets outperform most approaches in detecting *Pedestrians*.

PointRCNN follows a different strategy. It does not make use of 2D RGB images, but directly uses the 3D point cloud in a two stage network. The first stage generates 3D proposals in a bottom-up manner by segmenting the point cloud into foreground and background. The second stage refines the generated proposals in canonical coordinates. The result is an oriented 3D bounding box similar to the Frustum PointNets. This is the best publicly available approach for the classes *Car* and *Cyclist*. For further details we refer the interested reader to [129].

5.1.5 Adaptation of SORT

An easy point to start implementing a filter-based tracking framework in Python is the SORT [14] algorithm, which is publicly available[5] and combines a linear Kalman Filter (KF) with the Hungarian algorithm. By default, the tracker is used on the MOT16 dataset and therefore, parametrized for tracking in 2D image coordinates. We extended the state vector $\mathbf{x}_t$ with the coordinate z_t, the height h_t and the bounding box orientation φ_t to

$$\mathbf{x}_t = \begin{bmatrix} x_t & y_t & z_t & \varphi_t & s_t & r_t & h_t & v_x & v_y & v_s \end{bmatrix}, \tag{5.3}$$

with center coordinates x_t, y_t and z_t of the bounding box ground area, orientation φ_t, aspect ratio r_t and scale s_t of the bounding box ground area with related scaling velocity v_s, bounding box height h_t and the velocities v_x and v_y for changes of the bounding box ground area size. A velocity and scaling for z_t is not necessary because we assume no change of the targets height.

Furthermore, we added rows in the system matrix $\mathbf{F}$ and the measurement matrix $\mathbf{H}$ for these values. Additionally, we had to replace all 2D bounding boxes with its 3D counterparts. Because the state vector does not fit our defined representation, the tracker module has to take care of an appropriate external representation. The measurement vector has to be the same as for all trackers.

The described approach is not necessarily designed for our special needs represented by

[5] `https://github.com/abewley/sort` (accessed August 19, 2019)

maneuvering targets in a cluttered environment. Notwithstanding, it is easy to implement, has a very short runtime, comes up with a short number of adjustable parameters and performs pretty good as we will see later in our evaluations. Thus, this implementation is our first algorithm embedded in the tracker module.

5.2 Combined Bayesian Filter Approach

The previously presented architecture is valid for all in this thesis implemented approaches. One of them is the combination of Bayesian filters, derived and explained in Chapter 3. Within this section we combine the elucidated filters into one *MOT* pipeline as illustrated in Figure 5.3 and describe the implementation details. Because we make use of an *IMM* filter for handling various motion models, combined with an *UKF* for non-linear filtering and a clutter aware *JPDA*, the final pipeline is named Interacting Multiple Model Unscented Kalman Filter with Joint Probabilistic Data Association (IMM-UKF-JPDA).

We adapted this approach from [117, 126] to get a well performing baseline – together with the adapted 3D implementation of Bewley et al. [14] – on maneuvering targets in cluttered environments. Additionally, it builds the basis of our investigations on neural networks in combination with Bayes filters. However, we additionally implemented a less complex version by replacing the *JPDA* with a Global Nearest Neighbour (GNN) approach. It reduces not only the complexity, but also the tracker's runtime and offers the possibility for another association scheme based on *LSTMs*.

Moreover, we make use of the ego-vehicle velocity obtained from the *IMU* data of the KITTI dataset. In contrast to the filter explanations in Section 3.2.1 and Section 3.2.2, where we dropped the control input, we now use the parameter $\mathbf{u}_t$ of each time step to add the ego-motion as an artificial input to each tracker. All these variants, including *IMU* data or not and *JPDA* or *GNN* association, as well as the related filter parameters, are controllable by an external configuration file. We do not list all the filter parameters used for the different models, because there are many of them. Instead, we refer to Schreier [126], which is a good guideline of how to choose them properly.

5.2.1 Multiple Dynamic Models

The Interacting Multiple Model (*IMM*) (see Section 3.2.3) makes use of multiple dynamic models which should cover the motion behavior of all traffic participants. They are used to perform predictions for each individual filter. We decided to use a set of $\mathcal{M} = \{M_j\}_{j=1}^{r}$ models with $r = 3$, including Constant Velocity (CV), Constant Turn-Rate Velocity (CTRV) and Random Motion, as proposed in [117, 126]. While most motions are constant in speed and direction, the *CTRV* model is, for example, responsible for movements around road crossings. All kinds of motions which are not represented by both mentioned models, are processed by the random model, *i.e.* static targets and most

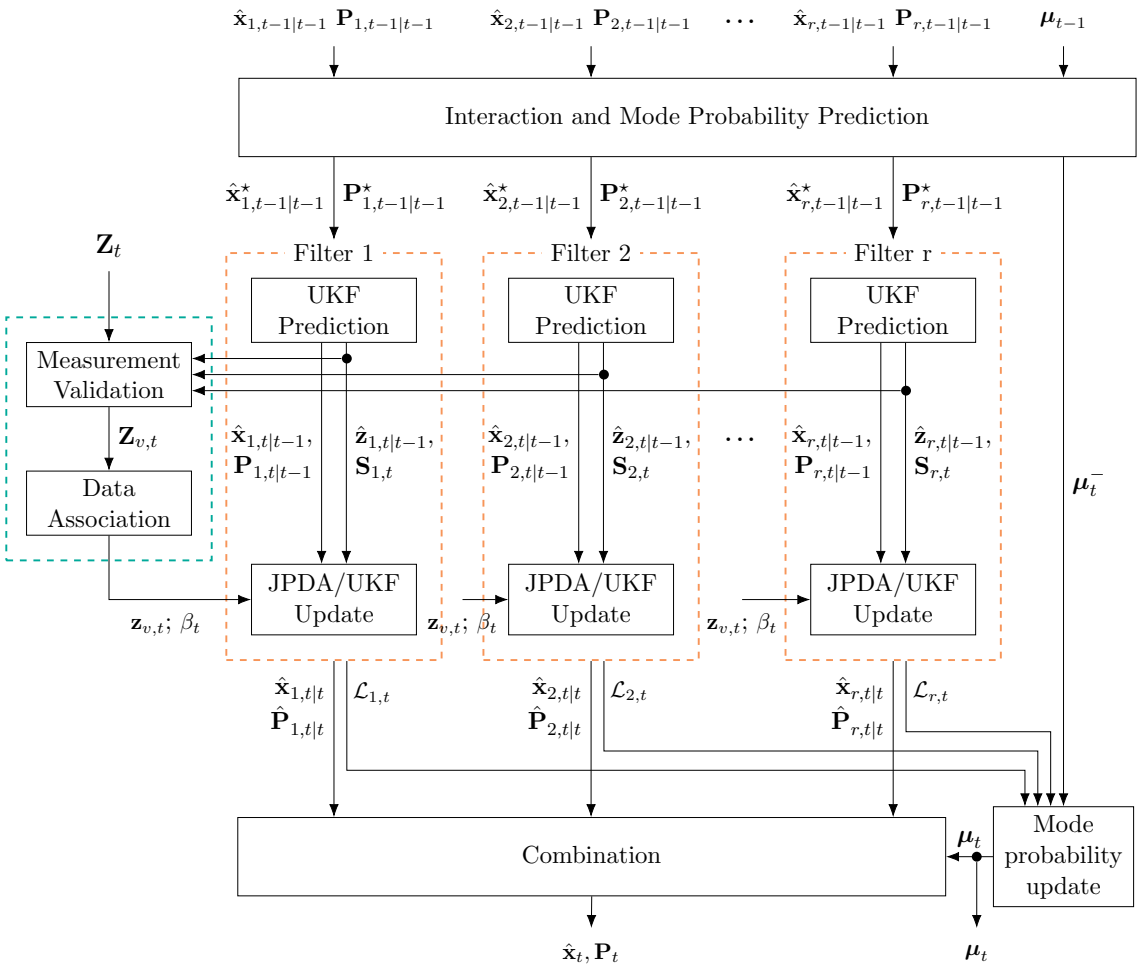


Figure 5.3: Overview of IMM-UKF approach with JPDA, Hungarian or LSTM update. All individual filter parts are derived in Chapter 3. The solid black part can be interpreted as the Interacting Multiple Model (IMM) filter, whereas the dashed orange blocks between represent the Unscented Kalman Filter (UKF) combined with either a *JPDA* or standard *UKF* update. The measurement validation and data association is a separate block (dashed green). Notice that we have omitted all fixed parameters. Adapted from [126].

pedestrians. A detailed explanation and derivations of used motion models can be found in [88].

Constant Velocity

One of the most common motion models is the Constant Velocity (CV) model. It represents all movements with a nearly constant velocity and heading direction. This means that the angular velocity is zero. We can write the motion model as a function of the state

vector as

$$\mathbf{f}_{CV,t}(\mathbf{x}_t) = \begin{bmatrix} x_t + v_t\ \Delta t\ \cos(\omega_t) \\ y_t + v_t\ \Delta t\ \sin(\omega_t) \\ z_t \\ \omega_t \\ v_t \\ 0 \\ D_{bb,t} \end{bmatrix}, \tag{5.4}$$

where x_t, y_t and z_t denote the position coordinates, Δt denotes the time difference and $D_{bb,t}$ represents the bounding box dimensions w_t, l_t, h_t and its yaw angle φ_t. Within this model, all values are constant except the movement in x and y direction, influenced by the velocity v_t and the steering angle ω_t. Notice, we assume no vertical motion or change in the bounding box size.

Constant Turn-Rate Velocity

Traffic participants performing a left or right turn, *i.e.* at a crossway, mostly expected in an urban environment, are represented best by the Constant Turn-Rate Velocity ($CTRV$) model. Characterized by a constant velocity and turning rate, we write it again as a function of the state vector

$$\mathbf{f}_{CTRV,t}(\mathbf{x}_t) = \begin{bmatrix} x_t + \frac{v_t}{\dot{\omega}_t}\ (-\sin(\omega_t) + \sin(\Delta t\ \dot{\omega}_t + \omega_t)) \\ y_t + \frac{v_t}{\dot{\omega}_t}\ (+\cos(\omega_t) - \cos(\Delta t\ \dot{\omega}_t + \omega_t)) \\ z_t \\ \omega_t + \Delta t\ \dot{\omega}_t \\ v_t \\ \dot{\omega}_t \\ D_{bb,t} \end{bmatrix}, \tag{5.5}$$

where x_t, y_t and z_t denote the position coordinates, Δt denotes the time difference and $D_{bb,t}$ represents the bounding box dimensions w_t, l_t, h_t and its yaw angle φ_t. The x and y coordinates now depend on the velocity v_t, the angular velocity $\dot{\omega}_t$ and the steering angle ω_t. Furthermore, the steering angle ω_t gets updated as well taking the current angular velocity $\dot{\omega}_t$ into account.

Random Motion

Static objects, *i.e.* targets without motion, are modeled by the Random Motion model. This could be a parking car, a resting pedestrian or a traffic participant which waits at a crossway. The static behavior is meant in relation to the real world coordinate system and not to the ego-vehicle movement. To model such a behavior, the function is simply

the state vector

$$\mathbf{f}_{RAND,t}(\mathbf{x}_t) = \begin{bmatrix} x_t & y_t & z_t & \omega_t & v_t & \dot{\omega}_t & w_t & l_t & h_t & \varphi_t \end{bmatrix}^T, \tag{5.6}$$

which is normally used in combination with higher process noise $\mathbf{Q}_t$. The model may also get active, if the observed targets - mostly pedestrians - move slowly or in frequently changing directions.

Artificial Input (Ego-Motion)

Because we model the tracked targets w.r.t. to the ego-vehicle, the resulting trajectories do not represent the true motion behavior of each target w.r.t. the world or global coordinate system. Instead, all paths are influenced by movements of the ego-vehicle. In general, this is no problem for the implemented filters. However, if the ego-vehicle suddenly changes its velocity or performs a turn, the filters are at risk of drifting away. Thus, we make use of the *IMU* data included in the KITTI dataset to overcome this issue. Therefore we use the artificial input

$$\mathbf{u}_t = \begin{bmatrix} -\frac{v_t}{\dot{\omega}_t}\left(-\sin(\omega) + \sin(\dot{\omega}\ \Delta t + \omega)\right) \\ -\frac{v_t}{\dot{\omega}_t}\left(+\cos(\omega) - \cos(\dot{\omega}\ \Delta t + \omega)\right) \\ \mathbf{0} \end{bmatrix}, \tag{5.7}$$

inside the individual filters, influenced by the ego-vehicle's velocity v_t, turn rate $\dot{\omega}_t$ and steering angle ω_t. The zero vector $\mathbf{0}$ represents all other values of the state which do not change by the ego-motion.

5.2.2 Interacting Multiple Model Unscented Kalman Filter

With the predefined motion models and the theory elucidated in Chapter 3, we have all tools needed to walk through one cycle of recursively estimating states and mode probabilities of a tracked target. Because of the modular design we decouple the data association from the closely related filter-update, which happens inside the tracker module, ensuring compatibility to all implemented assignment algorithms.

We extended the FilterPy implementations of *IMM* and *UKF* for our purposes which we explain in the following processing steps: Interaction and Mode Probability Prediction, Non-linear State Estimation, Data Association and Filter Update. The detailed steps are illustrated in Figure 5.3.

Interaction and Mode Probability Prediction

The *IMM* is implemented with $r = 3$ *UKF* filters, each containing a different motion model: *CV*, *CTRV* and Random Motion. It performs an interaction step (recall Equation (3.49)) where all individual state $\hat{\mathbf{x}}_{i,t-1|t-1}$ and covariance $\mathbf{P}_{i,t-1|t-1}$ estimates of the $i = 1, \ldots, r$ filters of the previous time step are mixed into an initial state $\hat{\mathbf{x}}^{\star}_{j,t-1|t-1}$

and covariance $\mathbf{P}^{\star}_{j,t-1|t-1}$ for each filter $j = 1, \ldots, r$ of the current time step t. Therefore, we have to calculate mixing probabilities $\mu_{i|j,t-1}$ (recall Equation (3.50)) using predicted mode probabilities $\mu^{-}_{j,t}$ (recall Equation (3.51)).

Non-linear State Estimation

Afterwards, each filter $j = 1, \ldots, r$ performs a prediction step based on its motion model producing the predicted state $\hat{\mathbf{x}}_{j,t|t-1}$ and corresponding covariance matrix $\mathbf{P}_{j,t|t-1}$. Moreover, we obtain the predicted measurement $\hat{\mathbf{z}}_{t|t-1}$ and corresponding innovation covariance matrix $\mathbf{S}_t$. The exact filter formulas of the *UKF* used for state and measurement prediction can be looked up in Section 3.2.2.

The following update procedure depends on the association algorithm, selected in the configuration file. We provide two different methodologies. On the one hand, we have the exclusive assignment strategy, where each target gets associated exactly to one or no measurement. On the other hand, we support a joint probabilistic approach combining all measurements within a certain area. While the former is implemented using the Hungarian algorithm and yields association pairs of tracks and measurements, the latter is realized via *JPDA* in combination with *Gating* (recall Section 3.3.1) and delivers association probabilities for each filter and track-measurement combination.

Filter and Mode Probability Update with Exclusive Assignment

In order to obtain the posterior mean $\hat{\mathbf{x}}_{t|t}$ and covariance matrix $\mathbf{P}_{t|t}$, each tracker performs a simple update step of the *UKF* as in Section 3.2.2, given by Equation (3.39). This action is only performed if an associated measurement is available. Otherwise, it will be skipped which leads to higher uncertainty expressed by the covariance matrix.

In addition to the state and covariance of each filter, we update the mode probabilities inside the *IMM* block (recall Equation (3.53)). Therefore, we use the filter likelihoods (recall Equation (3.52)) which represent how well the assigned measurement $\mathbf{z}_t$ fits each filter or motion model, respectively.

Filter and Mode Probability Update with Joint Probabilities

For measurements associated by *JPDA*, the update for each individual filter differs in contrast to exclusive methods. The tracker receives association probabilities, describing the influence of each measurement on the update. These association probabilities are separately available for each filter.

In order to obtain the posterior mean $\hat{\mathbf{x}}_{t|t}$ and covariance matrix $\mathbf{P}_{t|t}$, each tracker performs an update step following Equations (3.63) and (3.64) defined for the Probabilistic Data Association (PDA) filter.

At this point, incorrect bounding box orientation estimates, caused by the object detectors as well as clutter measurements with mostly different orientations lead to an

unpredictable behavior. As a consequence, we check, if the innovation value within the *JPDA* update implementation of the *UKF* is larger than 90 degrees. This mostly happens when an object detector predicts the bounding box orientation in the opposite direction (180 degree twisted). If this is the case, we allow a maximum innovation of 30 degrees. The reason for that is twofold. On the one hand, we are interested in small changes if there happens a wrong orientation estimate. On the other hand, if the initial guess was wrong, we want to change the orientation to the right direction as fast as possible.

The mode probability update follows the same equation as the exclusive assignment method. However, because the *JPDA* takes multiple measurements into account, we build the Gaussian mixture likelihood $\mathcal{L}_{j,t}$ for each individual *UKF* model j as

$$\mathcal{L}_{j,t} = \frac{1 - (P_D P_G)}{V_t^{N_v}} + \frac{P_D V_t^{(1-N_v)}}{N_v \sqrt{\det(2\pi\, \mathbf{S}_{j,t})}} \sum_{m=1}^{N_v} \exp\left(-\frac{1}{2} (\mathbf{z}_t - \hat{\mathbf{z}}_{j,t|t-1})^T \mathbf{S}_{j,t}^{-1} (\mathbf{z}_t - \hat{\mathbf{z}}_{j,t|t-1}) \right), \tag{5.8}$$

with detection and gating probability P_D and P_G, respectively, the validation region volume V_t defined in Equation (3.56), the number of valid measurements N_v and the summed model likelihood for each valid measurement.

Combination

The last step within one prediction and update cycle of the *IMM-UKF* is combination. Here, we merge the posterior mean and covariance of each filter to a single state $\hat{\mathbf{x}}_t$ and covariance $\mathbf{P}_t$ following Equation (3.54). These values represent the final result of one cycle and thus, are inserted into the state history by the track manager.

5.2.3 Data Association

Hungarian Algorithm

For data association, Global Nearest Neighbour (*GNN*) approaches are heavily used in tracking implementations. One representative solution is the Hungarian algorithm, which performs global optimization on a cost matrix. While low scores imply that a pair would make a good match, high scores represent independence. However, in the case of data association, we have similarity scores or probabilities between measurements and tracks instead of costs. In order to obtain costs, required by the algorithm, we multiply the similarity scores with -1.

Hence, the values of our cost matrix are the negative *IoUs* between tracks and measurements. Negative, because the *IoU* lies in the range $[0, 1]$. By negating the *IoU*, 0 is the maximum cost, describing a completely disjoint pair, and -1 is the lower bound which stands for a total overlap. For such algorithms, it is common to introduce an overlap threshold which ensures that only bounding boxes with a minimum overlap are considered.

This avoids the association of detections emerging from clutter. Typical threshold values are in the range $[0.1, 0.5]$.

The association function returns three lists. The matching pairs containing linked tracks and measurements, the unmatched tracks and the unmatched detections. Important for the update step are only the matches. Unmatched detections are used by the track management module to initiate new tracks, while unmatched tracks are eventually terminated.

Joint Probabilistic Data Association

A more sophisticated methodology is the Joint Probabilistic Data Association (*JPDA*). It requires some preliminary measurement validation called *Gating* (recall Section 3.3.1). In practice we use an elliptical validation gate defined by Equation (3.55). Following Schreier [126], we use the innovation covariance matrix $\mathbf{S}_t$ with the largest determinant for validation, assuming that its region covers most of the relevant measurements. Further, we exclude the Random Model innovation covariance from the validation process because of its high uncertainty. The validation region is then calculated by

$$V_t = V(q)\,|\gamma S_{j_r,t}|^{\frac{1}{2}}\,, \tag{5.9}$$

with

$$j_r = \arg\max_{j \in M} |\mathbf{S}_{j,t}|, \tag{5.10}$$

where j_r denotes the model with the largest innovation covariance matrix determinant and γ denotes the gating threshold. $V(q)$ is the q-dimensional unit hypersphere, recall Equation (3.57). The remaining N_v measurements $Z_{v,t} = \{\mathbf{z}_{m,t}\}_{m=1}^{N_v}$ passing the validation gate are part of the following calculation of association probabilities.

Therefore, we set up a validation matrix containing all measurements as rows and all tracks as columns. If a measurement passes the validation gate of a certain track, the row value of this track is set to 1, otherwise it is 0. With this validation matrix, the algorithm builds clusters of measurements with probably overlapping regions (recall Figure 3.3 for an example). Each cluster is then separately processed by each filter $j = 1, \ldots r$. Notice, all filters use the same validation gate. This is necessary because the likelihoods have to be conditioned on the same set of measurements [8].

Based on each cluster, we generate hypothesis trees which represent all possible permutations of associations. Afterwards, we use these hypotheses to calculate joint association probabilities. Then, we receive a mapping from clusters to a list of involved tracks and the corresponding association probabilities. Furthermore, we get all measurements whose row in the validation matrix contains zeros only. These measurements are used to start new tracks. A detailed *JPDA* explanation can be found in Section 3.3.3.

5.3 Multi-Object Tracking Networks

Inspired by the work of Milan et al. [103], we implemented and trained *RNNs* to improve or replace parts of the combined filter approach. The reasons for that are manifold. One disadvantage of Bayes filters is that we need to design fixed motion models which we have to combine with the *IMM* or use in a stand alone manner with an *UKF* or linear *KF*, respectively. Thus, wrong assumptions or models which do not cover the motion behavior of the tracked targets lead to poor predictions and, consequently, to tracking failures.

Additionally, the parametrization of such filters is complex and depends on the environment. For example, filter parameters which work well in an urban environment - with lots of parked cars, pedestrians and many road crossings - may not be the perfect choice for motorways. The traffic participants and their motion behavior, *e.g.* velocity or steering angle, differ significantly between these two settings.

Another issue is the data association performance regarding time and precision. Hand-crafted similarity scores, *e.g.* *IoU* or shortest distance, are not always a good decision. While the exclusive association strategy with hand-crafted similarity scores does not work for filters with inaccurate precisions, *e.g.* due to the lack of detections, joint association approaches are very slow because of the exponential runtime with growing number of targets and measurements. In addition, they do not work well for estimating the bounding box orientation.

Because of these issues, we trained three different networks. The first two, on the one hand, perform state prediction and data association for a variable number of tracks and measurements. The third one, on the other hand, was designed to solve the whole *MOT* procedure, including both previous networks, extended with state update and track management. We describe each network in three steps. First, we introduce the network structure. Second, we elucidate data preprocessing and training parameters. Finally, we explain where and how we used the trained network in our tracking framework.

5.3.1 State Prediction Network

For state prediction we created and trained a simple *LSTM* model. The aim of this network is to learn motion patterns from the training data and predict the next state from a given state sequence. Afterwards, the network serves as a single motion model inside an *UKF*.

Architecture

The network consists of four input neurons, two hidden layers - containing 256 *LSTM* cells each - and two output neurons, yielding the center coordinates x and y of the bounding box ground area. The architecture is listed in Table 5.2. As input the network receives center coordinate x and y of the bounding box ground area as well as Δx and Δy, representing

Layer	Type	Input	Output	Activation
1	LSTM	4	256	-
2	LSTM	256	256	-
3	fully-connected	256	2	identity

Table 5.2: Prediction network architecture. Because *LSTM* cells have more than one activation function, we list only activations of non-temporal elements.

the ego vehicle movement of one time step. For the reduction of our *LSTM* output we applied a projection layer after each time step activated by the identity function.

Training

Because we predict coordinate values, which is a regression problem, we applied the linear output activation to the last layer. While training, we need an additional loss layer which calculates the difference of prediction and ground truth values. For this kind of problems, the Mean Squared Error (MSE)

$$E(y,t) = \frac{1}{2D}\sum_{i=1}^{D}(y_i - t_i)^2, \tag{5.11}$$

with network outputs y_i and ground truth or target value t_i respectively, is a common choice.

We decided to implement a many-to-many network, where each time step of the *LSTM* produces a prediction. Another option is the many-to-one structure, where the network produces one single output after processing the whole input sequence. The idea behind this decision is, that the network should learn to produce meaningful or "careful" outputs when only little information is available.

For the prediction model we decided to separate all available track trajectories of the training set. This means that we do not make use of dependencies between different traffic participants, *e.g.* when one car follows another they are not allowed to occupy the same space without violating geometric constraints. Therefore, we extracted all available tracks and saved them in a list.

Each training iteration yields sequences $\mathbf{S}_i \in \mathbb{R}^{B\times T\times D}$ with batch size B and sequence length T containing tracking states of dimensionality $D = 4$, randomly chosen out of all available tracks starting at a random position. To ensure equal chances for each subsequence, the picking is weighted by the total length of each track. In order to achieve better generalization while training, we perform dataset augmentation. With a probability of 0.5 we apply a random shift in x-direction, a random shift in y-direction, or a mirroring at the x-axis. Additionally, we add random noise to the tracks. These augmentations are only applied if the manipulated track is still in the minimum and maximum range of all

tracks. Finally, the input data gets normalized to lie within the range $[-1, 1]$ which should prevent the network from running into saturation.

Network optimization is done with the Adam [80] optimizer with a learning rate of 0.003 and a decay of 0.95 after every 20k iterations. We train the network with an early stopping criterion which is fulfilled when the mean absolute error on the validation set does not decrease for a certain number of training iterations.

Inference

For inference, we embedded the network within an *UKF* tracker combined with Hungarian data association. Because of its ability to handle nonlinear dynamic functions without knowing its derivative, we can simply replace the filter's motion model with the trained network. Therefore, we have to adapt the internal representation of the filter. An additional list containing the last N sigma points is necessary to serve the network in each time step with correct input data. With this change, the network is able to do a state estimate based on the prediction network. Thus, all sigma points can be calculated in one step, by stacking up an appropriate batch. Because we are only interested in the last state, the network is used in a many-to-one manner for inference.

5.3.2 Variable Data Association Network

In contrast to the prediction network, the variable data association model works in a frame-based mode. This means that while training, the whole frame information, containing all active tracks and detections, is available to the network. As the name points out, this approach learns to associate a variable number of tracks and measurements. In particular, we adapted the work of Yoon et al. [160] and trained it on the KITTI dataset.

Architecture

The architecture is similar to the one explained in Section 4.4.2 and contains two parts. First, an encoder is responsible for encoding the input sequence into an encoded vector of size 64. However, we added an *RNN* layer in front of the dense layers and in turn removed one fully-connected layer. We think that the ability of *RNNs* to understand temporal

Layer	Type	Input	Output	Activation
1	RNN	$(T+1) \times D$	128	-
2	fully-connected	128	128	ReLU
3	fully-connected	128	64	tanh

Table 5.3: Implemented encoder network architecture. The RNN layer receives a batch of sequences containing the last T states and detections with D features each.

dependencies of the state sequence can improve the encoding procedure. Table 5.3 shows the used encoder layer structure. The network input is a batch - with variable size - of

measurement-to-track permutations. Each permutation contains the last T states and one detection. Both entries, states and detection, contain the same D features: Center coordinates x, y of the bounding box ground area and its bounding box dimensions, *i.e.* width w and length l. We do not make use of the detection confidence as proposed in the original model.

Second, this internal representation is used in the decoder to calculate a score for each pair of detections and tracks. Its architecture is shown in Table 5.4. The encoder output is fed into a Bidirectional Long Short-Term Memory (BLSTM) layer. It consists of

Layer	Type	Input	Output	Activation
4,5	BLSTM	64	128	-
6	fully-connected	128	64	ReLU
7	fully-connected	64	1	softmax

Table 5.4: Implemented decoder network architecture.

two connected *LSTM* layers fed with the input in forward and backward direction (recall Section 4.2.4). As an output, the *BLSTM* layer produces a concatenation of both layers. Hence, the *BLSTM* output is twice the size of its used hidden cells.

In contrast to the original implementation of Yoon et al. [160], we formulate the problem not as a regression task. Instead, we define the assignment of measurements to tracks as a multi-class classification task, requiring a *softmax* activation in the last dense layer. This leads to a more intuitive representation and returns probabilities for measurement-to-track assignments. Additionally, we applied a dropout wrapper to the *BLSTM* layer parametrized with a keep probability of 0.8 to achieve better generalization.

Training

For network training, we need an additional loss layer. Since the model solves a classification task, we apply the *cross-entropy* loss as explained in Equation (4.14). However, based on the classification formulation, the softmax function produces only probabilities of which measurement depends on which task. Hence, each row of the score matrix produces exactly a sum of one.

This violates the assumption that only one measurement per track is allowed, verified by a column sum smaller or equal to one (similar to the *JPDA* conditions in Equation (3.71)). Because of that, we multiply the loss with an additional penalty term depending on the column sums of the obtained score matrix. Each column with a sum bigger than one, except the first, contributes to the penalty multiplier. Thus, it is defined by the number of columns larger than one, times a predefined penalty factor. We exclude the first column, because it represents the *dummy* or *clutter* track, which allows associations to multiple measurements.

Because we use the training sequences as they are - frame by frame - we train the network in epochs, each using the whole training set once. Therefore, we randomly loop

through all training sequences, and therein, pick each start position once, in an unordered manner. This means, that each training sample is randomly chosen and contains the information of T consecutive frames. Hence, T denotes the sequence length while training. In contrast to the prediction model, we do not train our network on noisy ground truth data only. Instead, we replace all ground truth bounding boxes with the corresponding detections from our object detector, if available. These correspondences are found by the Hungarian algorithm using a cost matrix derived from the IoU of detections and ground truth objects. Additionally, we applied noise to the ground truth boxes without association. This imitates the real state estimates of the tracker.

One subsequence of T frames contains all tracks active within this timeslot, also tracks started or stopped within this subsequence. Hence, tracks which start within the subsequence are padded before with a default value and stopping tracks are constantly padded afterwards. This also includes false positive detections, generating a track with only one valid entry, the rest is filled with default values. These false positive tracks are handled as if they were regular ones. This means that the track position stays the same, except there is a stopping track beforehand which does not exist any more. In addition to that, we normalize the input to lie in the interval $[-1, 1]$. Thus, the chosen default value we

	a)	b)	c)	d)	e)
$\mathbf{x}_{t-4}$	$[2,\ 2,\ 2,\ 2]$	$[2,\ 2,\ 2,\ 2]$	$[x,\ y,\ w,\ l]$	$[x,\ y,\ w,\ l]$	$[2,\ 2,\ 2,\ 2]$
$\mathbf{x}_{t-3}$	$[2,\ 2,\ 2,\ 2]$	$[2,\ 2,\ 2,\ 2]$	$[2,\ 2,\ 2,\ 2]$	$[x,\ y,\ w,\ l]$	$[2,\ 2,\ 2,\ 2]$
$\mathbf{x}_{t-2}$	$[2,\ 2,\ 2,\ 2]$	$[2,\ 2,\ 2,\ 2]$	$[2,\ 2,\ 2,\ 2]$	$[x,\ y,\ w,\ l]$	$[x,\ y,\ w,\ l]$
$\mathbf{x}_{t-1}$	$[2,\ 2,\ 2,\ 2]$	$[2,\ 2,\ 2,\ 2]$	$[2,\ 2,\ 2,\ 2]$	$[x,\ y,\ w,\ l]$	$[2,\ 2,\ 2,\ 2]$
$\mathbf{x}_{t\|t-1}$	$[2,\ 2,\ 2,\ 2]$	$[x,\ y,\ w,\ l]$	$[2,\ 2,\ 2,\ 2]$	$[x,\ y,\ w,\ l]$	$[2,\ 2,\ 2,\ 2]$

Table 5.5: Data association track examples with $T = 5$ and default value 2. a) Clutter track which should be assigned to all false positive detections or new track detections. b) New initialized track. c) Track which gets terminated. d) Regular active track. e) False positive track existing already for three time steps.

use for the inputs is 2, which lies outside the input range and therefore, should be easier to recognize for the network. Table 5.5 shows possible track examples at a certain time index.

In order to obtain one input batch for the network, we build all the existing tracks, each containing its subsequence of T states. This includes also the clutter track representing all false positives. It is the first target in each batch and contains only default values. Afterwards, we generate permutations of these tracks with all existing detections for the chosen time frame and stack them together. Hence, this results in $(N+1)\times M$ permutations of N active tracks, including one clutter track, and M measurements. The encoder is able to process the whole batch in one step and the resulting encoded vectors are fed into the $BLSTM$ as one batch with $(N + 1) \cdot M$ time steps, *i.e.* 64 features each.

As proposed by Yoon et al. [160], we further trained the network to learn the initial cell state $\mathbf{C}_0$ and hidden state $\mathbf{h}_0$ of both *LSTM* layers inside the bidirectional structure. Hence, the initial cell and hidden state are initialized by

$$\begin{aligned} \mathbf{h_0} &= \tanh(\mathbf{W}_h \cdot \bar{\mathbf{e}} + \mathbf{b}_h), \\ \mathbf{C_0} &= \tanh(\mathbf{W}_c \cdot \bar{\mathbf{e}} + \mathbf{b}_c), \end{aligned} \tag{5.12}$$

where

$$\bar{\mathbf{e}} = \frac{1}{(N+1)\ M} \sum_{i=1}^{(N+1)\ M} \mathbf{e}_i, \tag{5.13}$$

denotes the mean of all encoded vectors. The weight matrix $\mathbf{W}$ and the bias term $\mathbf{b}$ represent one dense network layer with input $\bar{\mathbf{e}}$ for the initial hidden and cell state.

Network optimization is again done with the Adam [80] optimizer with a learning rate of 0.0001. We train the network with an early stopping criterion which is fulfilled when the validation accuracy does not increase for a certain number of training epochs.

Inference

For inference, we embed the trained model into our data association module. In addition to the current state, each tracker keeps a list of T previous states, initialized with default values and the current predicted state $\mathbf{x}_{t|t-1}$ at the last sequence position t. Active tracks, including the dummy track, combined with all available detections, build the network input for the next data association step.

Because the resulting score matrix does not take care of one-to-one assignments, we apply the Hungarian algorithm on it, excluding the first column which is reserved for the clutter or dummy track, respectively. Another post-processing step removes associations with a score lower than 0.5. All measurements which are assigned to clutter with a probability higher than 0.5 are also removed. The remaining matches are considered correct. Unmatched tracks and measurements are handled by the track management.

5.3.3 End-to-End MOT Network

The missing parts to train and perform *MOT* in an end-to-end manner are state update and track management. Hence, we propose a network which combines the explained state prediction and data association models, extended by an update network.

The following approach is an adaption of [103], handling a variable number of measurements and tracks. In addition, we do not treat each track separately, but combine them in an *BLSTM* layer, inspired by Alahi et al. [3], but with a simpler model (*BLSTM* instead of "social pooling").

Unfortunately, the proposed model is not competitive to the previous models regarding its tracking performance. The reasons for this are manifold. One issue is definitely the update step. While training, we figured out that the prediction step can be learned very

well. However, performing the update with received detections does not noticeably improve the state estimate. Sources of error are the data association which does not fulfill the one-to-one constraint and the handling of detections in general. We make use of one detection per time step and do not explicitly calculate an encoding - similarly to the tracking state - which contains additional information, *e.g.* how good previous detections improved the final state in the past.

Another source of error is the general model design. We take track sequences of length T as an input and assume they reflect the true or desired behavior of our model. Based on that information, we train the network to produce results for the next time step. In contrast to that, a sequence generating network receives one state value per *RNN* iteration only. It performs a prediction, followed by data association and updates the states with the provided detections. Afterwards, it delivers the new states, which are in turn the input for the next *RNN* time step. Such a network could implicitly learn relations between predictions and updates. However, it is much harder to implement and would need much more data to train.

Architecture

In order to perform prediction, data association and update within one network, we need not only the state sequences as an input, but also detections of the current time step. Therefore, we have six input neurons connected with an *RNN* encoder, similar to that in the data association model. However, these neurons receive only the state sequence without measurements, as shown in Table 5.6, where $D = 6$ denotes the feature size and T is the sequence length. Each state of size D contains center coordinates of the bounding box ground area x, y and its bounding box dimensions with w and length l, completed by the ego-vehicle movements Δx and Δy.

Layer	Type	Input	Output	Activation
1	RNN	$T \times D$	128	-
2	fully-connected	128	128	ReLU
3	fully-connected	128	64	tanh

Table 5.6: Encoder structure of our end-to-end network.

Afterwards, the encoded vector is used for state prediction within a *BLSTM* layer followed by two fully-connected projection layers, shown in Table 5.7. The last projection

Layer	Type	Input	Output	Activation
4,5	BLSTM	64	128	-
6	fully-connected	128	128	ReLU
7	fully-connected	128	4	identity

Table 5.7: Prediction structure of our end-to-end network.

layer, consisting of four units, gets activated by the identity function. As a consequence, we receive state predictions as intended. This prediction layers are not absolutely necessary, because one could also infer a prediction by adding an output layer on top of the encoder. Notwithstanding, we think that the bidirectional structure can improve performance for interacting targets, *e.g.* parking cars (same motion), cars driving next to each other and so on. In order to avoid overfitting and reach better generalization, we apply a dropout wrapper to the *BLSTM* layer, again with a keep probability of 0.8. In addition, we learn the internal state for this layer as explained in Equations (5.12) and (5.13).

At this point, the network holds encoded state vectors for each track only. Therefore, we add another four input nodes, receiving the detections and build a Cartesian product - each encoded vector of a track track gets concatenated with each measurement - resulting in $N \times M$ permutations. These permutations are the input forwarded to the data association network. As explained in the previous section, it contains a *BLSTM* layer (now 8 and 9) with two following dense layers (now 10 and 11) as listed in Table 5.4. The result is again a score matrix. By applying the softmax function, we get association probabilities, describing the probability of how good a detection can be assigned to each track. Again, we apply a dropout wrapper keeping 80 percent of the nodes and learn the internal states, this time with the measurement-track permutation mean.

Finally, the network receives another input, representing the track existence probability of the last time step for each track. In order to perform an update step, we generate track-measurement permutation pairs, this time using track predictions, where each measurement is multiplied with its corresponding association probability. Afterwards, we reorder the permutation list of dimension $(N + 1)$ $M \times 2D$, to get a batch of sequences, each containing all permutations for one track with dimension $N \times M \times 2D$. Multiplied with the track-existence probability of the last time step, each sequence is the input of another *BLSTM* layer as shown in Table 5.8. Again with two following fully-connected

Layer	**Type**	**Input**	**Output**	**Activation**
12,13	BLSTM	2 D	128	-
14a	fully-connected	128	128	ReLU
15a	fully-connected	128	4	identity

Table 5.8: Update structure of our end-to-end network.

projection layers with linear activation in the last layer. Dropout with a keep probability of 0.8 is also applied to this *BLSTM* layer, initialized with learned states depending on the predicted track states.

Two separate projection layers, receiving the concatenated output of the *BLSTM* layer (12, 13) predict the track existence probability for the current tracks. Because this is a single value for each track, the last output layer contains only one node with sigmoid activation, as listed in Table 5.9, to produce probabilities which are in the range $[0, 1]$. Together with both state outputs, containing predictions and updates of the bounding

Layer	Type	Input	Output	Activation
14b	fully-connected	128	128	ReLU
15b	fully-connected	128	1	sigmoid

Table 5.9: Track existence structure of our end-to-end network.

box ground plane center coordinates x, y and its dimensions w and l, the track existence probability builds the entire network output and thus, closes one *MOT* cycle.

Training

For the end-to-end architecture we need different loss layers for each problem. Beginning with the state prediction and update part, the *MSE* loss is an appropriate choice for both. Whereas, for data association, we apply the cross-entropy loss for multiple classes as usual. In addition, the data association loss is again penalized for ignoring the one-by-one assignment constraint (see Section 5.3.2). The track existence probability decides between two classes, active or not. As a consequence, the binary cross-entropy loss is used. Furthermore, we use the same regularization term for the track existence probability as proposed by Milan et al. [103], forcing the model to produce smooth probability switches.

Hence, to obtain the total loss, we sum up all calculated losses, including prediction, update, data association and existence probability loss as

$$E(\dots) = \overbrace{\frac{1}{2N}\sum ||x^* - \hat{x}||^2}^{\text{prediction}} + \overbrace{\frac{1}{2N}\sum ||x^* - x||^2}^{\text{update}} + E_\varepsilon + \hat{\varepsilon} + E_{DA}\ \Psi_{PEN}, \tag{5.14a}$$

$$E_\varepsilon = \varepsilon^* \log(\varepsilon) + (1 - \varepsilon^*) \log(1 - \varepsilon), \tag{5.14b}$$

$$E_{DA} = -\sum_{n}^{N} \mathbf{a}^*_{0,n} \log(a_{0,n}), \tag{5.14c}$$

where the prediction and update term is straight forward, the existence probability loss E_ε is smoothed with the regularization term $\hat{\varepsilon}$ and the data association loss E_{DA} is a multi-class cross-entropy error function, multiplied with the previously mentioned penalty term Ψ_{PEN}.

The training and validation data is processed and used similar to that of the data association network. We train in epochs, where each epoch contains all training sequences and for each iteration, we pick a random sample regarding sequence and start position. Again, we provide a batch of variable length to the network, depending on active tracks within the selected sample, containing the last T states for each track. Included are also false positive tracks. Both, real and false positive tracks, consist of noisy ground truth values with entries replaced by detections if they are assignable with the Hungarian algo-

rithm. In contrast to the data association network, we separately provide the detections in addition to track existence probabilities.

Each single network can be trained step-by-step without influencing the previous one. This means, that we are able to train the state prediction network first. Second, we fix the weights and train the data association part. Finally, the model is trained for updating the states with fixed weights for both previous models. Therefore, all individual loss functions are used and evaluated on the ground truth sequences. Afterwards, we are able to load all network weights and fine-tune the model, by training the whole network without fixing weights.

The network can also be trained in an end-to-end manner without training each part of the network first. However, this could lead to suboptimal solutions, because of the complex loss function and the small amount of data available in the used dataset. One way to improve the learning could be a weighting of each loss term. Notwithstanding, we performed a separate training of each network part followed by an end-to-end fine-tuning within this thesis.

Network optimization is again done with the Adam [80] optimizer with a learning rate of 0.0001. We train the network with an early stopping criterion which is fulfilled when the combined validation accuracy of all loss terms does not increase for a certain number of training epochs.

Inference

One of the biggest advantages of end-to-end models is that they are mostly easy to use. We just have to prepare the input data, containing state sequences of active tracks, detections and track existence probabilities, all of time $t-1$. Then, the model yields a state prediction for each track, a score matrix containing assignments of all detections to active tracks, updated states for each track and existence probabilities, each for the current time step t.

First, we evaluate the data association scores by performing the Hungarian algorithm to get one-to-one assignments. Furthermore, we again remove pairs with association probabilities smaller than 0.5 and measurements assigned to clutter with a probability larger than 0.5. Now, we can perform the track state updates in two different ways:

1. Update all tracks with the received update value.

2. Update all assigned tracks with received update values and all others with the received prediction values.

While we assume for the first approach that the updater learns to act correctly even with uncertain associations, the second one is more restrictive as we do not trust the updater in case of uncertain assignment situations. Because we first trained each network part separately and the data association model does not implicitly solve the one-to-one assignment problem, we decided to use processed prediction values for unassigned tracks.

Finally, track management deals with unmatched tracks and detections. Instead of counting consecutive updates, we use the track existence probabilities. Each unmatched detection gets a new potential track and is part of the next time step, starting with an existence probability of 0.5. Afterwards, all tracks with an existence probability larger than 0.7 switch into the active state and those with an existence probability smaller than 0.4 are terminated and thus, removed from the list of all tracks.

6 Evaluation

Contents

In this chapter, we present a detailed performance evaluation of our Multiple Object Tracking (*MOT*) framework. Therefore, we apply the implemented models on the publicly available KITTI dataset [48, 49]. First of all, we briefly describe the provided raw data and tracking data, their differences, as well as the selected sequences.

For the first experiments, we make use of the KITTI raw dataset, containing different scenes, *i.e. Road*, *City* and *Residential*, and various object classes, *i.e. Car*, *Van*, *Truck*, *Pedestrian*, *Sitting Person*, *Cyclist*, *Tram* and *Misc*. Because we want to simulate real traffic scenarios, we do not focus on a specific scene and do not filter out targets which are occluded or outside the laser scanner's range. Instead, we tried to find a balanced set of sequences, containing different environments, traffic participants and situations, where we make use of all available detections.

To compare the implemented models, we start with the overall tracking performance. Furthermore, we focus on the behavior of each tracker when we skip time steps, which is normally used to increase speed. This means, that we use only a subset of the available detections, *e.g.* detections of each second or third time step. Another evaluation reflects the tracking performance on the three main classes: *Car*, *Pedestrian* and *Cyclist*.

Afterwards, we evaluate our models on the KITTI tracking dataset which is partly included in the KITTI raw dataset and is commonly used in literature for evaluation. Because the tracking test set used for the online leader board does not provide ground truth labels, we evaluate on a commonly used validation split of the training data. We again start with an overall tracking performance comparison between all models where we

take all object classes into account. In addition to that, we investigate the performance of our models by using two different object detectors: Frustum PointNets [116] and PointR-CNN [129]. Moreover, we discuss state-of-the-art approaches and how we can interpret our results w.r.t. the online KITTI leaderboard[1]. Finally, we discuss individual sequences in detail, where we describe properties of the different approaches.

For all the practical experiments we use the Classification of Events, Activities and Relationships (CLEAR) MOT[2] measures [12] and additional quality measures [89], recall Section 2.5. Additionally, we provide the average runtime, given as Frames Per Second (FPS). We run all experiments on a PC with an Intel® Core™ i7 - 8700K processor at 3.7 GHz, equipped with 32 GB RAM and a Nvidia® GeForce GTX 1080 graphics card with 8 GB video memory, running a 64 bit Ubuntu 16.04 LTS operating system.

6.1 Evaluation on the KITTI Dataset

The Karlsruhe Institute of Technology and the Toyota Technological Institute of Chicago provide the publicly available KITTI Dataset (recall Section 2.6) containing, on the one hand, a raw dataset, sorted by category, *i.e. Road*, *City* and *Residential*. The tracking dataset, on the other hand, consists of 21 training sequences and 29 test sequences which partly overlap with the raw dataset, as listed in Table 6.1. Because there is no ground

Training (our)		**Validation**	
Tracking	Raw	Tracking	Raw
0001	0009	0000	0005
0002	0011	0003	0013
0005	0015	0004	0014
0007	0022	0006	0018
0008	0032	0010	0056
0009	0036	0012	0060
0011	0059	0013	0091
0017	-	0014	-
0018	-	0015	-
0019	-	0016	-
		0020	-

Table 6.1: The table shows a mapping between tracking training data sequences and raw data sequences, as well as the training/test split proposed in [112]. We can see that 14 out of the 21 sequences of the public KITTI training dataset are also contained within the raw dataset.

truth available for the test sequences, we make use of training sequences only. Hence, we split the 21 training sequences following Osep et al. [112].

[1] `http://www.cvlibs.net/datasets/kitti/eval_tracking.php` (accessed September 24, 2019)
[2] `https://github.com/cheind/py-motmetrics` (accessed September 24, 2019)

For the evaluation on the raw dataset we have chosen the sequences listed in Table 6.2. At first sight, it might look unbalanced. However, we have to take care of different cir-

City	Residential	Road
0001	0020	0070
0005	0039	
0014	0064	
0018		
0060		
0084		

Table 6.2: Validation sequences listed by category.

cumstances. First, the recorded sequences are unbalanced w.r.t. the categories (18 City, 13 Residential, 7 Road). Second, for training our models we need enough data of each category and object class. And third, we do not want to evaluate on sequences contained in the training set.

6.2 Comparison of Presented Models

To ensure a fair comparison between all models in different scenarios, we do not change the model parametrization. This means, that we tuned each model to perform well in the overall performance evaluation w.r.t. Multiple Object Tracking Accurency (MOTA) and keep the parameters fixed afterwards. Track management threshold parameters can be found in Table 6.3. Furthermore, we use Frustum PointNets [157] as our standard 3D

		IMM-UKF			RNN-UKF	
	SORT	**GNN**	**JPDA**	**ENC-DEC**	**GNN**	**ENC-DEC**
time-since-update	3	2	3	2	2	2
hit-streak	2	1	1	1	1	1
suspend counter	-	2	2	2	2	2

Table 6.3: Track management threshold parameters for all trackers. While the *hit-streak* threshold is responsible for the initialization of tracks, the threshold for *time-since-update* and *suspend counter* influence the termination of tracks. Each model except SORT is based on an IMM-UKF or RNN-UKF and named after its data association procedure. Each value defines the threshold for the listed parameter.

object detector. As a baseline we use the SORT [14] tracker which we modified to work in a 3D environment. In all following tables, the best score of each column is written in **green** and the second best score is written in **blue**. We do not highlight Partly Tracked (PT) targets, because Mostly Tracked (MT) and Mostly Lost (ML) targets w.r.t. the number of ground truth tracks are the important measures.

The basis of all models, except SORT, is the IMM-UKF (recall Section 5.2) with different association strategies. While ENC-DEC denotes the encoder-decoder network, Global Nearest Neighbour (GNN) represents a global data association strategy implemented by the Hungarian algorithm. The special form RNN-UKF represents an IMM-UKF with only one learned motion model based on Recurrent Neural Networks (RNNs). It implicitly contains various motion patterns and thus, does not need the Interacting Multiple Model (IMM) filter.

6.2.1 Validation on KITTI Raw Data

In order to obtain the overall performance of our models, we applied them to the selected set of sequences from the KITTI raw dataset. Because we are interested in real-world scenarios, we test the performance including all available object classes. Afterwards, we test the ability of tracking by repeatedly ignoring detections. Finally, we check the influence of objects apart from cars, pedestrians and cyclists.

All Object Classes

The best performing models considering *MOTA* are RNN-UKF-GNN and SORT, listed in Table 6.4. While SORT benefits from the lowest False Positive (FP) score and the second

	MT	PT	ML	IDS	FP	FN	MOTP	MOTA	FPS
SORT	161	78	16	**53**	**548**	2105	**75.5%**	**76.5%**	**412**
IMM-UKF-GNN	169	72	14	85	753	1930	**74.6%**	75.9%	**74**
IMM-UKF-JPDA	161	77	17	99	950	2164	73.4%	72.1%	50
IMM-UKF-ENC-DEC	**173**	71	**11**	**48**	824	**1867**	74.3%	76.2%	33
IMM-UKF-ENC-DEC*	139	95	21	65	940	2617	72.0%	68.5%	34
RNN-UKF-GNN	**174**	68	13	68	**695**	**1860**	71.5%	**77.2%**	40
RNN-UKF-ENC-DEC	172	73	**10**	78	849	1874	71.5%	75.6%	35
RNN-UKF-ENC-DEC*	139	93	23	143	1032	2666	68.1%	66.6%	37

Table 6.4: Evaluation results of all trackers on KITTI raw dataset and detections from the Frustum PointNets detector. Models containing ENC-DEC* do not use the Hungarian algorithm after encoder-decoder data association. The number of total ground truth tracks is 255.

best result regarding ID switches (IDS), the RNN-UKF-GNN tracker produces the best result concerning False Negatives (FNs), combined with a good *FP* value. Additionally, the latter one has the highest number of *MT* targets as well as a moderate number of *IDS*. The Multiple Object Tracking Precision (MOTP) for all trackers lies between 75.5% and 71.5% which is a solid result.

Because of the learned association strategy independent from the Intersection over Union (IoU) of tracked targets and detections, both encoder-decoder association based trackers, IMM-UKF-ENC-DEC and RNN-UKF-ENC-DEC, reach a very high number of *MT* targets and consequently also a very low number of *ML* targets. In particular, the

RNN-UKF-ENC-DEC tracker performs on par with both leading models. It has the second best *ML* and *MT* score. Moreover, it reaches the best result w.r.t. *IDS*, which indicates that the encoder-decoder data association works out very well.

Considering the averaged *FPS*, SORT is the fastest tracker. However, all listed models operate on average more than 3 times faster than the data recording rate of KITTI and thus, easily achieve real-time performance. Note, however, that we do not take the object detection time into account. Furthermore, an average value does not necessarily promise a constant processing speed. For example, the execution time of the IMM-UKF-JPDA filter increases exponentially w.r.t. the number of tracks and detections within one cluster. Hence, it can happen that the tracker needs more than 1 second per frame for a short subsequence and for the rest it operates 100 times faster. Summed up, this leads to an acceptable averaged value, but does not reflect real-time behavior.

In addition to our implemented trackers, Table 6.4 lists two approaches marked with an asterisk. They illustrate the tracking results without using the Hungarian algorithm on the encoder-decoder network output. Thus, the data association procedure does not necessarily fulfill the one-by-one assignment constraint. The significant drop in the respective tracking performance demonstrates that this constraint is crucial for Multiple Object Tracking (MOT).

All Object Classes and Skipped Detections

Within this section, we ignore detections of specific time steps to investigate the state prediction quality of our trackers. This is a more challenging task which requires the tracker to make good predictions and subsequently associate detections to them.

This experiment is the only exception regarding parameter tuning. We have to change the *hit-streak* threshold to 1, because the missing detections make consecutive updates impossible. This means, that each track is initialized immediately. Another problem is the *time-since-update* counter. Because of the removed detections, we need to adapt the threshold for this parameter as well. Thus, we increase the *time-since-update* threshold by 1 for each time step without detection. For example, if the default threshold is 2 and we skip the detections of each second time step, the new threshold is 3. If we drop the detections of two subsequent frames, the threshold is 4, and so on.

Table 6.5 shows the results of all trackers by skipping the detections of each second time step where we use only detections filtered as:

$$\text{filter}(\mathbf{d}_t) = \begin{cases} \mathbf{d}_t, & \text{if } \mod(t, 2) = 0, \\ \{\}, & \text{otherwise.} \end{cases} \tag{6.1}$$

Note, however, that we still calculate state estimates for all time steps. The first observation is an increase of the processing speed. Each tracker works faster by skipping the state update step for 50% of all frames.

	MT	PT	ML	IDs	FP	FN	MOTP	MOTA	FPS
SORT	81	103	71	159	723	5001	70.9%	48.8%	**657**
IMM-UKF-GNN	133	98	24	**46**	**619**	2599	**73.2%**	71.6%	**98**
IMM-UKF-JPDA	88	124	43	63	684	3853	67.8%	60.0%	77
IMM-UKF-ENC-DEC	**146**	93	**16**	**43**	799	**2366**	**71.2%**	**72.1%**	54
RNN-UKF-GNN	**137**	95	23	65	**654**	**2497**	66.8%	**72.0%**	52
RNN-UKF-ENC-DEC	130	108	**17**	72	673	2510	65.9%	71.7%	48

Table 6.5: Evaluation of all trackers on KITTI raw dataset omitting detections of each second time step, detected by the Frustum PointNets detector. The number of total ground truth tracks is 255.

More interesting is the *MOTA* result in comparison to the overall performance. All trackers suffer from the information loss what was to be expected. Notwithstanding, both models with encoder-decoder data association, RNN-UKF-ENC-DEC and IMM-UKF-ENC-DEC, still work very well and lose approximately only 4%. The reason for that is obvious. Since the learned data association model does not depend on the IoU, state predictions do not have to be as accurate as for the other models. This is also the reason why the performance of SORT drops significantly. The linear motion model which is responsible for state prediction fails for most of the tracks. In contrast, the other trackers deliver an acceptable result which is owed to the *IMM*, containing various motion models. An interesting observation is the still good performance of our RNN-UKF-GNN model which proves that the *RNN* state prediction works properly. In general, we can observe an increase of *FNs* and Partly Tracked (PT) targets, which is clearly expected.

When we go even further and drop detections of two subsequent time steps, considering detections filtered as

$$\text{filter}(\mathbf{d}_t) = \begin{cases} \mathbf{d}_t, & \text{if } \text{mod}(t,3) = 0, \\ \{\}, & \text{otherwise}, \end{cases} \tag{6.2}$$

the results confirm our hypotheses. Table 6.6 shows the comparison of all trackers. First of

	MT	PT	ML	IDs	FP	FN	MOTP	MOTA	FPS
SORT	14	68	173	352	1096	9074	55.5%	8.5%	**826**
IMM-UKF-GNN	102	92	61	**28**	**518**	3714	**71.5%**	63.0%	**112**
IMM-UKF-JPDA	49	142	64	54	726	5158	61.7%	48.4%	107
IMM-UKF-ENC-DEC	**105**	124	**26**	**44**	573	**3159**	**68.9%**	**67.2%**	73
RNN-UKF-GNN	**107**	98	50	48	**569**	3540	61.1%	63.8%	60
RNN-UKF-ENC-DEC	**105**	123	**27**	53	575	**3133**	58.7%	**67.3%**	56

Table 6.6: Evaluation of all trackers on KITTI raw dataset omitting detections of two subsequent time steps, detected by the Frustum PointNets detector. The number of total ground truth tracks is 255.

all, we can see another decrease of *MOTA* and a higher frame rate for all models. In detail, SORT is unusable in this setting, while all other trackers are at least around a *MOTA* of

50%. The best performing approaches are again both models based on Encoder-Decoder data association. In comparison to the overall evaluation results, we observe a decrease of only 9% for the best tracker by using approximately a third of the detections. However, the *MOTP* of RNN-UKF-ENC-DEC is very low, because we do not predict the bounding box orientation and thus, rely only on the detections.

Concluding we can say that the data association performance of a model has the most influence on the final results. Another observation is that models with a precise state prediction in combination with hand-crafted association metrics, *e.g. IoU*, are still able to track most targets. Not surprisingly, dropping detections leads to a slight decrease of *FPs*.

Main Object Classes

In order to obtain results which ensure better comparability to state-of-the-art approaches, we additionally evaluated our trackers just on the three main object classes, *i.e. Car*, *Pedestrian* and *Cyclist*. Therefore, we filtered out objects of the remaining classes from our detections and the ground truth data. Compliant to common practice, we treat objects of the class *Van* as a car and thus, it is also part of this evaluation. The results for this experiment are listed in Table 6.7. Because the sequences under consideration contain only

	MT	PT	ML	IDs	FP	FN	MOTP	MOTA	FPS
SORT	156	73	15	**43**	**530**	1987	**77.0%**	**76.7%**	**462**
IMM-UKF-GNN	164	66	14	78	745	1817	**76.1%**	75.9%	**58**
IMM-UKF-JPDA	158	69	17	91	919	2017	74.9%	72.4%	51
IMM-UKF-ENC-DEC	167	66	**11**	**50**	827	**1768**	75.8%	75.9%	35
RNN-UKF-GNN	**169**	62	13	70	**692**	**1765**	72.9%	**77.0%**	41
RNN-UKF-ENC-DEC	**168**	66	**10**	76	843	1772	72.7%	75.5%	36

Table 6.7: Evaluation results of all trackers on KITTI raw dataset, considering only objects from the main classes *Car/Van*, *Pedestrian* and *Cyclist*, detected by the Frustum PointNets detector. The number of total ground truth tracks is 244.

11 target tracks which are not included in the main classes, there is not a big difference to the overall evaluation in Table 6.4.

However, we can see a slight performance improvement w.r.t. *MOTA* for trackers depending on a data association performed by the Hungarian algorithm with a score matrix based on the *IoU* of detections and tracks. We think that, on the one hand, in the case of all object classes, these trackers are confused by targets with huge bounding box ground areas, *i.e. Tram*, *Truck* or small targets, *i.e. Sitting Person*. On the other hand, the trained association network benefits from distinguishable targets of different size.

The precision of each tracker regarding *MOTP* is higher for the main classes, because they are represented better in the dataset. Thus, detecting these object classes can be learned more precise by data driven object detectors. Furthermore, targets which are very

small, *i.e. Person Sitting* or very large, *i.e. Truck*, *Tram* are hard to match or detect as a whole, respectively.

6.2.2 Validation on KITTI Tracking Data

Another commonly used method to evaluate on the KITTI dataset is to test on the publicly available training data. Thus, following Osep et al. [112], we split the training sequences into two chunks, training and validation (see Table 6.1). The latter one is used within this section to evaluate again the overall performance of our implementations. Afterwards, we investigate the influence of using a different object detector.

All Object Classes

In order to obtain an overall performance measure on the KITTI tracking dataset we first evaluate our trackers considering all object classes. The results are listed in Table 6.8 and show a worse performance quality compared to the evaluation on the raw dataset in Table 6.4. The reasons for this are twofold. First, although using only one sequence more

	MT	PT	ML	IDs	FP	FN	MOTP	MOTA	FPS
SORT	163	168	40	**173**	**1250**	4688	**70.0%**	66.6%	**340**
IMM-UKF-GNN	**205**	140	**26**	202	1576	**3968**	69.4%	**68.6%**	**50**
IMM-UKF-JPDA	115	178	78	**107**	**1050**	6646	**70.5%**	57.4%	34
IMM-UKF-ENC-DEC	**192**	142	37	194	1708	4597	68.6%	64.5%	27
RNN-UKF-GNN	**205**	137	**29**	240	1530	**3996**	65.5%	**68.5%**	30
RNN-UKF-ENC-DEC	**192**	150	**29**	259	1802	4556	64.5%	63.8%	27

Table 6.8: Evaluation results of all trackers on KITTI tracking dataset and detections from the Frustum PointNets detector. The number of total ground truth tracks is 371.

for evaluation, the tracking data sequences contain 50% more tracks in comparison to the selected raw data sequences. This implies that several scenes are heavily crowded. Second, the number of pedestrians and cyclists is considerably larger. However, we do not change the parameters of our models, initially found for the selected sequences of the raw data. This ensures fair comparability and we are able to see the robustness of each tracker.

The top-performing trackers are IMM-UKF-GNN, followed by RNN-UKF-GNN and SORT. This leads us to the conclusion that our trained encoder-decoder association model has problems with heavily crowded scenes. Notwithstanding, the RNN-UKF-ENC-DEC tracker performs on par with the best models regarding *MT* and *ML* tracks. It shows that the tracker is able to deal with different situations and scenes. The big disadvantages are the high number of *FPs* which are caused by the simple input we use for training the encoder-decoder network and the high number of ID switches.

Main Object Classes

One of the main advantages of our tracking framework is the ability of using different object detectors. Because of that,we compare the performance of our models by processing detections of two different publicly available object detectors. Therefore, we apply the trackers on the KITTI tracking dataset and investigate the results by considering only the main object classes, *i.e. Car/ Van*, *Pedestrian* and *Cyclist*.

Frustum PointNets: Table 6.9 shows the results for the Frustum PointNets [157] detector, which we also used for all previously discussed evaluations. We observe, similar to

	MT	PT	ML	IDs	FP	FN	MOTP	MOTA	FPS
SORT	161	150	37	**160**	**1193**	4319	**71.6%**	67.2%	**358**
IMM-UKF-GNN	**198**	126	**24**	191	1520	**3662**	71.0%	**68.9%**	**55**
IMM-UKF-JPDA	116	162	70	**103**	**1023**	6170	**72.2%**	57.8%	39
IMM-UKF-ENC-DEC	**188**	124	36	185	1674	4273	70.3%	64.5%	28
RNN-UKF-GNN	**198**	124	**26**	229	1489	**3695**	67.0%	**68.7%**	32
RNN-UKF-ENC-DEC	184	136	28	240	1764	4214	66.1%	64.0%	28

Table 6.9: Evaluation results of all trackers on KITTI tracking dataset, considering only objects from the main classes *Car/Van*, *Pedestrian* and *Cyclist*, detected by the Frustum PointNets detector. The number of total ground truth tracks is 348.

the KITTI raw dataset evaluation, a slight performance increase for trackers based on a *GNN* data association strategy in comparison to the results in Table 6.8. IMM-UKF-GNN and RNN-UKF-GNN are still the leading trackers in this scenario.

PointRCNN: The second evaluated 3D object detector is PointRCNN [129]. The results in Table 6.10 give us new insights on the robustness of our trackers, but also on the detection quality of both detectors. Compared to Table 6.9 we have similar results

	MT	PT	ML	IDs	FP	FN	MOTP	MOTA	FPS
SORT	161	137	50	**64**	**1312**	4438	80.9%	66.4%	**400**
IMM-UKF-GNN	**191**	113	**44**	67	1567	**3816**	**81.2%**	**68.5%**	62
IMM-UKF-JPDA	129	146	73	**48**	**1264**	6503	**81.4%**	54.8%	45
IMM-UKF-ENC-DEC	175	118	55	77	1653	5154	**81.4%**	60.2%	31
RNN-UKF-GNN	**190**	118	**40**	91	1550	**3854**	76.7%	**68.2%**	36
RNN-UKF-ENC-DEC	180	118	50	128	1660	4701	76.3%	62.5%	32

Table 6.10: Evaluation results of all trackers on KITTI tracking dataset, considering only objects from the main classes *Car/Van*, *Pedestrian* and *Cyclist*, detected by the PointRCNN object detector. The number of total ground truth tracks is 348.

regarding *MOTA* for the best performing trackers IMM-UKF-GNN and RNN-UKF-GNN. In contrast, both trackers with learned data association model, IMM-UKF-ENC-DEC and RNN-UKF-ENC-DEC, lose 4.3% and 1.5% accuracy, respectively. We attribute this to

the trained encoder-decoder association model which learns from detections found by the Frustum PointNets detector.

The most obvious observation is the noticeable increase regarding *MOTP*. Each tracker reaches much better results compared to previous evaluations. This can be attributed to the more accurate bounding box detection provided by the PointRCNN detector. This result shows the importance of object detection in a tracking framework following an tracking-by-detection paradigm.

6.3 Comparison to the State-of-the-Art

Within this section we compare our evaluation results to state-of-the-art trackers which leads us to a number of problems. First and foremost, the KITTI tracking test data is not publicly available and trackers listed on the KITTI leaderboard[3] are evaluated on the 2D image plane, separated by object class. Second, there are only 4 out of 20 among the leading approaches on the score board using 3D information from the point cloud. Third, the inspiring approach proposed by Rachman [117] is evaluated only on carefully selected *City* sequences including restrictions w.r.t. occlusion and laser scanner range.

Because our 3D SORT implementation is similar to that of Weng and Kitani [150], we can compare and interpret our results w.r.t. their evaluation. They are on the 14^{th} place of the KITTI leaderboard for object class *Car* and on the 20^{th} place for object class *Pedestrian* by tracking in 3D space and projecting the 3D bounding boxes back into the 2D image plane. These projected bounding boxes are then used for evaluation. They reach 83.84% *MOTA* for cars which is approximately 1% less than the best approach using point cloud data. The accuracy for pedestrians lacks with 36.36% far behind the best approach which reaches 60.67%. The precision for cars given as *MOTP* is with 85.24% just 0.5% worse than the leading approach and for pedestrians with 64.86% approximately 10% worse, which is still acceptable.

Consequently, we can argue that the IMM-UKF based models work on par with the 3D implementation of SORT and thus, performs well compared to state-of-the-art approaches. Furthermore, they are more robust because of the precise state prediction mechanism due to multiple and non-linear motion models. Moreover, nearly all implemented models are able to deal with skipped time steps where we drop up to two thirds of the detections.

6.4 Detailed Tracker Analysis

For a better comparison between the best performing trackers on both selected datasets, KITTI raw data and KITTI tracking test data, we analyze the results of each sequence in detail. Therefore, we investigate the results of SORT and RNN-UKF-GNN for all

[3] `http://www.cvlibs.net/datasets/kitti/eval_tracking.php` (accessed October 1, 2019)

sequences of the raw dataset and IMM-UKF-GNN and RNN-UKF-GNN for all sequences of the tracking test dataset.

Sequence by sequence, we provide the results and an additional row, containing the improvements or degradations of the second tracker regarding the first one. Values denoting a performance increase are written in **green**, whereas values denoting a performance decrease are written in **red**. Again, we do not highlight Partly Tracked (PT) targets, because MT and ML targets w.r.t. the number of ground truth tracks are the important measures.

6.4.1 KITTI Raw Sequences

To gain more insights into the different behaviors of our trackers, we compare the two top-performing implementations SORT and RNN-UKF-GNN. By looking at the detailed evaluation in Table 6.11, we first observe the worse $MOTP$ values of the RNN-UKF-GNN tracker. This result is not really surprising by looking at the *RNN* prediction model implementation. We neither use the bounding box orientation as an input, nor as a model output. Thus, we do not predict the orientation with our motion model and the Unscented Kalman Filter (UKF) has to rely on the detected bounding box only. Hence, the tracking precision depends strongly on the object detector.

Apart from that, the overall $MOTA$ value is nearly the same and differs only by 0.7%. It is interesting to take a closer look at three sequences showing the biggest deviation regarding tracking accuracy: 0005, 0014 and 0020. Sequence 0005 is a city scenario with 12 cars, 2 pedestrians and 1 cyclist. The ego vehicle performs an s-shaped turn following another car. The object detector produces a lot of FP detections mostly caused by parking bicycles and mopeds. We have listed the most significant frames of sequence 0005 in Figure 6.1 where the results of both trackers are shown. While the track management of SORT is parametrized with a *hit-streak* threshold of 2 to perform best on all sequences, the RNN-UKF-GNN tracker starts tracks immediately indicated by a *hit-streak* threshold of 1. This prevents SORT from wrongly initializing tracks with the FP detections. However, a drawback of this parametrization is the lower number of MT targets.

On the contrary, sequence 0014 can be tracked better by the RNN-UKF-GNN model. It contains another city scenario with 30 cars, 5 pedestrians, 4 cyclists, 1 truck and 1 tram. In this case, the ego vehicle performs a relatively fast turn in the beginning and follows a straight road afterwards. We have listed the most significant frames of sequence 0014 in Figure 6.2 where the results of both trackers are shown. The difficulty is, on the one hand, the fast turn of the ego vehicle and, on the other hand, the different velocities of the traffic participants, *i.e.* static and oncoming objects, as well as various object classes (vehicles, pedestrians, cyclists). The simple linear Kalman Filter (KF) of the SORT tracker is not able to follow these fast changes within the first two frames and thus, misses a lot of targets. In contrast, the RNN-UKF-GNN tracker is able to yield precise predictions in such situations. Hence, it misses significantly less tracks which reflects the lower FN value.

Seq.	Tracker	MT	PT	ML	IDs	FP	FN	MOTP	MOTA	FPS
	(1)	10	5	0	5	26	97	63.3%	77.6%	275
0001	(2)	11	4	0	8	40	80	55.5%	77.6%	25
		1	-1	0	3	26	-17	-7.8%	0%	
	(1)	9	6	0	4	110	89	69.8%	71.7%	240
0005	(2)	11	4	0	4	172	90	70.2%	62.9%	23
		2	-2	0	0	62	1	0.4%	-8.8%	
	(1)	17	17	7	7	82	273	70.4%	68.3%	386
0014	(2)	26	13	2	8	70	168	63.2%	78.5%	35
		9	-4	-5	1	-12	-105	-7.2%	10.2%	
	(1)	11	4	0	3	55	118	77.1%	77.1%	465
0018	(2)	12	3	0	3	52	102	73.7%	79.6%	46
		1	-1	0	0	-3	-16	-3.4%	2.5%	
	(1)	4	4	0	6	20	76	78.4%	78.8%	251
0020	(2)	5	3	0	5	1	61	75.1%	86.1%	24
		1	-1	0	-1	-19	-15	-3.3%	7.3%	
	(1)	34	8	2	2	52	258	78.5%	83.4%	301
0039	(2)	33	8	3	2	68	258	75.8%	82.6%	29
		-1	0	1	0	16	0	-2.7%	-0.8%	
	(1)	4	0	0	2	68	24	73.5%	62.2%	321
0060	(2)	4	0	0	2	73	20	68.0%	61.8%	26
		0	0	0	0	5	-4	-5.5%	-0.4%	
	(1)	31	16	2	9	55	430	76.2%	79.1%	356
0064	(2)	33	14	2	8	93	409	72.8%	78.4%	34
		2	-2	0	-1	38	-21	-3.4%	-0.7%	
	(1)	3	3	2	1	11	111	68.7%	66.9%	1323
0070	(2)	5	1	2	4	28	81	60.8%	69.6%	138
		2	-2	0	3	17	-30	-7.9%	2.7%	
	(1)	38	15	3	14	69	629	78.8%	75.8%	198
0084	(2)	34	18	4	24	98	591	75.0%	75.8%	21
		-4	3	1	10	29	-38	-3.8%	0%	
	(1)	161	78	16	53	548	2105	75.5%	76.5%	444
OVERALL	(2)	174	68	13	68	695	1860	71.5%	77.2%	41
		13	-10	-3	15	147	-245	-4.0%	0.7%	

Table 6.11: Detailed comparison of SORT (1) and RNN-UKF-GNN (2) on KITTI raw data. Improvements of RNN-UKF-GNN in comparison to SORT are written in **green**, degradations are written in **red**. Partly Tracked (PT) targets are considered neither good nor bad.

The last sequence with a high deviation between both models regarding $MOTA$ is 0020. However, we have to take this sequence's result with a grain of salt. Because there are only 5 tracks, a single missing or incorrectly initialized track influences the result already by 20%. This is exactly the case in this evaluation. SORT initializes one track two times, because of some unlucky circumstances within the track management heuristics. This

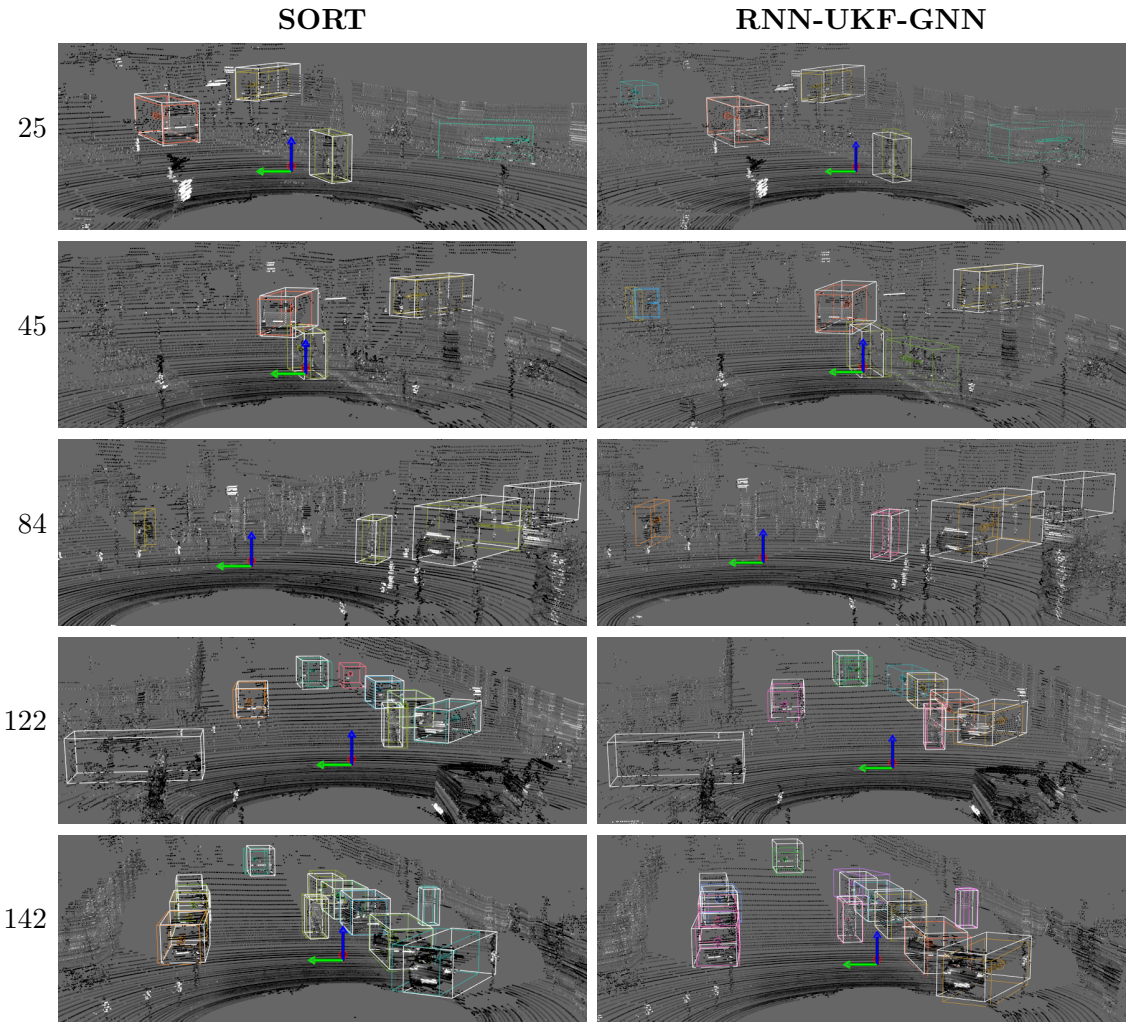


Figure 6.1: List of most significant frames of KITTI raw dataset sequence 0005, for both trackers, SORT on the left-hand side and RNN-UKF-GNN on the right-hand side. White bounding boxes describe ground truth objects and colored bounding boxes describe active tracks. While SORT has only 1 *FP* track in frame 25 and 84, RNN-UKF-GNN shows in sum 6 *FP* tracks for the first three listed frames. Notice that both trackers achieve a qualitatively robust performance across all frames.

means one *FP* track containing wrong instances over multiple time steps. However, this one sequence does not influence the overall result that much.

6.4.2 KITTI Tracking Sequences

The two top-performing implementations on the KITTI tracking dataset are IMM-UKF-GNN and RNN-UKF-GNN which are explicitly evaluated in Table 6.12. We can again observe the worse performance of the RNN-UKF-GNN tracker w.r.t. *MOTP*, due to missing orientation estimation as explained in the last section.

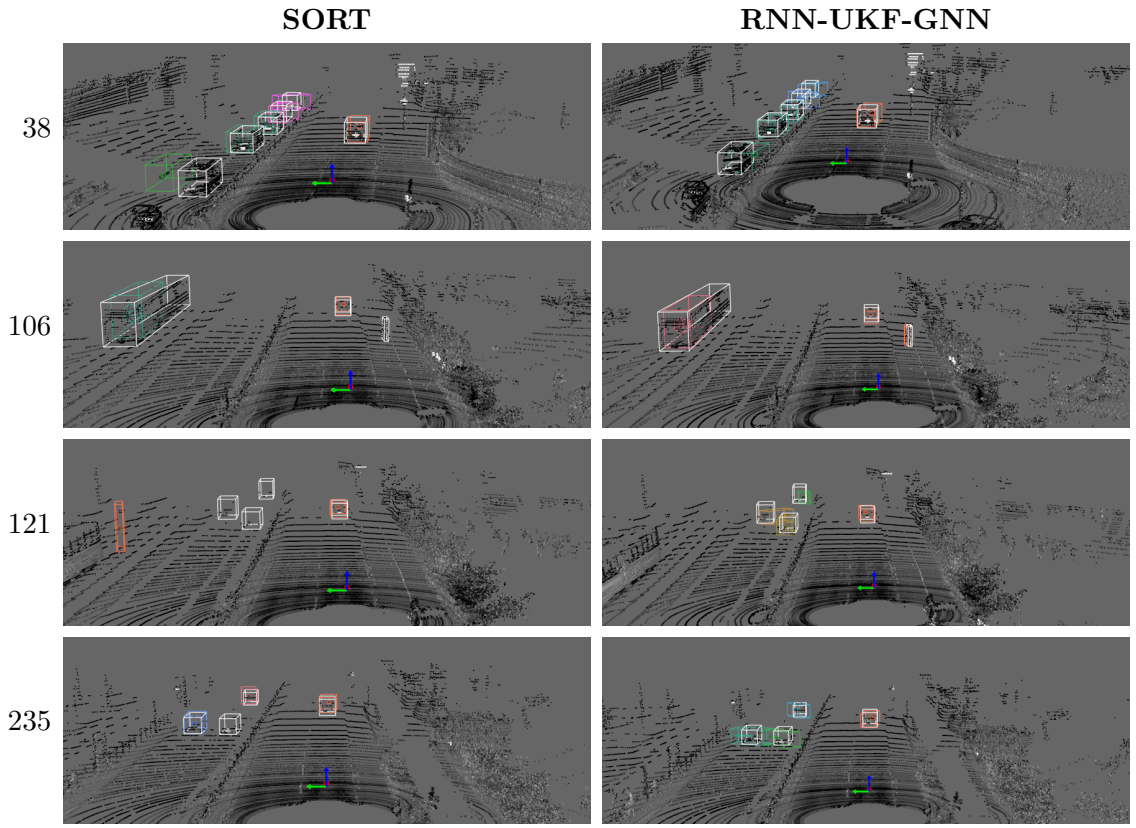


Figure 6.2: List of most significant frames of KITTI raw dataset sequence 0014, for both trackers, SORT on the left-hand side and RNN-UKF-GNN on the right-hand side. White bounding boxes describe ground truth objects and colored bounding boxes describe active tracks. The first frame shows that SORT has problems with the sharp turn in the beginning of the sequence. SORT reveals also problems on fast cars driving in the opposing direction of the ego-vehicle, *i.e.* frame 121 and 235. The RNN-UKF-GNN tracker does not lose a single track within these frames. However, its weaknesses regarding bounding box orientation can be observed as well.

The overall performance of the investigated trackers is nearly the same. However, there are differences in how each tracker reaches this value. We take a closer look at two sequences showing the biggest deviation regarding tracking accuracy: 0012 and 0014. Sequence 0012 contains only 4 ground truth instances containing 2 cars, 1 pedestrian and 1 cyclist. The RNN-UKF-GNN tracker with learned motion model is not able to follow the slowly walking pedestrian properly which leads to the higher number of *FN* targets. Additionally, the *MOTA* difference of 12.4% is owed to the small number of ground truth tracks, where one missing track has a huge impact of 25% on the total result.

Another interesting sequence is 0014, where the ego vehicle performs a relatively fast turn in the beginning, as shown in Figure 6.3. Because of the predefined motion models with fixed parameters, the IMM-UKF-GNN tracker is not able to follow the fast turn and

Seq.	Tracker	MT	PT	ML	IDs	FP	FN	MOTP	MOTA	FPS
	(1)	12	3	0	5	175	83	69.7%	63.0%	23
0000	(2)	10	5	0	3	156	84	72.2%	65.8%	36
		-2	2	0	-2	-19	1	2.5%	2.8%	
	(1)	6	3	0	2	18	61	73.6%	79.1%	46
0003	(2)	6	2	1	1	17	73	75.5%	76.5%	86
		0	-1	1	-1	-1	12	1.9%	-2.6%	
	(1)	26	13	2	11	68	154	65.0%	79.1%	33
0004	(2)	26	13	2	18	85	190	67.6%	73.7%	60
		0	0	0	7	17	36	2.6%	-5.4%	
	(1)	12	3	0	3	53	96	72.8%	80.1%	44
0006	(2)	12	3	0	7	59	106	76.3%	77.4%	78
		0	0	0	4	6	10	3.5%	-2.7%	
	(1)	12	12	4	10	52	166	65.3%	75.4%	39
0010	(2)	14	12	2	13	74	140	70.5%	75.5%	69
		2	0	-2	3	22	-26	5.2%	0.1%	
	(1)	3	1	0	5	88	34	68.5%	49.0%	26
0012	(2)	4	0	0	1	80	15	74.2%	61.4%	48
		1	-1	0	-4	-8	-19	5.7%	12.4%	
	(1)	25	22	7	18	256	282	57.4%	57.5%	30
0013	(2)	26	22	6	16	256	268	63.7%	58.7%	49
		1	0	-1	-2	0	-14	6.3%	1.2%	
	(1)	8	8	1	7	14	173	64.5%	70.1%	24
0014	(2)	5	12	0	18	34	201	68.0%	61.0%	40
		-3	4	-1	11	20	28	3.5%	-9.1%	
	(1)	11	13	2	47	78	826	53.6%	57.0%	31
0015	(2)	14	9	3	26	75	801	59.0%	59.2%	50
		3	-4	1	-21	-3	-25	5.4%	2.2%	
	(1)	11	16	1	86	226	950	59.2%	59.7%	14
0016	(2)	11	16	1	61	256	925	63.8%	60.4%	15
		0	0	0	-25	30	-25	4.6%	0.7%	
	(1)	79	43	12	46	502	1171	70.6%	74.9%	20
0020	(2)	77	46	11	38	484	1165	73.7%	75.4%	26
		-2	3	-1	-8	-18	-6	3.1%	0.5%	
	(1)	205	137	29	240	1530	3996	65.5%	68.5%	30
OVERALL	(2)	205	140	26	202	1576	3968	69.4%	68.6%	51
		0	3	-3	-38	46	-28	3.9%	0.1%	

Table 6.12: Detailed comparison of RNN-UKF-GNN (1) and IMM-UKF-GNN (2) on KITTI tracking data. Improvements of IMM-UKF-GNN in comparison to RNN-UKF-GNN are written in **green**, degradations are written in **red**. Partly Tracked (*PT*) targets are considered neither good nor bad.

thus, looses a lot of tracks. Especially a pair of pedestrians occluding each other produce a lot of detection errors, *i.e. FNs* as well as *FPs*. *IDS* are also a consequence of these circumstances. The RNN-UKF-GNN tracker uses its learned motion model to overcome

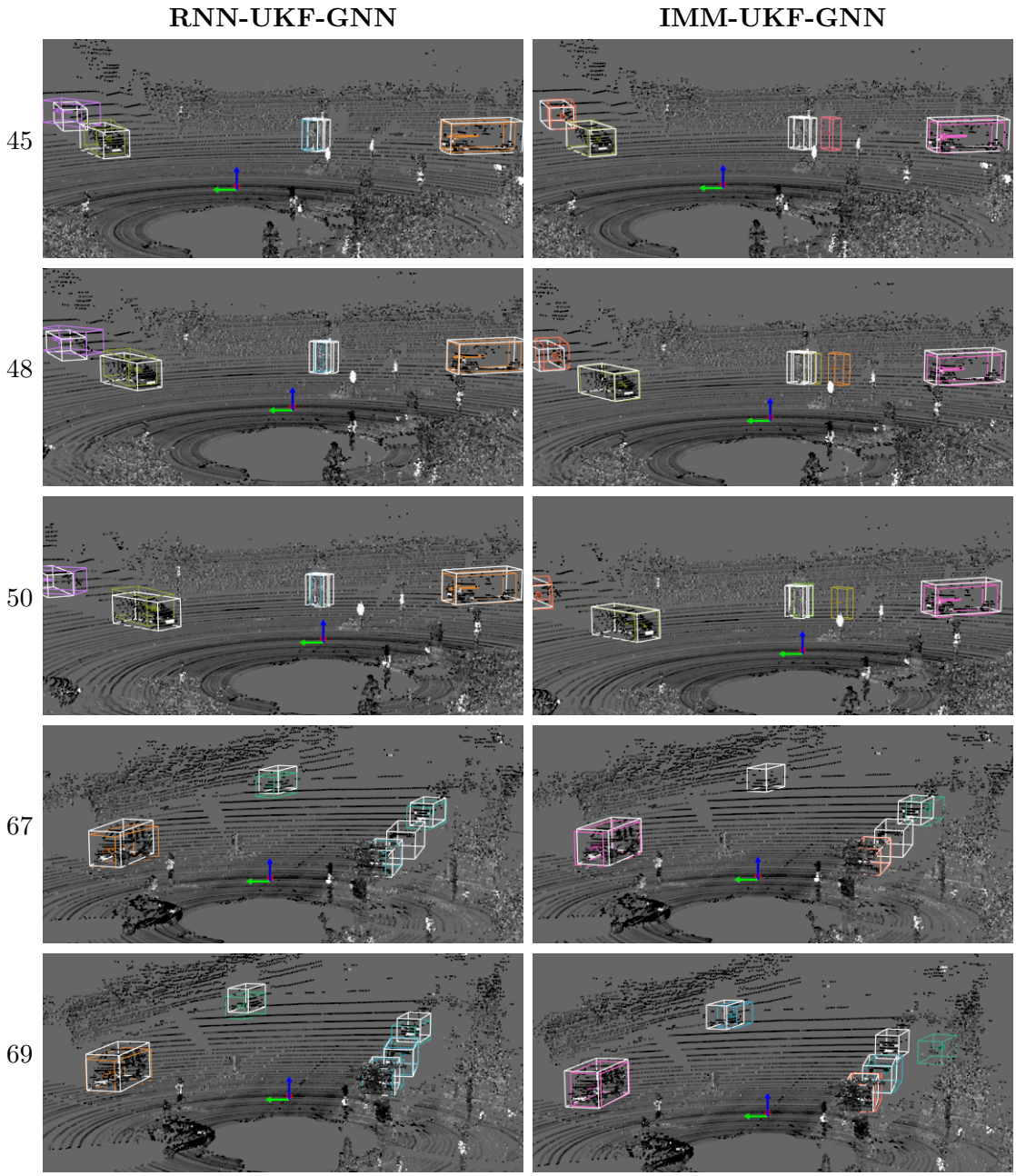


Figure 6.3: List of most significant frames of KITTI tracking dataset sequence 0014, for both trackers, RNN-UKF-GNN on the left-hand side and IMM-UKF-GNN on the right-hand side. White bounding boxes describe ground truth objects and colored bounding boxes describe active tracks. The tracking problems of the IMM-UKF-GNN approach are clearly visible by first looking at the pedestrians in the center. In the last two frames, the same problems can be observed by looking at the parking cars.

these problems, except the self-occlusion of pedestrians which leads also to a high number of *FNs*.

7 Conclusion and Future Work

Contents

7.1 Conclusion

In this thesis, we presented a modular Multiple Object Tracking (MOT) framework designed to track traffic participants in an autonomous driving scenario. To this end, we implemented several online approaches based on probabilistic filtering, as well as data-driven tracking models.

In particular, as a baseline we adapted SORT [14] to work in a 3D environment. Furthermore, to take non-linear motion behavior into account, we replaced the linear Kalman Filter (KF) by an Unscented Kalman Filter (UKF) and added the Interacting Multiple Model (IMM) to combine multiple motion models in a single tracker. The resulting filter is the IMM-UKF with exclusive global data association named IMM-UKF-GNN. In order to deal with cluttered detections, we implemented the IMM-UKF-JPDA filter, where we exchanged the exclusive data association with Joint Probabilistic Data Association (JPDA). Additionally, we trained a neural network for state prediction within the filter, as well as for data association. Hence, we defined an *UKF* with data-driven prediction model as RNN-UKF and a learned data association model as ENC-DEC, which leverages an encoder-decoder network architecture. Moreover, we presented an end-to-end trained network performing *MOT* which is not based on filtering and thus, does not require handcrafted parametrization anymore.

For all approaches, the input is restricted to 3D bounding boxes which allows the framework to deal with different object detectors and datasets. Thus, we evaluated the trackers on a publicly available autonomous driving dataset (*i.e.* KITTI [49]) and investi-

gated the performance by using two different object detectors (*i.e.* [129, 157]) and dropping detections in certain time steps. We have shown that our 3D adaption of SORT [14], containing a linear *KF* and a global exclusive data association, works well in comparison to state-of-the-art approaches. However, because the linear filter is not appropriate for the motion behavior of all traffic participants and strongly depends on reliable detections, it fails to track non-linear motions and suffer significantly from missing detections. Thus, we presented an adaption of the IMM-UKF-JPDA filter inspired by Rachman [117]. It is designed to work well for state estimation in cluttered environments and takes multiple detections into account. Because of compatibility reasons between all trackers implemented in our modular framework, we included the bounding box information to the state vector instead of using an external bounding box tracker. This extended state representation makes it hard to parametrize this tracker properly and additionally degrades it's performance. Furthermore, because of the exponentially increasing runtime w.r.t. the number of simultaneous tracks and detections, it is not the best choice for online tracking.

To overcome these issues, we presented an IMM-UKF approach combined with a global exclusive data association strategy. This implementation works on par with SORT and offers – because of the *IMM* – a more precise state prediction. When we drop detections of subsequent time steps, it is in contrast to SORT still able to follow most of the tracks. Another improvement to that is the RNN-UKF tracker with global exclusive data association. We demonstrated that the neural network trained for state prediction is able to produce even better results with less detections, as we have shown by dropping frames. This is especially the case in sequences where the ego-vehicle performs a sharp turn.

Since all approaches based on a global exclusive data association strategy with handcrafted similarity scores, *i.e.* Intersection over Union (IoU), rely on precise state estimation, they suffer from dropped detections as shown in our evaluation. In contrast, our implemented tracking approaches with an encoder-decoder association network are still able to perform on a reasonable level with as little as one third of the detections. Additionally, they work on par with the best implemented trackers using all available detections.

With our end-to-end model we presented a network which is designed to perform the whole *MOT* process without probabilistic filtering. It uses 3D bounding boxes as input and performs state prediction, data association, state update and track management for a variable number of tracks and measurements. Due to the limited availability of proper training data, however, this end-to-end model achieved significantly lower tracking performance and, thus, is not listed in our evaluations.

Concluding we can say that the implemented framework provides various object trackers which are able to robustly track multiple targets online in real traffic scenes. Additionally, all presented methods work on average approximately 4 times faster than the capturing rate of the sensor. The two main reasons which heavily influence the overall tracking performance are, on the one hand, unreliable detections caused by noisy sensor measurements, *e.g.* due to occlusions. On the other hand, environment changes cause large differences in the motion behavior of traffic participants or the ego-vehicle, *e.g.* cars

driving in the opposing direction, fast turns of the ego-vehicle or crowded scenes. These problems can be solved or mitigated in different ways which encourages further research.

7.2 Future Work

Although our evaluation shows promising results on the KITTI dataset, the complex task of *MOT* has ample room for improvement. We demonstrated that simple Recurrent Neural Networks (RNNs) are able to improve the robustness and the overall performance of filter-based trackers. However, recent research trends show that vision-based end-to-end trained networks which, for example, concurrently detect and track objects (*i.e.* [64, 95]) or use learned features from different sensors (*i.e.* [163]) also achieve promising results. These networks, however, need much more data which may be served by the recently released and publicly available nuScenes [23] dataset.

Another interesting research direction is *MOT* in combination with semantic segmentation, *e.g.* [146]. The big advantage of segmented objects in contrast to traditional bounding boxes is the smaller overlap of objects which are close to each other. This could lead to improved similarity scores and, subsequently, to less association errors between detections and tracks. Moreover, the segmentation of objects may implicitly help end-to-end networks to learn the tracking of multiple objects.

Bibliography

[1] Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M., Kudlur, M., Levenberg, J., Monga, R., Moore, S., Murray, D. G., Steiner, B., Tucker, P., Vasudevan, V., Warden, P., Wicke, M., Yu, Y., and Zheng, X. (2016). TensorFlow: A System for Large-scale Machine Learning. In *Proceedings of the USENIX Conference on Operating Systems Design and Implementation*. (page 69)

[2] Akashi, H. and Kumamoto, H. (1977). Random sampling approach to state estimation in switching environments. *Automatica*, 13(4):429–434. (page 10)

[3] Alahi, A., Goel, K., Ramanathan, V., Robicquet, A., Fei-Fei, L., and Savarese, S. (2016). Social LSTM: Human Trajectory Prediction in Crowded Spaces. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. (page 88)

[4] Andriluka, M., Roth, S., and Schiele, B. (2008). People-tracking-by-detection and people-detection-by-tracking. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. (page 16)

[5] Arulampalam, M., Maskell, S., Gordon, N., and Clapp, T. (2002). A tutorial on particle filters for online nonlinear/non-Gaussian Bayesian tracking. *IEEE Transactions on Signal Processing*, 50(2):174–188. (page 10)

[6] Asvadi, A., Peixoto, P., and Nunes, U. (2015). Detection and Tracking of Moving Objects Using 2.5D Motion Grids. *Proceedings of the IEEE International Conference on Intelligent Transportation Systems*. (page 9, 35)

[7] Bar-Shalom, Y., Daum, F., and Huang, J. (2009). The Probabilistic Data Association Filter: Estimation in the presence of measurement origin uncertainty. *IEEE Control Systems*, 29(6):82–100. (page 9, 35, 36, 37, 38, 42)

[8] Bar-Shalom, Y. and Li, X.-R. (1995). *Multitarget-Multisensor Tracking: Principles and Techniques*. YBS Publishing, first edition. (page 5, 9, 19, 35, 36, 37, 38, 82)

[9] Bar-Shalom, Y. and Tse, E. (1975). Tracking in a cluttered environment with probabilistic data association. *Automatica*, 11(5):451–460. (page 9, 35, 36, 37)

[10] Barth, A. and Franke, U. (2010). Tracking oncoming and turning vehicles at intersections. In *Proceedings of the IEEE International Conference on Intelligent Transportation Systems*. (page 5, 11)

[11] Bengio, Y., Simard, P., and Frasconi, P. (1994). Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, 5(2):157–166. (page 58)

[12] Bernardin, K. and Stiefelhagen, R. (2008). Evaluating multiple object tracking performance: The CLEAR MOT metrics. *Journal on Image and Video Processing*, 2008(1):1–10. (page 12, 13, 96)

[13] Betke, M., Hirsh, D. E., Bagchi, A., Hristov, N. I., Makris, N. C., and Kunz, T. H. (2007). Tracking large variable numbers of objects in clutter. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. (page 2)

[14] Bewley, A., Ge, Z., Ott, L., Ramos, F., and Upcroft, B. (2016). Simple online and realtime tracking. In *Proceedings of the IEEE International Conference on Image Processing*. (page 11, 73, 75, 76, 97, 111, 112)

[15] Bialkowski, A., Lucey, P., Carr, P., Yue, Y., Sridharan, S., and Matthews, I. (2015). Large-Scale Analysis of Soccer Matches Using Spatiotemporal Tracking Data. In *Proceedings of the IEEE International Conference on Data Mining*. (page 2)

[16] Bishop, C. M. (2006). *Pattern Recognition and Machine Learning*. Springer, first edition. (page 47)

[17] Blackman, S. S. (2004). Multiple hypothesis tracking for multiple target tracking. *IEEE Aerospace and Electronic Systems Magazine*, 19(1 II):5–18. (page 9)

[18] Blom, H. and Bar-Shalom, Y. (1988). The interacting multiple model algorithm for systems with Markovian switching coefficients. *IEEE Transactions on Automatic Control*, 33(8):780–783. (page 11, 24, 30)

[19] Britz, D. (2015). Recurrent Neural Networks Tutorial, Part3 - Backpropagation Through Time and Vanishing Gradients. `http://www.wildml.com/2015/10/recurrent-neural-networks-tutorial-part-3-backpropagation-through-time-and-vanishing-gradients/`. Online; accessed March 11, 2019. (page 57)

[20] Broggi, A., Cerri, P., Debattisti, S., Laghi, M. C., Medici, P., Molinari, D., Panciroli, M., and Prioletti, A. (2015). PROUD-Public Road Urban Driverless-Car Test. *IEEE Transactions on Intelligent Transportation Systems*, 16(6):3508–3519. (page 1)

[21] Buehler, M., Iagnemma, K., and Singh, S. (2007). *The 2005 DARPA Grand Challenge: The Great Robot Race*. Springer, first edition. (page 2)

[22] Buehler, M., Iagnemma, K., and Singh, S. (2009). *The DARPA Urban Challenge: Autonomous Vehicles in City Traffic*. Springer, first edition. (page 2)

[23] Caesar, H., Bankiti, V., Lang, A. H., Vora, S., Liong, V. E., Xu, Q., Krishnan, A., Pan, Y., Baldan, G., and Beijbom, O. (2019). nuscenes: A multimodal dataset for autonomous driving. *arXiv CoRR*, abs/1903.11027. (page 113)

[24] Cannons, K. (2008). A review of visual tracking. Technical Report CSE-2008-07, York University. (page 8)

[25] Ćesić, J., Marković, I., Jurić-Kavelj, S., and Petrović, I. (2014). Detection and tracking of dynamic objects using 3D laser range sensor on a mobile platform. In *Proceedings of the International Conference on Informatics in Control, Automation and Robotics*. (page 9)

[26] Challa, S., Morelande, M. R., Musicki, D., and Evans, R. J. (2011). *Fundamentals of Object Tracking*. Cambridge University Press, first edition. (page 6, 9, 19, 20, 22, 23, 24, 25, 26, 30, 32, 37)

[27] Chen, L.-C., Collins, M. D., Zhu, Y., Papandreou, G., Zoph, B., Schroff, F., Adam, H., and Shlens, J. (2018a). Searching for Efficient Multi-Scale Architectures for Dense Image Prediction. In *Proceedings of the Conference on Neural Information Processing Systems*. (page 11)

[28] Chen, L.-C., Zhu, Y., Papandreou, G., Schroff, F., and Adam, H. (2018b). Encoder-Decoder with Atrous Separable Convolution for Semantic Image Segmentation. In *Proceedings of the European Conference on Computer Vision*. (page 11)

[29] Cho, K., van Merrienboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., and Bengio, Y. (2014). Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*. (page 60)

[30] Choi, J., Ulbrich, S., Lichte, B., and Maurer, M. (2013a). Multi-Target Tracking using a 3D-Lidar sensor for autonomous vehicles. *Proceedings of the IEEE Conference on Intelligent Transportation Systems*. (page 9, 35)

[31] Choi, W., Pantofaru, C., and Savarese, S. (2013b). A general framework for tracking multiple people from a moving camera. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(7):1577–1591. (page 11)

[32] Cox, I. J. (1993). A review of statistical data association techniques for motion correspondence. *International Journal of Computer Vision*, 10(1):53–66. (page 10)

[33] Dickmanns, E., Behringer, R., Dickmanns, D., Hildebrandt, T., Maurer, M., Thomanek, F., and Schiehlen, J. (1994). The seeing passenger car 'VaMoRs-P'. In *Proceedings of the IEEE Intelligent Vehicles Symposium*. (page 1)

[34] Dickmanns, E., Mysliwetz, B., and Christians, T. (1990). An integrated spatio-temporal approach to automatic visual guidance of autonomous vehicles. *IEEE Transactions on Systems, Man, and Cybernetics*, 20(6):1273–1284. (page 1)

[35] Doucet, A., de Freitas, N., Murphy, K., and Russell, S. (2000). Rao-blackwellised Particle Filtering for Dynamic Bayesian Networks. In *Proceedings of the Conference on Uncertainty in Artificial Intelligence*. (page 11)

[36] Elfes, A. (1989). Using Occupancy Grids for Mobile Robot Perception and Navigation. *Computer*, 22(6):46–57. (page 2)

[37] Ess, A., Leibe, B., Schindler, K., and Van Gool, L. (2008). A mobile vision system for robust multi-person tracking. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. (page 2, 10)

[38] Fan, L., Wang, Z., Cai, B., Tao, C., Zhang, Z., Wang, Y., Li, S., Huang, F., Fu, S., and Zhang, F. (2017). A survey on multiple object tracking algorithm. In *Proceedings of the IEEE International Conference on Information and Automation*. (page 8)

[39] Fang, K., Xiang, Y., Li, X., and Savarese, S. (2018). Recurrent Autoregressive Networks for Online Multi-object Tracking. In *Proceedings of the IEEE Winter Conference on Applications of Computer Vision*. (page 8)

[40] Fontaine, E., Barr, A. H., and Burdick, J. W. (2007). Model-based tracking of multiple worms and fish. In *Proceedings of the IEEE International Conference on Computer Vision Workshop*. (page 2)

[41] Fortin, B., Lherbier, R., and Noyer, J.-C. (2015). A Model-Based Joint Detection and Tracking Approach for Multi-Vehicle Tracking With Lidar Sensor. *IEEE Transactions on Intelligent Transportation Systems*, 16(4):1883–1895. (page 10)

[42] Fortin, B., Noyer, J. C., and Lherbier, R. (2012). A particle filtering approach for joint vehicular detection and tracking in lidar data. In *Proceedings of the IEEE International Instrumentation and Measurement Technology Conference*. (page 2, 10)

[43] Fortmann, T., Bar-Shalom, Y., and Scheffe, M. (1983). Sonar tracking of multiple targets using joint probabilistic data association. *IEEE Journal of Oceanic Engineering*, 8(3):173–184. (page 9, 35, 38, 39)

[44] Franke, U., Gavrila, D., Görzig, S., Lindner, F., Paetzold, F., and Wöhler, C. (1998). Autonomous driving goes downtown. *IEEE Intelligent Systems and Their Applications*, 13(6):40–48. (page 1)

[45] Franke, U., Mehring, S., Suissa, A., and Hahn, S. (1994). The Daimler-Benz steering assistant: a spin-off from autonomous driving. In *Proceedings of the Intelligent Vehicles Symposium*. (page 1)

[46] Frossard, D. and Urtasun, R. (2018). End-to-end Learning of Multi-sensor 3D Tracking by Detection. In *Proceedings of the IEEE International Conference on Robotics and Automation*. (page 12)

[47] Garcia-Fernandez, A. F., Williams, J. L., Granstrom, K., and Svensson, L. (2018). Poisson Multi-Bernoulli Mixture Filter: Direct Derivation and Implementation. *IEEE Transactions on Aerospace and Electronic Systems*, 54(4):1883–1901. (page 10)

[48] Geiger, A., Lenz, P., Stiller, C., and Urtasun, R. (2013). Vision meets robotics: The KITTI dataset. *International Journal of Robotics Research*, 32(11):1231–1237. (page 14, 15, 16, 70, 95)

[49] Geiger, A., Lenz, P., and Urtasun, R. (2012). Are we ready for autonomous driving? The KITTI vision benchmark suite. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. (page 11, 12, 15, 95, 111)

[50] Gers, F. A., Schmidhuber, J., and Cummins, F. (2000). Learning to Forget: Continual Prediction with LSTM. *Neural Computation*, 12(10):2451–2471. (page 58)

[51] Gers, F. A., Schraudolph, N. N., and Schmidhuber, J. (2002). Learning precise timing with LSTM recurrent networks. *Journal of Machine Learning Research*, 3(1):115–143. (page 58)

[52] Girshick, R. (2015). Fast R-CNN. In *Proceedings of the IEEE International Conference on Computer Vision*. (page 3, 75)

[53] Goodfellow, I., Bengio, Y., and Courville, A. (2017). *The Deep Learning Book*. MIT Press, first edition. (page 45, 46, 47, 50, 51, 53, 57, 58, 60)

[54] Gordon, N., Salmond, D., and Smith, A. (1993). Novel approach to nonlinear/non-Gaussian Bayesian state estimation. *IEE Proceedings F Radar and Signal Processing*, 140(2):107. (page 10)

[55] Graves, A. (2012). *Supervised Sequence Labelling with Recurrent Neural Networks*. Springer, first edition. (page 46, 47, 48, 50, 51, 57, 58, 60)

[56] Greff, K., Srivastava, R. K., Koutnik, J., Steunebrink, B. R., and Schmidhuber, J. (2017). LSTM: A Search Space Odyssey. *IEEE Transactions on Neural Networks and Learning Systems*, 28(10):2222–2232. (page 60)

[57] Grisleri, P. and Fedriga, I. (2010). The BRAiVE autonomous ground vehicle platform. *IFAC Proceedings Volumes*, 43(16):497–502. (page 1)

[58] Gündüz, G. and Acarman, T. (2018). A Lightweight Online Multiple Object Vehicle Tracking Method. In *Proceedings of the IEEE Intelligent Vehicles Symposium*. (page 11)

[59] Handschin, J. E. and Mayne, D. Q. (1969). Monte Carlo techniques to estimate the conditional expectation in multi-stage non-linear filtering. *International Journal of Control*, 9(5):547–559. (page 10)

[60] He, K., Gkioxari, G., Dollar, P., and Girshick, R. (2017). Mask R-CNN. In *Proceedings of the IEEE International Conference on Computer Vision*. (page 11)

[61] Ho, Y. and Lee, R. (1964). A Bayesian approach to problems in stochastic estimation and control. *IEEE Transactions on Automatic Control*, 9(4):333–339. (page 25)

[62] Hochreiter, J. (1991). Untersuchungen zu dynamischen neuronalen Netzen. Master's thesis, Technische Universität München, Institut für Informatik. (page 58)

[63] Hochreiter, S. and Schmidhuber, J. (1997). Long Short-Term Memory. *Neural Computation*, 9(8):1735–1780. (page 58)

[64] Hu, H.-N., Cai, Q.-Z., Wang, D., Lin, J., Sun, M., Krähenbühl, P., Darrell, T., and Yu, F. (2018). Joint monocular 3d vehicle detection and tracking. In *Proceedings of the IEEE International Conference on Computer Vision*. (page 113)

[65] Huang, J., Rathod, V., Sun, C., Zhu, M., Korattikara, A., Fathi, A., Fischer, I., Wojna, Z., Song, Y., Guadarrama, S., and Murphy, K. (2017). Speed/Accuracy Trade-Offs for Modern Convolutional Object Detectors. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. (page 8)

[66] Isard, M. and MacCormick, J. (2001). BraMBLe: a Bayesian multiple-blob tracker. In *Proceedings of the IEEE International Conference on Computer Vision*. (page 10)

[67] Janai, J., Güney, F., Behl, A., and Geiger, A. (2017). Computer Vision for Autonomous Vehicles: Problems, Datasets and State-of-the-Art. *arXiv CoRR*, abs/1704.05519. (page 1, 8)

[68] Julier, S. J. (2002). The scaled unscented transformation. In *Proceedings of the IEEE American Control Conference*. (page 27, 28, 29)

[69] Julier, S. J. and Uhlmann, J. K. (1997). A New Extension of the Kalman Filter to Nonlinear Systems. In *Proceedings of SPIE 3068, Signal Processing, Sensor Fusion, and Target Recognition VI*. (page 24, 26, 30)

[70] Julier, S. J. and Uhlmann, J. K. (2004). Unscented filtering and nonlinear estimation. *Proceedings of the IEEE*, 92(3):401–422. (page 8)

[71] Kaempchen, N. and Dietmayer, K. (2004). IMM Vehicle Tracking for Traffic Jam Situations on Highways. In *Proceedings of the International Conference on Multisensor Information Fusion*. (page 11)

[72] Kalman, R. E. (1960). A New Approach to Linear Filtering and Prediction Problems. *Journal of Basic Engineering*, 82(1):35. (page 8, 24)

[73] Kalman, R. E. and Bucy, R. S. (1961). New Results in Linear Filtering and Prediction Theory. *Journal of Basic Engineering*, 83(1):95. (page 8)

[74] Karpathy, A. (2015). The Unreasonable Effectiveness of Recurrent Neural Networks. `http://karpathy.github.io/2015/05/21/rnn-effectiveness/`. Online; accessed March 11, 2019. (page 54, 55)

[75] Khan, Z., Balch, T., and Dellaert, F. (2003). Efficient particle filter-based tracking of multiple interacting targets using an mrf-based motion model. In *Proceedings of the IEEE International Conference on Intelligent Robots and Systems*. (page 10)

[76] Khan, Z., Balch, T., and Dellaert, F. (2005). MCMC-based particle filtering for tracking a variable number of interacting targets. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 27(11):1805–1819. (page 2, 11)

[77] Kim, C., Li, F., Ciptadi, A., and Rehg, J. M. (2015). Multiple Hypothesis Tracking Revisited. In *Proceedings of the IEEE International Conference on Computer Vision*. (page 11)

[78] Kim, D., Jo, K., Lee, M., and Sunwoo, M. (2018). L-Shape Model Switching-Based Precise Motion Tracking of Moving Vehicles Using Laser Scanners. *IEEE Transactions on Intelligent Transportation Systems*, 19(2):598–612. (page 5)

[79] Kim, Y.-s. and Hong, K.-S. (2004). An IMM Algorithm for tracking Maneuvering Vehicle in an Adaptive Cruise Control Environment. *International Journal of Control Automation and Systems*, 2(3):310–318. (page 11)

[80] Kingma, D. P. and Ba, J. (2015). Adam: A Method for Stochastic Optimization. In *Proceedings of the International Conference on Learning Representations*. (page 61, 85, 88, 92)

[81] Krebs, S., Duraisamy, B., and Flohr, F. (2017). A survey on leveraging deep neural networks for object tracking. In *Proceedings of the IEEE International Conference on Intelligent Transportation Systems*. (page 7, 8)

[82] Kristan, M. (2008). *Tracking people in video data using probabilistic models*. PhD thesis, University of Ljubljana. (page 19)

[83] Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). ImageNet classification with deep convolutional neural networks. In *Proceedings of the Conference on Neural Information Processing Systems*. (page 11)

[84] Leal-Taixé, L., Milan, A., Reid, I., Roth, S., and Schindler, K. (2015). MOTChallenge 2015: Towards a Benchmark for Multi-Target Tracking. *arXiv CoRR*, abs/1504.01942. (page 15)

[85] Lee, B., Erdenee, E., Jin, S., Nam, M. Y., Jung, Y. G., and Rhee, P. K. (2016). Multi-class Multi-object Tracking Using Changing Point Detection. In *Proceedings of*

the European Conference on Computer Vision Workshop on Benchmarking Multi-target Tracking: MOTChallenge. (page 11)

[86] Li, K., Miller, E. D., Chen, M., Kanade, T., Weiss, L. E., and Campbell, P. G. (2008). Cell population tracking and lineage construction with spatiotemporal context. *Medical Image Analysis*, 12(5):546–566. (page 2)

[87] Li, P., Wang, D., Wang, L., and Lu, H. (2018). Deep visual tracking: Review and experimental comparison. *Pattern Recognition*, 76:323–338. (page 8)

[88] Li, X. R. and Jilkov, V. P. (2003). Survey of maneuvering targettracking. Part I: Dynamic models. *IEEE Transactions on Aerospace and Electronic Systems*, 39(4):1333–1364. (page 11, 77)

[89] Li, Y., Huang, C., and Nevatia, R. (2009). Learning to associate: HybridBoosted multi-target tracker for crowded scene. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. (page 12, 14, 96)

[90] Liu, S., Qi, L., Qin, H., Shi, J., and Jia, J. (2018). Path Aggregation Network for Instance Segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. (page 11)

[91] Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C.-Y., and Berg, A. C. (2016). SSD: Single shot multibox detector. In *Proceedings of the European Conference on Computer Vision*. (page 3)

[92] Lu, C.-W., Lin, C.-Y., Hsu, C.-Y., Weng, M.-F., Kang, L.-W., and Liao, H.-Y. M. (2013). Identification and tracking of players in sport videos. In *Proceedings of the International Conference on Internet Multimedia Computing and Service*. (page 2)

[93] Luo, W., Kim, T. K., Stenger, B., Zhao, X., and Cipolla, R. (2014a). Bi-label propagation for generic multiple object tracking. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. (page 2)

[94] Luo, W., Xing, J., Milan, A., Zhang, X., Liu, W., Zhao, X., and Kim, T.-K. (2014b). Multiple Object Tracking: A Literature Review. *arXiv CoRR*, abs/1409.7618. (page 5)

[95] Luo, W., Yang, B., and Urtasun, R. (2018). Fast and Furious: Real Time End-to-End 3D Detection, Tracking and Motion Forecasting with a Single Convolutional Net. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. (page 2, 12, 113)

[96] MacCormick, J. and Blake, A. (1999). A probabilistic exclusion principle for tracking multiple objects. In *Proceedings of the IEEE International Conference on Computer Vision*. (page 10)

[97] Mahler, R. (2003). Multitarget bayes filtering via first-order multitarget moments. *IEEE Transactions on Aerospace and Electronic Systems*, 39(4):1152–1178. (page 10)

[98] Mahler, R. (2007). PHD filters of higher order in target number. *IEEE Transactions on Aerospace and Electronic Systems*, 43(4):1523–1543. (page 10)

[99] Mazor, E., Averbuch, A., Bar-Shalom, Y., and Dayan, J. (1998). Interacting multiple model methods in target tracking: a survey. *IEEE Transactions on Aerospace and Electronic Systems*, 34(1):103–123. (page 30)

[100] McCulloch, W. S. and Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. *The Bulletin of Mathematical Biophysics*, 5(4):115–133. (page 45)

[101] Meijering, E., Dzyubachyk, O., Smal, I., and van Cappellen, W. A. (2009). Tracking in cell and developmental biology. *Seminars in Cell and Developmental Biology*, 20(8):894–902. (page 2)

[102] Milan, A., Leal-Taixe, L., Reid, I., Roth, S., and Schindler, K. (2016). MOT16: A Benchmark for Multi-Object Tracking. *arXiv CoRR*, abs/1603.00831. (page 12, 14, 15, 16, 71)

[103] Milan, A., Rezatofighi, S. H., Dick, A., Reid, I., and Schindler, K. (2017). Online Multi-Target Tracking Using Recurrent Neural Networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*. (page 12, 62, 63, 64, 83, 88, 91)

[104] Montemerlo, M., Becker, J., Bhat, S., Dahlkamp, H., Dolgov, D., Ettinger, S., Haehnel, D., Hilden, T., Hoffmann, G., Huhnke, B., Johnston, D., Klumpp, S., Langer, D., Levandowski, A., Levinson, J., Marcil, J., Orenstein, D., Paefgen, J., Penny, I., Petrovskaya, A., Pflueger, M., Stanek, G., Stavens, D., Vogt, A., and Thrun, S. (2009). Junior: The stanford entry in the urban challenge. *Springer Tracts in Advanced Robotics*, 56(9):91–123. (page 2)

[105] Morales, N., Toledo, J., Acosta, L., and Sanchez-Medina, J. (2017). A Combined Voxel and Particle Filter-Based Approach for Fast Obstacle Detection and Tracking in Automotive Applications. *IEEE Transactions on Intelligent Transportation Systems*, 18(7):1824–1834. (page 10)

[106] Munkres, J. (1957). Algorithms for the Assignment and Transportation Problems. *Journal of the Society for Industrial and Applied Mathematics*, 5(1):32–38. (page 9, 35)

[107] Mušicki, D. and Evans, R. (2002). Joint Integrated Probabilistic Data Association - JIPDA. In *Proceedings of the International Conference on Information Fusion*. (page 10)

[108] Musicki, D. and La Scala, B. (2008). Multi-target tracking in clutter without measurement assignment. *IEEE Transactions on Aerospace and Electronic Systems*, 44(3):877–896. (page 10)

[109] Niknejad, H., Takeuchi, A., Mita, S., and McAllester, D. (2012). On-Road Multivehicle Tracking Using Deformable Object Model and Particle Filter With Improved Likelihood Estimation. *IEEE Transactions on Intelligent Transportation Systems*, 13(2):748–758. (page 2, 10)

[110] Nillius, P., Sullivan, J., and Carlsson, S. (2006). Multi-target tracking - Linking identities using Bayesian network inference. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. (page 2)

[111] Organization, W. H. (2018). Global Status Report on Road Safety 2018. `https://www.who.int/violence_injury_prevention/road_safety_status/2018/en/`. Online; accessed October 8, 2019. (page 1)

[112] Osep, A., Mehner, W., Mathias, M., and Leibe, B. (2017). Combined image- and world-space tracking in traffic scenes. In *Proceedings of the IEEE International Conference on Robotics and Automation*. (page 5, 96, 102)

[113] Otto, C., Gerber, W., Leon, F. P., and Wirnitzer, J. (2012). A Joint Integrated Probabilistic Data Association Filter for pedestrian tracking across blind regions using monocular camera and radar. In *Proceedings of the IEEE Intelligent Vehicles Symposium*. (page 10)

[114] Petrovskaya, A. and Thrun, S. (2009). Model based vehicle detection and tracking for autonomous urban driving. *Autonomous Robots*, 26(2-3):123–139. (page 2, 5, 11)

[115] Plaut, D. C., Nowlan, S. J., and Hinton, G. E. (1986). Experiments on Learning by Back Propagation. Technical Report CMU-CS-86-126, Carnegie-Mellon University. (page 60)

[116] Qi, C. R., Liu, W., Wu, C., Su, H., and Guibas, L. J. (2018). Frustum PointNets for 3D Object Detection from RGB-D Data. In *Proceedings of the Conference on Computer Vision and Pattern Recognition*. (page 8, 73, 75, 96)

[117] Rachman, A. S. A. (2017). 3D-LIDAR Multi Object Tracking for Autonomous Driving. Master's thesis, Delft University of Technology, Center for Systems and Control. (page 2, 5, 9, 11, 72, 76, 104, 112)

[118] Rangesh, A. and Trivedi, M. M. (2018). No Blind Spots: Full-Surround Multi-Object Tracking for Autonomous Vehicles using Cameras & LiDARs. *arXiv CoRR*, abs/1802.08755. (page 2, 9)

[119] Redmon, J., Divvala, S., Girshick, R., and Farhadi, A. (2016). You Only Look Once: Unified, Real-Time Object Detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. (page 75)

[120] Reid, D. (1979). An algorithm for tracking multiple targets. *IEEE Transactions on Automatic Control*, 24(6):843–854. (page 9)

[121] Reuter, S., Vo, B. T., Vo, B. N., and Dietmayer, K. (2014). The Labeled Multi-Bernoulli Filter. *IEEE Transactions on Signal Processing*, 62(12):3246–3260. (page 10)

[122] Rosenblatt, F. (1958). The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65(6):386–408. (page 45)

[123] Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986). Learning representations by back-propagating errors. *Nature*, 323(6088):533–536. (page 47, 52)

[124] Sanchez-Matilla, R., Poiesi, F., and Cavallaro, A. (2016). Online Multi-target Tracking with Strong and Weak Detections. In *Proceedings of the European Conference on Computer Vision Workshop on Benchmarking Multi-target Tracking: MOTChallenge.* (page 10)

[125] Scheidegger, S., Benjaminsson, J., Rosenberg, E., Krishnan, A., and Granstrom, K. (2018). Mono-Camera 3D Multi-Object Tracking Using Deep Learning Detections and PMBM Filtering. In *Proceedings of the IEEE Intelligent Vehicles Symposium.* (page 10, 11)

[126] Schreier, M. (2017). *Bayesian environment representation, prediction, and criticality assessment for driver assistance systems.* PhD thesis, Technische Universität Darmstadt, Department of Electrical Engineering and Information Technology. (page 2, 9, 11, 19, 30, 32, 34, 76, 77, 82)

[127] Schuster, M. and Paliwal, K. (1997). Bidirectional recurrent neural networks. *IEEE Transactions on Signal Processing*, 45(11):2673–2681. (page 57)

[128] Sharma, S., Ansari, J. A., Murthy, J. K., and Krishna, K. M. (2018). Beyond Pixels: Leveraging Geometry and Shape Cues for Online Multi-Object Tracking. In *Proceedings of the IEEE International Conference on Robotics and Automation.* (page 9, 11)

[129] Shi, S., Wang, X., and Li, H. (2019). PointRCNN: 3D Object Proposal Generation and Detection from Point Cloud. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition.* (page 3, 8, 73, 75, 96, 103, 112)

[130] Sivaraman, S. and Trivedi, M. M. (2013). Looking at Vehicles on the Road: A Survey of Vision-Based Vehicle Detection, Tracking, and Behavior Analysis. *IEEE Transactions on Intelligent Transportation Systems*, 14(4):1773–1795. (page 8)

[131] Spampinato, C., Chen-Burger, Y.-H., Nadarajan, G., and Fisher, R. (2008). Detecting, Tracking and Counting Fish in Low Quality Unconstrained Underwater Videos. In *Proceedings of the International Conference on Computer Vision Theory and Applications.* (page 2)

[132] Spampinato, C., Palazzo, S., Giordano, D., Kavasidis, I., Lin, F.-P., and Lin, Y. T. (2012). Covariance Based Fish Tracking in Real-Life Underwater Environment. *Proceedings of the International Conference on Computer Vision Theory and Applications*. (page 2)

[133] Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014). Dropout : A Simple Way to Prevent Neural Networks from Overfitting. *Journal of Machine Learning Research*, 15(Jun):1929–1958. (page 62)

[134] Sun, D., Yang, X., Liu, M.-Y., and Kautz, J. (2018). PWC-Net: CNNs for Optical Flow Using Pyramid, Warping, and Cost Volume. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. (page 11)

[135] Thorpe, C., Hebert, M. H., Kanade, T., and Shafer, S. A. (1988). Vision and Navigation for the Carnegie-Mellon Navlab. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 10(3):362–373. (page 1)

[136] Thrun, S., Burgard, W., and Fox, D. (2005). *Probabilistic Robotics*. MIT Press, first edition. (page 6, 19, 21, 26)

[137] Thrun, S., Montemerlo, M., Dahlkamp, H., Stavens, D., Aron, A., Diebel, J., Fong, P., Gale, J., Halpenny, M., Hoffmann, G., Lau, K., Oakley, C., Palatucci, M., Pratt, V., Stang, P., Strohband, S., Dupont, C., Jendrossek, L.-E., Koelen, C., Markey, C., Rummel, C., van Niekerk, J., Jensen, E., Alessandrini, P., Bradski, G., Davies, B., Ettinger, S., Kaehler, A., Nefian, A., and Mahoney, P. (2006). Stanley: The robot that won the DARPA Grand Challenge. *Journal of Field Robotics*, 23(9):661–692. (page 2)

[138] Tieleman, T. and Hinton, G. (2012). rmsprop: Divide the gradient by a running average of its recent magnitude. `http://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf`. Online; accessed Oktober 5, 2019. (page 61)

[139] Urmson, C., Anhalt, J., Bagnell, D., Baker, C., Bittner, R., Clark, M. N., Dolan, J., Duggins, D., Galatali, T., Geyer, C., Gittleman, M., Harbaugh, S., Hebert, M., Howard, T. M., Kolski, S., Kelly, A., Likhachev, M., McNaughton, M., Miller, N., Peterson, K., Pilnick, B., Rajkumar, R., Rybski, P., Salesky, B., Seo, Y. W., Singh, S., Snider, J., Stentz, A., Whittaker, W., Wolkowicki, Z., Ziglar, J., Bae, H., Brown, T., Demitrish, D., Litkouhi, B., Nickolaou, J., Sadekar, V., Zhang, W., Struble, J., Taylor, M., Darms, M., and Ferguson, D. (2008). Autonomous driving in Urban environments: Boss and the Urban Challenge. *Journal of Field Robotics Special Issue on the 2007 DARPA Urban Challenge, Part I*, 25(8):425–466. (page 2)

[140] Urmson, C., Ragusa, C., Ray, D., Anhalt, J., Bartz, D., Galatali, T., Gutierrez, A., Johnston, J., Harbaugh, S., Yu Kato, H., Messner, W., Miller, N., Peterson, K., Smith, B., Snider, J., Spiker, S., Ziglar, J., Red Whittaker, W., Clark, M., Koon, P., Mosher,

A., and Struble, J. (2006). A robust approach to high-speed navigation for unrehearsed desert terrain. *Journal of Field Robotics*, 23(8):467–508. (page 2)

[141] Van Der Merwe, R. (2004). *Sigma-Point Kalman Filters for Probabilistic Inference in Dynamic State-Space Models*. PhD thesis, Oregon Health & Science University, OGI School of Science & Engineering. (page 26, 30)

[142] Vatavu, A., Danescu, R., and Nedevschi, S. (2015). Stereovision-Based Multiple Object Tracking in Traffic Scenarios Using Free-Form Obstacle Delimiters and Particle Filters. *IEEE Transactions on Intelligent Transportation Systems*, 16(1):498–511. (page 11)

[143] Vo, B.-n., Mallick, M., Bar-shalom, Y., Coraluppi, S., Osborne, R., Mahler, R., and Vo, B.-T. (2015). Multitarget Tracking. In *Wiley Encyclopedia of Electrical and Electronics Engineering*, pages 1–15. John Wiley & Sons, Inc., first edition. (page 8, 9)

[144] Vo, B.-N., Singh, S., and Doucet, A. (2005). Sequential Monte Carlo methods for multitarget filtering with random finite sets. *IEEE Transactions on Aerospace and Electronic Systems*, 41(4):1224–1245. (page 10)

[145] Vo, B.-T. and Vo, B.-N. (2013). Labeled Random Finite Sets and Multi-Object Conjugate Priors. *IEEE Transactions on Signal Processing*, 61(13):3460–3475. (page 10)

[146] Voigtlaender, P., Krause, M., Osep, A., Luiten, J., Sekar, B. B. G., Geiger, A., and Leibe, B. (2019). MOTS: multi-object tracking and segmentation. In *IEEE Conference on Computer Vision and Pattern Recognition*. (page 113)

[147] Wan, E. A. and Van Der Merwe, R. (2000). The unscented Kalman filter for nonlinear estimation. In *Proceedings of the IEEE Adaptive Systems for Signal Processing, Communications, and Control Symposium*. (page 8, 26, 27)

[148] Wan, X., Wang, J., and Zhou, S. (2018). An Online and Flexible Multi-object Tracking Framework Using Long Short-Term Memory. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*. (page 8)

[149] Weiss, K., Stueker, D., and Kirchner, A. (2010). Target modeling and dynamic classification for adaptive sensor data fusion. In *Proceedings of the Intelligent Vehicles Symposium*. (page 5, 11)

[150] Weng, X. and Kitani, K. (2019). A Baseline for 3D Multi-Object Tracking. *arXiv CoRR*, abs/1907.03961. (page 104)

[151] Werbos, P. J. (1990). Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, 78(10):1550–1560. (page 56)

[152] Widrow, B. and Hoff, M. E. (1960). Adaptive Switching Circuits. In *1960 IRE WESCON Convention Record*, pages 96–104. IRE, first edition. (page 46)

[153] Williams, J. L. (2015). Marginal multi-bernoulli filters: RFS derivation of MHT, JIPDA, and association-based member. *IEEE Transactions on Aerospace and Electronic Systems*, 51(3):1664–1687. (page 10)

[154] Wojke, N. and Häselich, M. (2012). Moving vehicle detection and tracking in unstructured environments. *Proceedings of the IEEE International Conference on Robotics and Automation*. (page 2, 11)

[155] Xiao, B., Wu, H., and Wei, Y. (2018). Simple Baselines for Human Pose Estimation and Tracking. In *Proceedings of the European Conference on Computer Vision*. (page 11)

[156] Xing, J., Ai, H., Liu, L., and Lao, S. (2011). Multiple player tracking in sports video: A dual-mode two-way Bayesian inference approach with progressive observation modeling. *IEEE Transactions on Image Processing*, 20(6):1652–1667. (page 2)

[157] Yang, B., Luo, W., and Urtasun, R. (2018). PIXOR: Real-time 3D Object Detection from Point Clouds. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. (page 8, 97, 103, 112)

[158] Yang, H., Shao, L., Zheng, F., Wang, L., and Song, Z. (2011). Recent advances and trends in visual tracking: A review. *Neurocomputing*, 74(18):3823–3831. (page 8)

[159] Yilmaz, A., Javed, O., and Shah, M. (2006). Object tracking. *ACM Computing Surveys*, 38(4):13. (page 8)

[160] Yoon, K., Kim, D. Y., Yoon, Y. C., and Jeon, M. (2019). Data association for multi-object tracking via deep neural networks. *Sensors*, 19(3):1–15. (page 58, 62, 64, 65, 66, 67, 85, 86, 88)

[161] You, H., Jianjuan, X., and Xin, G. (2016). *Radar Data Processing With Applications*. Publishing House of Electronics Industry, first edition. (page 34, 39, 40, 41, 42)

[162] Zhang, S., Benenson, R., Omran, M., Hosang, J., and Schiele, B. (2016). How Far are We from Solving Pedestrian Detection? In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. (page 8)

[163] Zhang, W., Zhou, H., Sun, S., Wang, Z., Shi, J., and Loy, C. C. (2019). Robust multi-modality multi-object tracking. In *Proceedings of the IEEE International Conference on Computer Vision*. (page 113)

[164] Zhu, H., Yuen, K. V., Mihaylova, L., and Leung, H. (2017). Overview of Environment Perception for Intelligent Vehicles. *IEEE Transactions on Intelligent Transportation Systems*, 18(10):2584–2601. (page 8)