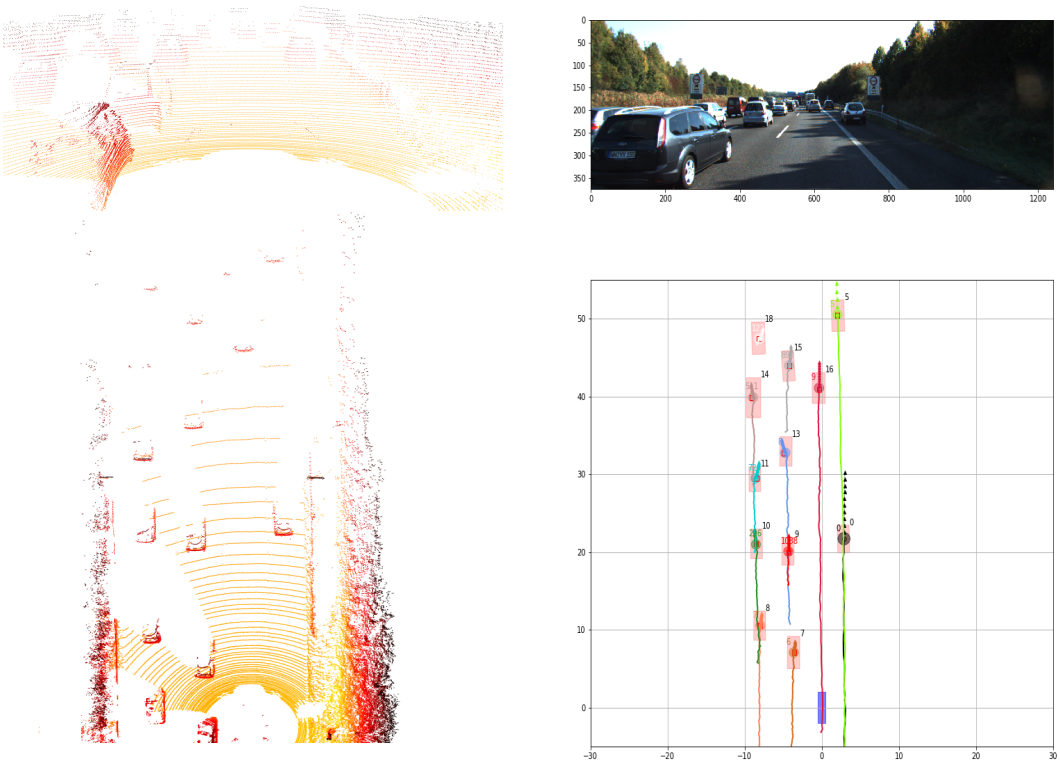
CHALMERS
UNIVERSITY OF TECHNOLOGY



Multi-Object Tracking using either End-to-End Deep Learning or PMBM filtering

Master's thesis in Systems, Control and Mechatronics

Erik Bohnsack, Adam Lilja

Department of Electrical Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2019

Master's thesis 2019:05

Multi-Object Tracking using either End-to-End Deep Learning or PMBM filtering

ERIK BOHNSACK
ADAM LILJA

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Electrical Engineering
Division of Signal Processing
Chalmers University of Technology
Gothenburg, Sweden 2019

Multi-Object Tracking using either End-to-End Deep Learning or PMBM filtering
ERIK BOHNSACK
ADAM LILJA

Supervisor: Selcuk Cavdar, Volvo Group Trucks Technology
Examiner: Karl Granström, Department of Electrical Engineering

Master's Thesis 2019
Department of Electrical Engineering
Division of Signal Processing
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: To the upper left a point cloud from an oracle view. To the lower left a Bird's Eye View snapshot of the same point cloud. To the right, the corresponding image and tracking visualization showing the PMBM filter using a Constant Acceleration motion model on simulated data from KITTI[1].

Typeset in LaTeX
Gothenburg, Sweden 2019

Multi-Object Tracking using either End-to-End Deep Learning or PMBM filtering
ERIK BOHNSACK
ADAM LILJA
Department of Eletrical Engineering
Chalmers University of Technology

Abstract

A vital part of autonomous vehicles is the ability to perceive the surrounding environment in order to navigate safely. Multi-Object Tracking, solving the issues of how many objects there are and their current dynamic state, is key to a vehicle's perception system. Tracking has traditionally been done in a tracking-by-detection way, relying on receiving measurements from a detector of some sort. Recent research has introduced end-to-end tracking using Deep Learning, jointly reasoning over detection, tracking and prediction. The aim of this thesis is to compare a conventional Bayesian filtering algorithm, a Poisson Multi-Bernoulli Mixture (`PMBM`) filter, with an end-to-end Deep Learning Neural network.
The end-to-end neural network (`ETENN`) show to be hard to generalize for data it has not trained on, but could possibly improve if more training data is available. The `PMBM` filter performs well, even when fed with noisy measurements.

All code can be found at `https://github.com/erikbohnsack/pmbm` and `https://github.com/erikbohnsack/etenn`.

Videos can be found via `www.adamlilja.com/m-thesis`

Keywords: Multi-Object Tracking, MOT, Deep Learning, Neural Networks, Bayesian Filtering, Poisson Multi-Bernoulli Mixture, PMBM, Fast-and-Furious, Artificial Intelligence, Autonomous Vehicles.

Acknowledgements

We would like to extend our humblest gratitude to our academic supervisor Karl Granström for his help throughout the spring. This implementation of a PMBM algorithm wouldn't have existed without you.

Also, thanks to professor Lennart Svensson for initial discussion and ideas regarding the thesis. Thanks to Yuxuan Xia for quick and elaborative answers on complicated questions.

Thanks to Volvo Group Trucks Technology for having us, where a special thanks go out to our supervisor, Selcuk Cavdar. You may eat slow, but you think fast. Thanks to Saimir Baci for all help, quick wit and wise quotes. Also, thanks to visiting scholar Eren Erdal Aksoy for asking the uncomfortable questions.

Erik Bohnsack, Adam Lilja, Gothenburg, May 2019

Contents

List of Figures

List of Tables

List of Algorithms

1

Introduction

Autonomous vehicles have over the last decade headlined tech news due to its potentially huge impact on our society. As we all rely on individual, public and industrial transport it is safe to say that improved efficiency in transportation leads to higher efficiency in society. In the race for full automation we have seen several vehicle automation applications in production vehicles such as adaptive cruise control and lane-keeping assistance emerge. More holistic systems are, however, yet to see the light of being production-ready.

A vital part of autonomous vehicles is the ability to perceive the surrounding environment in order to safely navigate. The perception accounts for many different aspects such as detecting nearby vehicles, pedestrians, drivable road, traffic lights, and road signs. High-performing Computer Vision algorithms using Deep Learning have over the last couple of years revolutionized the possibilities to detect surrounding objects in real-time, using monocular cameras, stereo cameras, or `LiDAR` sensors. However, an object detector will only tell you features such as where the object is, what it is and the size of it. To plan ahead it is important to estimate the surrounding objects' motion and predict their future positions. Therefore a multiple object tracker is needed, being able to estimate the surrounding objects' dynamic state.

Multi-Object Tracking (`MOT`) can be described as the problem of both determining the number of dynamic objects and each object's dynamic state. In automotive systems, this has earlier been done using a cascade approach, propagating detections of objects as measurements to Bayesian filtering methods. This is by all means a complex task, due to possible appearance changes of objects and surroundings, occlusion of objects, and data association, i.e. determining which measurements should be associated with which objects. Different to the cascade approach, recent Deep Learning progress has enabled end-to-end tracking algorithms using raw sensor data [8, 9, 10].

The focus of this thesis is exploring the differences between a Bayesian filtering algorithm and an end-to-end deep learning algorithm. For tracking-by-detection, the Poisson Multi-Bernoulli Mixture (`PMBM`) filter [11] has shown to be a promising tracking algorithm by yielding high performance while keeping the computational cost down [12]. Using `LiDAR` data has shown very promising results in object detection using PointPillars [6]. In combination with [8] and [9] this opens up for a novel end-to-end neural network (`ETENN`) approach.

1.1 Problem Formulation

The goal of this thesis is to investigate the difference between a Bayesian tracking-by-detection algorithm and a Deep Learning End-to-End method by implementing state-of-the-art for the two cases. A Poisson Multi-Bernoulli Mixture (PMBM) filter algorithm for the Bayesian case and a neural network inspired by Fast and Furious [8] for the End-to-End deep learning case. The PMBM filter is to be implemented for several different motion models and to be tested on both simulated data and detections from the End-to-end neural network (ETENN). The end-to-end deep learning network is to be tested using two input processes and two network structures.

1.2 Related Work

Starting already in the 1960's with tracking aeroplanes using radar for air-traffic safety research within object tracking is still on-going. Traditionally, object tracking has been rooted in statistics and filtering, using models describing measurements and states. In the tracking survey Tracking the Trackers [13], Leal-Taixé et al. analyses state-of-the-art in multiple object tracking based on submissions to the MOT15 [14] or MOT16 [15] benchmarks.

Lately, a new branch of tracking has emerged using neural networks for solving not only the detection of objects, but also the tracking problem as of whole. The survey [16] focuses on providing an overview of the machine learning approaches currently available to solve the data association problem in multiple object tracking. They do not compare any methods, but they provide a good overview of the current literature available in the different areas.

We will in this chapter try to further discuss the current state-of-the-art in Object detection, Conventional tracking, and Deep Learning tracking.

1.2.1 2D Object Detection

Deep Learning has been proven to perform very well for object detection in images. So called two-staged detectors, proposing interesting regions and classifying those regions, performed very well, but were far too slow to apply in real-time (24Hz) operation [17, 18, 19]. S. Ren et al. introduced region proposal networks in [20], almost achieving real-time while maintaining high accuracy. Later one-staged detectors [21] [22] performed slightly worse when it comes to accuracy, but were profoundly quicker. Accuracy for one-staged detectors have lately increased by novelties introduced in [23] [24] [25].

1.2.2 3D Object Detection - Camera

There are currently algorithms not only for predicting 2D-bounding boxes, but also 3D-bounding boxes, despite using only mono cameras [26]. However, using stereo cameras and LiDARs has shown better performance [27].

1.2.3 3D Object Detection - LiDAR

The combination of a point cloud's huge amount of data and the sparsity of the data, does not fall as natural with CNNs as images. PointNet [28] combat this by using the raw point cloud, point wise, as input together with fully connected layers. Instead of processing each point, it is common to divide the point cloud into a grid of some sort to use convolution layers. In [29, 30] they use extended 2D object detection by using 3D convolutions over the 3 dimensional space, which show good results, but are notoriously slow. To gain speed, [31] and [32] project the point cloud as a 2D image with channels, similar to an RGB image, but from a Bird's eye view (BEV) perspective. In [31] they encode height channels, while [32] creates a three channel image with height, intensity and density of the point cloud in each pixel. Similarly, [33] and [34] projects to a BEV, but with both height maps, intensity and density. Chen et al. [33] also includes a front-view projection and an RGB image as well. To find an optimal representation, [7] introduces a Voxel Feature Encoding (VFE) layer, which learns how to represent each voxel as optimal as possible. Both [35] and [6] extend this work, [35] by introducing a sparse convolutional network to increase efficiency, and [6] by encoding pillars which reduce the point cloud to a 2D image with a number of feature channels.

1.2.4 Conventional Tracking

There are several developed techniques to battle the problem of multi-object tracking using filtering. In [36] they name Joint Probabilistic Data Association (JPDA), Multiple Hypothesis Tracking (MHT), and Random Finite Sets (RFS) as the most common approaches. In [37] they state that JPDA and MHT are well established, while RFS is an emerging paradigm that has gotten a great deal of attention during the last decade. JPDA is a method to associate measurements at each time instance with existing objects by using a joint probabilistic score. The MHT revisited [38] shows good results in the 2017 survey by Leal Taixé et al. [13], while JPDA revisited [39] performance lags behind the other algorithms.

For keeping the label information from detectors one can use Labeled Random Finite Sets or Generalized Labeled Finite Sets, something which is done in [40, 41]. Here both undetected and detected objects as well as clutter is modelled as labeled multi-Bernoulli distributions.

Another interesting way of using RFS is called Poisson Multi-Bernoulli Mixture (PMBM), first introduced in [42] and [43] and is thoroughly investigated in [11] where also derivation and implementation is presented. Here all detected objects are modelled as Bernoulli distributions, whereas the undetected objects and the clutter is modeled with Poisson's distribution (hence the name Poisson Multi-Bernoulli mixture). In [12], the authors evaluates different multi-Bernoulli conjugate prior filters and argue that PMBM has the overall best performance concerning the metric GOSPA [44] and computational time.

1.2.5 Deep Learning Tracking

In a tracking-by-detection sense, receiving detections as measurements from an object detector, [45] uses Recurrent Neural Networks (RNN) and Long Short Term Memory (LSTM) networks to track and solve the data association. Xiang et al. formulates the problem as a reinforcement learning problem, where they model the objects as Markov Decision Processes.

Algorithms for visual tracking performs well, often by crafting affinity measures comparing two consecutive video frames. In [46] they use RNNs with affinity measures for appearance, motion and interaction. Zhu et al. [47] proposes a dual matching attention network to handle the data association, using matching patterns of image pairs. Tang et al. [48] formulates the problem as a graph-based problem, using appearance matching. Choi [49] proposes a local flow descriptor, encoding the relative motion pattern between two temporally different detections. In [50] they use a deep feature extractor, an affinity estimator and the Hungarian Algorithm. GOTURN [51] is a simple but highly effective single object tracker, using the past detection to search and compare regions over time. Leal-Taixé et al. [52] matches associations by learning a Siamese CNN to match based spatial-temporal features.

Luo et al. [8] proposes an end-to-end approach called *Fast and Furious*, due to its efficiency, which jointly reasons about 3D detection, tracking and motion forecasting using only a single convolutional neural network. They input past n frames of LiDAR data to the network to be able to reason over time and predict the object's location n frames ahead. Frossard et al. [10] uses both camera and LiDAR data, inferring for each time step separately, to output detections which they input to a matching network suggesting connections over time.

1.3 Structure of the Report

The report starts with outlining the theory behind and our implementation of the PMBM filter in chapter 2. We proceed by going through the theory of neural networks and the implementation of our end-to-end neural network ETENN in chapter 3. The results of the experiments discussed in Chapter 4 are then presented in chapter 5, Results. Discussion regarding the method and results are held in chapter 6 and the thesis is concluded in chapter 7.

2

Poisson Multi-Bernoulli Mixture Filter

The Poisson Multi Bernoulli Mixture (PMBM) filter is a, relatively, new approach to the Multiple Object Tracking (MOT) problem. It is able to track object by using Bayesian recursion, while it models undetected objects and detected objects as two different probability distributions. It has been proven to outperform other techniques [11].

2.1 Theory

In order to understand how to implement a PMBM filter, some basic theory is necessary. The PMBM filter is a Bayesian recursion algorithm for Multi-Object Tracking, it naturally lies on the foundation of Probability theory, Set Theory and Bayesian State Estimation.

2.1.1 Probability theory

The Bayesian interpretation of probability, describes the quantities of interest as random. It regards probability as a degree of belief of a quantity or event, where the degree of belief is set from a prior knowledge and updated based on retrieved measurements/data. Another interpretation of probability is the frequentism, which describes the quantities of interest as unknown, but deterministic. The work in this thesis relies on Bayesian statistics.

The probability density function (PDF) represents a distribution of a continuous random variable and is used to specify the probability of a random variable falling within a certain range of values. Integrating over the PDF will give the probability that the random variable will fall in that specified integral interval. The integral for a PDF naturally sums to one. The probability mass function (PMF) is similar to PDF, but represents the discrete random variables probability of taking a specific exact.

Probability theory holds many types of distributions. Bernoulli, Poisson, Gaussian and Uniform are important distributions used in the thesis.

2.1.1.1 Bernoulli Distribution

The Bernoulli distribution is a special case of the Binomial distribution, but for a single trial only. The binomial can be described of the discrete probability of how many successes k you get when you conduct n independent experiments, each with a probability p to result in 1 or $q = 1 - p$ to result in 0 with the PMF

$$f(k, n, p) = \begin{pmatrix} n \\ k \end{pmatrix} p^k (1-p)^{n-k} \tag{2.1}$$

The factor $\begin{pmatrix} n \\ k \end{pmatrix}$ is the binomial coefficient, which is how many number of different ways k successes can come to be out of n trials. The Bernoulli distribution is the discrete probability distribution of a trial with two possible outcomes, either 1 or 0, with respective probability p and $q = 1 - p$. The probability mass function can be written as

$$f(k; p) = \begin{cases} p & \text{if } k = 1 \\ q = 1 - p & \text{if } k = 0 \end{cases} \tag{2.2}$$

where k is the possible outcome, and p and q are the respective possibilities. The Bernoulli distribution is used to model the objects' distributions given that the object is either detected ($k = 1$) or missdetected ($k = 0$).

2.1.1.2 Poisson distribution

The probability mass function of a Poisson distribution is

$$f(k, \lambda) = e^{-\lambda} \frac{\lambda^k}{k!} \tag{2.3}$$

where k is the number of events and λ is the average number of events occurring. λ can also be seen as the expected number of events in an interval [a,b], noted $N(a, b)$, namely $E[(a, b)] = \lambda(b - a)$.

A Poisson Point Process (PPP) is a random process being tightly connected to the Poisson distribution. The intensity of a PPP, λ, equals the expected number of events within a interval of unit length (0, 1). If a random number of points within a finite interval is a PPP, the number of points is a random variable with a Poisson distribution. The Poisson distribution is a discrete random probability which represents the probability of events occurring. Thus can the Poisson distribution be used to model the birth of new, previously unseen, objects as well as identify clutter measurements.

2.1.1.3 Gaussian distribution

The Gaussian distribution's PDF is bell-shaped and specified by the mean μ and variance σ^2:

$$f(x|\mu, \sigma^2) = \mathcal{N}(x; \mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}} \tag{2.4}$$

By modelling each object's states as Gaussian distributions it is possible to account for noisy measurements. The states are assumed to be normally distributed with an uncertainty instead of having a firm value.

2.1.1.4 Uniform distribution

In this thesis the continuous uniform distribution is of interest. It distributes the probability evenly throughout the entire interval. For the one-dimensional case the PDF is

$$p(x|b,a) = \begin{cases} \frac{1}{b-a}, & \text{if } x \in [a,b] \\ 0, & \text{otherwise} \end{cases} \tag{2.5}$$

This can be used in the PMBM filter's birth process if assuming it is equally likely that an object spawns throughout the entire field-of-view.

2.1.2 Bayesian State Estimation

Bayesian state estimation, also called Bayesian filtering, is a probabilistic approach to estimate an unknown probability density function. Bayesian probability relies on the Bayes theorem

$$p(x|z) = \frac{p(z|x)p(x)}{p(z)} \propto p(z|x)p(x) \tag{2.6}$$

which describes the posterior probability of the random variable x given z. $p(z)$ is often seen as a normalization factor giving the proportionality.

2.1.2.1 Bayesian distributions

Prior distribution $p(\mathbf{x})$ is a probability distribution of the state $\mathbf{x}$ which reflects your beliefs before any evidence is observed.

The likelihood function $p(\mathbf{z}|\mathbf{x})$ is the probability of getting a measurement $\mathbf{z}$ given the state $\mathbf{x}$.

The posterior distribution is the distribution of the state $\mathbf{x}$ conditioned on the evidence observed $\mathbf{z}$. It is proportionate to the prior times the likelihood, as the denominator is only a normalization constant. The posterior is computed using the Bayes' theorem:

$$p(\mathbf{x}|\mathbf{z}) = \frac{p(\mathbf{z}|\mathbf{x})p(\mathbf{x})}{\int p(\mathbf{z}|\mathbf{x})p(\mathbf{x})d\mathbf{x}} \propto p(\mathbf{z}|\mathbf{x})p(\mathbf{x}) \tag{2.7}$$

2.1.2.2 Prediction and measurement update

In order to estimate a state using Bayesian estimation, two updates are subsequently performed. When the posterior distribution and the prior distribution are of the same probability density family it is called a Conjugate prior. This enables a closed-form expression of the posterior. First, based on a prior/posterior from the last time instance $p(\mathbf{x}_k|\mathbf{z}_{k-1})$, a prediction is done to estimate the state x_k. The prediction is performed using the Chapman-Kolmogorov equation

$$p(\mathbf{x}_k|\mathbf{z}_{k-1}) = \int p(\mathbf{x}_k|\mathbf{x}_{k-1})p(\mathbf{x}_{k-1}|\mathbf{z}_{k-1})d\mathbf{x}_{k-1} \tag{2.8}$$

Here $p(\mathbf{x}_k|\mathbf{x}_{k-1})$ represents a transition density, which is often described by a motion model.

Once a measurement has been observed, a measurement update is performed to correct the prediction. The measurement update is derived from the posterior, Equation (2.7), giving:

$$p(\mathbf{x}_k|\mathbf{z}_{1:k}) = \frac{p(\mathbf{z}_k|\mathbf{x}_k)p(\mathbf{x}_k|\mathbf{z}_{1:k-1})}{\int p(\mathbf{z}_k|\mathbf{x}_k)p(\mathbf{x}_k|\mathbf{z}_{1:k-1})d\mathbf{x}_k} \propto p(\mathbf{z}_k|\mathbf{x}_k)p(\mathbf{x}_k|\mathbf{z}_{1:k-1}) \tag{2.9}$$

Here, $p(\mathbf{z}_k|\mathbf{x}_k)$ represents the likelihood function, which is modelled using measurement models.

2.1.2.3 Kalman Filter

Generally, the Bayesian filtering updates do not have a closed form solution. However, if one assumes that the models are linear with additive Gaussian noise, the Kalman filter is the closed form solution, likewise being the optimal minimum mean square error (`MMSE`) estimator. Using a motion model with state transition F and covariance process noise Q, a measurement model with observation matrix H and observation noise covariance of R, and assuming no external input added to the system in any time step, k, the next time step's predicted states, $\mathbf{x}_{k|k-1}$, are:

$$\mathbf{x}_{k|k-1} = F\mathbf{x}_{k-1|k-1} \tag{2.10}$$

and the predicted error covariance

$$\mathbf{P}_{k|k-1} = F\mathbf{P}_{k-1|k-1}F^T + Q \tag{2.11}$$

It is then possible to update these priors using the measurements in the current time step:

$$\mathbf{x}_{k|k} = \mathbf{x}_{k|k-1} + K_k\mathbf{y}_k \tag{2.12}$$

$$\mathbf{P}_{k|k} = (I - K_kH)\mathbf{P}_{k|k-1} \tag{2.13}$$

where the measurement residual is

$$\mathbf{y}_k = \mathbf{z}_k - H\mathbf{x}_{k|k-1} \tag{2.14}$$

the innovation:

$$S_k = R + H\mathbf{P}_{k|k-1}H^T \tag{2.15}$$

and the Kalman gain

$$K_k = \mathbf{P}_{k|k-1}H^TS_k^{-1} \tag{2.16}$$

2.1.2.4 Unscented Kalman Filter

The ordinary Kalman Filter relies on linear state transition and measurement models to complete the prediction update and measurement update. However, if it is the case of nonlinear dynamics, there are special methods to encounter the non-linearities. Extended Kalman Filter (`EKF`) requires differential state transition and measurement models, as it uses linearization before using the basic Kalman filter update. Unscented Kalman Filter (`UKF`) uses a deterministic sampling technique, which propagates specifically placed points through the non-linear function, from which it forms a new mean and covariance in what they call an Unscented Transform. The work in this thesis uses `UKF` to perform nonlinear Kalman updates, hence only this method will be explained in detail.

The $2L + 1$ points to be propagated are called sigma points. The sigma points are sampled around the prior distribution $\mathcal{N}(\mathbf{x},\ \mathbf{P_x})$ according to:

$$\mathcal{X}_0 = \mathbf{x} \tag{2.17}$$

$$\mathcal{X}_i = \mathbf{x} + \left(\sqrt{(L+\lambda)\mathbf{P_x}}\right)_i \qquad i = 1, ..., L \tag{2.18}$$

$$\mathcal{X}_i = \mathbf{x} - \left(\sqrt{(L+\lambda)\mathbf{P_x}}\right)_{i-L} \qquad i = L+1, ..., 2L \tag{2.19}$$

Each sigma point have a corresponding weight W_i:

$$W_0^{(m)} = \frac{\lambda}{\lambda + L} \tag{2.20}$$

$$W_0^{(c)} = \frac{\lambda}{\lambda + L} + (1 - \alpha^2 + \beta) \tag{2.21}$$

$$W_i^{(m)} = W_0^{(c)} = \frac{1}{2(\lambda + L)} \qquad i = 1, ..., 2L \tag{2.22}$$

$$\lambda = \alpha^2(L + \kappa) - L \tag{2.23}$$

Where λ is a scaling parameter, set from the hyperparameters α and κ. α affects the spread of the points around $\mathbf{x}$, often set to a small positive value, while κ is often set to 0 [53]. β incorporates prior knowledge of the states' distributions, and for Gaussian distributions $\beta = 2$ is optimal.

For the prediction step of the Bayesian recursion, the sigma points are propagated through the nonlinear state process function

$$\mathcal{Y}_i = g(\mathcal{X}_i), \qquad i = 0, ..., 2L \tag{2.24}$$

The predicted mean and covariance are approximated using the weighted mean and covariance from the propagated points

$$\hat{\mathbf{x}}_{k|k-1} \approx \sum_{0}^{2L} W_i^{(m)}\, \mathcal{Y}_i \tag{2.25}$$

$$\mathbf{P}_{k|k-1} \approx \sum_{i=0}^{2L} W_i^{(c)} [\mathcal{Y}_i - \mathbf{y}][\mathcal{Y}_i - \mathbf{y}]^T + \mathbf{Q} \tag{2.26}$$

where $\mathbf{Q}$ is the process noise matrix.

To perform the update step, new sigma points $\mathcal{X}_i$ from the predicted mean and covariance are propagated through the measurement function h:

$$\mathcal{Z}_i = h(\mathcal{X}_i), \qquad i = 0, ..., 2L \tag{2.27}$$

and with the measurement noise matrix $\mathbf{R}$, the empirical measurement mean, $\hat{\mathbf{z}}$, and covariance, $\hat{\mathbf{S}}$, is given through:

$$\hat{\mathbf{z}} = \sum_{0}^{2L} W_i^{(m)} \mathcal{Z}_i \tag{2.28}$$

$$\hat{\mathbf{S}} \approx \sum_{i=0}^{2L} W_i^{(c)} [\mathcal{Z}_i - \hat{\mathbf{z}}][\mathcal{Z}_i - \hat{\mathbf{z}}]^T + \mathbf{R} \tag{2.29}$$

After propagating the predicted state to measurement space, one needs to perform the Kalman update equations, (2.30) - (2.33), similar to the basic linear case:

$$\mathbf{P}_{z,x} = \sum_{i=0}^{2L} W_i^{(c)} [\mathcal{Z}_i - \hat{z}][\mathcal{X}_i - \hat{\mathbf{x}}_{k|k-1}] \tag{2.30}$$

$$\mathbf{K} = \mathbf{P}_{z,x} \hat{\mathbf{S}}^{-1} \tag{2.31}$$

$$\hat{\mathbf{x}}_{k|k} = \hat{\mathbf{x}}_{k|k-1} - \mathbf{K}(\mathbf{z} - \hat{\mathbf{z}}) \tag{2.32}$$

$$\mathbf{P}_{k|k} = \mathbf{P}_{k|k-1} - \mathbf{K}\hat{\mathbf{S}}\mathbf{K}^T \tag{2.33}$$

2.1.3 Dynamics

Models of how the states of objects change with time are essential for predicting the potential movement in between measurements and are referred to as Motion Models. In order to extract the needed information from sensors' output and map it to the filtered states a Measurement Models is used. These two models are further discussed below.

2.1.3.1 Motion Models

Motion models are modelling the states' behaviour in between measurements. Given a certain state one can predict the next state by propagating through a motion model. Not only the state, e.g. position and velocity, is of interest, but also the accuracy of the new state. Mathematically a motion model is defined as the propagation of the old state through a function with some added noise

$$x_k = f_{k-1}(x_{k-1}) + q_{k-1} \tag{2.34}$$

There are different well known motion models, one is **Constant Velocity** (CV) which uses the state vector containing positions and velocities $\mathbf{x} = \begin{bmatrix}\mathbf{p} & \mathbf{v}\end{bmatrix}^T$. The update is linear and performed through the matrix operations:

$$\mathbf{x}_k = \mathbf{F}x_{k-1} + q_{k-1} \tag{2.35}$$

where the motion noise is expected to be Gaussian $q_{k-1} \sim \mathcal{N}(0, \mathbf{Q})$. The position is updated by Euler integrating the velocity ($\mathbf{p} = \Delta t * \mathbf{v}$) and the velocity is remains unchanged. The matrices $\mathbf{F}$ and $\mathbf{Q}$ for the two dimensional position case ($\mathbf{x} = [x, y, \dot{x}, \dot{y}]$), which is of interest in this thesis, are defined as

$$\mathbf{F} = \begin{bmatrix} 1 & 0 & \Delta t & 0 \\ 0 & 1 & 0 & \Delta t \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.36}$$

$$\mathbf{Q}^{CV} = \sigma_Q^2 \begin{bmatrix} \frac{\Delta t^3}{3} & 0 & \frac{\Delta t}{2} & 0 \\ 0 & \frac{\Delta t^3}{3} & 0 & \frac{\Delta t^2}{2} \\ \frac{\Delta t^2}{2} & 0 & \Delta t & 0 \\ 0 & \frac{\Delta t^2}{2} & 0 & \Delta t \end{bmatrix} \tag{2.37}$$

Where σ_Q^2 is the motion noise which needs to be tuned to fit the application and Δt is how long time in the future to predict.

Taking it one step further than the CV-model is the **Constant Acceleration** (CA) model. For the two dimensional case the state vector is $\mathbf{x} = [x, y, \dot{x}, \dot{y}, \ddot{x}, \ddot{y}]$, and the transition model and process noise are

$$\mathbf{F} = \begin{bmatrix} 1 & 0 & \Delta t & 0 & \frac{T^2}{2} & 0 \\ 0 & 1 & 0 & \Delta t & 0 & \frac{T^2}{2} \\ 0 & 0 & 1 & 0 & \Delta t & 0 \\ 0 & 0 & 0 & 1 & 0 & \Delta t \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.38}$$

$$\mathbf{Q}^{CA} = \sigma_Q^2 \begin{bmatrix} \frac{\Delta t^5}{20} & 0 & \frac{\Delta t^4}{8} & 0 & \frac{\Delta t^3}{6} & 0 \\ 0 & \frac{\Delta t^5}{20} & 0 & \frac{\Delta t^4}{8} & 0 & \frac{\Delta t^3}{6} \\ \frac{\Delta t^4}{8} & 0 & \frac{\Delta t^3}{3} & 0 & \frac{\Delta t^2}{2} & 0 \\ 0 & \frac{\Delta t^4}{8} & 0 & \frac{\Delta t^3}{3} & 0 & \frac{\Delta t^2}{2} \\ \frac{\Delta t^3}{6} & 0 & \frac{\Delta t^2}{2} & 0 & \Delta t & 0 \\ 0 & \frac{\Delta t^3}{6} & 0 & \frac{\Delta t^2}{2} & 0 & \Delta t \end{bmatrix} \tag{2.39}$$

Another motion model is the non-linear **Bicycle Model**. There are several ways of defining this model and the definition used in this thesis is based on [2]. The state vector is, for the 2D case, $\mathbf{x} = \begin{bmatrix} x & y & \psi & v & \delta \end{bmatrix}^T$, where ψ is the heading, v the speed and δ the steering angle, as seen in Figure 2.1. The state vector, assuming

constant velocity and steering angle, is updated through:

$$\mathbf{x} = \begin{bmatrix} x \\ y \\ \psi \\ v \\ \delta \end{bmatrix} = \begin{bmatrix} x + v \cdot \cos(\psi + \beta)\Delta t \\ y + v \cdot \sin(\psi + \beta)\Delta t \\ \psi + \frac{v}{l_r} \cdot \sin(\beta)\Delta t \\ v \\ \delta \end{bmatrix} \tag{2.40}$$

where

$$\beta = \operatorname{atan}_2\left(l_r \cdot \tan(\delta),\ (l_r + l_f)\right)$$

and l_r and l_f are lengths of the rear and forward part of the objects. The process noise is a matrix of tunable parameters.

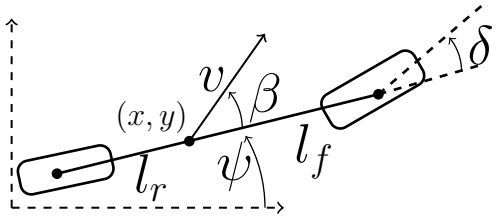


Figure 2.1: Variable definitions of Bicycle Model in accordance to [2]

2.1.3.2 Measurement Models

The measurement model defines the relationship between the sensors' outputs and the states currently being filtered. For the current time step k, the raw sensor output y_k, and some known, or model of, noise or bias r_k yield the observations through a measurement model,

$$z_k = h_k\left(y_k\right) + r_k \tag{2.41}$$

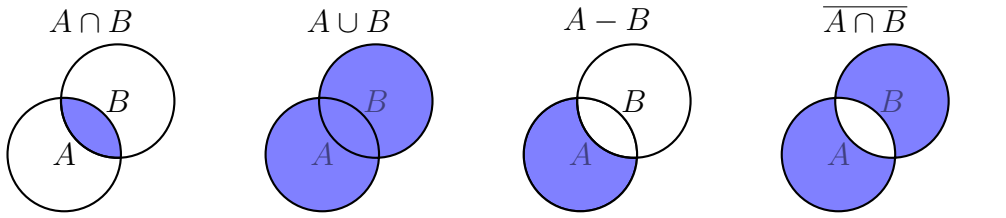
Of course, this is conditioned on that the information is extractable from the raw sensor data. If the measurement model is linear with Gaussian noise our measurement becomes:

$$z_k = H y_k + r\ , \quad r \sim \mathcal{N}(0, R) \tag{2.42}$$

where R is the measurement noise covariance matrix.

2.1.4 Set Theory

A set may contain anything: a digit, vector, matrix or another set. The content of a set is unordered and can be empty, infinite or anything in between. In Figure 2.2 some basic set relations are seen. The intersection $\cap$ means only the content two (or more) sets have in common, whereas the union $\cup$ represents the sum of sets' content. The exclusive or, $\overline{A \cap B}$, is the content where the sets differs. The subtraction is, as you would suspect, removing one part from another.
An element is *in*, $\in$, a set if one can find the element in the set. In Figure 2.3 for instance A is in S_1, or mathematically: $A \in S_1$. In the same figure one can se that the set S_1 also contains B. The element C, however, is not part of set S_1 but it is



(a) Intersection **(b)** Union **(c)** Subtraction **(d)** Exclusive or

Figure 2.2: Some basic set operations where the result of respective operation is marked blue

in S_2, i.e. $C \in S_2$ and $C \notin S_1$. In Figure 2.3 it can also be see that the intersection between S_1 and S_2 is non-zero, i.e. $S_1 \cap S_2 \neq \emptyset$ and thus having some elements in common, B for instance. S_3 on the other hand is *disjoint* from both S_1 and S_2, i.e. $S_1 \cap S_3 = S_2 \cap S_3 = \emptyset$. Two sets are disjoint when the sets have no elements in common. For a family of sets S, if all pairs in the set is disjoint to each other, the sets would be mutually disjoint. Elements of mutually disjoint sets is denoted $S_1 \uplus ... \uplus S_n = S$. Note that the sets in Figure 2.3 are not mutually disjoint since $S_1 \cap S_2 \neq \emptyset$.

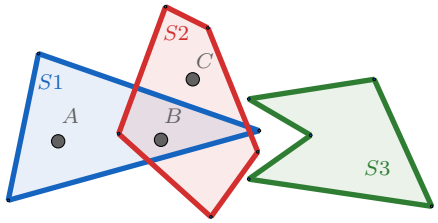


Figure 2.3: Three sets and three points. $A \in S_1$, $B \in S_1$, $B \in S_2$, and $C \in S_2$ are all true. S_1 and S_2 are joint since their intersection is non-empty. S_3 however is disjoint from both S_1 and S_2.

A Random Finite Set (RFS) is a set where both the cardinality and the variables within the set is random [54, 55]. Due to the nature of the MOT problem, where the amount of objects and the object's state is unknown, combining Bayesian recursion with RFS has shown to be a promising method to derive solutions from. Using RFS for MOT, one models both the single object state as a stochastic variable as well as the number of objects. During the update step in the Bayesian recursion the measurements of the object state is given. Modelling the object states like such and given a prior multi-object state as $f(\cdot)$ and a measurement likelihood of $\mathbf{Z}$ given $\mathbf{X}$, $l(\mathbf{Z}|\mathbf{X})$, Bayes' rule gives the posterior multi-object density [11]

$$p(\mathbf{X}|\mathbf{Z}) = \frac{l(\mathbf{Z}|\mathbf{X})p(\mathbf{X})}{\rho(\mathbf{Z})} \tag{2.43}$$

2.1.5 Poisson Multi-Bernoulli Mixture Filter

The Poisson Multi-Bernoulli Mixture (PMBM) filter is a RFS-based algorithm, first derived in [42, 43] and later extended in [11]. The key part of the filter is the ability to model undetected and detected objects differently. The undetected objects are modelled using a distribution, while detected objects are modelled by a Multi-Bernoulli Mixture (MBM). Combining the two densities gives a density shown in equation (2.44), where $\mathcal{P}_{k|k}$ is the Poisson density, $\mathcal{MBM}_{k|k}$ is the Multi-Bernoulli mixture. It also allows to compute the prediction and update steps for single object densities instead of full multi-object predictions and updates, reducing the complexity [11]. By letting $\mathbf{X}$ be the set of all objects, which can be divided into the disjoint union of the set of undetected objects $\mathbf{X}^u$ and the set of detected objects $\mathbf{X}^d$, the distribution becomes

$$\mathcal{PMBM}_{k|k}(\mathbf{X}) = \sum_{\mathbf{X}^u \uplus \mathbf{X}^d = \mathbf{X}} \mathcal{P}_{k|k}(\mathbf{X}^u)\mathcal{MBM}_{k|k}(\mathbf{X}^d) \tag{2.44}$$

In [11] they derive a proof of conjugacy for the PMBM filter, meaning that the distribution after the prediction and update step will be of the same distribution form as the initial prior. Thus, given the updated PMBM density from previous time step $\mathcal{PMBM}_{k|k}(\mathbf{X}_k)$ and motion model $p(\mathbf{X}_{k+1}|\mathbf{X}_k)$ the next time step's predicted PMBM density can be written:

$$\mathcal{PMBM}_{k+1|k}(\mathbf{X}_k) = \int p(\mathbf{X}_{k+1}|\mathbf{X}_k)\mathcal{PMBM}_{k|k}(\mathbf{X}_k)\delta\mathbf{X}_k \tag{2.45}$$

Furthermore, by adding information from the measurement model $p(\mathbf{Z}_k|\mathbf{X}_k)$, one can update the PMBM density with:

$$\mathcal{PMBM}_{k+1|k+1}(\mathbf{X}_{k+1}) = \frac{p(\mathbf{Z}_{k+1}|\mathbf{X}_{k+1})\mathcal{PMBM}_{k+1|k}(\mathbf{X}_{k+1})}{\int p(\mathbf{Z}_{k+1}|\mathbf{X'}_{k+1})\mathcal{PMBM}_{k+1|k}(\mathbf{X'}_{k+1})\delta\mathbf{X'}_k} \tag{2.46}$$

Before explaining the Bayesian recursion, hypotheses methodology, reduction, and estimation steps in detail, the PMBM filter's structure in a holistic sense is shown in Figure 2.4. Furthermore Algorithm 2.1 combines the details explained in upcoming sections below.

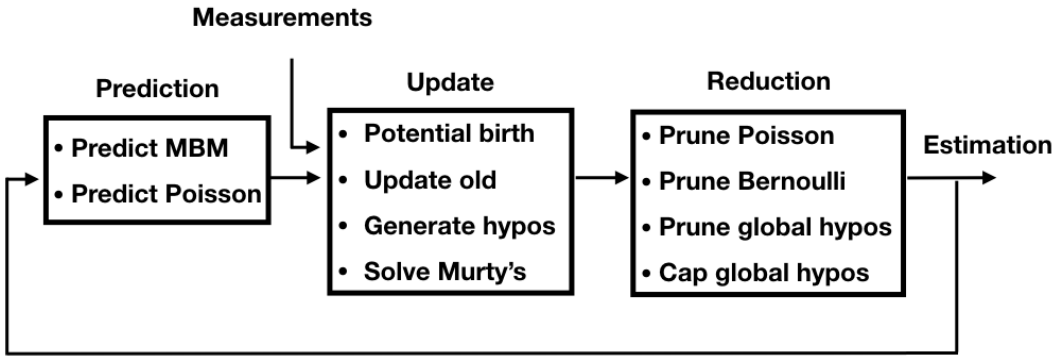


Figure 2.4: Flowchart of the PMBM Filter algorithm, showing the three steps, prediction, update and reduction, with their respective details. Here *Solve Murty's* regards the association optimization algorithm called Murty's algorithm.

Algorithm 2.1 Pseudo code for one prediction, update, estimation and reduction step for the PMBM algorithm

Inputs:
Posterior from previous time step: $\mathcal{PMBM}_{k-1|k-1}$
New set of measurements: Z
Output:
Posterior for current time step: $\mathcal{PMBM}_{k|k}$

- Perform prediction, i.e. acquire $\mathcal{PMBM}_{k|k-1}$

Find potential new targets among the Poisson components:
for $z \in Z$ **do**
  - Perform gating w.r.t all Poisson components
  if *Number of gated Poisson components* > 1 **then**
    - Perform Moment Matching
  end
  if z *within gate of* $\bar{x}^{\mu}_{j,i}$ **then**
    - Create new detected target $\bar{x}^{d}_{j,i}$
  end
end
Update the previously detected targets:
for $i \in [1, N_d]$ **do**
  for $s \in [1,$ *Number of Single Target Hypotheses for* $\bar{x}^{d}_{j,i}]$ **do**
    - Create missed detection hypothesis
    - Perform gating and create plausible hypotheses
  end
end
for $j \in [1,$ *Number of Global Hypotheses*$]$ **do**
  - Create cost matrix
  - Generate the K best global hypotheses using Murty's Algorithm
end
- Estimate the target states

Perform reduction:
- Prune Poisson components with $w^{u}_{i} < \Gamma^{u}$
- Prune Bernoulli components with $w^{d}_{j,i} < \Gamma^{d}$
- Prune Global Hypotheses with $w^{j} < \Gamma^{j}$

2.1.5.1 Hypotheses

For each measurement at each time step, there are three possible outcomes, either it's a newly detected target, a previously detected target or clutter (a faulty measurement not belonging to any real object). To combat the uncertainty in the association, which measurement belongs where, a methodology of hypotheses is needed and there are currently two ways one can go about to do this, hypothesis-oriented [56] and track-oriented [57]. The work in this thesis uses a track-oriented approach as it's computationally more efficient [3], hence the following section will explain how a track-oriented PMBM filter works.

As mentioned in section 2.1.5, each detected target is modelled as a Bernoulli distribution. In a track-oriented approach, this Bernoulli distribution is included in a "track". Figure 2.5 displays a track-oriented PMBM hypothesis tree, displaying three tracks. Here the track of object x^1 includes the single target association history hypotheses, showing that at time 1, the object was associated to measurement z_1^1, while at time 2 it was not detected. At time 3, it shows two possible associations, so called *Single Target Hypothesis*, (STH), for the object. Either a missdetection, 0, or an association with z_3^1.

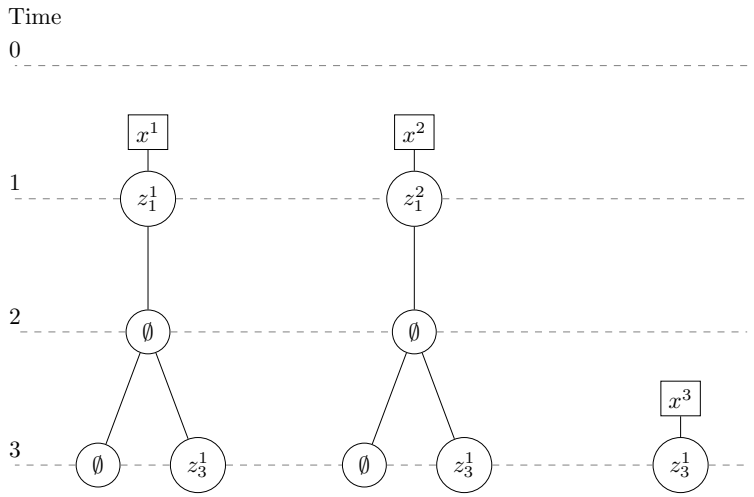


Figure 2.5: Example of track-oriented PMBM hypothesis tree. The tree shows the single target hypothesis association history for object x^1 and x^2 and a new potentially detected object x^3. The figure is kindly borrowed from [3].

Track-oriented PMBM forms global association hypotheses from the possible combinations of the single target hypotheses, containing one single target hypothesis from each potential target, with the restriction that one measurement can only be associated with one STH. For instance, in the example in Figure 2.5, a feasible global hypotheses at time 3 would be { x^1:missdetected, x^2:missdetected, x^3: new object from z_3^1} or { x^1:missdetected, x^2:connected to z_3^1}.

2.1.5.2 Undetected Objects

The Poisson components are henceforth assumed to be Gaussian mixtures (GM) such that the distribution of all undetected objects ($\mathbf{x}^u \in \mathbf{X}^u$) belong to a mixture of normal distributions. $\bar{\mathbf{x}}^u$ and P^u is the undetected objects state and variance respectively. On top of these two values, an undetected object is also parameterized with a weight $w^u_{j,i}$ defining how important this object is for the total distribution. One can write the total undetected object intensity with N_u undetected targets as:

$$\rho^u(\mathbf{x}^u) = \sum_{i=1}^{N_u} w^u \mathcal{N}(\mathbf{x}^u; \bar{\mathbf{x}}^u, P^u) \tag{2.47}$$

In order to model predicted newly born undetected object the new born object PPP-intensity for all global hypotheses are defined as follows:

$$\rho^b(x) = \sum_{i=1}^{N_u} w^b_i \mathcal{N}(\mathbf{x}; \bar{\mathbf{x}}^b_i, P^b_i) \tag{2.48}$$

It is also possible to model the Poisson intensity as partially uniform to decrease the computional cost. In [58] they show that a uniform birth model has similar accuracy as GM birth models with many Gaussian components, and better accuracy than GM birth models with few Gaussian components. As described in [58], if one would use a GM, the Gaussian components needs to be many and well placed in order to describe an (approximately) uniform distribution. This would model the birth as the weighted sum

$$\rho^b(\mathbf{x}^u) = \sum_{i=1}^{N_u} w^u \gamma_k(\theta, \phi) \tag{2.49}$$

where each componet is described as

$$\gamma_k(\theta, \phi) = w^b_k \mathcal{U}(\theta; \beta) \mathcal{N}(\phi; \bar{\phi}, \sigma^2_\phi) \tag{2.50}$$

where $\mathcal{U}(\theta, \beta)$ is the undetected uniform distribution, β is the uniform distribution volume. Here, θ represents the states in the state vector x that are measureable, and ϕ represents the unmeasurable states.

In the **prediction** step we can use the Bayesian filtering prediction step to get the posterior Poisson intensity:

$$\rho^u_{k+1|k}(\mathbf{x}^u) = \rho^b(\mathbf{x}^u) + p_s \sum_{i=1}^{N_u} w^u_i \mathcal{N}(\mathbf{x}^u; F\bar{\mathbf{x}}^u_i, FP^u_i F^T + Q) \tag{2.51}$$

where p_s is the survival probability defining how likely an object is to survive from one time step to another and F and Q are the motion model.

For the **update** step however, as per definition, the objects that do not have any measurement associated to them remain undetcted. The Bayesian update will thus not change the states or variances for the Poisson distribution since no more information is added. On the other hand, if the probability of an object being detected

is constant p_d the object ought to have been detected with that probability. In the undetected intensity, the weight of each undetected object, is thus decreased with a factor $(1 - p_d)$ as to account for the decreased probability of existing:

$$\rho^u_{k+1|k+1}(\mathbf{x}^u) = (1 - p_d)\rho_{k+1|k}(\mathbf{x}^u) \tag{2.52}$$

2.1.5.3 Detected Objects

The Multi-Bernoulli Mixture (`MBM`) components are also henceforth assumed to have Bernoullis with Gaussian state densities, such that the distribution of a detected object, i, in a certain global hypothesis j, can be written:

$$\mu_{j,i}(\mathbf{x}^d) = \mathcal{N}(\mathbf{x}^d; \bar{\mathbf{x}}^d_{j,i}, P^d_{j,i}) \tag{2.53}$$

where $\bar{\mathbf{x}}^u_{j,i}$ and $P^u_{j,i}$ is the detected objects state and variance respectively. On top of these two values a detected object holds a probability of existence $r^d_{j,i}$ and a weight $w^d_{j,i}$ that is related to the posterior probability of the global hypothesis j, see [11]. Note that the probability of existence does not affect the distribution, but is being important in a later stage. Furthermore, can the interested reader find the definition of the density function in [11].

Prediction for the linear and Gaussian case, using the basic Bayesian filtering prediction step, the predicted `MBM` distribution becomes:

$$\mu_{j,i}(\mathbf{x}^d) = \mathcal{N}(\mathbf{x}^d; F\bar{\mathbf{x}}^d_{j,i}, FP^d_{j,i}F^T + Q) \tag{2.54}$$

where N_d is the number of detected objects and F, and Q are the state transition matrix and noise covariance matrix respectively, defined by the motion model. The weight $w^d_{j,i}$ remains unchanged whereas the probability of existence, $r^d_{j,i}$, is scaled with p_s as follows

$$w^d_{j,i} = w^d_{j,i} \tag{2.55}$$

$$r^d_{j,i} = p_s r^d_{j,i} \tag{2.56}$$

In the **Update** step a non-empty set of measurements and the measurement model $p(\mathbf{z}_k|\mathbf{x}_k) = \mathcal{N}(\mathbf{z}; H\mathbf{x}, R)$ are given, and the predicted state is updated by weighing in the information contained in the measurements. There are two different types of updates to perform: the objects being detected for the first time as well as the detected objects from the previous time step. The update steps share the majority of the process, but they are here discussed individually for clarity.

Since an object must be connected to a measurement in order to be classified as **detected for the first time** all components in the Poisson distribution must be considered by performing gating in order to see which measurements are close enough to the PPP intensity components. Then, when all measurements have been gated, Moment Matching is performed with regards to all the components connected to each measurement in order to combine the different updated PPP intensity components

into one fused distribution. The probability of existence $r^d_{j,i}$ and object state density $\mu^d_{j,i}$ for an object detected for the first time are

$$r^d_{j,i} = \frac{e_j(\mathbf{z})}{\rho_{j,i}(\mathbf{z})} \tag{2.57}$$

$$\mu^d_{j,i}(\mathbf{x}_{j,i}) = \sum_{i=1}^{N_u} \widehat{w}_{j,i} \mathcal{N}(\mathbf{x}; \widehat{\mathbf{x}}_{j,i}(\mathbf{z}), \widehat{P}_{j,i}) \tag{2.58}$$

where $c(\mathbf{z})$ is the clutter intensity and

$$e_j(\mathbf{z}) = p_d \sum_{i=1}^{N_u} w^u_{j,i} \mathcal{N}(\mathbf{z}; \widehat{\mathbf{x}}_{j,i}(\mathbf{z}), \widehat{S}_{j,i}) \tag{2.59}$$

$$\rho_j(\mathbf{z}) = e_j(\mathbf{z}) + c(\mathbf{z}) \tag{2.60}$$

$$\widehat{w}_{j,i} \propto w^u_{j,i} \mathcal{N}(\mathbf{z}; \widehat{\mathbf{x}}_{j,i}(\mathbf{z}), \widehat{S}_{j,i}) \tag{2.61}$$

$$\widehat{\mathbf{x}}_{j,i}(\mathbf{z}) = \bar{\mathbf{x}}^u_{j,i} + \widehat{K}_{j,i} \widehat{S}^{-1}_{j,i} (\mathbf{z} - H\bar{\mathbf{x}}^u_{j,i}) \tag{2.62}$$

$$\widehat{P}_{j,i} = P^u_{j,i} - \widehat{K}_{j,i} \widehat{S}^{-1}_{j,i} \widehat{K}^T_{j,i} \tag{2.63}$$

$$\widehat{K}_{j,i} = P^u_{j,i} H^T \tag{2.64}$$

$$\widehat{S}_{j,i} = H P^u_{j,i} H^T + R \tag{2.65}$$

To reduce the computational cost of the tracking algorithm, the mixture density in Equation 2.58 is reduced to a single Gaussian using moment matching. The weight of the newly created components is set to $w^d_{j,i} = \rho_j(z)$ if the current global hypothesis includes the object. If the current global hypothesis does not include the object $w^d_{j,i} = 1$ and the existence probability is $r^d_{j,i} = 0$.

On the other hand, if an object was **detected in a previous time step** it adds the possibility of the object still being present but that the measurement is missing. Furthermore, all potentially detected targets and investigate every possible measurement connections. Here, too, gating can be used to reduce the number of possible connections. For the misdetection hypothesis the object component's existence probability and weight are changed according to

$$r^d_{j,i} = \frac{r^d_{j,i}(1 - p_d)}{1 - r_{j,i} + r_{j,i}(1 - p_d)} \tag{2.66}$$

$$w^d_{j,i} = w^d_{j,i} \left(1 - r_{j,i} + r_{j,i}(1 - p_d)\right) \tag{2.67}$$

where p_d is the probability of the sensor detecting the object. The states and variances remain unchanged for the misdetection hypothesis. For the detected object $x_{j,i}$ associated to a measurement z the standard Kalman filter update yields:

$$\mu_{j,i}(\mathbf{x}_{j,i}) = \mathcal{N}(\mathbf{x}_{j,i}; \hat{\mathbf{x}}_{j,i}(\mathbf{z}), \widehat{P}_{j,i}) \tag{2.68}$$

where

$$\hat{\mathbf{x}}_{j,i}(\mathbf{z}) = \bar{\mathbf{x}}^d_{j,i} + \widehat{K}_{j,i}\widehat{S}^{-1}_{j,i}(\mathbf{z} - H\bar{\mathbf{x}}^d_{j,i}) \tag{2.69}$$

$$\widehat{P}_{j,i} = P^d_{j,i} - \widehat{K}_{j,i}\widehat{S}^{-1}_{j,i}\widehat{K}^T_{j,i} \tag{2.70}$$

$$\widehat{K}_{j,i} = P^d_{j,i}H^T \tag{2.71}$$

$$\widehat{S}_{j,i} = HP^d_{j,i}H^T + R \tag{2.72}$$

The probability of existence is

$$r^d_{j,i} = 1 \tag{2.73}$$

and the updated weight is

$$w^d_{j,i} = w^d_{j,i} r_{j,i} p_d \mathcal{N}(\mathbf{z}; \bar{\mathbf{x}}^d_{j,i}, \widehat{S}_{j,i}) \tag{2.74}$$

Additional details and derivations can be found in [11].

2.1.5.4 Generating new hypotheses

After updating based on a new set of measurements as described in section 2.1.5.3, one still needs to solve the association problem, solving which measurement is connected to which component. This is done by creating cost matrices, one for each previous global hypothesis, where each row corresponds to a measurement and each column corresponds either a previous object or a possible new object. Solving the cost matrices using an optimal assignment algorithm solves the data association.

For the previous global hypothesis h with N^h many single target hypotheses, measurements $\{z_i, \dots z_{m_k}\} \in Z_k$, the cost matrix is

$$L^h = \begin{bmatrix} -l^{1,1,h} & -l^{1,2,h} & \dots & -l^{1,N^h,h} & -l^{1,0} & \infty & \dots & \infty \\ -l^{2,1,h} & -l^{2,2,h} & \dots & -l^{2,N^h,h} & \infty & -l^{2,0} & \dots & \infty \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ -l^{m_k,1,h} & -l^{m_k,2,h} & \dots & -l^{m_k,N^h,h} & \infty & \infty & \dots & -l^{m_k,0} \end{bmatrix} \tag{2.75}$$

The first N^h columns are for the previous components, where the last m_k columns are for the possible new detections. The elements for the previous components are the negative logarithm of the corresponding single target hypothesis-measurement pair weight divided by the components missdetection weight

$$l^{j,i,h} = \log(w^d_{j,i}) - \log(w^d_{0,i}) = \log\left(\frac{w^d_{j,i}}{w^d_{0,i}}\right) \tag{2.76}$$

For the possible new detections, the diagonal elements for corresponding measurement j is the corresponding weight, $w^d_{j,i}$, which as explained in section 2.1.5.3 is given by equation (2.60), and becomes

$$l^{j,0} = \log(w^d_{j,0}) = -\log(\rho_j(\mathbf{z})) \tag{2.77}$$

The Hungarian algorithm [59] is an optimization algorithm which solves the assignment problem. In the case of the data association problem and cost matrix, it will find the optimal assignment, giving pairs of measurements and targets (j, i) which minimizes the cost, thus maximizing the combination of logarithmic weights. An extension of the Hungarian algorithm is Murty's algorithm [60], which outputs the K best assignments instead of only the optimal. This algorithm is here used to create new global hypotheses.

2.1.5.5 Reduction

As by definition, the assignment problem in multiple object tracking will theoretically become NP-hard[61], it is of high importance to reduce the computational complexity. Luckily, there are several methods to decrease the computational complexity, namely pruning, capping, gating, recycling and merging.

Pruning can be done for both Poisson distributions, `MB` distributions, and global hypotheses by simply removing distributions or hypotheses with weights below a certain threshold. For Poisson, $w^u_i < \Gamma^u$. For `MB` distributions, remove the ones with $w^d_i < \Gamma^d$. For global hypotheses, remove the ones with $w^h < \Gamma^h$.

Capping refers to reducing the amount of global hypotheses and detected objects. If there are more than N^d_{max} detected objects, keep the ones with highest weights. Same, with the global hypotheses, only keep the N^h_{max} best global hypotheses.

Gating is the limitation of the possibility to associate an object, either undetected or detected, with a measurement. Ellipsoidal gating is a widely used technique using the Mahalanobis distance, equation(2.78), which measures how many standard deviations z_j is from $\hat{z}_i$. If $d^2_M \leq G$ than measurement z_j is relevant for object i.

$$d^2_M = [\hat{z}_i - z_j]^T S^{-1} [\hat{z}_i - z_j] \tag{2.78}$$

As the Mahalanobis distance uses the innovation covariance, it takes into account the uncertainties of the prediction and the measurement.
Euclidean distance (2.79) can also be used.

$$d^2_E = [\hat{z}_i - z_j]^T [\hat{z}_i - z_j] \tag{2.79}$$

Another neat reduction trick for `PMBM` is the idea of recycling. Recycling refers to moving the distribution of detected objects with probability of existence below a threshold, $r^i_{k|k} < \Gamma^r$, to the undetected distribution. This allows to reduce the complexity by not having to account for as many objects to connect associations with,

while at the same time not losing information as pruning would force. Mathematically, the undetected density is after recycling

$$\rho^u_{k|k}(\mathbf{x}^u) = \rho^u_{k|k}(\mathbf{x}^u) + \sum_{i:r^i_{k|k}<\Gamma^r} w^d_{j,i} r^i_{k|k} \mu_{j,i}(\mathbf{x}_{j,i}) \tag{2.80}$$

There is no need for non-unique global hypotheses, hence merging identical global hypotheses reduces the computational complexity without affecting performance. While merging identical global hypotheses, one should add together the individual global hypotheses weight's. One should also look to merge similar Poisson distributions, especially after recycling. For a track-oriented approach, it is also possible to merge the local hypotheses.

2.1.5.6 Estimation

The final step in the `PMBM` filter is to yield an estimate of the current objects and their states. We choose the global hypothesis with highest weight, i.e. the most probable one

$$j^* = \text{argmax}_j \prod_{i=1}^{N_d} w^d_{j,i} \tag{2.81}$$

where the total global hypothesis weight is the product of its detected object weights. After finding the *best* global hypothesis j^* the state values $x_{j^*,i}$ for all `MBM` components with a probability of existence greater than a certain threshold, $r_{j^*,i} > \gamma$ are considered the output of the algorithm for this time step. Note that as the probability of existence will decrease, with a rate depending on the survival probability p_s and detection probability p_d, if misdetected, the threshold γ will have a great impact on how many consecutive time steps an object will survive although not being connected to any measurement.

2.2 Implementation

The `PMBM` filter was implemented with object-oriented Python. All objects are tracked in the longitudinal and lateral 2D coordinates with position and direction with respect to the ego vehicle. As mentioned in section 2.1.5.1 the implementation is done with a track-oriented approach, meaning that each object is modelled as a track, containing both object state properties and single target association hypotheses history. Each track can have several measurement associations at each time instance, each responding to what we call a single target hypothesis. The state of the filter contains several global hypotheses, each global hypothesis holding, at one time instance, combinations of track IDs with respective single target hypothesis IDs, corresponding to different measurements.

The filter was implemented for three different motion models, the `CV`, `CA` and the Bicycle model, described in section 2.1.3.1.

There is a vast hypothesis space which grows exponentially for each time step. In order to make the algorithm computationally tractable we use a number of simplifications. We use gating for only creating *reasonable* connections between an object and a measurement and Murty's algorithm to only further investigate the most probable global hypotheses. On top of that we use weights from the log domain in order to avoid very small numbers, with the cost of having to reformulate some equations.

2.2.1 Remarks on the Bayesian Filtering

Two different filtering methods were applied to deal with the linear `CV/CA` models and the nonlinear Bicycle model. A basic Kalman prediction and update, as described in Equations (2.10) - (2.16), was used for the `CV/CA` models. Meanwhile, a `UKF` prediction and update, Equation (2.24) and (2.33), was used for the Bicycle model.

2.2.1.1 Models

The state vector looks different depending on which motion model is connected to the target. The state vector for the `CV` and `CA` modes are:

$$\mathbf{x}_{CV} = \begin{bmatrix} x & y & \dot{x} & \dot{y} \end{bmatrix}^T \tag{2.82}$$

$$\mathbf{x}_{CA} = \begin{bmatrix} x & y & \dot{x} & \dot{y} & \ddot{x} & \ddot{y} \end{bmatrix}^T \tag{2.83}$$

and for the Bicycle Model:

$$\mathbf{x}_{Bicycle} = \begin{bmatrix} x & y & \psi & v & \delta \end{bmatrix}^T \tag{2.84}$$

where the position coordinates x, y are relative to the ego vehicle, i.e. in the middle of the ego vehicle, x is defined rightward in the lateral direction and y forward in the longitudinal. The rest of the states, however, are relative the real world in order for the motion models to be realistic. For instance is the bicycle model fitted for replicating how an object moves in the real world rather than the movement relative another object. The coordinate system is thus translated and rotated for each time step in order to fit the ego vehicle's motion. Another perk of this is that we don't have to transform the measurements since the sensors always yields distances with respect to the ego vehicle. Using the notation: object states $\mathbf{x}$, imu data with lateral, longidtudinal and rotational velocities v_x, v_y, r_z, and sampling time Δt, we can get the transformed states by first Euler integrating the angle: $\alpha = -r_z \Delta t$ and then using rotation and translation matrices according to:
`CV :`

$$\begin{bmatrix} x^+ \\ y^+ \\ \dot{x}^+ \\ \dot{y}^+ \end{bmatrix} = \begin{bmatrix} \cos(\alpha) & -\sin(\alpha) & 0 & 0 \\ \sin(\alpha) & \cos(\alpha) & 0 & 0 \\ 0 & 0 & \cos(\alpha) & -\sin(\alpha) \\ 0 & 0 & \sin(\alpha) & \cos(\alpha) \end{bmatrix} \begin{bmatrix} x \\ y \\ \dot{x} \\ \dot{y} \end{bmatrix} - \begin{bmatrix} v_x \Delta t \\ v_y \Delta t \\ 0 \\ 0 \end{bmatrix} \tag{2.85}$$

CA:

$$\begin{bmatrix} x^+ \\ y^+ \\ \dot{x}^+ \\ \dot{y}^+ \\ \ddot{x}^+ \\ \ddot{y}^+ \end{bmatrix} = \begin{bmatrix} \cos(\alpha) & -\sin(\alpha) & 0 & 0 & 0 & 0 & 0 \\ \sin(\alpha) & \cos(\alpha) & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \cos(\alpha) & -\sin(\alpha) & 0 & 0 & 0 \\ 0 & 0 & \sin(\alpha) & \cos(\alpha) & 0 & 0 & 0 \\ & 0 & 0 & 0 & 0 & \cos(\alpha) & -\sin(\alpha) \\ & 0 & 0 & 0 & 0 & \sin(\alpha) & \cos(\alpha) \end{bmatrix} \begin{bmatrix} x \\ y \\ \dot{x} \\ \dot{y} \\ \ddot{x} \\ \ddot{y} \end{bmatrix} - \begin{bmatrix} v_x \Delta t \\ v_y \Delta t \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \tag{2.86}$$

Bicycle:

$$\begin{bmatrix} x^+ \\ y^+ \\ \psi^+ \\ v^+ \\ \delta^+ \end{bmatrix} = \begin{bmatrix} \cos(\alpha) & -\sin(\alpha) & 0 & 0 & 0 \\ \sin(\alpha) & \cos(\alpha) & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ \psi \\ v \\ \delta \end{bmatrix} - \begin{bmatrix} v_x \Delta t \\ v_y \Delta t \\ r_z \Delta t \\ 0 \\ 0 \end{bmatrix} \tag{2.87}$$

As we created our own data loader which loads the detections as the x and y center positions, and the rotation r_z around the object's upward axis we get a measurement:

$$z = \begin{bmatrix} x, y, r_z \end{bmatrix}^T \tag{2.88}$$

As to fit the states for each motion model we set the linear measurement model for `CV` :

$$H_{CV} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \tag{2.89}$$

with measurement noise:

$$R_{CV} = \sigma_{CV} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} \tag{2.90}$$

And for the `CA` model:

$$H_{CA} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{2.91}$$

with measurement noise:

$$R_{CA} = \sigma_{CA} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} \tag{2.92}$$

And for the bicycle model:

$$H_{BC} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{bmatrix} \tag{2.93}$$

with measurement noise:

$$R_{BC} = \sigma_{BC} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.94}$$

where we ended up using $\sigma_{CV} = \sigma_{CA} = \sigma_{BC} = 0.1$.

2.2.1.2 Uncented Kalman Filter

As the Bicycle Model is rather non-linear we need to do some sort of approximation. We use the Uncented Kalman Filter (`UKF`) with weights always yielding positive weight to the centre point:

$$\lambda = \alpha^2(L + \kappa) - L = \alpha^2(5 + \kappa) - 5 \tag{2.95}$$

$$w_0 = \frac{\lambda}{\lambda + L} \tag{2.96}$$

$$w_i = \frac{1}{2(\lambda + L)} \tag{2.97}$$

Setting the condition $\lambda > 0$ to ensure positive weights, one can derive a relation between α and κ. Knowing that for the bicycle model, the state dimension $L = 5$, the following relation is found.

$$\frac{5}{5 + \kappa} < \alpha^2 \tag{2.98}$$

Using $\alpha = 0.01$ gives $\kappa > 5 \cdot 10^4 - 5$.
Using the same α with a more generic κ, say a small integer gives a negative λ severely affecting the results. Even though it is theoretically sound as the weight sums up to one, it gives w_0 as a large negative number. The consequence of this becomes a prediction in the wrong direction, as the speed becomes negative due to the weights. Using the derived weights mentioned above gives expected results.

2.2.2 Computational stability

Several methods were used to increase the computational stability. All weights were transferred to the logarithmic domain, both for single target hypotheses and global hypotheses. The covariance matrix from the Kalman update needs to be semi-positive definite, and to ensure this, Equation (2.99) is performed after each time a covariance matrix is altered.

$$\mathbf{S} = \frac{1}{2}(\mathbf{S} + \mathbf{S}^T) \tag{2.99}$$

2.2.3 Hypotheses structures

The detected objects are modelled as Multi-Bernoulli Mixture distributions, where the distributions are normally distributed with the mean value being the current state and the variance the uncertainty of that state. Each possible new object is starting a new track with an unique Track ID (`TID`) and since an object's state and variance will be different depending on whether it in the future time steps is detected or not we store all these possibilities in different Single Target Hypotheses (`STH`). In each `STH` we store the information about each object's states and variances separately (as opposed to storing the entire probability density function in one variable). Each `STH` also holds a weight and probability of existence as discussed in

the theory section. However, here we will use logarithmic weights in order to gain numerical stability. Weights in the normal domain tend to approach zero and thus will machine errors impact too much. Using log-weights instead will prevent this, but equations need some alterations (as will be discussed see further down).

On top of these values, the time of birth of this `STH`, the index of the measurement being connected to the particular object in the current time step and the cost for Murty's Algorithm to pick this `STH` are stored. In order to keep track of all `STH` they are also given a unique Single target ID, `SID`. The `SID` is unique in the sense of that particular track. On top of holding all `STH`s, the Track contains information of when the track was born (time of birth), what class the current track is of (car, pedestrian, cyclist, etc). The latter in order to make it possible to use different parameters depending on the track's class (change of motion model for instance). This architecture enables us to methodically go through all `STH`s in each Track and perform necessary operations. It also makes it possible to use a look-up-table when it comes to global hypotheses as a single global hypothesis only needs to point at which `TID`s and `SID`s to include.

As to avoid accumulating memory usage we also store which `SID`s descends from which `SID` in order to remove old or unused `STH`s. Global hypotheses holds the `TID`s and `SID`s that it believes are the correct ones. Different global hypotheses can hold different amounts of `TID-SID`-pairs and can also be empty.

2.2.4 Partial Uniform distribution for undetected objects

We use a partial uniform distribution for the undetected objects, where the birth model is modelled as a uniform distribution. The area of the uniform distribution is formed as a circle sector, with radius = 100m and angles $\in [0.78, 2.35]$ radians, based on the KITTI [1] ground truth measurements, visualized in Figure 2.6. We will also accumulate Gaussian Mixture (`GM`) components from recycled Multi Bernoulli Mixture tracks. By instead of removing the detected objects below the pruning threshold, reformulate and store them as undetected components in the Poisson distribution we can keep information acquired when the object was detected. This enables the same track to disappear and re-appear while still being considered the same object and keeping the state information.
Comparable with Equation (2.47) the partial uniform distribution for undetected objects is

$$\rho^u_{k|k}(\mathbf{x}^u) = w_{\mathcal{U}}\mathcal{U}\Big(\theta; V_{\mathcal{U}}\Big)\mathcal{N}(\phi; \bar{\phi}, \sigma^2_\phi) + \sum_{i\in\mu} w^i \mathcal{N}(\mathbf{x}^\mu; \bar{\mathbf{x}}^u_i, P^u_i) \tag{2.100}$$

Where μ refers to the recycled components. Furthermore, here θ and ϕ are, as described in Section 2.1.5.2 in relation to Equation (2.50), the measurable and unmeasurable states respectively. For the `CV` model the measurable states are $[x, y]$ and the unmeasurable $[\dot{x}, \dot{y}]$. Respectively for the Bicycle model $[x, y, \psi]$ was the measurable states with $[v, \delta]$ being the unmeasurable ones. The area over the uniform range is declared $V_{\mathcal{U}}$. With a partial uniform distribution for the Poisson

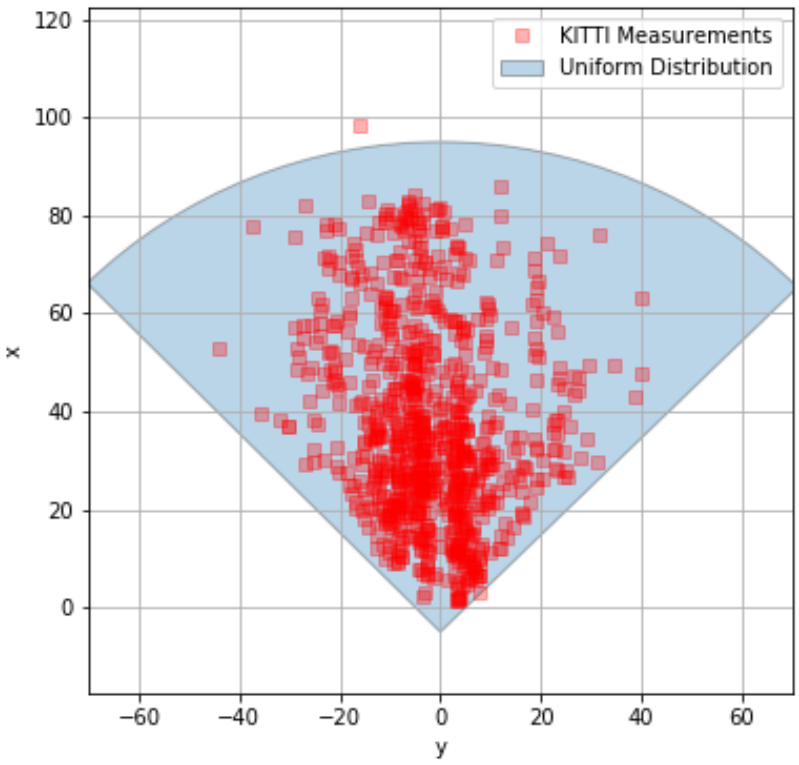


Figure 2.6: The uniform distribution area together with the KITTI [1] ground truth positions of all objects over all training sequences. The circle sector has radius = 100m and ranges between angles 0.78 and 2.35 radians.

distribution, the birth process described in relation to only GM components in Section 2.1.5.3, will differ. Equation (2.59) is rewritten to equation (2.101), which also becomes the weight for the newly created object.

$$e(\mathbf{z}) = p_d\left(\frac{w_{\mathcal{U}}}{V_U} + \sum_{i=1}^{N_\mu} w_i^\mu \mathcal{N}(\mathbf{z}; \hat{\mathbf{x}}_i^\mu(\mathbf{z}), \hat{S}_i)\right) \tag{2.101}$$

The posterior of detected objects for the first time therefore becomes:

$$\mu_{k+1|k+1}^d(\mathbf{x}^\mu) = \frac{w_{\mathcal{U}}}{V_U}\mathcal{N}(\theta; \mathbf{z}, R)\mathcal{N}(\phi; \bar{\phi}, \sigma_\phi^2) + \sum_{i=1}^{N_\mu}\mathcal{N}(\mathbf{x}; \hat{\mathbf{x}}_i(\mathbf{z}), \hat{P}_i) \tag{2.102}$$

Here are the mean unmeasurable states, $\bar{\phi}$ be set to $\mathbf{0}$, and the covariance σ_ϕ^2 becomes a tuning parameter. The `GM` Poisson components $\hat{\mathbf{x}}_i$ and $\hat{P}_i$ are updated according to equations (2.62) - (2.65). R is the measurement noise of the measurement model.

2.2.5 Prediction

In the two prediction steps, Poisson and Multi-Bernoulli Mixtures, are independent of each other. By looping through all tracks' single target hypotheses (`STH`) we first perform coordinate transform of the previous state using the ego vehicle's movement data. Euler integrations of the lateral, latitudal, and angular velocities from the `IMU` are used to get the translation and rotation matrices. For the `CV` model the position states x, y needs to be rotated and translated whereas the velocities $\dot{x}, \dot{y}$ only needs to be rotated. For the Bicycle model the position states x, y needs to be rotated and translated whereas the heading ψ needs to be added to the angular change and the rest are left unchanged. When the previous states have been transformed into the current coordinate system we can apply the respective Bayesian filtering prediction step. Note that we for each predicted variance ensure it is symmetrical by applying $P = 0.5(P + P^T)$ for numerical stability.

When it comes to the undetected objects the uniform part will not change, whereas the targets that have been recycled will. These Gaussian distributions are propagated through the coordinate transform and the motion model in the same manner as the detected objects.
The algorithm for the prediction step can be seen as pseduo code in 2.2:

Algorithm 2.2 Pseudo code for detected object's prediction in `PMBM` algorithm

Notations: x-state, P-variance, w-weight, r-existence probability, p_s-survival probability

for *Track* $\in$ *All Tracks* **do**
  for *`STH`* $\in$ *All `STH`s in Track* **do**
    - Perform coordinate transform
    - Perform Bayesian Prediction:
    $x^+, P^+ = \text{motionmodel}(x, P)$
    $w^+ = w$
    $r^+ = p_s r$
    - Ensure variance is symmetric:
    $P^+ = 0.5(P^+ + P^{+T})$
  end
end

2.2.6 Update

Here too we loop through all tracks' `STH`s. For each `STH` we need to add one misdetection hypothesis, and as many hypothesis as there are measurements within the gate. Gating, as described in Section 2.1.5.5 is performed with the described Mahalanobis distance. The distance threshold G is set to 3, which corresponds to the measurement having to lie within three standard deviations of the object to be deemed relevant. We also need to add hypotheses where possible new targets have emerged. A brief overview of this update step is seen in Algorithm 2.3 and further details and definitions of variables are discussed in the paragraphs below.

Algorithm 2.3 Pseudo code for the update step in `PMBM` algorithm

Notations: x-state, P-variance, w-weight, r-existence probability, c-cost, p_d-detection probability

for *Track* $\in$ *All Tracks* **do**
  for *`STH`* $\in$ *All `STH`s in Track* **do**
    - Create **Misdetection hypothesis** with:
    $x^+ = x$
    $P^+ = P$
    $r^+ = r(1 - p_d)/(1 - r + r(1 - p_d))$
    $w^+ = w + \log(1 - r + r(1 - p_d))$
    $c_{missed} = w^+$
    - Perform gating for all measurements w.r.t current `STH`
    - Create **Detection hypotheses**
    for $z \in$ *gated measurements* **do**
      - Create detection hypothesis with:
      $x^+ =$ Equation 2.69
      $P^+ =$ Equation 2.70
      $r^+ = 1$
      $w^+ = w + \log(r \cdot p_d \cdot l_z)$
      $c = w^+ - c_{missed}$
    end
  end
end
- Create **Possible new targets hypotheses**
- Update **Undetected Gaussian distributions**

2.2.6.1 Misdetection hypothesis

We create a new `STH` with `SID`: sid^+ having unchanged states and variances compared to the parent `STH` with `SID`: sid. The new existence is defined in Equation 2.66 and the weight update equation in Equation 2.67. However, the weight equation is altered to follow the logarithmic rules and instead implemented as:

$$w_{tid,sid^+} = w_{tid,sid} + \log(1 - r_{tid,sid} + r_{tid,sid}(1 - p_d)) \tag{2.103}$$

Furthermore is sid^+ added to the list of children in sid and the cost of this `STH` is simply the weight of hypothesis: $c_{missed} = w_{tid,sid^+}$

2.2.6.2 Detection hypotheses

We go through all measurements and evaluate which passes the gating. We then create one `STH` for each of these gated measurement. The state x and the variance P are both calculated with the bayesian filter update steps given the gated measurement. While doing this we also note the measurement likelihood $l_z = \mathcal{N}(z; \bar{x}^d_{tid,sid}, \widehat{S}_{tid,sid})$ as it is used for calculating the weight according to Equation 2.74. The measurement likelihood can be thought of as how likely it was that we got that exact measurement and is thus just drawn from the multivariate normal probability density function. Here, too, we must rewrite the weight equation into the log domain

$$w_{tid,sid^+} = w_{tid,sid} + \log(r_{tid,sid} \cdot p_d \cdot l_z) \tag{2.104}$$

The cost for this `STH` is normalized with the cost of the misdetection weight, namely

$$c = w_{tid,sid^+} - c_{missed} \tag{2.105}$$

The probability is naturally set to one, $r_{tid,sid^+} = 1$ since we here by definition hypothesize that the object exists.

2.2.6.3 Possible new targets hypotheses

Possible new targets are found by first using gating to see if any of the measurements fit with any of the Gaussian distributions, and then adding the effect of the uniform distribution. When creating a new possible track with a new target id we calculate the cost for picking the newly created track as

$$w^+ = \log(e + c(z)) \tag{2.106}$$

and the probability of existence as

$$r^+ = e/(e + c(z)) \tag{2.107}$$

where

$$e = p_d \cdot \mathcal{U}_{weight}/\mathcal{U}_{area} \tag{2.108}$$

and $c(z)$ is the clutter intensity. Later on Murty's algorithm will pick this new possible track if it's more likely that the measurement belong to a new track than any of the old tracks.

2.2.6.4 Undetected Gaussian distributions

For the Gaussian distributions within the undetected objects the weight is decayed in the log domain according to

$$w^+ = w + \log(1 - p_d) \tag{2.109}$$

whereas the states, variances, and existence probabilities remain unchanged.

2.2.7 Generating global hypotheses

For each global hypothesis a cost matrix is created, as described in section 2.1.5.4, with the costs $c_{tid,sid}$ calculated in the update step. In the case of no previous global hypothesis (e.g., first iteration of the algorithm), the step of creating a cost matrix is skipped, and a new global hypothesis is formed with each new possible target associated with the measurement it was created from.

As depicted in algorithm 2.1, for each global hypothesis, we aim at generating K new global hypotheses, where $K = \lceil N^{new} \cdot \exp(w_h) \rceil$. The hyperparameter N^{new} decides how many global hypotheses we desire having in total and w_h is the weight of the current global hypothesis. This allows better global hypotheses to spawn more new global hypotheses and worse to spawn fewer. K is clamped to be within the interval $[1, 10]$ to ensure that we get at least one new global hypothesis while also avoiding redundancy, i.e. all new global hypotheses originates from the same. N^{new} was tuned to 20, but if there are $K^- < K$ possible new global hypotheses we will only create K^- new ones.

The weight for the new global hypothesis w_h^+ is calculated in the log-domain as

$$w_h^+ = w_h - C_h \tag{2.110}$$

where C_h is the total cost for that particular hypothesis according to Murty's algorithm. We used a Python binding of a C++ implementation of Murty's algorithm [62] in order to very efficiently generate K new global hypotheses from the cost matrix.
After generating new global hypotheses we normalize each global hypothesis weight with the log-sum of all weights:

$$w_h := w_h - w_h^{smallest} + \log(1 + \sum(\exp(w_h^{rest} - w_h^{smallest}))) \tag{2.111}$$

where $w_h^{smallest}$ is the smallest weight among all current global hypotheses. Pseudo code for generating new global hypotheses can be seen in Algorithm 2.4.

Algorithm 2.4 Pseudo code for generating new global hypotheses in `PMBM` algorithm

```
for Global hypothesis ∈ All global hypotheses do
    - Create cost matrix for current GH using the information stored in each STH
    for k ∈ K do
        - Pick and remove Murty's algorithm solution with lowest cost
        - Create new global hypothesis with:
        w_h = solution's cost
    end
end
- Normalize all Global Hypotheses
```

2.2.8 Estimation and predicted trajectory

In order to find the best global hypothesis we loop through all current global hypotheses and note which one has the highest weight. We then go through all objects in this particular global hypothesis and collect the `STH` if the probability of existence is greater than a threshold, $r_{tid,sid} > \Gamma^r = 0.01$. In order to get a predicted trajectory of these tracks we simply use the motion model recursively as many times as we want to predict forward in time. If not specified otherwise we predict 10 time steps ahead. In order to get the tracks tail we connect the tracks with same track id over time.

2.2.9 Reduction

As to reduce the number of hypothesis a number of reduction techniques were used.

2.2.9.1 Recycling Tracks

Instead of recycling the tracks as described in Section 2.1.5.5, where an object's existence probability is compared against a threshold, we used a weighted sum over all global hypotheses and single target hypotheses (`STH`). If a `STH` is in the global hypothesis, the single target hypothesis' probability of existence multiplied with the global hypothesis' weight is added to the weighted sum

$$\hat{r}_i = \sum_{j}^{N^h} \sum_{k}^{N_i^d} r_k^i \cdot w_j^h \tag{2.112}$$

This way a track occurring in many global hypotheses with many single target hypotheses will be favoured. This makes sense because it is the entire track being recycled, rather than a single `STH`.
If the track's weighted sum is below threshold $\Gamma^{\hat{r}}$, the track is recycled. Upon recycling, each single target hypothesis Gaussian components of that track is degraded in to a Poisson component.

2.2.9.2 Hypothesis Reduction

Several reduction methods, extending the ones mentioned in Section 2.1.5.5, were used to decrease the number of hypotheses, both global and single target hypotheses, in order to decrease the complexity of the algorithm.

The **Global Hypotheses** were decreased in three ways, pruning, capping and merging. The three methods are implemented as described in Section 2.1.5.5. With parameters according to:

- Max number of global hypotheses, $N_{max}^h = 25$
- Global hypothesis weight threshold: Γ^h = -6

For **Single Target Hypotheses**, as mentioned in Section 2.1.5.5, the Multi-Bernoulli distributions with a weight below a certain threshold should be pruned. This was not implemented as it's redundant when using the track-oriented approach. With a

track-oriented approach, all new single target hypothesis are generated before new global hypotheses are generated. When generating the new global hypotheses, single target hypotheses with low weight will not be selected by Murty's algorithm per design. The single target hypotheses which have not been included in any global hypothesis are automatically discarded.

A pro-active reduction technique, specific for a track-oriented approach, was implemented. When generating new global hypotheses, a single target hypothesis is only added if the single target hypothesis' probability of existence is over a threshold Γ^r

Single target hypotheses in each track that were older than the current time were removed as to not accumulate data over time. In other words, the single target hypothesis association history including old states were removed.

2.2.10 Tuning

A lot of parameters needed to be set. Some could be reasoned to, e.g. the uniform distributed area as seen in Figure 2.6, which were set based on the simulated measurements from KITTI data. Pruning parameters could also be set with an iterative process, running to see the amount of hypotheses, adjusting parameters and so forth until a reasonable, i.e. computationally tractable while still being accurate, amount of hypotheses were achieved.

To help the search of good values for the parameters for the process noise and initial covariance for new objects, a genetic algorithm [63] was used with a combination of `GOSPA`, `MOTP` and `MOTA`, metrics described in Section 4.1, as evaluation function. The parameters from the genetic algorithm was manually fine-tuned afterwards.

3

ETENN - Artificial Neural Network

This Chapter gives a brief overview of the most important mathematical operations used or mentioned when discussing the Artificial Neural Network in this thesis. Generally, Artificial Neural Network (ANN) is a way of trying to replicate the behaviour of biological brains. Even though a human's brain is superior to today's ANNs, there are fields where an ANN is competitive with its biological counterpart, due to its ability to theoretically learn an arbitrarily hard mathematical mapping between input and output. One example of a field ANNs are thriving is in Computer Vision. For the interested reader we recommend the Deep Learning book [64] for more information regarding Neural Networks.

This Chapter will also discuss how the neural network used in this thesis is designed. The backbone network structure of our End-To-End-Neural Network (ETENN) is heavily inspired by Fast and Furious [8] and IntentNet [9]. Five consecutive LiDAR point-clouds are fed into a network performing detection and prediction of the nearby objects. These outputs are then used to reason about the dynamic states of each object.

3.1 Theory

An ANN consists of interconnected groups of nodes. Each group is called a layer and each node is called a neuron. The connection between neurons, replicating synapses in biological brains, transfers information between layers. A very simple ANN can be seen in Figure 3.1 where two inputs are going through a hidden layer in order to produce an output.

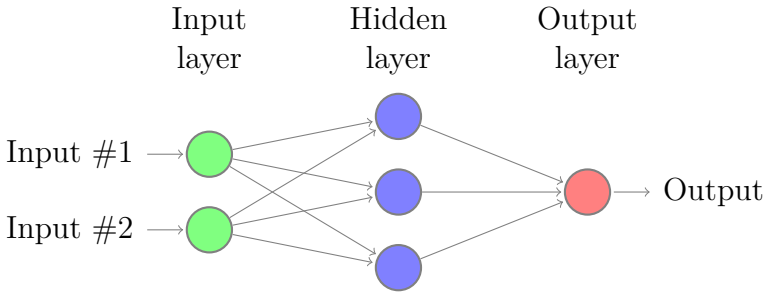


Figure 3.1: Example of an Artificial Neural Network with one input, one hidden and one output layer. The information from the input is combined to produce a single output.

3.1.1 Basic Components and Layers

To replicate the functionality of the human brain, the ANN components are modelled mathematically. The most important models will be discussed in this section. Also some basic layers, i.e. a set of basic components, will also be discussed.

3.1.1.1 Neurons

The artificial neurons in neural networks is a mathematical model of a biological neuron, modelled to replicate the function of being excitable depending on input signals. The equation of a neuron, visualized as one of the blue circles in Figure 3.1, is simply a weighted sum of all the inputs plus a bias term, i.e.

$$s(\mathbf{x}) = \sum_{i=0}^{n} w_i x_i + b_i \tag{3.1}$$

where n is the number of inputs and w_i the weights in the neuron and b_i the bias term. The combination of weights and input signals, giving the neuron its value together with the bias term, represents the excitability.

3.1.1.2 Convolutional layer

A convolutional filter (CONV) slides, or convolves, a window over the inputs' spatial dimension, performing dot product operations, in order to create an output. It uses the original elements' values as inputs to a weighted sum, where the weights are decided by the convolutional filter. An interpretation of these filters is that they are

trying to find certain features and outputs a high value if it's a match, and a low if it's not. It is these filters (and in particular the features of each filter) one refers to when training a network. A toy example of a horizontal line finding features is seen in Figure 3.2. Important notations when talking about convolutional layers are stride and padding, where stride is the step size when sliding the window. Padding is the term of adding values outside the borer of the original input to be able to start the filter in the out-most regions of the input. Note that depending on filter size, stride and padding the output can be either bigger, same or smaller size compared to the input.

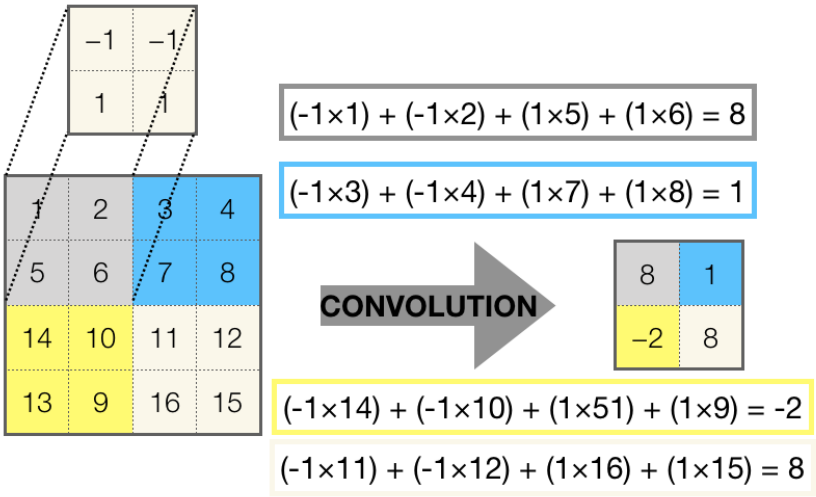


Figure 3.2: Toy example of convolution with an example filter of size 2×2 with stride 2. A linear combination of the input and the filter is output.

3.1.1.3 Fully connected layer

A Fully Connected layer (`FC`) is different to the `CONV` layer talked about above. Here we connect all the input neurons from one layer, to all the neurons in the next layer. One example of this is seen in Figure 3.1, where the input and hidden layers are fully connected.

3.1.1.4 Activation functions

In order to further imitate a brain, and making it possible to learn more complex (possibly non-linear) behaviour, the neuron is only to trigger a signal to the next layer when the weighted sum $s(\mathbf{x})$ is within a certain region. This is called an activation function, and there exists several different ones.
The **Logistic Sigmoid** is defined as:

$$y(s) = \frac{1}{1 + e^{cs}} \tag{3.2}$$

where c is a constant most commonly set to -1. Another activation function is the

Hyperbolic tangent, i.e.

$$y(s) = 1 - \frac{e^{-2s}}{1 + e^{-2s}} \tag{3.3}$$

where the output is in the range $[-1, 1]$. Furthermore, the most used activation function, is the **Rectified Linear units** (`ReLu`), which is defined as

$$y(s) = \max(0, s) \tag{3.4}$$

3.1.1.5 Pooling

A pooling function narrows down statistics from nearby elements in order to down-sample big matrices. This is very useful when the input is an image, as it not only reduces the number of computations needed but also makes features invariant to changes in scale and/or, rotation. For instance maxpooling, which is the most common type of pooling, simply takes a fixed region and stores the maximum value within that region. In Figure 3.3 a toy example can be seen where the region is a square of 2×2, i.e., the filter is of size 2×2, and we only apply the pooling once on each subsquare within the larger matrix, i.e., the stride in this case is 2. It is seen that the size of the matrix, in this case, after applying maxpooling has been reduced from containing 16 elements down to 4.

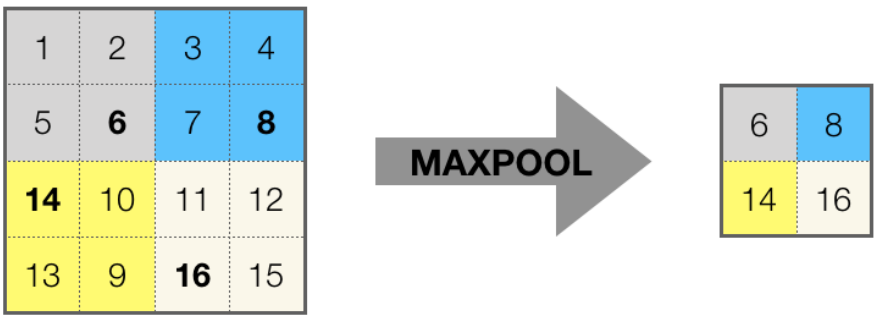


Figure 3.3: Toy example of maxpooling with a filter being 2×2 with stride 2. The maximum value within the filter kernel is passed as output.

3.1.1.6 Batch Normalization

Batch Normalization is a method which helps to speed up training a network by reducing the internal covariate shift. Internal covariate shift is defined as the change of distributions of the network activations due to the change of the networks parameters [65]. By subtracting the batch mean and dividing by the batch standard deviation, batch normalization enables more stability in the training, enabling higher learning rates and thus faster training. Batch normalization also helps as a regularizer, as it forces the network to not only learn particular patterns, but rather the overall tendency.

3.1.1.7 Deconvolution (up-sampling) layer

As exemplified in Figure 3.2 convolution layers tend to down-sample the input. Stacking several convolutions in a row will thus eventually end up in a very dense matrix. As to increase the size of the matrix again one may want to do the opposite of convolution, namely deconvolution. In order to up-sample the matrix the filter is slid over a (zero-)padded input, as seen in Figure 3.4. The filter can be either hand-crafted or learned. Another name of this layer is Transposed Convolution.

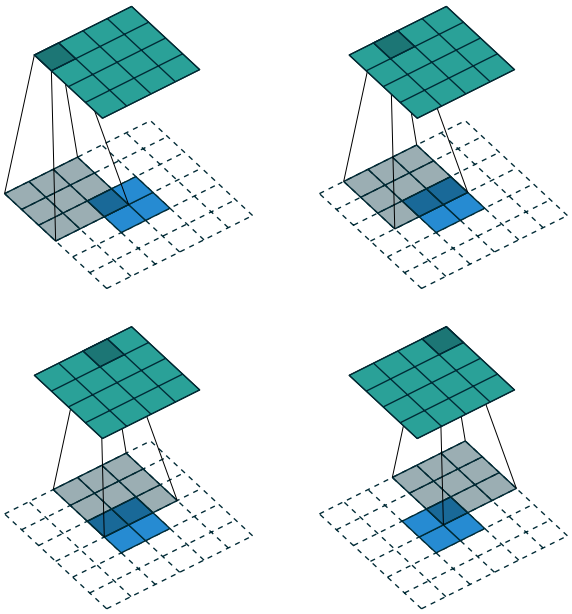


Figure 3.4: Deconvolution example from [4]. The information in the blue square is together with the applied filter (gray) outputting the information in the green layer.

3.1.1.8 Recurrent layer

Recurrent layers are used to store information through a sequence, e.g. time. The recurrent layer has two sources of inputs: new and previous information. A simple example can be seen in Figure 3.5 where the blue neuron in the middle takes previous output(s) as inputs.

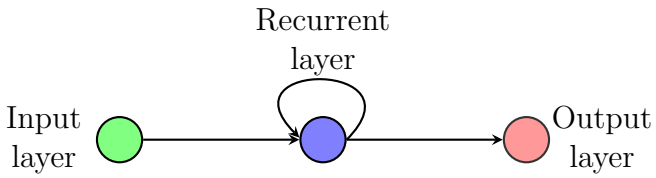


Figure 3.5: The output of the layer in one iteration is saved to, in combination with the new input, affect the next output.

3.1.1.9 Residual Building Block (ResBlock)

As He et. al show in [5] deeper neural networks are hard to train. In the same paper they introduce Deep Residual Learning building blocks to ease the training while also increasing the accuracy. A building block (`ResBlock`) consists of a few stacked layers where the residual mapping is learned and a shortcut connection in order to keep previous information. The block is mathematically defined as:

$$\mathbf{y} = \mathcal{F}(\mathbf{x}, W_i) + \mathbf{x} \tag{3.5}$$

and exemplified in Figure 3.6.

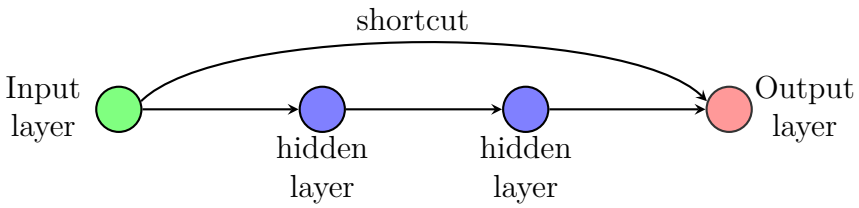


Figure 3.6: Residual Learning Building Block introduced by He et. al in [5]. In this figure two hidden layers are shown and the output of the last is simply element-wise added to the input.

3.1.2 Basic Operations

Training an `ANN` basically means altering the parameters, i.e., weights and bias terms of the neurons in the network, in order to optimize the network. During training we first need to evaluate the current state of the network, i.e., examining the output given an input. This is done by propagating the input, executing the sequential mathematical operations, through all layers. By comparing the outcome of this forward propagation with the true values we can calculate how to alter the weights in order to make the output fit the true states better. How the difference between output and true values is calculated is decided by the loss function and the actual alternation of weights is done in the backward propagation. These three operations are further reasoned about below.

3.1.2.1 Forward Propagation

A full forward propagation pass includes computing all layer's neurons values as Equation (3.1), sequentially, with optional alterations using activation and pooling functions. The output of the network is received after propagating through the output layer, which decides the dimensions of the output.

A trivial example is shown in figure 3.7, displaying a one-layer network, with only one neuron. It sums the weighted inputs, adds a bias and lastly propagates through an activation function to become the output.

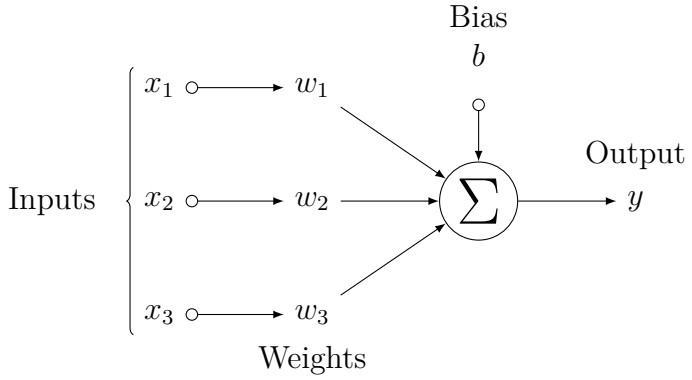


Figure 3.7: Forward propagation through one layer. Three inputs are weighted and summed. A bias is also added before entering an activation function yielding the output.

3.1.2.2 Loss Function

To measure the inconsistency of a neural network, one needs to compare the predicted output from the network with the actual target value. This is done using so called loss functions, calculating the error between the prediction and the ground truth. There are several different loss functions and via their differences they give different characteristics to the training/optimization of the networks. Some of the common ones used for regression are Mean absolute error (`MAE`)

$$\texttt{MAE}(y, x) = \frac{1}{n}\sum_{i=1}^{n}|y_i - x_i| \tag{3.6}$$

Mean squared error (`MSE`)

$$\texttt{MSE}(y, x) = \frac{1}{n}\sum_{i=1}^{n}(y_i - x_i)^2 \tag{3.7}$$

and the Huber Loss

$$\texttt{SmoothL1}(y, x) = \frac{1}{n}\sum_{i=1}^{n} z_i \tag{3.8}$$

$$z_i = \begin{cases} \frac{1}{2}(x_i - y_i)^2, & \text{if } |x_i - y_i| < \delta \\ |x_i - y_i| - 0.5, & \text{otherwise} \end{cases} \tag{3.9}$$

During the training, the model adjusts to minimize the loss, which depending on the loss function, it will adjust accordingly. For example, as the `MSE` squares the error, single outliers giving a large error will give a high loss value, hence the model will adjust to minimize the outlier error, at the expanse of other samples. On the other hand `MAE` will not be as affected by outliers. The Huber loss, also known as the `SmoothL1`-loss, combines the robustness of L1 loss and the preciseness of L2 loss. By conditioning on the magnitude of the error, it uses a loss similar to L1 loss if the error magnitude is large and a loss similar to L2 if the error magnitude is below a set value.

For classification, other loss functions are used, where one of the most common is the Cross Entropy (`CE`) loss, equation (3.10). The cross entropy loss uses the predicted probability $\hat{y}$ for each class in a multiple class classifier and compares it to the, one-hot encoded, ground truth y by computing the cross entropy between the two probability distributions for the class vector.

$$\texttt{CE}(y, \hat{y}) = -\sum_i y_i \log \hat{y}_i \tag{3.10}$$

When the case regards the binary classification of two classes, such as object versus background, the `CE` loss can be simplified to a Binary Cross Entropy Loss (`BCE`), Equation (3.11).

$$\begin{aligned} \texttt{BCE}(y, \hat{y}) = -\sum_i y_i \log \hat{y}_i = -\sum_{i=0,1} -y_0 \log \hat{y}_0 - y_1 \log \hat{y}_1 \\ = -y_0 \log \hat{y}_0 - (1 - y_0) \log(1 - \hat{y}_0) \end{aligned} \tag{3.11}$$

3.1.2.3 Back Propagation

To alter the weights of the network in order to learn, a method called back propagation is used. Back propagation is, as the named suggests, a method which works its way backward in the neural network. It starts by calculating the loss functions gradient with respect to each parameter in the network. As it's possible to use the chain rule, it starts from the end of the network and sequentially works its way to the first layer, using the chain rule to save computational dependency. An example of how to calculate the gradients in a single-neuron network is displayed in figure 3.8. After computing all gradients, Back propagation uses the gradient descent approach to alter the weights. A general gradient descent step is showed in Equation (3.12), where γ is the learning rate, a hyper parameter which affects the length of each step.

$$a_{n+1} = a_n - \gamma \nabla F(a_n) \tag{3.12}$$

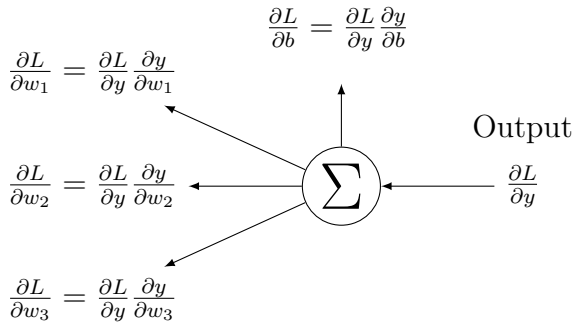


Figure 3.8: Back propagation through one layer. Three inputs are weighted and summed. A bias is also added before entering an activation function yielding the output.

3.2 Implementation

We implemented the network using PyTorch and trained it on KITTI data [1]. The basic idea for the network is to fuse K consecutive time steps and output detection for the current time step and predictions on where the objects will be in the future $K-1$ time steps. Two different approaches are evaluated: 1) using a hand crafted Bird's Eye View representation of the point cloud, and 2) letting the network learn the encoding of a discretized point cloud.

3.2.1 Data Representation

The data needs to be fed to the network in a compact way such that as much information as possible is saved, without requiring too large tensors. This goes for both images and point clouds, but is of more importance for point clouds as they generally are larger and thus more computational power can be saved. Not only because of the vast amount of points, but also because of the sparsity between them. In this thesis Bird's Eye View (`BEV`) and Voxel Feature Encoding (`VFE`) representations were implemented and this section aims to clarify these. In order to keep the complexity down and fit the used dataset the field of interest was set to span from the ego-vehicle and 70m forward (longitudinal direction), from 40m on the left hand side to 40m on the right hand side (lateral direction), and from the ground and 3.7m up in the air. All points outside this region were disregarded.

3.2.1.1 Bird's Eye View Representation

The `BEV` branch is encoded as a 2D map from above where the point cloud is discretized into cells with resolution 0.2×0.2m, values chosen to be the same as in [8]. Each cell in the 2D map contains of $(M+2)$ channels as the height (z-direction) is sliced into M slices. The slices are called height maps and are stacked on top of each other. The values for each slice is binary, 0 if there are no points in the slice and 1 if there are points. Naturally, if M is large this is a fairly similar approach as a traditional Voxel representation, such as [**?**]. A larger M means that more information is saved, but it also means a high computational cost.

3.2.1.2 Voxel Feature Encoding

The main idea behind a `VFE` network is to learn the `VFE` network how to represent the information in the point cloud in a optimal way. It uses the information of the points in a voxel and transform to represent each voxel to a feature vector. This training is done simultaneously as the training of the backbone network.

Our `VFE` is inspired by PointPillars [6], which divides the raw pointcloud into tall voxels, hence the name pillars, and propagates each pillar through a Feature Encoding Network, creating a pseudo image, as visualized in Figure 3.9. We use square pillars with the horizontal side length of 0.2m and a height of 3.7 meters. Each pillar contains a maximum of $N = 35$ points. If there are more in the point cloud, points are randomly sampled, and if there are fewer it zero-pads up to 35.

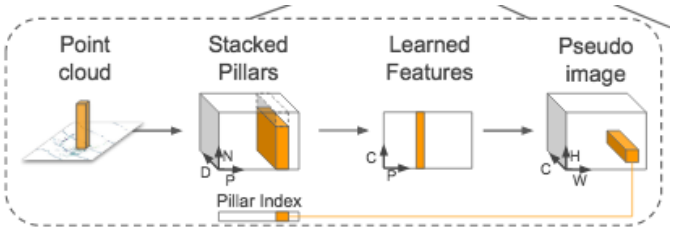


Figure 3.9: PointPillars Feature Encoding Network overview. Figure is kindly borrowed from [6].

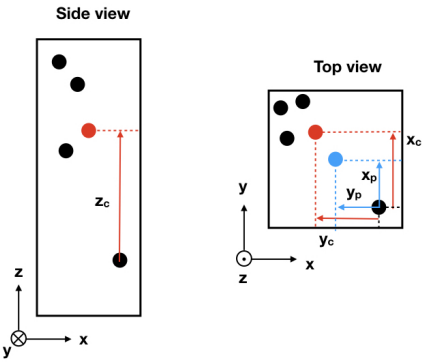


Figure 3.10: Visualization of additional information added in the point encoding used in the `VFE`. The red point is the arithmetic mean of all non-zero points in the pillar and the blue point is the pillar's center.

Each point in the pillar, $\mathbf{p}_i = [x, y, z]$, is encoded to include more information before propagating through the `VFE`. On top of x, y, z, each point is encoded to include both the distance to the arithmetic mean of all non-zero points in the pillar, x_c, y_c, z_c and the distance in x and y to the pillar's center, x_p, y_p. These distances are visualized in Figure 3.10, and all points after encoding are of dimension $D = 8$:

$$\mathbf{p}_i^{encoded} = \begin{bmatrix} x & y & z & x_c & y_c & z_c & x_p & y_p \end{bmatrix}^T \tag{3.13}$$

As mentioned in [6], not only a limit on number of points in each pillar is set, but also a number P of non-empty pillars. This limits the dense tensor vector to be of size (D, P, N) which is propagated through the Feature Encoding Network (`FEN`). The `FEN` is built the same way as described in [6], which in turn is inspired by the building blocks in VoxelNet[7]. These building blocks, called `VFE`-Layer, is displayed in Figure 3.11. Each point goes through a linear layer, a batch norm and a `ReLu`, generating a (C, P, N) tensor. This is followed by a max operation, resulting in a (C, P) tensor for all non-empty pillars. Each pillar's feature vector of size C is then scattered back to their position from where they were taken from, giving a pseudo image of size (C, H, W), where H and W is the pixel dimensionality in height/width for the pseudo image. We chose $C = 64$ and H and W is given from:

$$H = \frac{x_{\max} - x_{\min}}{x_{\text{gridsize}}} \tag{3.14}$$

$$W = \frac{y_{\max} - y_{\min}}{y_{\text{gridsize}}} \tag{3.15}$$

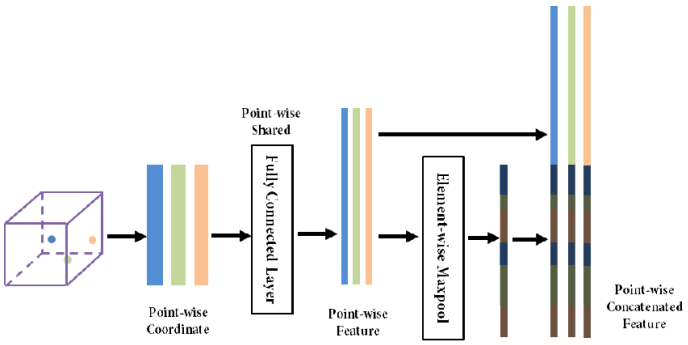


Figure 3.11: Voxel Feature Encoding Layer introduced by Zhou and Tuzel in VoxelNet [7].

3.2.2 Data Fusion

The input, may it be combining images with point clouds or stacking temporal information, is either fused before, after, or during propagation through a convolutional neural network. These methods are called Early, Late, and Deep Fusion, respectively. As can be seen in Figure 3.12 and Figure 3.13 we use Early Fusion to follow IntentNet's [9] approach stacking the `BEV` maps from each time step in the z-dimension. Not only does this decrease the number of parameters to learn compared to Late or Deep Fusion, but we can also use traditional 2D convolutions. For the `VFE`-approach one could argue that a Late or Deep Fusion is used as the point cloud is propagated through `VFE`-layers before being concatenated and fed through the rest of the network.

3.2.3 Prior boxes

Liu et al. introduced prior boxes in [22] and it is used in plenty of well known networks [21, 24, 25, 23, 66]. The goal of this method is to pre-define a set of bounding boxes and letting the network choose and tweak these. After analyzing the data we got at hand we could see that most ground truth bounding boxes can be reached using the three priors seen in Table 3.1. In order to keep the number of parameters down, we decided to not include the z-dimension and thus is only the length and width of the objects are of relevance.

Prior	Length	Width
1	3.4	1.5
2	3.9	1.6
3	4.4	1.8

Table 3.1: The prior boxes the network can be tweaked in order to fit the object as well as possible.

3.2.4 Network Architecture

The network structure without the input processing can be seen in Figure 3.12. The backbone network structure is heavily inspired by Fast and Furious [8] and IntentNet [9]. We use 3 consecutive Residual Blocks with 2 layers (`ResBlock`-2) followed with a single 3-layered Residual Block (`ResBlock`-3). On top of that we have two heads, namely the Detection Head and the Regression Head. The Detection Head is responsible for the classification and the Regression Head for the bounding boxes. The outputs from the heads are two feature maps where each element contains information of a certain region from the input point cloud. The backbone is the same for the PointPillar inspired version, but there is an additional `VFE`-layer for each input before entering the first `ResBlock`-2.

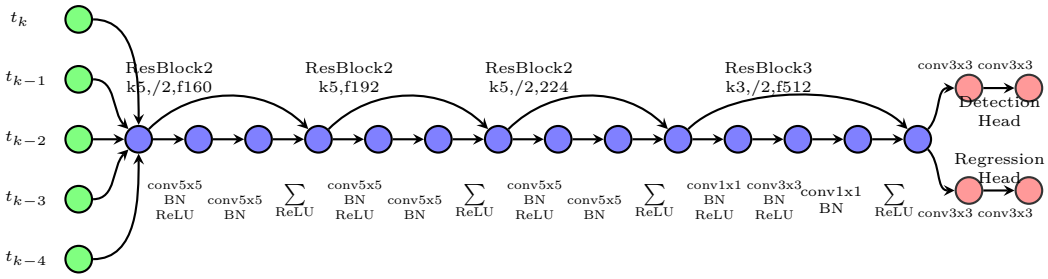


Figure 3.12: The `ETENN` network structure. Three consecutive 2-layered `ResBlock` followed by a single 3-layered `ResBlock` and the two network heads.

A smaller network (`ETEnn`) was also implemented, seen in Figure 3.13, in order to combat overfitting, as larger networks has the potential to learn higher complexity, such the noise in the training data. It only uses two `ResBlock`-2 before a `ResBlock`-3, using fewer channels in the `ResBlock`-3.
The two outputs of the network are reshaped such that for each predicted time step a cell on a certain row and column represents an area in the real world. Each and every one of these cells contain a feature vector for each prior box. The feature vectors are differently sized for the two output heads, containing only the information needed to perform the task at hand. The shapes of the tensors passing through the network is seen in Figure 3.14, ending in the detection and regression tensor sizes illustrated in Figure 3.15. This section proceeds by going into more detail of these two heads.

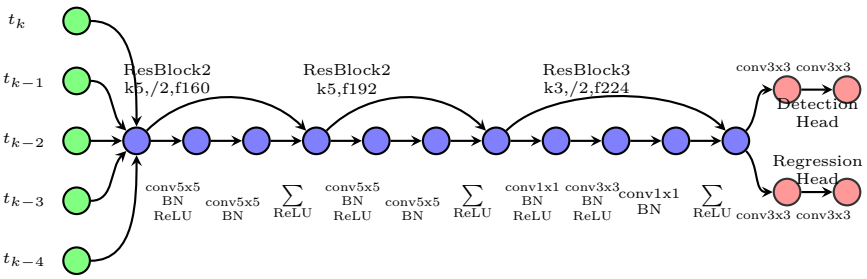


Figure 3.13: The `ETEnn` network structure. Two consecutive 2-layered `ResBlock` followed by a single 3-layered `ResBlock` and the two network heads.

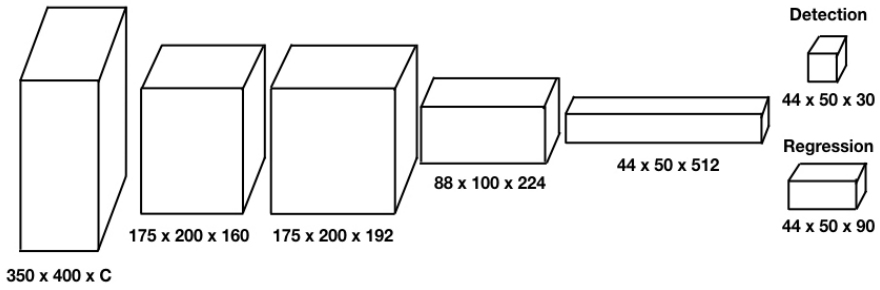


Figure 3.14: Illustration showing the sizes of the tensors passing through the `ETENN` network. The downsamplings are defined in Figure 3.12 and the last dimension of the incoming tensor vector, C, depends on the choice of input method, being larger for PointPillar encoding compared to `BEV` encoding.

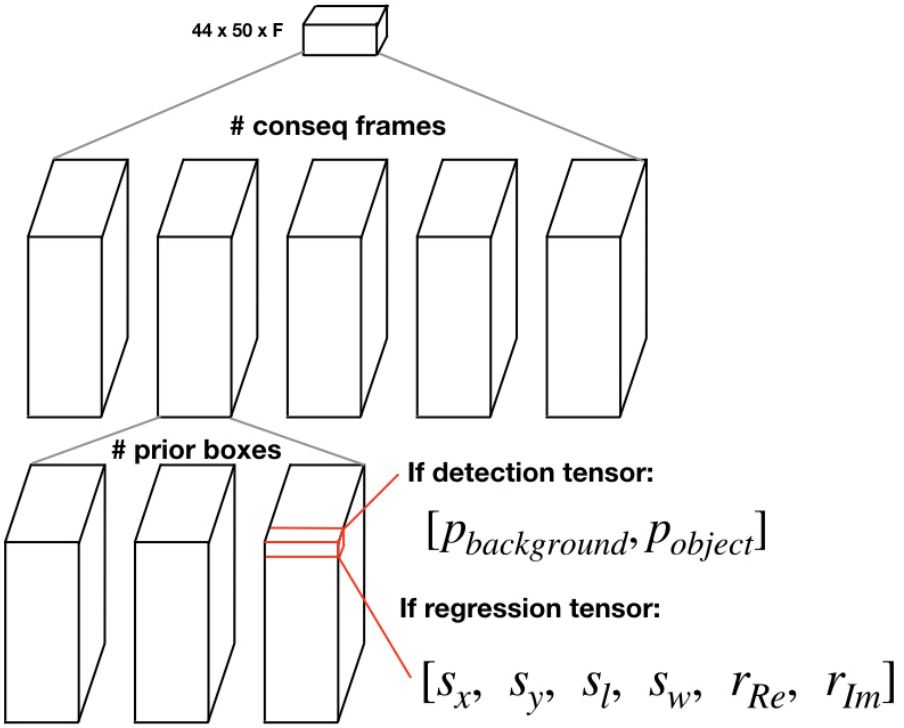


Figure 3.15: Illustration of the detection and regression tensors coming out of the networks respective head. The incoming tensor's last dimension F is a generalization to cover the different sizes of detection versus regression.

3.2.4.1 Regression Head

The Regression Head is trained for reshaping the prior boxes in size and rotation. The feature vectors of the Regression Head contain 6 values: $\begin{bmatrix} s_x & s_y & s_l & s_w & r_{Re} & r_{Im} \end{bmatrix}$ where each element shifts the prior box positions (s_x and s_y), sizes in length and width (s_l and s_w), as well as clockwise rotation around the objects with the imaginary and real parts of the angle (r_{Re}, r_{Im}). In order to transform these scaling factors into real world coordinates, sizes, and rotations we apply the following transforms for each predicted prior box:

$$\hat{p}_x = (i_{row} + \sigma(s_x)) \frac{H_{fov}}{H_{featuremap}} \tag{3.16}$$

$$\hat{p}_y = (i_{column} + \sigma(s_y)) \frac{W_{fov}}{W_{featuremap}}$$

$$\hat{p}_l = \exp(\sigma(s_l))a_l$$

$$\hat{p}_w = \exp(\sigma(s_w))a_w$$

$$\hat{p}_{Re} = r_{Re}$$

$$\hat{p}_{Im} = r_{Im}$$

Where H_{fov} is the field of view length in the longitudinal direction, W_{fov} in the lateral direction. $H_{featuremap}$ and $W_{featuremap}$ are the sizes of the region tensor. i_{row} and i_{column} are the indices of the feature map location.
The angle can be extracted using

$$\varphi = \text{atan}_2(\hat{p}_{Re}, \hat{p}_{Im}) \tag{3.17}$$

3.2.4.2 Detection Head

For the Detection Head the feature vector simply contains the class probabilities $\begin{bmatrix} p_{background} & p_{object} \end{bmatrix}$ for the prior boxes in all positions for all time steps. If one want to have the probabilities for different types of classes then one can only extend this feature vector dimension. A softmax layer is added on each feature vector in order to normalize the element values such that they sum to 1 and thus become a probability.

3.2.5 Matching

To prevent all the prior boxes into fitting all the ground truths we use a matching strategy. This will hinder the network from forcing a poorly sized prior box, for representing that particular object, to match a certain ground truth. As introduced in MultiBox [67] and used in, among others, [22, 6, 8, 9] we match each prior box and ground truth with the highest intersection of union (`IoU`), as long as the overlap is greater than 0.5. If there is a ground truth without any match it will be assigned to the prior box with highest overlap greater than zero. This strategy basically decouples the problem of sizing and rotating the bounding boxes from the classification task.

3.2.6 Loss

We split the loss function into three separate parts: target regression, angle regression and classification. They are responsible for tweaking the size of the prior box, rotating the heading and deciding which elements contain an object respectively. The output from the Regression Head is used in the Target and Angle Regressions, whereas the output of the Detection Head is input to the Classification Loss. We then take a weighted sum of these separate losses, resulting in the total loss:

$$\mathcal{L} = \beta_{target}\mathcal{L}^{target} + \beta_{angle}\mathcal{L}^{angle} + \beta_{cls}\mathcal{L}^{cls} \tag{3.18}$$

The weights of the loss were $\beta_{target} = 0.1$, $\beta_{angle} = 0.1$, and $\beta_{cls} = 1$. For the Target Regression Loss $\mathcal{L}^{target}$, we use a `SmoothL1`-loss for the matched prior boxes and the ground truth. The values used in the loss are regression targets described in 3.2.4.1. The mean `SmoothL1`-loss over all matches ($i \in [1, N]$) are added for all time steps ($k \in [0, K-1]$)

$$\mathcal{L}^{target} = \sum_{k=0}^{K-1} \lambda^k \cdot \frac{1}{N} \sum_{i}^{N} \texttt{SmoothL1}(\hat{\mathbf{p}}_i, \mathbf{p}_j^{gt}) \tag{3.19}$$

where $\hat{\mathbf{p}}_i = \begin{bmatrix}\hat{p}_x & \hat{p}_y & \hat{p}_l & \hat{p}_w\end{bmatrix}$ is the regression prediction matched with the ground truth $\mathbf{p}_j^{gt} = \begin{bmatrix}p_x^{gt} & p_y^{gt} & p_l^{gt} & p_w^{gt}\end{bmatrix}$. In order to give less weight to the predictions further away from the current time step, we added an exponential forgetting factor $\lambda = 0.95$.

For the Angle Regression Loss, we follow Complex Yolo's [32] lead to use a real and imaginary fraction to estimate the heading of the objects. This not only avoids singularities and helps generalization but also makes it possible to directly feed the output from the Regression Head into the loss function. On the other hand we need to transform the ground truth angle to this complex format, i.e extracting the cosine and sine values $\varphi_j^{gt} = \begin{bmatrix}\cos(\varphi^{gt}) & \sin(\varphi^{gt})\end{bmatrix}$. Having the two complex entities in place we can calculate the angular regression loss for each time step ($k \in [0, K-1]$) over all matches ($i \in [1, N]$) with

$$\mathcal{L}^{angle} = \sum_{k=0}^{K-1} \lambda^k \cdot \frac{1}{N} \sum_{i}^{N} \texttt{SmoothL1}(\hat{\varphi}_i, \varphi_j^{gt}) \tag{3.20}$$

where $\hat{\varphi}_i = \begin{bmatrix}\hat{p}_{Re} & \hat{p}_{Im}\end{bmatrix}$ and $\lambda = 0.95$ here as well.

For the classification loss we use a focal loss [23] to battle the imbalance of amount of negative versus positive examples in the data. As we have limited our work to classify the category *Car*, a Binary Cross Entropy loss is used to compute the loss-term:

$$\mathcal{L}^{cls} = \sum_{k=0}^{K-1} \lambda^k \cdot \frac{1}{N} \sum_{i}^{N} \texttt{BCE}(\hat{\gamma}_i, \gamma_j^{gt}) \tag{3.21}$$

where $\hat{\gamma}_i$ is the predicted class probability, and γ_j^{gt} the one-hot encoded ground truth.

3.2.7 Decoding Tracklets

As the output from the network is detections in the current time plus predictions in the future frames, these do not correspond to an output from an ordinary tracker. For example, it does not give any indication if an object in one time step is the same as in the next. To counter this, we've implemented an algorithm decoding the network's output to link objects over time. The implementation is inspired by the explanation of a similar algorithm in [8].

The output of the network is at each time $t_{k+i}\ \forall\ i \in [0, n-1]$ a random finite set of measurements Z_{k+i}. This data, together with the available predictions from the past $n-1$ time steps, is reasoned as shown in Figure 3.16. From the current time step and future $n-1$ time steps forward, the measurements which align over time, black rectangles in Figure 3.16, are reasoned. With reasoned we mean that for each time step $t_{k+i}\ \forall\ i \in [0, n-1]$ a new set Y_{k+i} is formed, being the union of the sets of measurements for this time step from either the current output or the previous predictions. If measurements in Y_{k+1} are too close they are merged. Continuing, the next step is to match reasoned detections in Y_k with reasoned predictions from Y_{k+i}, to link objects detected with their corresponding predictions. Predictions that do not match with detections from Y_k, are excluded. This yields $\hat{X}$, a set of detections with 0 to $n-1$ corresponding predictions. Note that the predictions omitted due to not being connected to a detection are still saved in the data state, being considered in the next iteration.

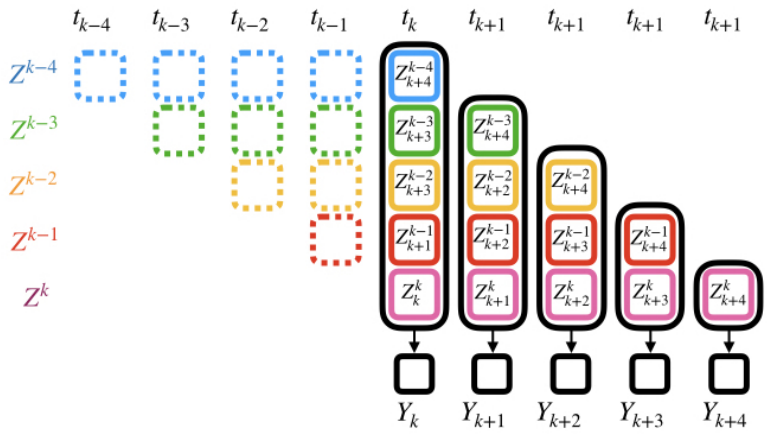


Figure 3.16: Illustration of the logic when reasoning over time for the decoding tracklets algorithm. At the current time t_k and for $n-1$ time steps in the future, the algorithm jointly reasons between the predictions from the current time step output and the past time steps output, which in this figure is represented by the black rectangles. For measurements Z the superscript refers to the measurement's time of origin and the subscript the time of prediction of the measurement.

The last step is to match the old object state X with $\hat{X}$. If objects in $\hat{X}$ is matched with an old object from X, it takes over that object's ID. New unmatched objects are given new IDs. The pseudo code of this algorithm is given by Algorithm 3.1.

Algorithm 3.1 Pseudo code for decoding tracklets algorithm

Inputs:
Detections for current time, Z_k^k, and $n-1$ predictions forward, $Z_{k+1}^k \dots Z_{k+4}^k$
Old $n-1$ predictions, $Z_{k+1\dots k+4}^{k-1}$, $Z_{k+2\dots k+4}^{k-2}$, $Z_{k+3,k+4}^{k-3}$, Z_{k+4}^{k-4}
Object state X

Output:
Updated object state X'

Match predictions over time:
for $i \in [0, n-1]$ **do**
  Match old predictions and current detection/prediction for time t_{k+i}
  yielding reasoned state Y_{k+i}
end
Link reasoned detections with predictions to give $\hat{X}$:
for $i \in [1, n-1]$ **do**
  Match reasoned detections Y_k with reasoned predictions Y_{k+i}
end
Match existing objects X with new objects in $\hat{X}$ giving updated object state X':
for *`object`* $\in X$ **do**
  Match `object` with new objects from $\hat{X}$
end

4
Experiments

4.1 Evaluation Measures

Experiments were done to test the performance for both the PMBM and the ETENN. In order to quantify the experiments to achieve comparable results, some measure of performance is needed. In this thesis the Generalized Optimal Sub-Pattern Assignment (GOSPA) [44], and the CLEAR-MOT metric suite [68] are used. Despite its name the CLEAR-MOT has not been proven to be a true metric, unlike the GOSPA metric, but is rather a measure of performance. This thesis, however, will not further discuss the theoretical differences between a metric and a non-metric. Instead the use of *metric* can be seen as synonymous to *performance measure*.

4.1.1 GOSPA

GOSPA is a metric which penalizes both the localization error and the cardinality error in a mathematically sound way. The metric is defined as

$$d_p^{(c,a)}(X,Y) \triangleq \min_{\gamma\in\Gamma}\left[\Big(\sum_{(j,i)\in\gamma} d(x_i,y_j)^p + \frac{c^p}{a}(|X|+|Y|-2|\gamma|)\Big)\right]^{1/p} \tag{4.1}$$

Here X is the set of ground truth objects and Y the estimates. $d(x_i, y_j)$ is the distance measure between an assigned pair of ground truth i and estimate j from the assignment set γ. The metric is minimized over all possible assignments sets Γ. The parameters c and a determines the cardinality mismatch error while p penalizes the localization error.

4.1.2 CLEAR-MOT

CLEAR-MOT is a suite of several performance measures, with Multi-Object Tracking Accuracy (MOTA) and Multi-object Tracking Precision (MOTP) as the more prevalent ones. MOTP is the total error over all frames between estimates and their assigned truth

$$\text{MOTP} = \frac{\sum_{i,t} d_t^i}{\sum_t c_t} \tag{4.2}$$

Here d_t^i is the distance between assignment i's estimate and ground truth for frame t. MOTP is averaged by c_t, which is the total number of assignments.
MOTA takes false negatives fn_t, false positives fp_t and mismatches mme_t into account.

It is averaged by the total number of ground truth, g_t

$$\text{MOTA} = 1 - \frac{\sum_t \text{fn}_t + \text{fp}_t + \text{mme}_t}{\sum_t g_t} \tag{4.3}$$

We also use two other metrics: Mostly Tracked (`MT`) and ID switches (`IDsw`) from [69]. Mostly Tracked is the percentage of ground truth trajectories which are covered by the tracker output for 80% of its length. ID switches is the total amount of times a trajectory gets a new track ID even though the ground truth identity remains the same.

4.2 Dataset

We use KITTI tracking dataset [1] totaling 8009 data time steps from 20 different scenarios. The sampling rate is $\Delta t = 0.1$s and includes images, point clouds, IMU data, and annotations. In Table 4.1 the number of frames per sequence can be seen.

Sequence	#Frames	#Objects	#Cars	#Pedestrians
0	155	15	12	2
1	447	98	92	3
2	233	20	16	1
3	144	9	9	0
4	314	39	29	5
5	297	36	34	0
6	270	15	13	0
7	800	63	57	2
8	390	27	25	0
9	803	89	87	1
10	294	28	16	2
11	373	60	55	5
12	78	4	2	1
13	340	68	3	42
14	106	17	15	2
15	376	26	9	11
16	209	28	4	19
17	145	11	0	9
18	339	21	21	0
19	1059	106	10	62
20	837	133	125	0
Total	8 009	913	634	167

Table 4.1: Number of frames and unique dynamic objects (cars, pedestrians, cyclists, vans, etc) in each sequence in the Kitti Tracking Dataset. For clarity the portion of cars and pedestrians are also displayed separately.

4.3 PMBM

`PMBM` was tested using four different motion models, `CV`, Bicycle, `CA` and `Mixed`. `Mixed` uses `CV` for pedestrians, and Bicycle Model for cars and bicyclists. As to avoid bias from our detection network, and to show that `PMBM` can be used to track any object class, we simulate measurements by applying noise and clutter to KITTI ground truth data. The applied noise is chosen as to resemble a state-of-the-art-detector, e.g. PointPillars [6]. We independently and identically sampled the noise from a normal distribution, $\mathcal{N}(x; \mathbf{0}, \sigma^2)$, where σ^2 was an identity matrix with $\sigma_i^2 = 0.1$ on the diagonal elements for either $i = \{x, y\}$ or $i = \{x, y, \psi\}$ depending on the measurable states. Clutter and miss-detections were added with probabilities $p_{clutter} = 0.02$ and $p_{miss} = 0.05$, where a clutter was uniformly sampled in the field of view. Note that the probability of clutter was drawn at every ground truth, i.e. more ground truths generated more clutter and no ground truth did not generate any clutter. Also note that for a certain sequence all filter configurations are run with the same measurements as to make a fair comparison.

To evaluate the different motion models' performance on different objects a test was performed where the `CV` model and Bicycle model were single-handedly used on Pedestrian and Cars respectively.

A test with noisier data was also performed to test the robustness of the algorithm, where $p_{miss} = 0.1$ and noise $\sigma_i^2 = 0.2$. No changes in the tuning of the process/measurement noise matrices were done in order to improve the results.

A test with fewer allowed global hypotheses was performed in order to see how the performance was affected. The original configuration uses a max of 100 global hypothesis, where the test uses 5.

4.4 ETENN

Four different networks were trained and tested, overlapping on two degrees of freedom in the overlying structure.

1. `BEV` input with `ETENN` network structure: `bev_NN`
2. Pointpillars input with `ETENN` network structure: `pp_NN`
3. `BEV` input with the smaller `ETEnn` network structure: `bev_nn`
4. Pointpillars input with the smaller `ETEnn` network structure: `pp_nn`

The different networks were trained using one NVIDIA Tesla V100 Tensor Core GPU with 32GB memory with different batch sizes for the different network and input structures, as the memory usage was different.

When training, we split the KITTI training dataset into a training and a validation set with the intention of using the separate test set for evaluating the actual performance. At first the data was split in 70% training and 30% validation data, but since the networks didn't generalized we also tried training on all sequences but

one. Different sequences for validation was tested but with negligible difference.

We used the Adam Optimizer with learning rate $1 \cdot 10^{-4}$ and weight decay $1 \cdot 10^{-4}$ for the final training. Test on both the validation set and the training set were performed, computing performance metrics over complete sequences. Test on the training data were done to show the potential of the network if more data was present.

4.5 PMBM with ETENN detections

In order to compare the difference between the two approaches, tests were done where we fed the detections in the current time step from the `ETENN` networks directly to the `PMBM` filter.

4.6 Hardware setup

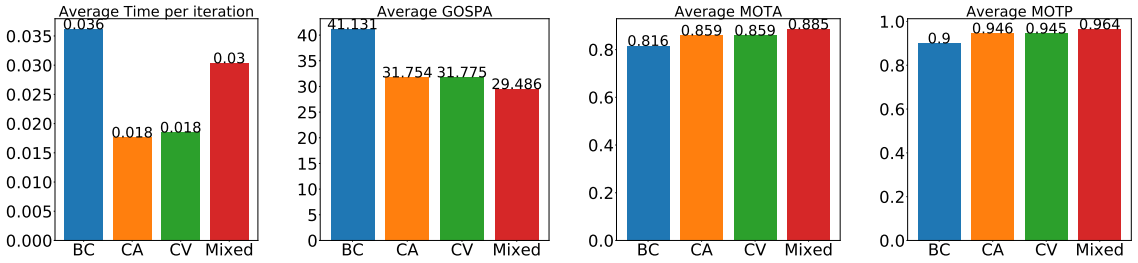
Both `PMBM` and `ETENN` tests were performed on a computer with an Intel Core i7-6820HQ CPU @ 2.70GHz 4-core processor together with graphics card: NVIDIA GM107GLM [Quadro M1000M] GPU Clock 993 MHz, Memory Clock, 1253 MHz, Memory size 2GB. The `PMBM` algorithm runs on CPU while the `ETENN` takes help from the GPU. `ETENN` inference is performed on the GPU, then the output is moved to the CPU for post-inference tracklets decoding. On this setup the `PMBM` algorithm takes on average 0.02s per iteration and `ETENN` on average 0.5s per iteration. However, evaluating the `ETENN` algorithm on the training computer with a NVIDIA Tesla V100 Tensor Core GPU we get the inference times displayed in the results below, where on average one iteration barely takes 0.1 seconds. This is the same period time as the Kitti data is annotated, and the `PMBM` algorithm is tested on.

5
Results

5.1 PMBM

A qualitative visualization from the tracker when running Sequence 12 can be seen in Figure 5.1. Each track gets assigned a color and a unique track ID. The full drawn lines are the past positions, and the triangles the predicted states. The ellipses are the 3σ-variances, meaning the object is with 99% certainty within that area. The ground truths are visualized through faded and filled red rectangles where the ground truth widths and heights are used for visualization purposes only. The ego vehicle is the blue square with center coordinates close to the origin.

Figure 5.2 and Table 5.1 displays the average metrics over all sequences for the `PMBM` with measurements generated as explained in Section 4.3. The sequence-wise values leading to these average metrics can be found in Appendix A.1 and Figure 5.3. It can also be seen that the `Mixed` model performs best, but takes considerably longer time than `CV` and `CA`. The Bicycle model has inferior results over all metrics, being the slowest as well.

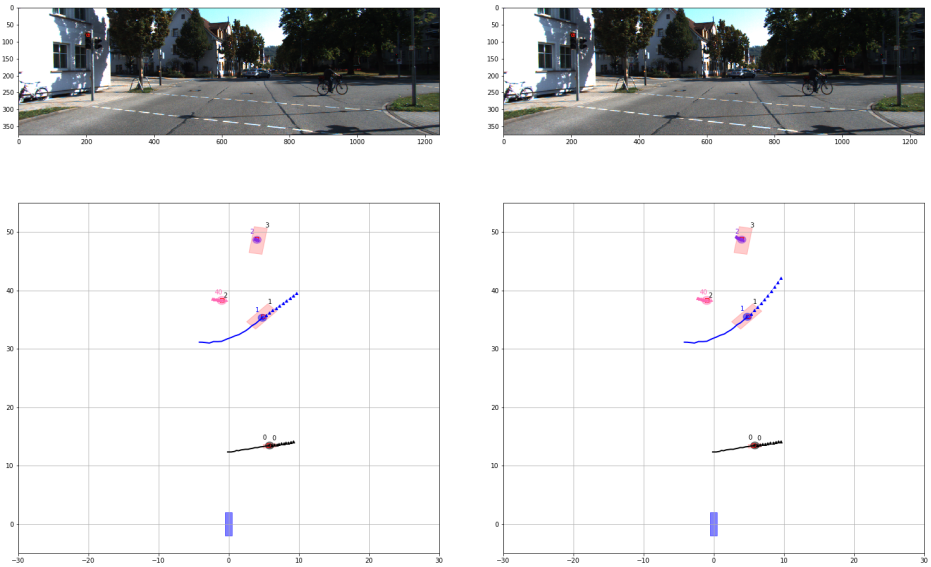


(a) Lower better **(b)** Lower better **(c)** Higher better **(d)** Higher better

Figure 5.2: Average metrics over all sequences for the `PMBM` filter with different Motion Models.

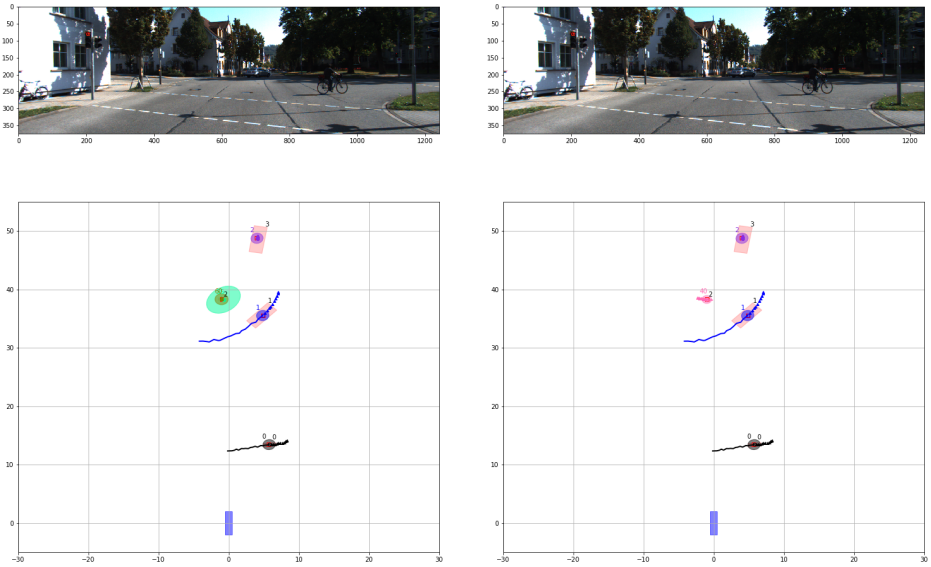
Model	AvgTime	Avg`MOTA`	Avg`MOTP`	Avg`GOSPA`	Avg`MT`	Avg`IDsw`
`BC`	0.0361	0.8157	0.9000	41.1310	43.476	114.333
`CA`	0.0176	0.8590	0.9457	31.7543	43.476	17.762
`CV`	0.0185	0.8590	0.9452	31.7748	43.476	17.810
`Mixed`	0.0303	0.8853	0.9643	29.4857	43.476	16.333

Table 5.1: Average metrics for sequences 0-20.



(a) Constant Velocity

(b) Constant Acceleration



(c) Bicycle Model

(d) Mixed

Figure 5.1: Image and Bird's Eye View of the world in the same frame (18) of sequence (12) for four different models. The trajectory of a track is displayed as a full drawn line, the current state as a circle, the predictions for 10 time steps ahead as triangles. The current measurements are the tiny red edge boxes (transparent within) whereas the ground truths are seen as slightly faded red rectangles, where the ground truth width and height is used for visualization purposes only.

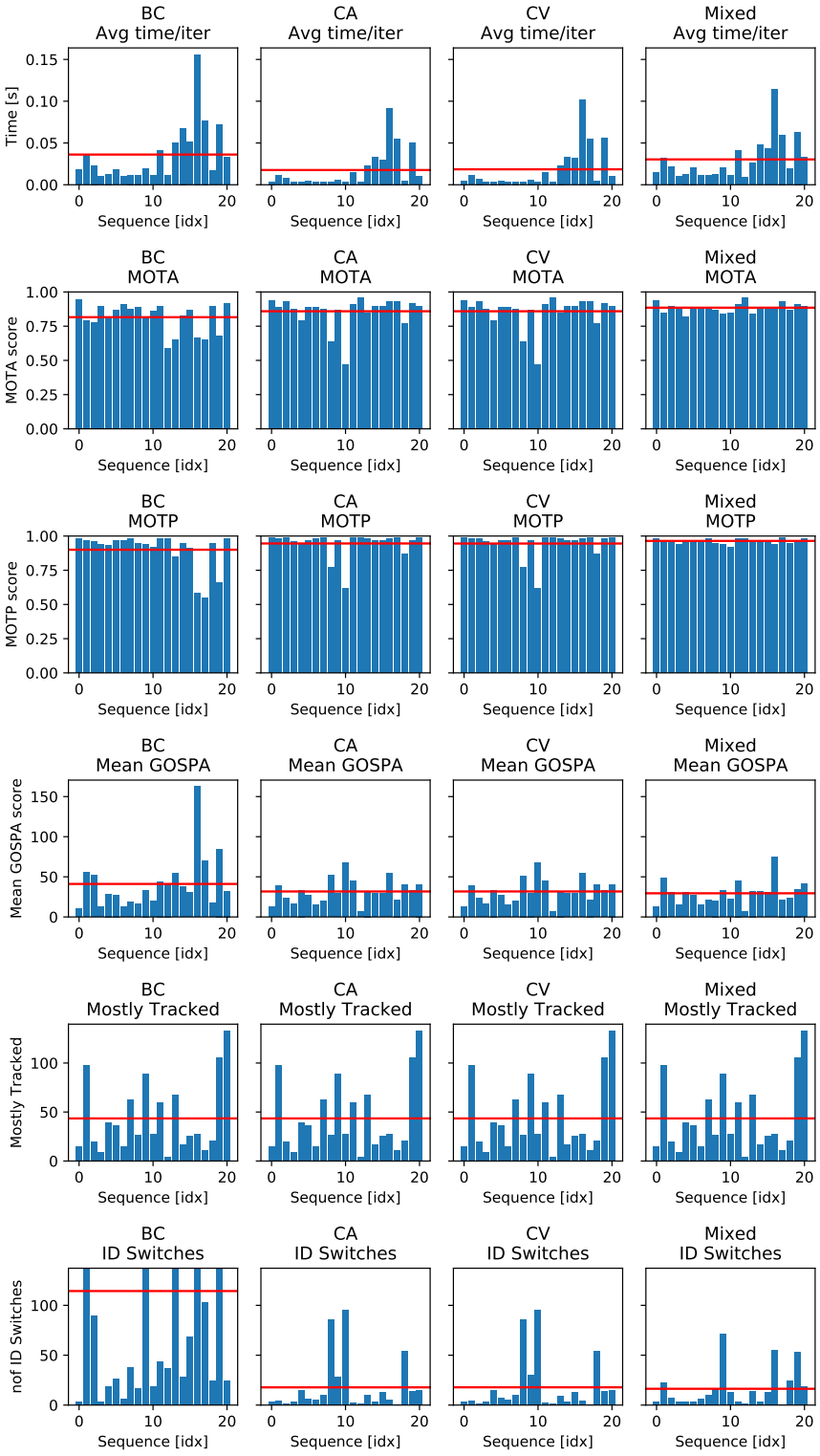


Figure 5.3: All metrics for all sequences. The red horizontal line is the mean for that motion model. All numbers can be seen in the tables in Appendix A.1. Note that the y-axis limit for the number of ID switches is capped for visualization purposes.

5.1.1 Predictions

The average GOSPA score, over all sequences, of the predictions for each time step ahead can be seen in Figure 5.4. Note that the predictions' GOSPA score varies a lot depending on the sequence, as can be seen in Figure 5.5 where a subset of all the sequences have been randomly chosen and displayed. Furthermore, the chosen number of predicted time steps is rather arbitrary and could be set to a higher or lower value dependent on the use case.

It is seen that the GOSPA score increases with higher number of time steps ahead, and that the GOSPA score has doubled ten time steps ahead. The mixed motion model performs best for the first 5 time steps ahead, but then yields a higher GOSPA than both CV and CA motion models. The Bicycle Model has a relatively high GOSPA and is also increasing with the highest rate. In particular, it can be seen that the predictions from the Bicycle Model are very poor in sequence 17, where there are only pedestrians present. However, in sequence 18, containing only cars, the predictions are rather good.

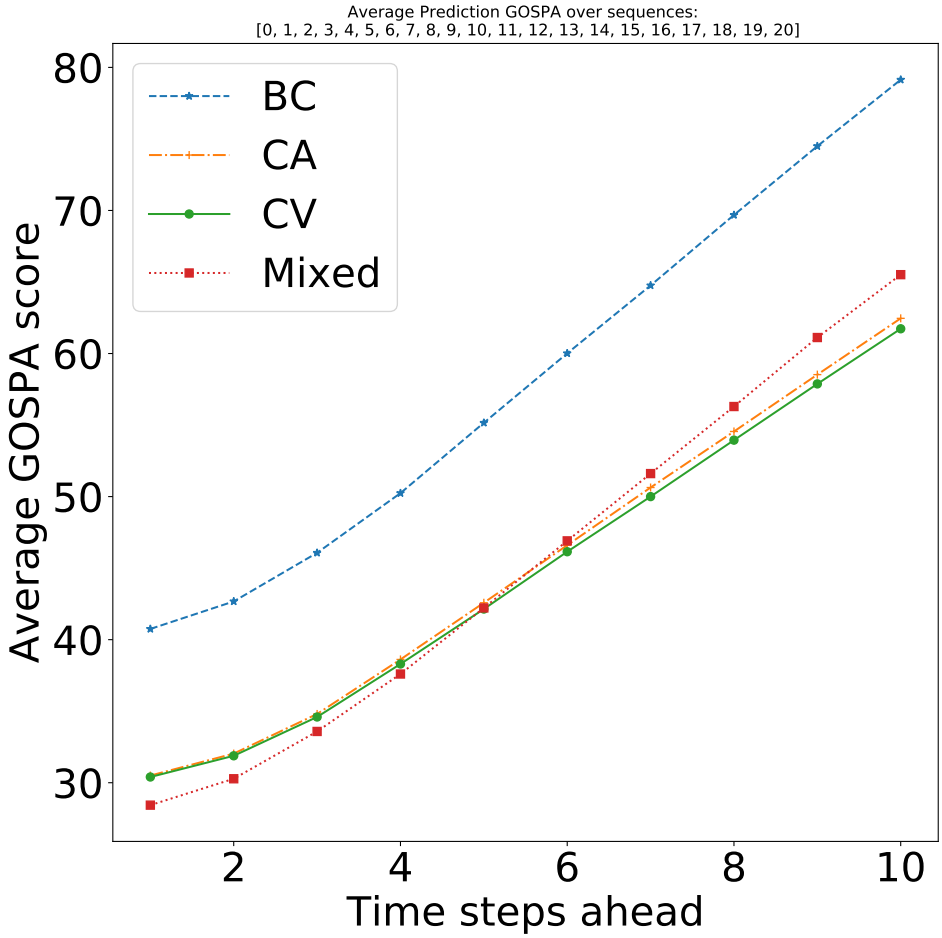
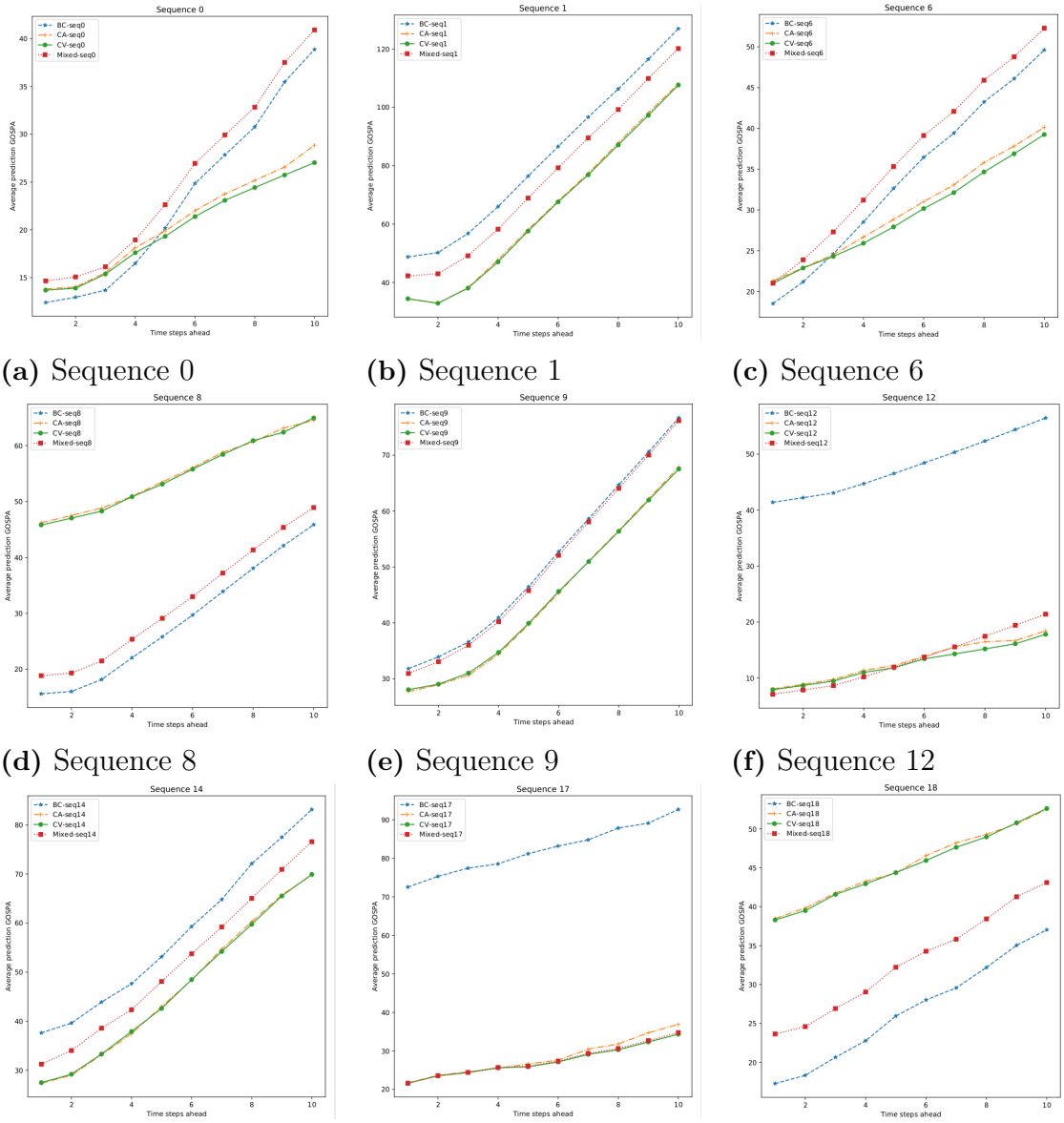


Figure 5.4: The GOSPA score of the prediction for each time step ahead ($k \in [1, 10]$). Lower is better.



(a) Sequence 0 **(b)** Sequence 1 **(c)** Sequence 6

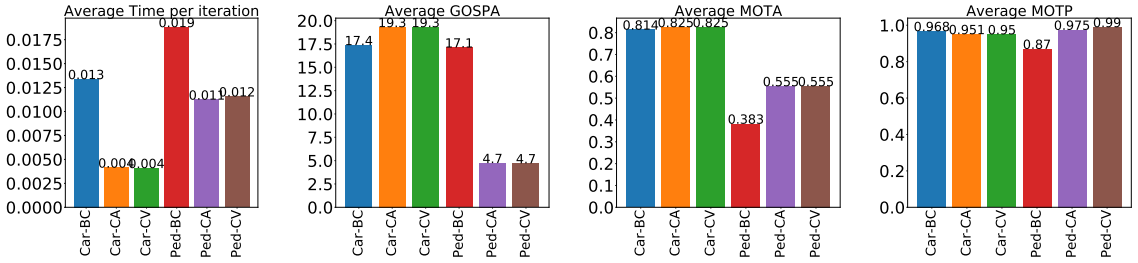
(d) Sequence 8 **(e)** Sequence 9 **(f)** Sequence 12

(g) Sequence 14 **(h)** Sequence 17 **(i)** Sequence 18

Figure 5.5: The GOSPA score of the prediction for each time step ahead ($k \in [0, 10]$) of a sub-set of the sequences. Lower is better. It is seen that the domestic performance among the motion models varies a lot depending on the sequence. For instance, the predictions from the Bicycle Model is very poor in sequence 17, where there are only pedestrians, and very good in sequence 18 where there only are cars.

5.1.2 Single classes

The results of tracking a certain class, i.e. car and pedestrian, can be seen in Figure 5.6 and Table 5.2. Here it is clear that the CA and CV models outperform the bicycle model for Pedestrians. They are faster due to the Bicycle model being non-linear, and thus propagated via the UKF, but it also yields more than three times lower GOSPA, higher MOTA and higher MOTP. It is not as clear difference for the Car category, where Bicycle model has slightly better GOSPA and marginally better MOTP, but just lower MOTA. The CA/CV models are more than three times faster than the bicycle model for pedestrians, but with similar performance. The CA/CV models are also faster for cars than pedestrians. This is because the pedestrians, often located in group and tightly located together, creates many more plausible hypotheses from the measurements. This in turn creates more global hypotheses to optimize over, taking more time. Furthermore, it is seen that the Bicycle model is struggling to connect objects over time as the average number of IDsw is high. This applies not only when tracking cars, but also when tracking pedestrians. The differences between CA and CV are rather negligible.



(a) Average time per iteration. Lower is better.

(b) Average GOSPA score. Lower is better.

(c) Average MOT Accuracy. Higher is better.

(d) Average MOT Precision. Higher is better.

Figure 5.6: Average metrics over all sequences for the PMBM filter tracking only cars and pedestrians respectively.

Metric	AvgTime	AvgMOTA	AvgMOTP	AvgGOSPA	AvgMT	AvgIDsw
Car-BC	0.0133	0.8138	0.9676	17.3723	30.1905	33.6190
Car-CA	0.0041	0.8252	0.9509	19.3028	30.1905	15.3333
Car-CV	0.0040	0.8252	0.9500	19.3047	30.1905	15.3333
Ped-BC	0.0188	0.3828	0.8700	17.1171	7.9524	74.3809
Ped-CA	0.0112	0.5547	0.9747	4.7385	7.9524	2.7619
Ped-CV	0.0115	0.5547	0.9895	4.7104	7.9524	2.0952

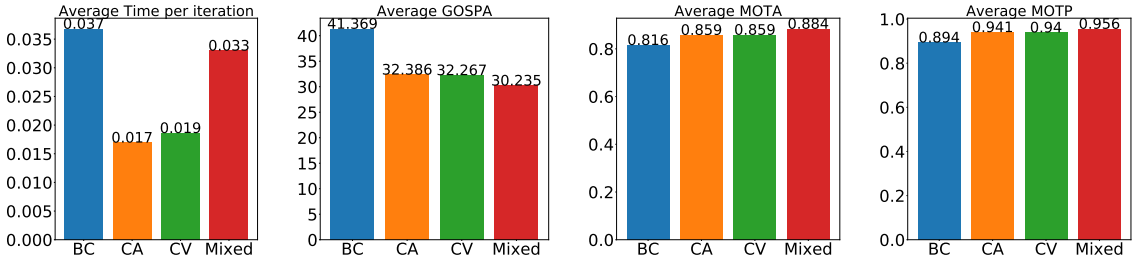
Table 5.2: Average metrics over all sequences for the PMBM filter tracking only cars and pedestrians respectively.

5.1.3 Noisier data

Results for the tests with noisier data, described in Section 4.3, is shown in Table 5.3 and Figure 5.7. A qualitative comparison is shown in Figure 5.8 for frame 198 of sequence 5. The quantitative results barely differ from the less noisy data, presented in Table 5.1. However, it can be seen in the qualitative comparison that the bicycle model overestimates the steering angle, e.g. for the orange coloured car in Figure 5.8(c) and (d). This leads to incorrect predictions, especially further in the future, as it predicts the car to turn. The quantitative results is unaffected because the filter only uses the prediction for one time step ahead. Despite the overshoot of steering angle it is able to create correct trajectories, however somewhat uneven due to the overshooting predictions.

Model	AvgTime	AvgMOTA	AvgMOTP	AvgGOSPA	AvgMT	AvgIDsw
BC	0.0368	0.8161	0.8942	41.3690	43.476	119.4761
CA	0.0169	0.8585	0.9409	32.3861	43.476	17.1904
CV	0.0186	0.8590	0.9395	32.2666	43.476	16.8571
Mixed	0.0330	0.8842	0.9561	30.2347	43.476	15.2380

Table 5.3: Average metrics over all sequences for the PMBM filter tracking using noisier simulated data.



(a) Average time per iteration. Lower is better.

(b) Average GOSPA score. Lower is better.

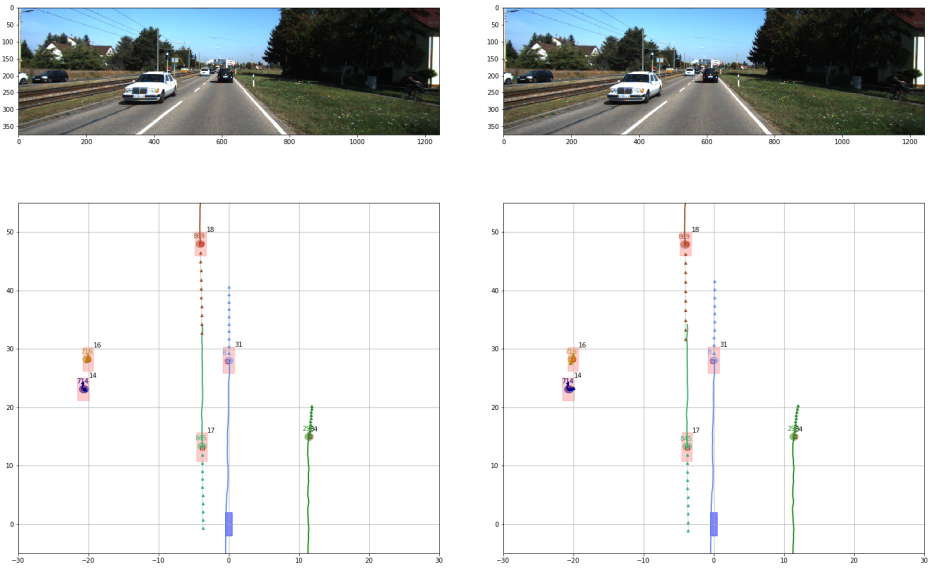
(c) Average MOTAccuracy. Higher is better.

(d) Average MOTPrecision. Higher is better.

Figure 5.7: Average metrics over all sequences for the PMBM filter tracking using noisier data.

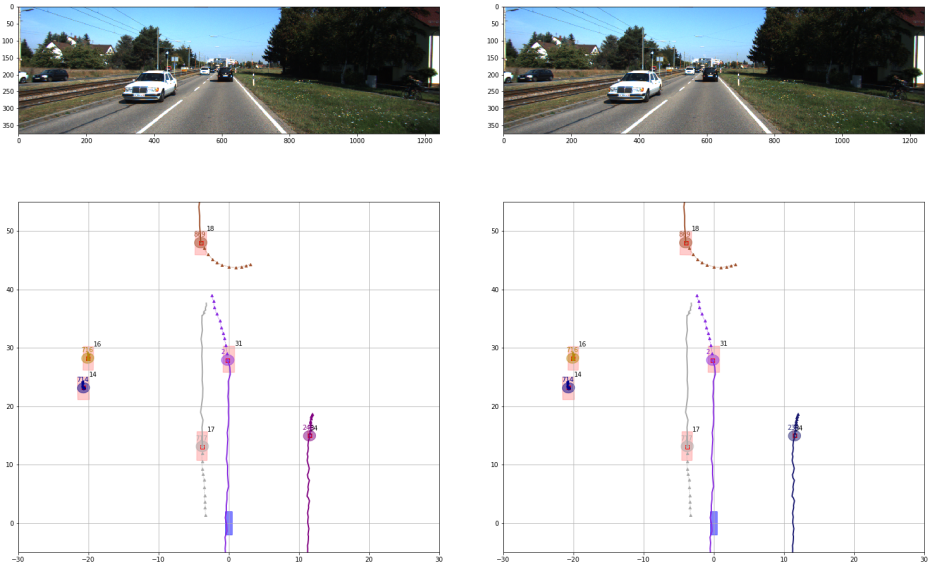
5.1.4 Hard cap global hypotheses

Results from running PMBM with a cap of max five global hypothesis is shown in Table 5.4. For CV/CA, it roughly halves the average time per iteration landing below 9 *ms*, while not dropping considerably in performance compared to Table 5.1, which uses a cap of max 100 global hypotheses.



(a) Constant Velocity

(b) Constant Acceleration



(c) Bicycle Model

(d) Mixed

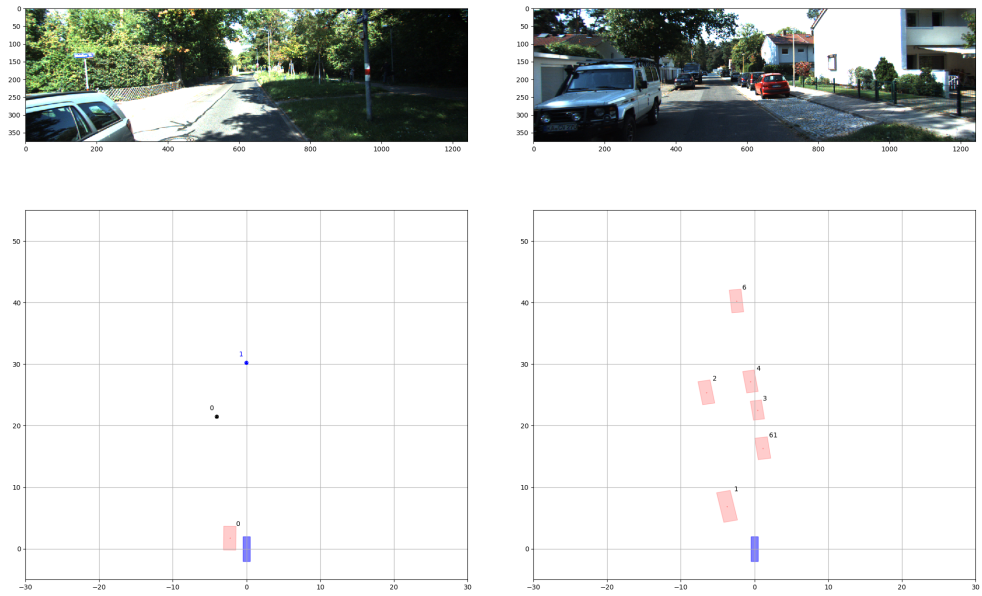
Figure 5.8: Image and Bird's Eye View of the world in the same frame (198) of sequence (5) for four different models when running on noisier data. Same structure of plotting as explained in Figure 5.1.

Model	AvgTime	AvgMOTA	AvgMOTP	AvgGOSPA	AvgMT	AvgIDsw
BC	0.0286	0.8138	0.8971	41.8380	43.4761	113.4761
CA	0.0089	0.8528	0.9385	33.5980	43.4761	17.6190
CV	0.0087	0.8547	0.9395	33.4690	43.4761	17.3333
Mixed	0.0230	0.8804	0.9576	31.5328	43.4285	14.8095

Table 5.4: Average metrics over all sequences for the PMBM filter tracking using max 5 cap of global hypotheses.

5.2 ETENN

The networks were not able to generalize during training, even though extensive tests of parameters were conducted. Running the network on validation data does not give any reasonable output, as can be seen in Figure 5.9, which is a good example of how the network guesses randomly. Why the network does not succeed to generalize for unseen data is discussed in Section 6.2. Henceforth, this section will now show results when running on training data, data the network has seen before, to showcase the potential of the algorithm.

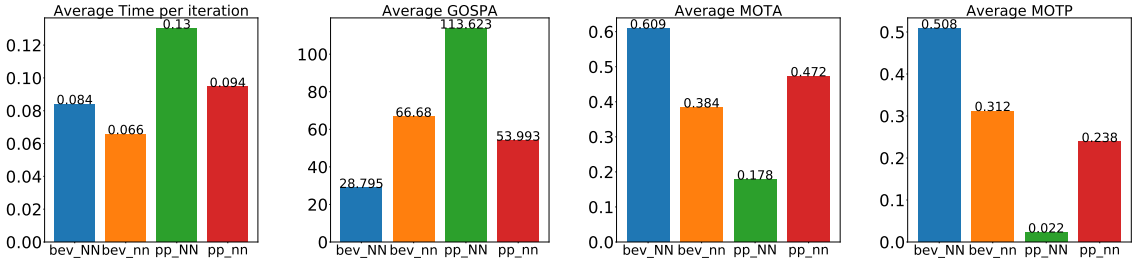


(a) BEV ETENN, frame 12 sequence 7

(b) BEV ETENN, frame 12 sequence 7

Figure 5.9: Tracking results on validation data. The left figure shows how the network misses an object and spawns two false positives. The right figure shows how the network misses all present objects.

The results from running ETENN including tracklet on training data, specifically on all sequences but 4, 7, 15, 16, and 17, can be seen in Figure 5.10 and Table 5.5. The ETENN column specifies which network structure was used, bev represents the BEV input processing and pp is the PointPillar input processing. NN represents the ETENN network structure and nn represents the smaller network structure. Here it can be seen that the BEV input with the larger network, bev_NN, and the PointPillar input with the smaller network, pp_nn, performs best. The average time for one time step includes the network's inference together with the decoding tracklet algorithm. It can be seen that the average time per iteration is more dependent on how the input is processed than the network size. The PointPillar input with the large ETENN structure is clearly the worst on all performance metrics.



(a) Average time per iteration. Lower is better.

(b) Average GOSPA score. Lower is better.

(c) Average MOTAccuracy. Higher is better.

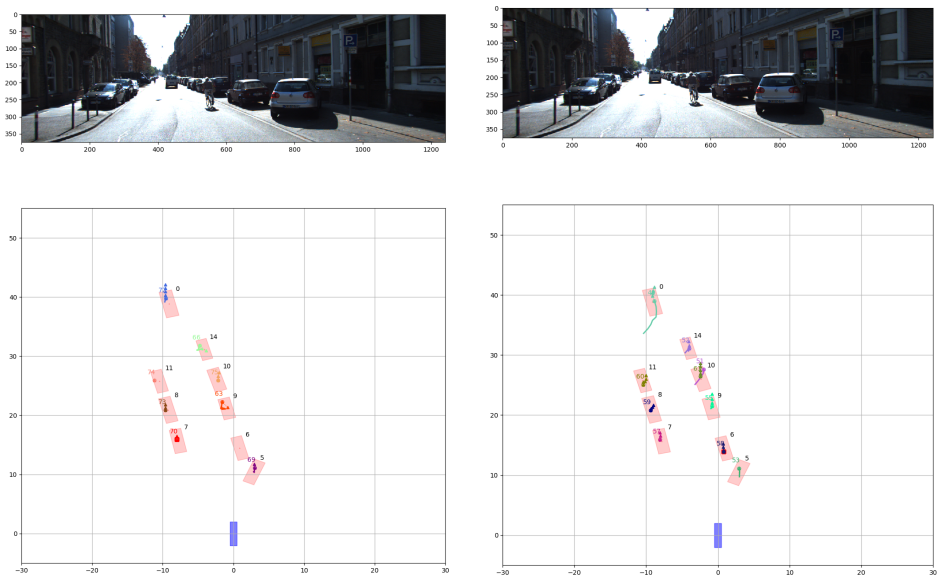
(d) Average MOTPrecision. Higher is better.

Figure 5.10: Average metrics over all sequences for the ETENN using different network architectures.

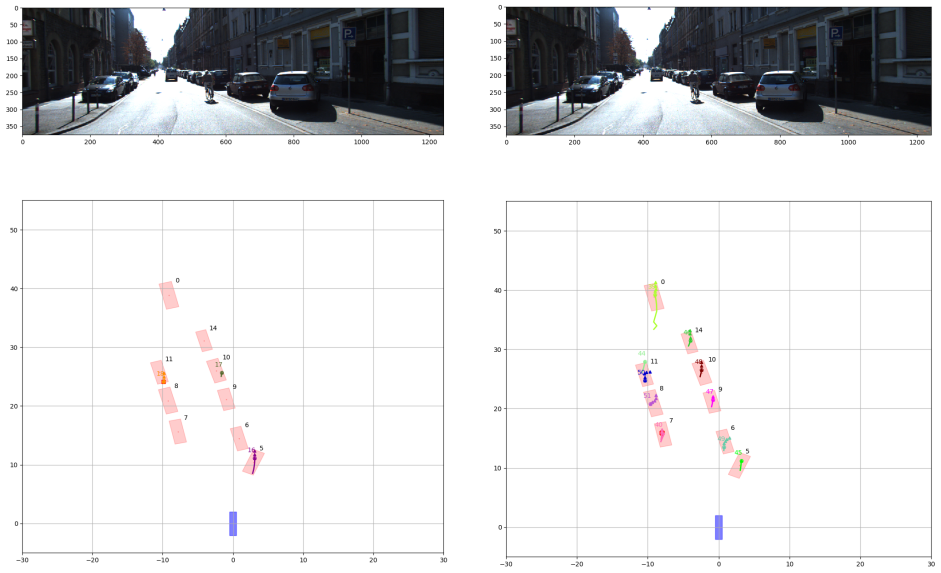
ETENN	AvgTime	AvgMOTA	AvgMOTP	AvgGOSPA	AvgMT	AvgIDsw
bev_NN	0.0840	0.6087	0.5081	28.7950	25.6250	291.1875
bev_nn	0.0656	0.3837	0.3118	66.6800	7.4375	180.6250
pp_NN	0.1300	0.1781	0.0225	113.6231	1.1250	106.6250
pp_nn	0.0945	0.4725	0.2381	53.9925	27.3125	303.3750

Table 5.5: Metrics for different network architectures. All sequences but 4, 7, 15, 16, and 17 as they were used for validation when training.

A qualitative look on the performance is seen in Figure 5.11, showing the tracking output at frame 132 for the training data sequence 0. Here it can be seen that the pp_NN is barely able to track any present objects. The other versions are tracking all objects, except bev_NN missing one.



(a) BEV `ETENN` **(b)** BEV smaller `ETENN`



(c) PointPillars `ETENN` **(d)** PointPillars smaller `ETENN`

Figure 5.11: Image and Bird's Eye View of the world in the same frame (132) of sequence (0) for four different network structures. The trajectory of a track is displayed as a full drawn line, the current state as a circle, the predictions for 10 time steps ahead as triangles. The current measurements are the tiny red edge boxes (transparent within) whereas the ground truths are seen as slightly faded red rectangles, where the ground truth width and height is used for visualization purposes only.

5.2.1 Prediction

The average prediction GOSPA of ETENN for sequence 0 can be seen in 5.12. It can be seen that the smaller networks perform better for longer predictions, as bev_nn passes bev_NN for the third prediction step and forward.

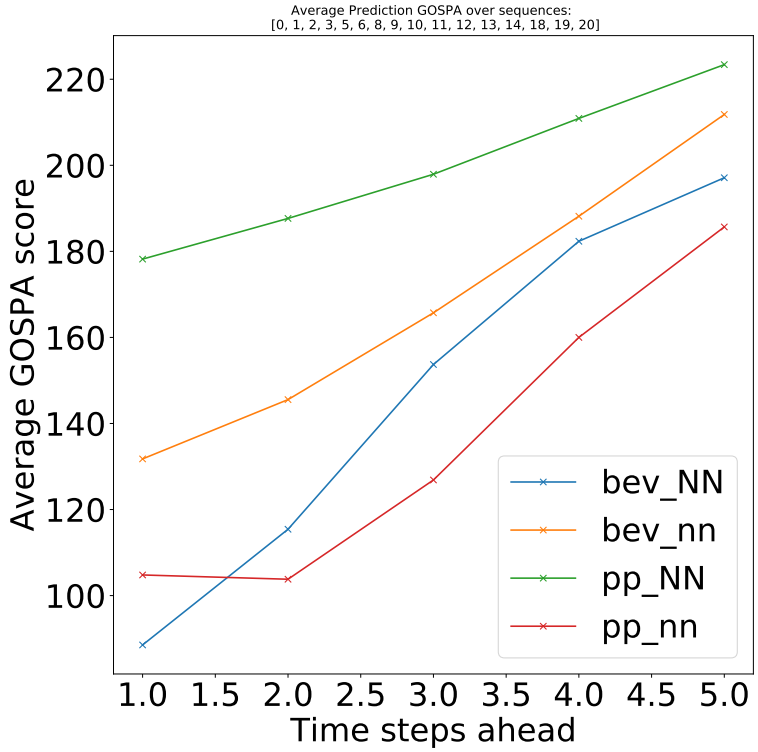
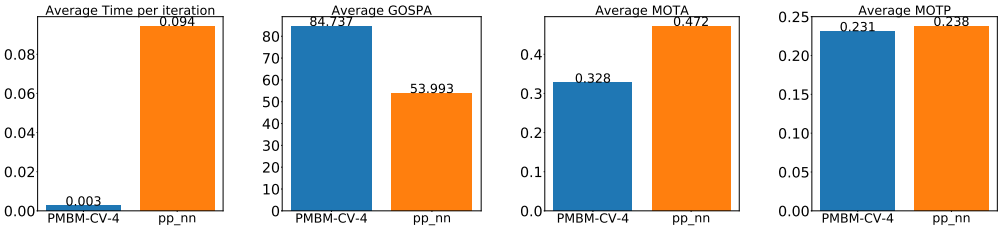


Figure 5.12: The GOSPA score of the prediction for each time step ahead ($k \in [1, 5]$) for the different network structures of ETENN.

5.3 PMBM with ETENN detections

In this section the detections from ETENN are used as measurements to the PMBM filter with Constant Velocity (CV) motion model. This section also re-visit the results of the tracking/prediction output from the ETENN algorithm as to easen the comparison between the two approaches. All sequences except for 4, 7, 15, 16, and 17 are used when evaluating the performance as those sequences were used as validation data when training the ETENN networks.

In Figure 5.13 and Table 5.6 the average metrics are displayed. The PMBM filter with CV model does not manage to improve the detection GOSPA, MOTA or MOTP. It does, however, manage to keep the number of average IDsw down. That means that we get better estimations on each tracks velocities. In turn, this leads to better predictions, as seen in Figure 5.14 where the predictions of the PMBM algorithm outperforms the ones of the pp_nn network. The most notable difference between the two is when it comes to iteration time, the ETENN is much slower.



(a) Average time per iteration. Lower is better.

(b) Average GOSPA score. Lower is better.

(c) Average MOTAccuracy. Higher is better.

(d) Average MOTPrecision. Higher is better.

Figure 5.13: Average metrics over all sequences for PMBM using ETENN detections and ETENN tracking.

Tracker	AvgMOTA	AvgMOTP	AvgGOSPA	AvgMT	AvgIDsw
PMBM-CV	0.3275	0.2312	84.7368	4.3125	68.1875
pp_nn	0.4725	0.2381	53.9925	27.3125	303.3750

Table 5.6: Results for all sequences except for 4, 7, 15, 16, and 17

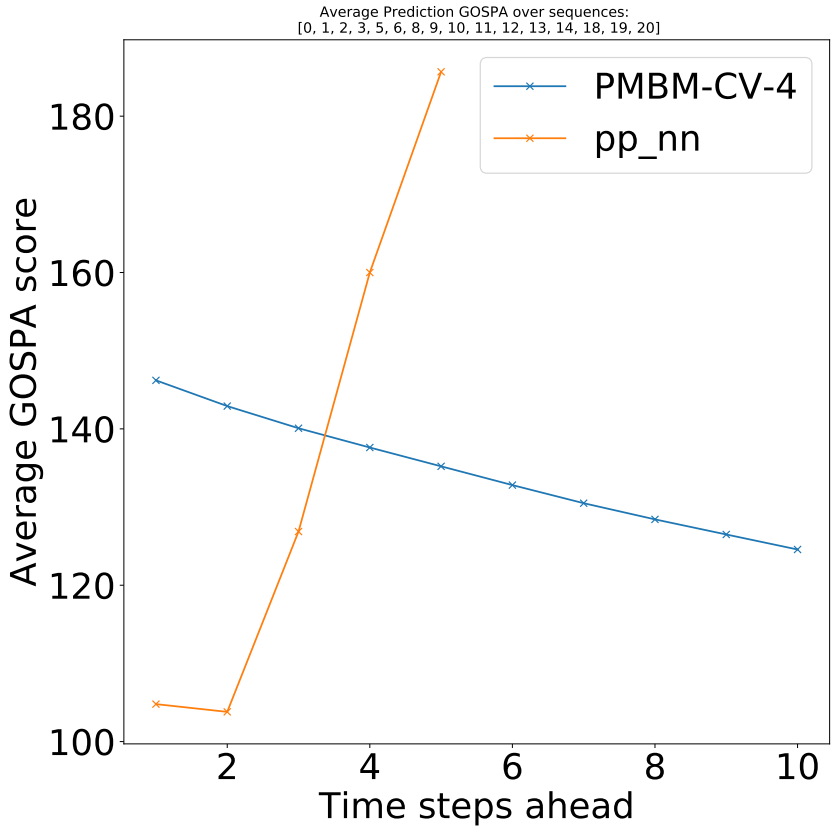


Figure 5.14: The GOSPA score of the prediction for each time step ahead for ETENN detections + PMBM and ETENN tracking predictions ($k \in [1, 10]$ and $k \in [1, 5]$ respectively).

6
Discussion

The aim of comparing the two approaches could unfortunately not be done in a fair way due to the ETENN not being able to generalize during training. The two approaches plus the combination of using PMBM with ETENN detections are therefore discussed separately. The chapter ends with Section 6.4, Future Work, where recommendations are proposed to be able to individually improve the two approaches and thus be able to thoroughly compare them.

6.1 PMBM

In order to mimic a state-of-the-art detector, the noise on the simulated data was independently and identically sampled from a normal distribution for all categories. This is arguably inaccurate in several ways. To begin with, as can be seen in Table 1 from [6], the performance of object detectors differs between the object categories. Continuing, sampling the angle measurements with the same variance as the position measurements might not reflect well. Partly because there is a difference between performance in orientation and bounding box regression, as seen when comparing Table 1 and 3 in [6], but also because the same level of noise is arguably relatively different in radians than in meters. Lastly, the simulated noise does not take into account the difficulty of the detections, such as occlusions or distance to object.

The PMBM performs well on the simulated data, for both levels of noise applied to the simulated data, generating smooth trajectories and good predictions. As can be seen in Figure 5.2 the Bicycle model's performance is inferior, while also being computationally heavier. Being the least efficient is reasonable, since it is non-linear and thus needs to be filtered with a non-linear Kalman Filter, in this case UKF. However, if one looks at the per-sequence detailed statistics, Figure 5.3, it shows that it has huge problems with a sub-set of the sequences, while performing fairly well on others. It performs especially poorly on sequence 16 and 17, having a high GOSPA, and low MOTP and MOTA . These are sequences with a lot of pedestrians, and sequence 16 in particular has a lot of pedestrians walking closely together. This is further emphasized in Figure 5.6, clearly showing the CV outperforming the bicycle model for pedestrians. Looking at the nature of the motion models compared to the nature of pedestrians movement, it makes sense that the bicycle model does not fit pedestrians movement. For example, the steering angle of the bicycle model does not resemble a natural state of the pedestrian. When it comes to the avarage number of switches the Bicycle model is inferior as well. This could very well be due

to it being harder to tune, but is also likely to originate from the poor predictions of pedestrians' movement.

The `Mixed` motion model is according to Figure 5.2 performing best, something that is underlined in Figure 5.3. Here it can be seen that the `Mixed` model covers the scenarios where the individual models lag behind. To be more specific, the `Mixed` performs well on sequence 16 and 17, sequences with a lot of pedestrians, because the `CV` model is more accurate for pedestrians. It also performs better on sequence 8 and 10 compared to `CA/CV` where there are many cars. The `Mixed`, using bicycle for cars, can change the speed and direction necessarily fast compared to `CA/CV`, because the bicycle model uses the orientation as measurement, compared to the `CA/CV` only using position. The `CA/CV` looses track and births new objects because it can't alter the direction and speed fast enough.

Despite being better than the bicycle model on pedestrians, Figure 5.6 clearly shows that both `CV` and `CA` has low `MOTA` for pedestrians, meaning that `CV` and `CA` has problems with tracking pedestrians as well. As mentioned above, the sequences with a lot of pedestrians are difficult tracking scenarios, but one thing that could affect the result is the tuning of the noise matrices. The matrices are the same for the different object classes, which might not reflect reality. Having different matrices, reflecting a difference in uncertainty of movement and measurement between cars and pedestrians, might have performed better.

The predictions for `PMBM` are only compared against future ground truth which are present in the current frame. That is, if a new object emerges in the future it will not affect the prediction `GOSPA` score. Another remark on the prediction `GOSPA` is that if there is a cardinality error in the prediction in the current frame, the cardinality error will be present for all future prediction steps.

The birth process of an object is not a straightforward choice. As described in Section 2.2.4, we use a Partial Uniform distribution for the undetected objects. When birthing, if no recycled Poisson component is connected to a measurement, the unmeasurable states are set to 0. This creates a problem. As discussed above, `CA/CV` do not perform as well on cars moving with some speed in the opposite direction, as the velocity cannot change fast enough. Using Poisson components as a Gaussian mixture instead of a uniform distribution does not solve the problem. Even if one set the velocity to a value depending on location of the measurement, e.g. velocity in the opposite direction of the ego vehicle for Poisson components located in location representing the opposite direction, it is not general enough. For example, the lane to the left of the ego vehicle is different depending on which road and which lane the ego vehicle is on. An idea would be to use a multi-modal Poisson distribution, changing models depending on the information available. If there is information regarding speed limits, ego-vehicle position in relation to road lanes, ego vehicle speed etc., it would be possible to create a system optimizing the birth process.

The test comparing the result of different cap of global hypothesis shows that the

PMBM does not need many global hypotheses to perform well. We would argue that this is because of letting the best global hypotheses generate more new global hypotheses than the not as good ones. By generating more new global hypotheses from the better ones, the need of keeping global hypotheses with lower weight is redundant. This might of course be a generalization proved wrong in certain cases, but our argument is that it is unlikely. Pruning for both weight and max cap is a method which should cover the case of not keeping around unnecessary ones while being able to ramp up when many measurements are close to many objects, creating many hypotheses.

The coordinate transform before predicting is simply an Euler integration of the data given by the IMU. Therefore the noise from the IMU becomes infused into the tracker and thus affects the performance of the tracking algorithm. Using a more sophisticated method for ego-motion could possibly improve the performance.

6.2 ETENN

The first and foremost problem with ETENN was underestimating the amount of data we needed. Even though training on all but one training sequence on KITTI, the network is far from generalizing and the validation loss, especially the classification loss, does not improve. The authors of Fast-and-Furious [8] use a dataset 20 times the size of KITTI.

We do not use any data augmentation, which we in hindsight believe clearly affected the inability to generalize. The authors of PointPillars [6] state the importance and suggest several methods. The problem with data augmentation for ETENN, and the reason behind us not implementing it, is the situation of adding realistic data over all time steps. Adding random ground truths, to balance positive/negative samples, becomes a very complex problem when it is to be done for 10 consecutive frames in a row. To augment the movement of an object over all these time steps in a realistic way seemed too far fetched. Not only do one have to ensure the added objects does not clash with an already existing object in any time step, it must also always be on the road. The added objects must be in the field of view and cannot be occluded by any other objects because then the LiDAR beams would not in reality be able to reach it. However, flipping, small rotations and translations could, and probably should, have been tested in order to generalize the data slightly.

In [8] they describe the importance of coordinate transforming the input data to account for the ego motion of the ego vehicle. We did not implement this, inputting data for each time step in relation to that current position of the ego vehicle. This means that the network learns to track objects relative to itself. This adds unnecessary complexity since the NN not only needs to learn how the surrounding objects, but also how the ego vehicle move. As it is realistic to assume the IMU data is available in real time this coordinate transform would be a relatively simple way to favour the learning process.

Furthermore, we did not encode reflectivity when using PointPillars input processing. This is something which we would do different if starting over training the networks, as it without adding much complexity adds important information. Reflectivity can be a good source of information to when deciding if it is a car or not, and it is reasonable to hypothesize that including reflectivity could have improved the performance.

There are currently a lot of unnecessary computations being done, as the input point cloud is quite a lot bigger than the FOV holding the ground truths. Due to the necessity of symmetry in the convolutions of the network, the input point cloud was not formed as the FOV but rather a rectangle. Except for unnecessary computations, this will further complicate the training, as it adds more imbalance to the positive/negative sample ratio, as the area outside the FOV only contain negative samples. It might also affect the end-to-end training with the PointPillar VFE layer, as it is one VFE layer for all pillars.

It is hard to argue for characteristics of the algorithm based on the results showed on the training data, as it is simply not possible to say what is actually learned and what is overfitted learning. The results however show the potential of the decoding tracking algorithm, which by jointly reasoning between current detection and predictions performs quite well despite bad detections and predictions.

The decoding tracklet algorithm is a simple algorithm logic-wise, jointly reasoning over detections and given predictions. A thought could be to use Bayesian recursion filter after the ETENN instead of this.

6.3 PMBM with ETENN detections

PMBM performs adequately given the measurements from ETENN. It does not improve the GOSPA, MOTA, or MOTP compared to ETENN. It does, however, improve the predictions further ahead and keeps the number of IDsw at a lower level. Further, the PMBM filter makes it possible to predict further ahead in time, possibly due to being able to keeping the same trackID for an object and thus accumulating more information about the velocity and heading. However, it should be noted that since the ETENN predictions are based on the same data it was trained on it is impossible to know how overfitted they are.

6.4 Future work

To perform well for a vast variety of tracking scenarios the two approaches needs to be improved in different ways. However, as ETENN never generalized, it is necessary to improve ETENN to be able to compare the two approaches at all.

For ETENN, improving the issues mentioned Section 6.2 is a key together with implementing data augmentation. After that, the next step is to use more data. We

looked into using NuScenes [70], which offer more data, but only has annotations every fifth time frame, clocking in at 2Hz. It would be interesting to see if despite the sparsity in annotation it would be possible to train an ETENN-like network.

We firmly believe, based on the performance of [6], that the combination of a more sophisticated point cloud representation as the presented and implemented PointPillar can improve the performance of the complete ETENN compared to the BEV representation.

Using a real object detector would validate the performance of the PMBM. One could, e.g., submit a run on the test data to the KITTI benchmark, being able to compare against the state-of-the-art tracking algorithms. To overall improve the performance, the algorithm needs to be tuned for different models depending not only on object class, but also on the current state and environment. Using a more sophisticated birth process such as discussed above in Section 6.1 would arguably also improve the performance.

Neither our ETENN-structures nor Fast-and-Furious propagate any uncertainty. Luo mentioned in the [8] conference talk that they are working on it. This could be an interesting perspective to look into.

The decoding tracklet post ETENN could possibly be replaced by some kind of RNN, such as [45]. This would truly make it End-to-End Deep Learning. The decoding tracklet could also be replaced by a more sophisticated filter method, such as the PMBM. Using the predictions as additional measurements could help the performance of the PMBM compared to only receiving measurements containing location, orientation and size.

Neither ETENN nor PMBM has any Affinity Model, such as appearance modelling of objects and measurements, implemented, which as described in [13] is one of the key characteristics of the top trackers studied in that paper. Extending ETENN with a RNN would e.g. allow for matching of extracted feature maps at a lower level in the network. To implement this for PMBM, the object detector would need to be modified to output the affinity measurement.

7
Conclusion

In this thesis, the problem of Multi-Object Tracking was approached in two different ways. A Poisson Multi-Bernoulli Mixture filter (PMBM) was implemented and an end-to-end deep learning neural network (ETENN) was implemented and trained. As the ETENN approach did not generalize we were not able to compare the two methods explicitly. However, the individual performances were studied.

The PMBM filter performs well for several motion models on simulated noisy data from KITTI [1] ground truth labels. The added noise aims to imitate the behaviour of a state-of-the-art 3D object detector. The PMBM filter manages to connect information from objects over time and fairly accurately predicts future states. The motion model heavily affects the performance of the tracking algorithm, and to use different motion models depending on object class is shown to be superior. The algorithm is rather robust to noise, but further fine-tuning and adding the possibility of using different tuning for different scenarios can enhance the performance further.

The ETENN was implemented using two different input processes, one Bird's Eye View grid and one self-learning feature extractor, and two different backbone networks, both implemented using ResNet blocks of different sizes. None of the network structures were able to generalize from the training, which arguably was partly due to implementation issues and partly due to the amount of data available.

Comparing the PMBM fed with ETENN detections as measurements with the ETENN tracking on training data, the GOSPA, MOTA, and MOTP were superior for the latter case. However, the PMBM filter is superior when it comes to predictions further ahead in time, as well as keeping the same track ID for a certain object, i.e. understanding it is the same object over a longer period of time.

Suggestions on how to improve the PMBM and the ETENN were proposed to be able to fully compare the two approaches.

All code can be found at https://github.com/erikbohnsack/pmbm and https://github.com/erikbohnsack/etenn. Videos can be found via www.adamlilja.com/m-thesis

Bibliography

[1] A. Geiger, P. Lenz, and R. Urtasun, "Are we ready for autonomous driving? the kitti vision benchmark suite," pp. 3354–3361, 2012.

[2] C. K. Law, D. Dalal, and S. Shearrow, "Robust model predictive control for autonomous vehicles/self driving cars," *CoRR*, vol. abs/1805.08551, 2018.

[3] K. Granstrom and L. Svensson, "Multi-object tracking for automotive systems," 2019. `https://www.edx.org/course/multi-target-tracking-for-automotive-systems`,.

[4] V. Dumoulin and F. Visin, "A guide to convolution arithmetic for deep learning," *ArXiv e-prints*, vol. abs/1603.07285, 2016.

[5] S. R. J. S. Kaiming He, Xiangyu Zhang, "Deep Residual Learning for Image Recognition," *ArXiv e-prints*, vol. abs/1512.03385v1, 2015.

[6] A. H. Lang, S. Vora, H. Caesar, L. Zhou, J. Yang, and O. Beijbom, "Pointpillars: Fast encoders for object detection from point clouds," *arXiv e-prints*, vol. abs/1812.05784, 2018.

[7] Y. Zhou and O. Tuzel, "Voxelnet: End-to-end learning for point cloud based 3d object detection," *arXiv e-prints*, vol. abs/1711.06396, 2017.

[8] W. Luo, B. Yang, and R. Urtasun, "Fast and furious: Real time end-to-end 3d detection, tracking and motion forecasting with a single convolutional net," pp. 3569–3577, 2018.

[9] S. Casas, W. Luo, and R. Urtasun, "Intentnet: Learning to predict intention from raw sensor data," in *Proceedings of The 2nd Conference on Robot Learning* (A. Billard, A. Dragan, J. Peters, and J. Morimoto, eds.), vol. 87 of *Proceedings of Machine Learning Research*, pp. 947–956, PMLR, 2018.

[10] D. Frossard and R. Urtasun, "End-to-end learning of multi-sensor 3d tracking by detection," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 635–642, 2018.

[11] Angel F. García-Fernandez, Jason L. Williams, Karl Granstrom, and Lennart Svensson, "Poisson multi-Bernoulli mixture filter: direct derivation and implementation," 2017.

[12] Y. Xia, K. Granstrom, L. Svensson, and A. F. García-Fernández, "Performance evaluation of multi-bernoulli conjugate priors for multi-target filtering," pp. 1–8, 2017.

[13] L. Leal-Taixé, A. Milan, K. Schindler, D. Cremers, I. D. Reid, and S. Roth, "Tracking the trackers: An analysis of the state of the art in multiple object tracking," *arXiv e-prints*, vol. abs/1704.02781, 2017.

[14] L. Leal-Taixé, A. Milan, I. D. Reid, S. Roth, and K. Schindler, "Motchallenge 2015: Towards a benchmark for multi-target tracking," *arXiv e-prints*, vol. abs/1504.01942, 2015.

[15] A. Milan, L. Leal-Taixé, I. D. Reid, S. Roth, and K. Schindler, "MOT16: A benchmark for multi-object tracking," *arXiv e-prints*, vol. abs/1603.00831, 2016.

[16] P. Emami, P. M. Pardalos, L. Elefteriadou, and S. Ranka, "Machine learning methods for solving assignment problems in multi-target tracking," *arXiv e-prints*, vol. abs/1802.06897, 2018.

[17] R. B. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," *arXiv e-prints*, vol. abs/1311.2524, 2013.

[18] R. B. Girshick, "Fast R-CNN," *arXiv e-prints*, vol. abs/1504.08083, 2015.

[19] J. Dai, Y. Li, K. He, and J. Sun, "R-FCN: object detection via region-based fully convolutional networks," *arXiv e-prints*, vol. abs/1605.06409, 2016.

[20] S. Ren, K. He, R. B. Girshick, and J. Sun, "Faster R-CNN: towards real-time object detection with region proposal networks," *arXiv e-prints*, vol. abs/1506.01497, 2015.

[21] J. Redmon, S. K. Divvala, R. B. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," *arXiv e-prints*, vol. abs/1506.02640, 2015.

[22] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. E. Reed, C. Fu, and A. C. Berg, "SSD: single shot multibox detector," *arXiv e-prints*, vol. abs/1512.02325, 2015.

[23] T. Lin, P. Goyal, R. B. Girshick, K. He, and P. Dollár, "Focal loss for dense object detection," *arXiv e-prints*, vol. abs/1708.02002, 2017.

[24] J. Redmon and A. Farhadi, "YOLO9000: better, faster, stronger," *arXiv e-prints*, vol. abs/1612.08242, 2016.

[25] J. Redmon and A. Farhadi, "Yolov3: An incremental improvement," *arXiv e-prints*, vol. abs/1804.02767, 2018.

[26] X. Chen, K. Kundu, Z. Zhang, H. Ma, S. Fidler, and R. Urtasun, "Monocular 3d object detection for autonomous driving," pp. 2147–2156, 2016.

[27] X. Chen, K. Kundu, Y. Zhu, H. Ma, S. Fidler, and R. Urtasun, "3d object proposals using stereo imagery for accurate object class detection," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 40, no. 5, pp. 1259–1272, 2018.

[28] C. R. Qi, H. Su, K. Mo, and L. J. Guibas, "Pointnet: Deep learning on point sets for 3d classification and segmentation," *arXiv e-prints*, vol. abs/1612.00593, 2016.

[29] B. Li, "3d fully convolutional network for vehicle detection in point cloud," *arXiv e-prints*, vol. abs/1611.08069, 2016.

[30] M. Engelcke, D. Rao, D. Z. Wang, C. H. Tong, and I. Posner, "Vote3deep: Fast object detection in 3d point clouds using efficient convolutional neural networks," *arXiv e-prints*, vol. abs/1609.06666, 2016.

[31] B. Yang, W. Luo, and R. Urtasun, "PIXOR: real-time 3d object detection from point clouds," *arXiv e-prints*, vol. abs/1902.06326, 2019.

[32] M. Simon, S. Milz, K. Amende, and H. Gross, "Complex-yolo: Real-time 3d object detection on point clouds," *arXiv e-prints*, vol. abs/1803.06199, 2018.

[33] X. Chen, H. Ma, J. Wan, B. Li, and T. Xia, "Multi-view 3d object detection network for autonomous driving," *arXiv e-prints*, vol. abs/1611.07759, 2016.

[34] J. Ku, M. Mozifian, J. Lee, A. Harakeh, and S. L. Waslander, "Joint 3d proposal generation and object detection from view aggregation," *arXiv e-prints*, vol. abs/1712.02294, 2017.

[35] Y. Yan, Y. Mao, and B. Li, "Second: Sparsely embedded convolutional detection," *Sensors*, vol. 18, p. 3337, 2018.

[36] K. Granström, M. Baum, and S. Reuter, "Extended object tracking: Introduction, overview and applications," *Journal of Advances in Information Fusion*, vol. 12, pp. 139–174, Dec. 2017.

[37] B.-N. Vo, M. Mallick, Y. bar shalom, S. Coraluppi, R. Osborne III, R. Mahler, and B.-T. Vo, "Multitarget tracking," *Wiley Encyclopedia*, pp. 1–25, 2015.

[38] C. Kim, F. Li, A. Ciptadi, and J. M. Rehg, "Multiple hypothesis tracking revisited," in *2015 IEEE International Conference on Computer Vision (ICCV)*, pp. 4696–4704, 2015.

[39] S. H. Rezatofighi, A. Milan, Z. Zhang, Q. Shi, A. Dick, and I. Reid, "Joint probabilistic data association revisited," in *2015 IEEE International Conference on Computer Vision (ICCV)*, pp. 3047–3055, 2015.

[40] B. Vo and B. Vo, "Labeled random finite sets and multi-object conjugate priors," *IEEE Transactions on Signal Processing*, vol. 61, no. 13, pp. 3460–3475, 2013.

[41] S. Reuter, B. Vo, B. Vo, and K. Dietmayer, "The labeled multi-bernoulli filter," *IEEE Transactions on Signal Processing*, vol. 62, no. 12, pp. 3246–3260, 2014.

[42] J.L. Williams, "Marginal multi-Bernoulli filters: RFS derivation of MHT, JIPDA and association-based MeMBer," *ArXiv e-prints*, 2012.

[43] J. Williams, "Hybrid Poisson and multi-Bernoulli filters," *ArXiv e-prints*, 2012.

[44] A. S. Rahmathullah, Á. F. García-Fernández, and L. Svensson, "A metric on the space of finite sets of trajectories for evaluation of multi-target tracking algorithms," *arXiv e-prints*, vol. abs/1605.01177, 2016.

[45] A. Milan, S. H. Rezatofighi, A. R. Dick, K. Schindler, and I. D. Reid, "Online multi-target tracking using recurrent neural networks," *arXiv e-prints*, vol. abs/1604.03635, 2016.

[46] A. Sadeghian, A. Alahi, and S. Savarese, "Tracking the untrackable: Learning to track multiple cues with long-term dependencies," *arXiv e-prints*, vol. abs/1701.01909, 2017.

[47] J. Zhu, H. Yang, N. Liu, M. Kim, W. Zhang, and M. Yang, "Online multi-object tracking with dual matching attention networks," *arXiv e-prints*, vol. abs/1902.00749, 2019.

[48] S. Tang, B. Andres, M. Andriluka, and B. Schiele, "Multi-person tracking by multicut and deep matching," *arXiv e-prints*, vol. abs/1608.05404, 2016.

[49] W. Choi, "Near-online multi-target tracking with aggregated local flow descriptor," *arXiv e-prints*, vol. abs/1504.02340, 2015.

[50] S. Sun, N. Akhtar, H. Song, A. Mian, and M. Shah, "Deep affinity network for multiple object tracking," *arXiv e-prints*, vol. abs/1810.11780, 2018.

[51] D. Held, S. Thrun, and S. Savarese, "Learning to track at 100 FPS with deep regression networks," *arXiv e-prints*, vol. abs/1604.01802, 2016.

[52] L. Leal-Taixé, C. Canton-Ferrer, and K. Schindler, "Learning by tracking: Siamese CNN for robust target association," *arXiv e-prints*, vol. abs/1604.07866, 2016.

[53] E. A. Wan and R. Van Der Merwe, "The unscented kalman filter for nonlinear estimation," *Proceedings of the IEEE 2000 Adaptive Systems for Signal Processing, Communications, and Control Symposium (Cat. No.00EX373)*, pp. 153–158, 2000.

[54] R. Mahler, *Advances in Multisource-Multitarget Information Fusion*. 2014.

[55] R. Mahler, *Statistical Multisource-Multitarget Information Fusion*. 2007.

[56] D. Reid, "An algorithm for tracking multiple targets," *IEEE Transactions on Automatic Control*, vol. 24, no. 6, pp. 843–854, 1979.

[57] T. Kurien, "Issues in the design of practical multi-target tracking algorithms," *Multitarget-Multisensor Tracking: Advanced Applications*, 1990.

[58] M. Beard, B.T Vo, B.N Vo, S. Arulampalam, "Gaussian Mixture PHD and CPHD Filtering withPartially Uniform Target Birth," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 49, pp. 2835–2844, 2013.

[59] H. W. Kuhn, "The Hungarian method for the assignment problem," *Naval Research Logistics Quarterly*, 1955.

[60] Katta G. Murty, "An Algorithm for Ranking all the Assignments in Order of Increasing Cost ," *Naval Research Logistics Quarterly*, vol. 16, pp. 682–687, 1968.

[61] P. Emami, P. M. Pardalos, L. Elefteriadou, and S. Ranka, "Machine learning methods for solving assignment problems in multi-target tracking," *arXiv e-prints*, vol. abs/1802.06897, 2018.

[62] "Python Package for Murty's algorithm." `https://github.com/erikbohnsack/murty`.

[63] M. Wahde, *Biologically Inspired Optimization Methods*. WIT Press, 2008.

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

[65] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *arXiv e-prints*, vol. abs/1502.03167, 2015.

[66] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *arXiv e-prints*, vol. abs/1704.04861, 2017.

[67] D. Erhan, C. Szegedy, A. Toshev, and D. Anguelov, "Scalable object detection using deep neural networks," *arXiv e-prints*, vol. abs/1312.2249, 2013.

[68] K. Bernardin and Stiefelhagen, "Evaluating multiple object tracking performance: The clear mot metrics," *EURASIP Journal on Image and Video Processing*, vol. 2008, 2008.

[69] C. Y. Li, C. Huang, and R. Nevatia, "Learning to associate: Hybridboosted multi-target tracker for crowded scene," *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2953–2960, 2009.

[70] H. Caesar, V. Bankiti, A. H. Lang, S. Vora, V. E. Liong, Q. Xu, A. Krishnan, Y. Pan, G. Baldan, and O. Beijbom, "nuscenes: A multimodal dataset for autonomous driving," *arXiv e-prints*, vol. abs/1903.11027, 2019.

A

Appendix - PMBM

A.1 Statistics for all sequences comparing motion models

SeqId	Cfg	#Frames	Time	MOTA	MOTP	GOSPA	MT	FP	IDsw
0	BC	153	0.018	0.95	0.98	10.13	15	25	3
0	CA	153	0.004	0.94	0.99	12.95	15	34	3
0	CV	153	0.005	0.94	0.99	12.94	15	34	3
0	Mixed	153	0.015	0.94	0.98	13.07	15	34	3

Table A.1: Results for sequence 0

SeqId	Cfg	#Frames	Time	MOTA	MOTP	GOSPA	MT	FP	IDsw
1	BC	426	0.035	0.79	0.97	56	98	494	146
1	CA	426	0.011	0.89	0.98	38.49	98	337	4
1	CV	426	0.011	0.89	0.98	38.48	98	337	4
1	Mixed	426	0.032	0.85	0.97	48.91	98	432	23

Table A.2: Results for sequence 1

SeqId	Cfg	#Frames	Time	MOTA	MOTP	GOSPA	MT	FP	IDsw
2	BC	223	0.023	0.78	0.96	52.27	20	238	90
2	CA	223	0.008	0.93	0.99	23.7	20	106	1
2	CV	223	0.007	0.93	0.98	23.78	20	106	1
2	Mixed	223	0.022	0.9	0.97	30.9	20	139	7

Table A.3: Results for sequence 2

SeqId	Cfg	#Frames	Time	MOTA	MOTP	GOSPA	MT	FP	IDsw
3	BC	143	0.01	0.9	0.94	12.64	9	30	3
3	CA	143	0.003	0.88	0.96	16.07	9	40	3
3	CV	143	0.003	0.88	0.96	16.05	9	40	3
3	Mixed	143	0.01	0.88	0.94	15.09	9	37	3

Table A.4: Results for sequence 3

SeqId	Cfg	#Frames	Time	MOTA	MOTP	GOSPA	MT	FP	IDsw
4	BC	313	0.013	0.82	0.93	28.82	39	160	19
4	CA	313	0.003	0.79	0.94	33.65	39	191	15
4	CV	313	0.004	0.79	0.94	33.64	39	191	15
4	Mixed	313	0.013	0.82	0.96	30.51	39	172	3

Table A.5: Results for sequence 4

SeqId	Cfg	#Frames	Time	MOTA	MOTP	GOSPA	MT	FP	IDsw
5	BC	296	0.018	0.87	0.97	27.63	36	159	26
5	CA	296	0.005	0.89	0.97	27.06	36	156	6
5	CV	296	0.005	0.89	0.97	27.34	36	158	7
5	Mixed	296	0.021	0.89	0.97	26.62	36	153	3

Table A.6: Results for sequence 5

SeqId	Cfg	#Frames	Time	MOTA	MOTP	GOSPA	MT	FP	IDsw
6	BC	268	0.01	0.91	0.97	12.46	15	59	6
6	CA	268	0.003	0.89	0.98	14.78	15	72	5
6	CV	268	0.003	0.89	0.97	14.84	15	72	5
6	Mixed	268	0.011	0.89	0.97	15.24	15	74	6

Table A.7: Results for sequence 6

SeqId	Cfg	#Frames	Time	MOTA	MOTP	GOSPA	MT	FP	IDsw
7	BC	679	0.012	0.88	0.98	18.53	63	277	38
7	CA	679	0.004	0.88	0.99	19.73	63	299	10
7	CV	679	0.004	0.88	0.99	19.72	63	299	10
7	Mixed	679	0.012	0.88	0.98	20.74	63	314	10

Table A.8: Results for sequence 7

SeqId	Cfg	#Frames	Time	MOTA	MOTP	GOSPA	MT	FP	IDsw
8	BC	389	0.012	0.89	0.95	16.37	27	118	17
8	CA	389	0.004	0.64	0.77	51.83	27	392	86
8	CV	389	0.003	0.64	0.77	51.57	27	390	86
8	Mixed	389	0.013	0.87	0.95	20.36	27	149	16

Table A.9: Results for sequence 8

SeqId	Cfg	#Frames	Time	MOTA	MOTP	GOSPA	MT	FP	IDsw
9	BC	783	0.02	0.82	0.94	33.59	89	521	152
9	CA	783	0.006	0.87	0.97	29.35	89	452	28
9	CV	783	0.006	0.87	0.97	29.89	89	461	30
9	Mixed	783	0.021	0.84	0.94	33.34	89	515	71

Table A.10: Results for sequence 9

SeqId	Cfg	#Frames	Time	MOTA	MOTP	GOSPA	MT	FP	IDsw
10	BC	293	0.011	0.86	0.92	19.83	28	110	19
10	CA	293	0.003	0.47	0.62	67.74	28	395	95
10	CV	293	0.003	0.47	0.62	67.73	28	395	95
10	Mixed	293	0.011	0.85	0.92	22.05	28	125	13

Table A.11: Results for sequence 10

SeqId	Cfg	#Frames	Time	MOTA	MOTP	GOSPA	MT	FP	IDsw
11	BC	372	0.041	0.9	0.98	44.23	60	317	44
11	CA	372	0.015	0.91	0.99	44.57	60	325	2
11	CV	372	0.015	0.91	0.99	44.55	60	325	2
11	Mixed	372	0.041	0.91	0.98	45.37	60	330	3

Table A.12: Results for sequence 11

SeqId	Cfg	#Frames	Time	MOTA	MOTP	GOSPA	MT	FP	IDsw
12	BC	77	0.012	0.59	0.98	41.93	4	63	37
12	CA	77	0.003	0.96	0.99	6.79	4	10	1
12	CV	77	0.003	0.96	0.99	6.82	4	10	1
12	Mixed	77	0.009	0.96	0.98	6.85	4	10	1

Table A.13: Results for sequence 12

SeqId	Cfg	#Frames	Time	MOTA	MOTP	GOSPA	MT	FP	IDsw
13	BC	339	0.05	0.65	0.85	54.46	68	357	152
13	CA	339	0.023	0.85	0.98	31.23	68	202	10
13	CV	339	0.023	0.85	0.98	31.21	68	202	9
13	Mixed	339	0.026	0.84	0.97	32.43	68	210	14

Table A.14: Results for sequence 13

SeqId	Cfg	#Frames	Time	MOTA	MOTP	GOSPA	MT	FP	IDsw
14	BC	105	0.068	0.83	0.95	38.09	17	76	28
14	CA	105	0.033	0.9	0.97	29.24	17	58	3
14	CV	105	0.033	0.9	0.97	29.21	17	58	3
14	Mixed	105	0.048	0.89	0.97	31.66	17	63	3

Table A.15: Results for sequence 14

SeqId	Cfg	#Frames	Time	MOTA	MOTP	GOSPA	MT	FP	IDsw
15	BC	375	0.051	0.87	0.91	30.54	26	221	69
15	CA	375	0.03	0.9	0.97	29.22	26	210	13
15	CV	375	0.032	0.9	0.97	29.22	26	210	13
15	Mixed	375	0.043	0.89	0.97	30.49	26	219	13

Table A.16: Results for sequence 15

SeqId	Cfg	#Frames	Time	MOTA	MOTP	GOSPA	MT	FP	IDsw
16	BC	208	0.156	0.67	0.58	162.43	28	664	364
16	CA	208	0.092	0.93	0.98	54.59	28	225	5
16	CV	208	0.102	0.93	0.98	54.56	28	225	4
16	Mixed	208	0.114	0.88	0.94	74.26	28	309	55

Table A.17: Results for sequence 16

SeqId	Cfg	#Frames	Time	MOTA	MOTP	GOSPA	MT	FP	IDsw
17	BC	144	0.077	0.65	0.55	69.48	11	198	103
17	CA	144	0.055	0.93	0.99	21.7	11	60	0
17	CV	144	0.055	0.93	0.99	21.67	11	60	0
17	Mixed	144	0.06	0.93	0.99	21.69	11	60	0

Table A.18: Results for sequence 17

SeqId	Cfg	#Frames	Time	MOTA	MOTP	GOSPA	MT	FP	IDsw
18	BC	300	0.017	0.9	0.95	17.31	21	111	24
18	CA	300	0.005	0.77	0.87	40.45	21	269	54
18	CV	300	0.005	0.77	0.87	40.43	21	269	54
18	Mixed	300	0.019	0.87	0.95	24.11	21	157	24

Table A.19: Results for sequence 18

SeqId	Cfg	#Frames	Time	MOTA	MOTP	GOSPA	MT	FP	IDsw
19	BC	1058	0.072	0.68	0.66	84.73	106	1775	1037
19	CA	1058	0.05	0.92	0.97	33.1	106	691	14
19	CV	1058	0.056	0.92	0.98	33.03	106	690	14
19	Mixed	1058	0.063	0.91	0.97	34.6	106	724	53

Table A.20: Results for sequence 19

SeqId	Cfg	#Frames	Time	MOTA	MOTP	GOSPA	MT	FP	IDsw
20	BC	836	0.033	0.92	0.98	32.28	133	524	24
20	CA	836	0.01	0.9	0.99	40.6	133	665	15
20	CV	836	0.01	0.9	0.99	40.59	133	665	15
20	Mixed	836	0.033	0.9	0.98	40.91	133	668	19

Table A.21: Results for sequence 20

A.2 Statistics for all sequences comparing tracked class

Figure A.1: Each sequence's metrics for all sequences. The red horizontal line is the mean for that motion model.

B

Appendix - ETENN

B.1 Statistics for all sequences comparing ETENN