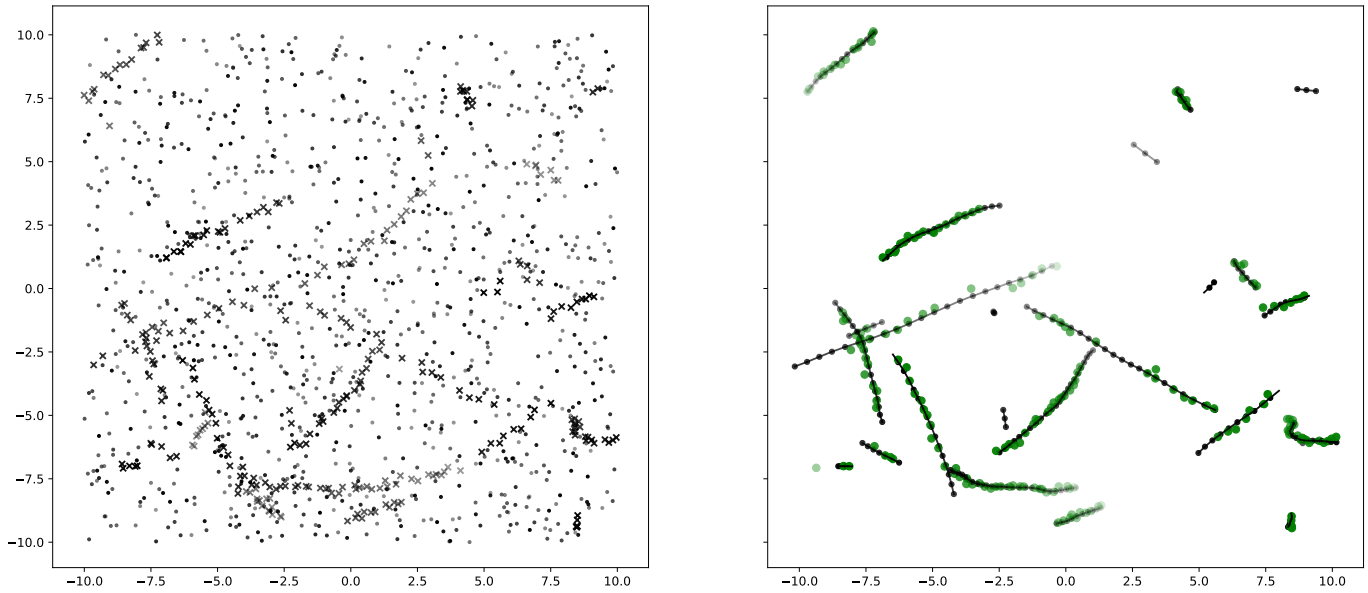

Transforming the Field of Multi-object Tracking

Master's thesis in Systems, Control and Mechatronics

GEORG HESS
WILLIAM LJUNGBERGH

Department of Electrical Engineering

CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2021
www.chalmers.se

Master's thesis 2021

Transforming the Field of Multi-object Tracking

GEORG HESS
WILLIAM LJUNGBERGH

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Electrical Engineering
Division of Signal Processing and Biomedical Engineeering
Chalmers University of Technology
Gothenburg, Sweden 2021

Transforming the Field of Multi-object Tracking
GEORG HESS
WILLIAM LJUNGBERGH

Supervisor and examiner: Lennart Svensson, Department of Electrical Engineering

Master's Thesis 2021
Department of Electrical Engineering
Division of Signal Processing and Biomedical Engineeering
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: On the left, all measurements fed to the model are shown. Here, circles represent clutter measurements and crosses represent measurements originating from objects. This scenario is generated using the point object base case described in Section 4.1.2. On the right, the ground truth trajectories (black) with predictions (green) obtained with the MOTT architecture are shown. Trajectories and predictions are faded with time. The model is applied iteratively over 55 time steps with a sliding window size of 20.

Typeset in LaTeX
Printed by Chalmers Reproservice
Gothenburg, Sweden 2021

Transforming the Field of Multi-object Tracking
Georg Hess & William Ljungbergh
Department of Electrical Engineering
Chalmers University of Technology

Abstract

One of the main challenges in making autonomous vehicles a reality is to provide them with an accurate representation of their surroundings. Multi-object tracking is a perception task enabling such a representation by tracking the states of an unknown number of objects using noisy measurements. For kinematic state tracking, state-of-the-art performance has thus far been achieved by model-based Bayesian filters which, in theory, these have the potential to provide Bayes-optimal estimates. Apart from relying on models, these methods must resort to approximations in order to remain computationally tractable in complex scenarios, thus impacting their performance.

In contrast, model-free methods based on deep learning have the potential to learn the optimal filter from data, thus providing an attractive alternative to model-based methods. However, to the best of our knowledge, such approaches have never been compared to the high-performing Bayesian filters, especially in a setting where accurate models are available. This thesis proposes new deep learning methods based on the transformer architecture to perform the multi-object tracking task. The performance of these methods is compared to state-of-the-art model-based methods, in a setting where the correct model is assumed to be given. While this gives an advantage to the model-based methods, it also allows us to train the deep learning models on an unlimited amount of data. The transformer models are shown to outperform the Bayesian filters on complex tasks, while performing on par for simpler scenarios. Thus, we display the applicability of the transformer in yet another field, and show the potential of data-driven approaches in a territory dominated by model-based approaches.

Keywords: Transformers, Deep Learning, Multi-Object Tracking, Multi-Target Tracking, Poisson Multi-Bernoulli Mixture Filter, delta-Generalized Labeled Multi-Bernoulli

Acknowledgements

First and foremost, we would like to express our sincerest gratitude to our academic supervisor Lennart Svensson. Not only did he come up with the idea that spawned this thesis, but also provided us with new avenues to explore throughout this spring. Without him, the multi-object tracking field would remain un-transformed.

Also, we would like to thank Juliano Pinto who has been instrumental throughout this thesis, both in debugging and brainstorming. Moreover, we thank Yuxuan Xia for providing the code for our model-based baselines and for all the assistance in making them work in our framework. We would also like to thank all of the people mentioned above, as well as Henk Wymeersch, for their valuable advice and ideas during our weekly meetings.

We also acknowledge that our computations were enabled by resources provided by the Swedish National Infrastructure for Computing (SNIC) at C3SE, partially funded by the Swedish Research Council through grant agreement no. 2018-05973.

Lastly, we would like to give a shoutout to our beloved friends that have had to endure our endless discussions about (to them) meaningless transformer-related issues.

Georg Hess & William Ljungbergh, Gothenburg, May 2021

Contents

List of Figures

List of Tables

Acronyms

δ-GLMB Delta-Generalized Labeled Multi-Bernoulli. xvii, 2, 5, 15, 41–44, 61

AD Autonomous Driving. 1
ADAS Advanced Driver-Assistance Systems. 1
AS Active Saftey. 1

DETR Detection Transformer. xiii, xv, xvii, 18, 19, 21, 30–33, 35, 39, 41–44, 48, 50, 53, 54
DML Deep Machine Learning. 1, 2
DNNs Deep Neural Networks. 1, 25

EMOT Extended Multi-Object Tracking. 5–7, 21, 24, 57
ESOT Extended Single-Object Tracking. 44, 57

FFN Feed Forward Network. 29, 30, 32–36, 59
FOV Field of View. I, III–VII, 14, 22, 23

GLMB Generalized Labeled Multi-Bernoulli. 15
GOSPA Generalized Optimal Sub-Pattern Assignment. xv, xvii, 5, 6, 36, 37, 40–49, 51, 53, 55–58, 61

MBM multi-Bernoulli mixture. 12–15
MOT Multi-Object Tracking. 1–6, 8–15, 21, 22, 25, 28–30, 32, 36, 39, 54, 55, 57, 58
MOTT Multi-Object Tracking Transformer. iv, xiii–xv, xvii, X, 21, 30, 32, 33, 39–44, 46–50, 52–58
MTT Multi-Target Tracking. 1

NLP Natural Language Processing. 2, 17

PMBM Poisson Multi-Bernoulli Mixture. xiii, xiv, xvii, 2, 5, 12, 13, 15, 35, 40–48, 55, 56, 61
PMOT Point Multi-Object Tracking. xv, 5, 6, 21–24, 41, 43, 48, 49, 55, 57
PPP Poisson Point Process. 9, 12–14

RFS Random Finite Set. 8, 12, 15

STRAT Single-Trajectory Transformer. xiii–xv, xvii, xviii, XI, 21, 30, 35, 36, 41, 43–49, 51, 53–58

1

Introduction

Advanced Driver-Assistance Systems (ADAS), Active Saftey (AS) systems, and Autonomous Driving (AD) all denote different techniques that allow vehicles to drive by themselves in some sense, and aim at assisting or replacing human drivers. They have the potential to make road transport more efficient and safer, both for the passengers in the vehicle and for other road users. Further, starting in 2022, some ADAS functionality will become mandatory for all new cars sold in the EU to improve safety in the automotive sector [4]. Examples of such features include advanced emergency braking, intelligent speed assistance, and lane-keeping assist.

One fundamental function in enabling vehicles to take sensible actions is having an accurate representation of the environment surrounding the vehicle, a task commonly denoted as perception. An accurate representation often covers the same information humans utilize when driving a vehicle, which includes the detection and classification of objects or semantic segmentation of the surroundings. Specific examples are the localization of other vehicles, vulnerable road users and traffic signs, or knowing which part of the road is driveable. The same way humans rely on their eyes, vehicle sense their surroundings using a suite of sensors such as cameras, lidars and radars. Based on their input, different representations can be built and used by downstream tasks to safely navigate the vehicle.

Multi-Object Tracking (MOT), or Multi-Target Tracking (MTT), is a perception task that, given noisy detections, should locate multiple objects, track their identities, and output their individual trajectories [5]. Challenges include data association, i.e., determining which detections should be associated with what object, handling occlusions, and the initialization and termination of trajectories. A robust and high-performing tracker gives accurate estimates of the number of objects in a scene as well as their states, e.g., their position and velocities. Such a tracker allows other AD features to make sound decisions and enables the prediction of future object locations.

The field of MOT can be divided into two sub-fields. The first one is concerned with tracking objects in images and video sequences by leveraging Deep Machine Learning (DML) methods. The rapid increase in available computational power has made it feasible to use Deep Neural Networks (DNNs) in computer vision, which has sparked the development of such methods for MOT. For this field, there ex-

ist common datasets and benchmarking frameworks with leaderboards [6, 7, 8, 9], where many contestants constantly push the state-of-the-art forward, see [10, 11] for examples. In contrast, the second sub-field, which is the focus of this thesis, is concerned with performing tracking using low-dimensional measurements such as position and velocity, also known as kinematic states. For this modality, the state-of-the-art is instead achieved by model-based Bayesian methods such as the Poisson Multi-Bernoulli Mixture (PMBM) filter [12] or the Delta-Generalized Labeled Multi-Bernoulli (δ-GLMB) filter [13]. For this type of tracking, potential sensor modalities include lidar and radar. Some datasets for this exist [14, 15], however it is common for these methods to evaluate their performance on custom scenarios with synthetic data.

While these model-based algorithms, in theory, can provide Bayes-optimal estimates, this is not the case in practice due to multiple reasons. Firstly, the complexity of the data association and track management makes it intractable to compute the optimal solutions [12, 13]. Therefore, these methods must resort to approximations that decrease in accuracy as the data association problem becomes increasingly complex. Secondly, they rely on models to represent reality, e.g., describing the birth, death and motion of objects or modelling sensor characteristics. For the filters to perform adequately, selecting suitable models and proper tuning of associated hyper-parameters becomes essential. This is both time-consuming and introduces modelling errors that reduce the estimates' quality even further.

In order to mitigate the issues associated with model-based approaches, this thesis explores to what extent DML methods can achieve state-of-the-art performance for kinematic state tracking. The data-driven approach has the potential to alleviate any modelling errors since any expressive neural network may learn the true data distribution, given enough data. The DML approach also removes the need for repeated model selection and parameter re-tuning when changing data distribution. So far, such methods have relied on image and video data, high-resolution lidar data, or a combination of both [16, 17, 18, 19]. All these sensor modalities usually allow objects to be detected in a single frame, reducing the data association problem to be frame-by-frame. Thus, the applicability of DML methods in a domain where long-range temporal reasoning is needed for proper data associations remains to be examined.

Specifically, in this thesis, a neural network architecture known as the transformer is utilized to perform MOT. While traditionally used for Natural Language Processing (NLP) [2] in models such as BERT [20] or GPT-3 [21], transformers have displayed great performance in other areas such as object detection [3] and more recently also for MOT in images [10, 22, 23]. Besides producing great results for many tasks, the transformer has several traits that make it a suitable candidate to solve kinematic MOT. Firstly, the architecture is tailored to be used as a set-to-set function approximation. In a MOT setting, this can be used to map a set or sequence of measurements to a set of predictions. Secondly, the transformer can model long-range dependencies in its input sequence, giving it the capabilities to reason about data associations and produce context-aware predictions. Nevertheless, the performance

of transformers for kinematic MOT has not been explored in research prior to this thesis and needs further investigations.

1.1 Problem formulation

This thesis aims to apply a neural network structure known as the transformer to solve the task of MOT. Concretely, given a sequence of noisy measurements that span over a certain time frame, the objective is to predict the state vectors of all objects currently alive. We do so by devising novel transformer architectures and modifying existing ones to suit our needs. As is customary in the field, we work with synthetic data where the underlying models are assumed to be known. This is usually done to ensure that the model-based methods can perform at their maximum level by reducing modelling errors that arise when using real-world data. However, it brings several benefits to our data-driven approach as well. The use of synthetic data removes the need for manual labelling labour and, in theory, allows for an infinite amount of training data. Further, this allows for precise control over the complexity of the data that is generated. This type of control is not possible with real-world data sets, which already are close to non-existent for kinematic MOT in general.

Subsequently, to get a complete understanding of each model's performance, they are subjected to tasks of varying complexity, and are compared with baselines consisting of state-of-the-art model-based methods. These tasks include point object tracking, which assumes that objects can only generate at most one measurement per time step, and extended object tracking, where objects can generate an arbitrary number of measurement per time step. For point object tracking, both single- and multi-object tracking tasks are considered. However, for extended object tracking, only single-object tracking is considered.

To summarize, in this thesis we aim to do the following:

- Design novel, and modify existing, transformer-based architectures to solve different types object tracking tasks. These tasks include single-, and multi-object tracking at varying levels of complexity.
- Using synthetic data, compare the performance of our transformer-based models with current state-of-the-art model-based approaches.

1.2 Limitations

The transformer-based models, as well as the model-based approaches, are exclusively evaluated on synthetic data. Finding a relevant real-world data set and training the neural networks on it, as well as tuning the baselines for said data set, is considered out of scope for this project. Furthermore, we only consider 2d-data in this thesis. The reason for this is mainly for ease of visualization. The models proposed can easily be configured to produce 3d predictions.

1.3 Contributions and main results

The main contributions of this thesis are:

- We present multiple transformer-based models, methods of how to train them for the kinematic MOT setting and display their usability in this domain.
- We compare these methods to model-based approaches and establish new state-of-the-art performance for a variety of MOT tasks.
- We adapt the contrastive learning framework to kinematic MOT and show that it boosts performance.

Further, the works of this thesis have resulted in a paper [24] presenting a comparison between model-based methods and a developed transformer model.

1.4 Thesis outline

In Chapter 2 the preliminaries needed for understanding this thesis are introduced. The concept of MOT is defined mathematically, and related performance metrics commonly used within the field are presented. Further, we provide an overview of the model-based methods later used as baselines. Next, we give a background of the transformer architecture and the intuition behind it. Lastly, one application of the transformer that has inspired the methods designed in this thesis is presented.

Next, Chapter 3 gives a detailed explanation of the methodology used in the thesis. First, the synthetic data set and the generation thereof is presented. Secondly, we present the loss functions used for training the neural networks, followed by an in-depth description of all transformer-based models designed and their motivation. Lastly, we present our evaluation framework, used to evaluate our models and our baselines.

Afterwards, the experimental evaluation is described in Chapter 4. This chapter also provides insights into the workings of the transformer models. The results are subsequently discussed and analyzed in Chapter 5. The chapter also gives potential directions of future work. The thesis concludes with Chapter 6, where the work is summarized.

2

Theory

This chapter aims to introduce the concepts, and the theory behind them, required to understand the subsequent chapters. First, a mathematical definition of the multi-object tracking problem is given. This definition is split into Point Multi-Object Tracking (PMOT) and Extended Multi-Object Tracking (EMOT). Here, we also introduce a performance metric, Generalized Optimal Sub-Pattern Assignment (GOSPA), used throughout the thesis. Secondly, we provide an overview of the existing model-based filtering approaches, which we use as baselines when evaluating our models. These methods include the current state-of-the-art Poisson Multi-Bernoulli Mixture (PMBM) filter, as well as the Delta-Generalized Labeled Multi-Bernoulli (δ-GLMB) filter. Thirdly, we present the relevant background of the transformer architecture and its variants, including the concept of self-attention and cross-attention, which are the main building blocks of the transformer encoder and decoder.

2.1 Multi-object tracking

Before diving deeper into transformers and traditional methods to solve multi-object tracking, we give a formal definition of the task itself. Further, we introduce the GOSPA metric used to compare performance between different methods.

2.1.1 Formal definition

While one may come across slight variations in the literature, in this section, we aim to introduce the definition of MOT that is used throughout this thesis. As mentioned previously, the field of MOT can be divided into two fields: tracking objects in a high-dimensional space using images and video sequences, and tracking the state of objects using low-dimensional measurements. Henceforth, whenever a reference to MOT is made, it refers to the latter of the two. Concretely, it refers to estimating the state of an unknown number of objects in the scene and maintaining their identities over time to form trajectories using low-dimensional measurements.

More specifically, at each time-instant t, a set of measurements $\mathbb{Z}^t$ is collected. This set is the union of the set of measurements originating from objects alive at the current time-instant $\mathbb{Z}^t_{obj}$ and possible clutter measurements $\mathbb{Z}^t_{clutter}$. Clutter mea-

surements, also sometimes referred to as false measurements, are measurements that do not originate from any object, but are used to model some of the shortcomings in the sensors used to obtain measurements in a real-world setting. Formally, the measurement set obtained at time t can be written as

$$\mathbb{Z}^t = \mathbb{Z}^t_{obj} \cup \mathbb{Z}^t_{clutter}. \tag{2.1}$$

It should be pointed out that it is assumed that any object has a probability $p_{\text{meas}} \in [0, 1]$ of being measured at any time step. In other words, the fact that there are objects alive at time-instant t does not guarantee that $\mathbb{Z}^t_{obj}$ contains any measurements. Furthermore, one of the differences between extended and point object tracking can be highlighted here. In PMOT it is assumed that each object may at most give rise to one detection per time-instant, while in EMOT this assumption is not made. In summary, this means that the following can be said about the cardinality of the object measurement set for the two different tracking settings

$$\begin{aligned} \text{PMOT:} \quad & 0 \leq |\mathbb{Z}^t_{obj}| \leq n \\ \text{EMOT:} \quad & 0 \leq |\mathbb{Z}^t_{obj}| < \infty \end{aligned}$$

where n is the number of objects alive at time-instant t.

The task of multi-object tracking then becomes, given access to the measurements from the current and previous time-instances $\mathbb{Z}^t, \mathbb{Z}^{t-1}, \cdots, \mathbb{Z}^0$, to estimate the unknown number of objects n and their individual states $\mathbb{X}^t = \{x^t_1, x^t_2, \cdots, x^t_n\}$ at the current time-instant. The state vector that is to be estimated varies from application to application. However, in PMOT the state vector is commonly composed of a position and a velocity. Moreover, in EMOT, the state additionally contains the spatial extent of the object.

The main challenges of MOT can coarsely be summarized as three sub-challenges. The first one is to determine the unknown number of n objects currently alive in the scene. While this is seemingly easy in a scenario where the objects are well separated and the clutter intensity is low, this sub-problem gets more challenging as the scenarios become increasingly realistic. This task can also increase in difficulty when objects give rise to fewer measurements, as these can easily be mistaken as clutter. Secondly, measurements should be associated with the object from which the measurement originated. This task is known as the data association and is arguably the most challenging part of the MOT problem. Lastly, the state x^t_i of each object $\{o_i\}^n_{i=1}$ has to be estimated using the measurements associated with that object, which is commonly denoted filtering.

2.1.2 GOSPA

In multi-object tracking, both the ground truth and estimates are commonly represented as a set of object states. Depending on the performance of the applied tracking algorithm, the cardinality of these sets might differ. Further, the task of assigning estimates to ground truths is non-trivial. In [25], the GOSPA metric was

presented, which provides a mathematically sound way of measuring the distance between two finite sets of objects often used to evaluate the performance of tracking algorithms. The metric can be expressed as an optimization over assignment sets

$$d_p^{c,2}(X,Y) = \left[\min_{\gamma\in\Gamma}\left(\sum_{(i,j)\in\gamma} d(x_i,y_j)^p + \frac{c^p}{2}\left(|X|+|Y|-2|\gamma|\right)\right)\right]^{\frac{1}{p}}. \tag{2.2}$$

Here $X = \{x_1,\ldots,x_i\}$ denotes the set of ground truth objects and $Y = \{y_1,\ldots,y_j\}$ is the set of estimates. Let γ denote an assignment set, containing pairs (i,j) of ground truths i and estimates j that are assigned to each other. Formally, an assignment set γ between the sets $\{1,\ldots,|X|\}$ and $\{1,\ldots,|Y|\}$ holds the following properties:

$$\gamma \subseteq \{1,\ldots,|X|\}\times\{1,\ldots,|Y|\}, \tag{2.3a}$$

$$(i,j),(i,j')\in\gamma \implies j=j', \tag{2.3b}$$

$$(i,j),(i',j)\in\gamma \implies i=i'. \tag{2.3c}$$

Further, the metric is minimized over all possible assignment sets Γ, meaning we search for an optimal γ^* that minimizes (2.2). Note however that the assignment set can be smaller than both the set of ground truths and estimates. In (2.2), the parameter c is referred to as cut-off distance and regulates at which distance a ground truth object should be considered to be missed and similarly when an estimate should be considered to be a false positive. The value of p can be used to regulate how much outliers are penalized, commonly chosen to be 1. For point objects, the metric $d(x_i,y_j)$ is often set to be Euclidean distance between the points x_i and y_j. Equation (2.2) facilitates the decomposition of the GOSPA metric into three parts:

- sum of state errors, localization error: $\sum_{(i,j)\in\gamma} d(x_i,y_j)^p$,
- misdetection error: $\frac{c^p}{2}\left(|X|-|\gamma|\right)$, and,
- false detection error: $\frac{c^p}{2}\left(|Y|-|\gamma|\right)$.

To clarify, in the misdetection error, the number of ground truth objects that have not been matched to any estimate are multiplied by a scalar $\frac{c^p}{2}$. Similarly, for the false detection error, the number of unmatched estimates is multiplied by the same scalar.

2.1.3 GOSPA for extended objects

When doing EMOT, several options of selecting $d(\cdot)$ in (2.2) exist, all of which must incorporate both localization and shape errors. It has been shown that for extended elliptic objects, the Gaussian Wasserstein metric is suitable to use as a distance metric [26]. However, in order to use the Gaussian Wasserstein metric, an ellipse $\mathbf{x}$ firsts has to be represented as a multivariate Gaussian distribution as

$$\mathcal{N}_{\mathbf{x}} = \mathcal{N}(m_{\mathbf{x}},\Sigma_{\mathbf{x}}), \tag{2.4}$$

where $m_{\mathbf{x}}$ specifies the center of the ellipse and $\Sigma_{\mathbf{x}}$ the extent. The extent can be parameterized as

$$\Sigma_{\mathbf{x}} = \mathbf{R}\mathbf{D}\mathbf{R}^T, \tag{2.5a}$$

$$\mathbf{R} = \begin{bmatrix} \cos\alpha & -\sin\alpha \\ \sin\alpha & \cos\alpha \end{bmatrix}, \tag{2.5b}$$

$$\mathbf{D} = \begin{bmatrix} (l^1)^2 & 0 \\ 0 & (l^2)^2 \end{bmatrix}, \tag{2.5c}$$

where $\mathbf{R}$ denotes a rotation matrix with ellipse rotation α, and $\mathbf{D}$ a diagonal matrix with the ellipse's semi-major and semi-minor axes l^1 and l^2 respectively. The squared L_2 Wasserstein distance between two multivariate Gaussians can, according to [27], be written as

$$d_{GW}(\mathcal{N}_{\mathbf{x}}, \mathcal{N}_{\hat{\mathbf{x}}})^2 = \|m_{\mathbf{x}} - m_{\hat{\mathbf{x}}}\|^2 + \mathrm{Tr}\left(\Sigma_{\mathbf{x}} + \Sigma_{\hat{\mathbf{x}}} - 2\sqrt{\sqrt{\Sigma_{\mathbf{x}}}\Sigma_{\hat{\mathbf{x}}}\sqrt{\Sigma_{\mathbf{x}}}}\right). \tag{2.6}$$

2.2 Probability distributions

Both the generation of synthetic data and the model-based methods used as baselines utilize a variety of probability distributions. This section aims at giving a brief introduction to a selection of relevant distributions.

2.2.1 Bernoulli distribution

The Bernoulli distribution is a discrete probability distribution of a random variable x that takes the value 1 with probability p and 0 with probability $q = 1 - p$. The mean of the distribution is p and its variance is $p(1-p) = pq$. The probability mass function over different outcomes k is

$$f(k; p) = \begin{cases} 1 - p & \text{if } k = 0, \\ p & \text{if } k = 1. \end{cases} \tag{2.7}$$

In a MOT setting, the Bernoulli density is commonly considered to be a Random Finite Set (RFS), i.e., a set whose cardinality is a random variable, and its set members also are random variables. This extends the expression of a Bernoulli density to

$$f(X) = \begin{cases} 1 - r & \text{if } X = \emptyset, \\ rp(x) & \text{if } X = \{x\}., \\ 0 & \text{otherwise,} \end{cases} \tag{2.8}$$

where r denotes the probability that an object x exists and $p(x)$ denotes the objects state density.

2.2.2 Binomial distribution

The binomial distribution is a discrete probability distribution modelling the number of successes when doing a sequence of n independent experiments, each with probability of success p. When doing a single trial $n = 1$, the binomial distribution is a Bernoulli distribution. For a random variable x that is binomially distributed $x \sim \mathcal{B}(n, p)$, its probability mass function is

$$f(k, n, p) = \binom{n}{k} p^k (1-p)^{n-k}, \tag{2.9}$$

where k is the number of successful trials and $\binom{n}{k} = \frac{n!}{k!(n-k)!}$.

2.2.3 Poisson distribution

The Poisson distribution is a discrete probability distribution used to model the number of occurrences of an event over a fixed interval, e.g., in time or space, given the events occur with known constant rate and independently of each other. The distribution is parameterized by $\lambda > 0$ and the probability mass function of a Poisson distributed random variable x is

$$f(k, \lambda) = \Pr(x = k) = \frac{\lambda^k e^{-\lambda}}{k!}. \tag{2.10}$$

Further, λ is both the expected value and the variance of x. To denote that a random variable follows a Poisson distribution one writes $x \sim \text{Pois}(\lambda)$.

A random process with a close connection to the Poisson distribution is the Poisson Point Process (PPP). Let λ be the intensity of a PPP. The intensity equals the expected number of events within a unit length interval, which, for example, can be an interval over time or space. Given that a random number of points within a finite interval are a PPP, then their cardinality is a Poisson distributed random variable. In a MOT context, PPP are often used to model clutter measurements.

2.2.4 Gaussian distribution

The Gaussian, or normal, distribution is a continuous probability distribution for a real-valued random variable. In one dimension, its probability density function is

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2}, \tag{2.11}$$

where μ denotes the mean, or expected value of the distribution, and σ is the standard deviation. The variance of the distribution is σ^2. When a random variable $x \in \mathbb{R}$ is normally distributed this is often written $x \sim \mathcal{N}(\mu, \sigma^2)$.

The generalization of the normal distribution to higher dimensions is commonly denoted multivariate normal distribution, or multivariate Gaussian distribution. The multivariate normal distribution of a vector $\mathbf{x} \in \mathbb{R}^k$ is given by $\mathbf{x} \sim \mathcal{N}(\mu, \mathbf{\Sigma})$, where

$\mu \in \mathbb{R}^k$ is the mean and $\mathbf{\Sigma} \in \mathbb{R}^{k \times k}$ is a positive semi-definite covariance matrix. However, its probability density function exists only if $\mathbf{\Sigma}$ is positive definite, and is then

$$f(\mathbf{x}; \mu, \mathbf{\Sigma}) = \frac{\exp\left(-\frac{1}{2}(\mathbf{x}-\mu)^T \mathbf{\Sigma}^{-1}(\mathbf{x}-\mu)\right)}{\sqrt{(2\pi)^k |\mathbf{\Sigma}|}}, \tag{2.12}$$

with $|\mathbf{\Sigma}|$ being the determinant of $\mathbf{\Sigma}$.

2.2.5 Gamma distribution

The gamma distribution is a continuous probability distribution for a random variable $x \in (0, \infty)$. It can be parameterized using two parameters, the shape parameter $k > 0$ and the scale parameter $\theta > 0$. A random variable x that is gamma-distributed is denoted $x \sim \Gamma(k, \theta)$ and has a probability density function

$$f(x; k, \theta) = \frac{x^{k-1} e^{-\frac{x}{\theta}}}{\theta^k \Gamma(k)}, \tag{2.13}$$

where $\Gamma(k)$ is the gamma function evaluated at k,

$$\Gamma(z) = \int_0^\infty t^{z-1} e^{-t} dt. \tag{2.14}$$

The gamma distribution can be used in Bayesian statistics as the conjugate prior for a Poisson density. In short, for a Poisson likelihood function $p(x|\theta)$, choosing a prior $p(\theta)$ to follow the gamma distribution ensures that the posterior $p(\theta|x)$ also follows the gamma distribution. The concept of conjugacy is used in many Bayesian filters, specifically also for the model-based approaches used as baselines in this thesis.

2.2.6 Inverse Wishart distribution

The inverse Wishart distribution is a probability distribution for real-valued positive-definite matrices. If a $p \times p$ positive definite matrix $\mathbf{X}$ follows the inverse Wishart distribution, this is denoted by $\mathbf{X} \sim \mathcal{W}^{-1}(\mathbf{\Psi}, \nu)$. Here, $\mathbf{\Psi}$ is a $p \times p$ positive definite scale matrix and $\nu > p - 1$ is a real scalar denoting the degrees of freedom. In a MOT setting it is used as the conjugate prior to a multivariate normal likelihood with unknown mean and covariance. The probability density function is

$$f(\mathbf{X}; \mathbf{\Psi}, \nu) = \frac{|\mathbf{\Psi}|^{\nu/2}}{2^{\nu p/2} \Gamma_p(\frac{\nu}{2})} |\mathbf{X}|^{-(\nu+p+1)/2} e^{-\frac{1}{2}\operatorname{Tr}(\mathbf{\Psi}\mathbf{X}^{-1})}, \tag{2.15}$$

where $|\cdot|$ denotes the determinant and $\Gamma_p(\cdot)$ the multivariate gamma function,

$$\Gamma_p(a) = \int_{S>0} \exp\left(-\operatorname{Tr}(S)\right) |S|^{a-(p+1)/2} dS. \tag{2.16}$$

2.3 Model-based approaches

As mentioned previously, the field of kinematic MOT is currently dominated by model-based approaches since accurate models are available for this problem formulation. These approaches are used as baselines for this thesis and are therefore introduced to give an understanding of how they solve the MOT task. All these methods rely on the Bayesian framework, thus, a short overview of Bayesian statistics and filtering in general is given prior to the detailed model descriptions. For an in-depth description of Bayesian filtering, the interested reader is referred to [28].

2.3.1 Bayesian state estimation

Optimal filtering refers to methods used for estimating the state of a time-varying system, observed through noisy measurements [28]. Here, optimal is meant in terms of statistical optimality. Further, in Bayesian filtering, the filtering is formulated using Bayesian statistics. In mathematical terms, the objective in filtering is to estimate a vector-valued time series $\mathbf{x}_{0:T} = \{\mathbf{x}_0, \mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_T\}$ by observing a set of noisy measurements $\mathbf{y}_{1:T} = \{\mathbf{y}_1, \mathbf{y}_2, \dots \mathbf{y}_T\}$ [28]. In a Bayesian setting this can also be formulated as trying to find the posterior density $p(\mathbf{x}_{0:T}|\mathbf{y}_{1:T})$, i.e., modelling the true states $\mathbf{x}_{0:T}$ as random variables given $\mathbf{y}_{1:T}$. This can be solved using Bayes' theorem

$$\begin{aligned} p(\mathbf{x}_{0:T}|\mathbf{y}_{1:T}) &= \frac{p(\mathbf{y}_{1:T}|\mathbf{x}_{0:T})p(\mathbf{x}_{0:T})}{p(\mathbf{y}_{1:T})} \\ &= \frac{p(\mathbf{y}_{1:T}|\mathbf{x}_{0:T})p(\mathbf{x}_{0:T})}{\int p(\mathbf{y}_{1:T}|\mathbf{x}_{0:T})p(\mathbf{x}_{0:T})d\mathbf{x}_{0:T}} \\ &\propto p(\mathbf{y}_{1:T}|\mathbf{x}_{0:T})p(\mathbf{x}_{0:T}), \end{aligned} \tag{2.17}$$

which shows that the posterior $p(\mathbf{x}_{0:T}|\mathbf{y}_{1:T})$ is proportional to the likelihood model for the measurements $p(\mathbf{y}_{1:T}|\mathbf{x}_{0:T})$ times the prior $p(\mathbf{x}_{0:T})$. The prior reflect the beliefs about $\mathbf{x}_{0:T}$ prior any observed measurements, while the likelihood models the probability of observing measurements $\mathbf{y}_{1:T}$ given $\mathbf{x}_{0:T}$.

Using (2.17) is however not computationally tractable, as the full posterior distribution would have to be recomputed each time a new measurement arrives. By limiting the class of dynamic models to probabilistic Markov sequences, the posterior of the state at time k can be found by Bayesian filtering in a recursive fashion. The recursion relies on two steps, namely the prediction and the measurement update. In the prediction step, the estimate is updated based on measurements received up until time $k-1$, giving $p(\mathbf{x}_k|\mathbf{y}_{1:k-1})$. Specifically, the Chapman-Kolmogorov equation is utilized

$$p(\mathbf{x}_k|\mathbf{y}_{1:k-1}) = \int p(\mathbf{x}_k|\mathbf{x}_{k-1})p(\mathbf{x}_{k-1}|\mathbf{y}_{1:k-1})d\mathbf{x}_{k-1}, \tag{2.18}$$

where $p(\mathbf{x}_k|\mathbf{x}_{k-1})$ denotes the transition density, also known as the motion model. Again, it is assumed that the state progresses as a Markov chain, i.e., conditioned on the current state state $\mathbf{x}_k$, the system's past and future states are independent. Further, we assume the measurements to be independent of each other.

The measurement update step then updates the estimate once a new measurement has been observed, improving the estimate. The update step is based on Bayes' theorem (2.17)

$$p(\mathbf{x}_k|\mathbf{y}_{1:k}) = \frac{p(\mathbf{y}_k|\mathbf{x}_k)p(\mathbf{x}_k|\mathbf{y}_{1:k-1})}{\int p(\mathbf{y}_k|\mathbf{x}_k)p(\mathbf{x}_k|\mathbf{y}_{1:k-1})} \propto p(\mathbf{y}_k|\mathbf{x}_k)p(\mathbf{x}_k|\mathbf{y}_{1:k-1}), \tag{2.19}$$

where the likelihood $p(\mathbf{y}_k|\mathbf{x}_k)$ is described by measurement models, and the prior $p(\mathbf{x}_k|\mathbf{y}_{1:k-1})$ was calculated in the prediction step.

In general, the Bayesian filtering equations have no closed-form solution. However, by utilizing certain types of distributions to describe $\mathbf{x}$ and by resorting to approximations, one can find closed-form solutions. The equations then yield the posterior $p(\mathbf{x}_k|\mathbf{y}_{1:k})$, from which one can extract the Bayes optimal estimate. However, depending on the approximations made, the quality of the estimates can vary. Examples of approximations can be only using linear motion and measurement models to describe non-linear processes or describing variables as approximately Gaussian distributed. One of the most famous Bayesian filters is the Kalman filter, which for instance assumes linear models with additive Gaussian noise [28].

2.3.2 Poisson multi-Bernoulli mixture filter

The Poisson Multi-Bernoulli Mixture filter was first derived in [29, 30] and later extended by [12]. However, the description and equations below are based on the explanations given in [31] unless specified otherwise. The PMBM filter is an algorithm based on Random Finite Set (RFS) theory and is used to solve the multi-object tracking problem. The filter is highly configurable, and multiple formulations of it exist, depending on the specific MOT task at hand. For instance, it is applicable for both point object [12] and extended object tracking [32]. A general overview of the steps taken by the algorithm is provided in Figure 2.1. On a high level, the algorithm consists of $\mathcal{PMBM}$ predictions, data association, $\mathcal{PMBM}$ update and reduction of hypotheses. Here $\mathcal{PMBM}$ refers to a Poisson multi-Bernoulli mixture density, which is used to model the multiple targets that are to be tracked. Before describing the building blocks in Figure 2.1, this density and why it is useful for MOT is explained.

PMBM distribution. In the PMBM filter, the set of objects $\mathbf{x}$ at any given time is separated into two disjoint subsets, the set of objects that have been detected $\mathbf{x}^d$, and the set of undetected objects $\mathbf{x}^u$ [33]. For the undetected objects a PPP density is used, while the detected objects are modelled using a multi-Bernoulli mixture (MBM). Explicitly modelling undetected objects allows the filter to handle occluded objects, e.g., a car located behind a truck and thus not visible to a sensor. The $\mathcal{PMBM}$ density at time t can be written

$$\mathcal{PMBM}_t(\mathbf{x}) = \sum_{\mathbf{x}^u \uplus \mathbf{x}^d = \mathbf{x}} \mathcal{P}_t(\mathbf{x}^u)\mathcal{MBM}_t(\mathbf{x}^d), \tag{2.20}$$

where $\uplus$ denotes the disjoint union, $\mathcal{P}$ is a PPP density and $\mathcal{MBM}$ is a multi-Bernoulli mixture density.

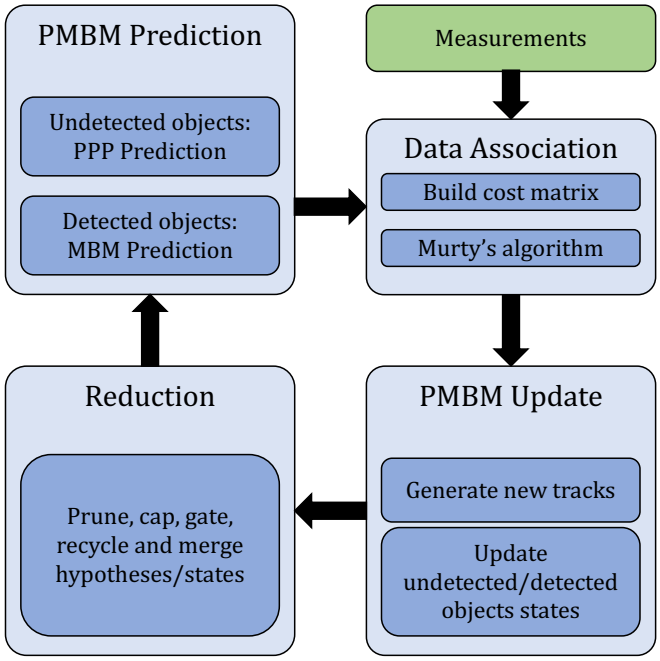

Figure 2.1: Overview of the different parts of the PMBM filter. The filter iteratively repeats these steps in the given order. Illustration inspired by [1].

The PPP density used for undetected objects $\mathbf{x}^u$ is typically modelled with a mixture intensity, meaning it is a weighted sum of densities. Let N^u be the number of mixture components, then the intensity can be written

$$\lambda^u(\mathbf{x}^u) = \sum_{i=1}^{N^u} w^{u,i} p^i(\mathbf{x}^u), \tag{2.21}$$

where $w^{u,i}$ is the weight of density $p^i(\mathbf{x}^u)$.

The multi-Bernoulli mixture density is useful for modelling detected objects in MOT since it allows the handling of multiple possible hypotheses, where each hypothesis contains an estimate for the set of current targets and their states. The density enables the PMBM filter to capture different potential data associations. Formally, the MBM density is defined as a weighted sum of multi-Bernoullis

$$\mathcal{MBM}(\mathbf{x}^d) = \sum_{h=1}^{\mathcal{H}} w^h \mathcal{MB}^h(\mathbf{x}^d), \tag{2.22}$$

where h denotes a current hypothesis and $\mathcal{H}$ is the total number of hypotheses. Further, w^h is the weight of a multi-Bernoulli component, reflecting the belief in that hypothesis. A multi-Bernoulli is a distribution consisting of multiple Bernoullis. For MOT, each Bernoulli represents a potential object with a probability of existence and state estimate as in (2.8).

Prediction. The PMBM filter follows the typical Bayesian filtering approach with a prediction step and an update step. Given a posterior distribution from the previous

timestep $\mathcal{PMBM}_{k-1|k-1}(\mathbf{x}_{k-1})$, the prediction is calculated as

$$\mathcal{PMBM}_{k|k-1}(\mathbf{x}) = \int p(\mathbf{x}_k|\mathbf{x}_{k-1})\mathcal{PMBM}_{k-1|k-1}(\mathbf{x}_{k-1})d\mathbf{x}_{k-1} \tag{2.23}$$

where $p(\mathbf{x}_k|\mathbf{x}_{k-1})$ is the transition density. This density models multiple things. Firstly, it handles the probability of an object surviving to the next timestep, e.g., setting low probability for objects leaving the FOV. Secondly, it contains a transition density describing the progression of object states, i.e., the motion model. Lastly, it has a PPP birth model to model the possibility of new objects appearing in the scene.

Concretely, the prediction updates the intensity of the PPP process density in the $\mathcal{PMBM}$ distribution and the multi-Bernoullis contained in the MBM part. For the multi-Bernoulli mixture, each multi-Bernoulli/hypothesis can be predicted independently of the other multi-Bernoullis. This is possible since no new information is added in the prediction step.

Data association. A key challenge in MOT is the unknown association between measurements and objects. For each collected measurement, there are three possible associations. One possibility is that the measurement originated from an object that is already considered to be detected. Alternatively, the measurement came from an undetected object, thus making that object detected. The final option is that the measurement did not originate from any object, i.e., it is a clutter measurement.

Given the current multi-object estimate $\mathcal{PMBM}_{k|k-1}(\mathbf{x})$, for each hypothesis, i.e., each multi-Bernoulli mixture component, a cost matrix is created to reflect the likelihood of measurements being assigned to an old object, a new object or clutter. Using a solver such as Murty's algorithm [34] the M assignments with the lowest cost are extracted and used to create new hypotheses. Selecting only M assignments is done to remove unlikely data associations and reduce computational complexity by limiting the number of hypotheses.

Update. After acquiring the potential data associations, the estimates are updated based on how the measurements are assigned. This is done according to

$$\mathcal{PMBM}_{k|k}(\mathbf{x}_k) = \frac{p(\mathbf{y}_k|\mathbf{x}_k)\mathcal{PMBM}_{k|k-1}(\mathbf{x}_k)}{\int p(\mathbf{y}_k|\mathbf{x}_k)\mathcal{PMBM}_{k|k-1}(\mathbf{x}_k)d\mathbf{x}_{k-1}}, \tag{2.24}$$

where $p(\mathbf{y}_k|\mathbf{x}_k)$ is the multi-object measurement model. Similar to the motion model in the prediction step, this contains three main components. Firstly, it models the probability for an object to be detected. Secondly, it contains a measurement model for individual object states. Lastly, it models the presence of clutter measurements as a PPP.

Hypothesis reduction. Although the data association can contain some hypothesis gating, it only limits how many new hypotheses each old hypothesis can spawn. To decrease computational complexity and run the filtering steps within a reasonable time frame, further reduction of the number of hypotheses is usually needed.

Examples of standard techniques include pruning, capping, gating, recycling and merging. Pruning removes both hypotheses with low weights from the MBM, as well as objects with low probability from the respective multi-Bernoullis. Capping can do the same but is based on a maximum number of hypotheses and objects allowed. Gating reduces the number of possible data associations by not considering assignments where the estimated object state and the measurement are far from each other. Recycling refers to moving detected objects with a low probability of existence to the set of undetected objects. Lastly, merging can be used to merge multiple hypotheses into a single one.

Estimate extraction. In order to evaluate the PMBM filter, the final step is to extract a single hypothesis used as the current estimate of object states. Multiple techniques exist, but a simple strategy is to use the multi-Bernoulli with the largest weight and extract the object state densities with existence probability above some threshold, for example, objects with existence probability above 0.5. For these densities, the final state estimate could be extracted using, for instance, the expected value or maximum a posteriori estimate, i.e., the mode of the posterior state density.

2.3.3 Delta-generalized labeled multi-Bernoulli filter

The Generalized Labeled Multi-Bernoulli (GLMB) filter was first shown in [35, 36] and later an efficient implementation was derived in [37]. It is an algorithm using the RFS framework and relying on the GLMB density for the Bayesian filtering equations to solve the MOT task. In implementation, the GLMB filtering density is expressed differently and referred to as Delta-Generalized Labeled Multi-Bernoulli (δ-GLMB). The δ-GLMB shares a common structure with the PMBM filter, and this section aims to highlight some key differences and their implications. For detailed expositions, the reader is referred to the original works [35, 36, 37], or for an implementation-oriented introduction see [38].

In [12], the connection between the PMBM and δ-GLMB filters was shown, as well as the fact that the δ-GLMB density is a labeled MBM but with less efficient parameterization. A labeled MBM can be seen as a type of MBM where each Bernoulli, besides the probability of existence and the state density, also contains a unique label. The labels are tuples specifying the birth time of an object and holding a unique identifier.

Because of this inefficient parameterization, [12] argues that the δ-GLMB filter must resort to approximations in its prediction step, something that is not needed by the PMBM filter. Further, they show that certain types of birth models can easily be handled by the PMBM filter, while needing an infinte amount of hypotheses/mixture components to be represented as a δ-GLMB density. Thus, we expect the performance of the δ-GLMB filter to be surpassed by the PMBM filter, but nevertheless it is included as a baseline for completeness.

2.4 Transformers

The transformer is a specific type of neural network architecture introduced in [2] and is based on the idea to solely utilize a mechanism called attention. The overall architecture follows an encoder-decoder structure, where the encoder maps a sequence of inputs $\mathbb{Z} = [z_0, \ldots, z_n]$ to some encoding representation $\mathbb{E} = [e_0, \ldots, e_n]$. Using $\mathbb{E}$, the decoder then produces an output sequence $\mathbb{Y} = [y_0, \ldots, y_m]$. An overview of the transformer architecture can be seen in Figure 2.2.

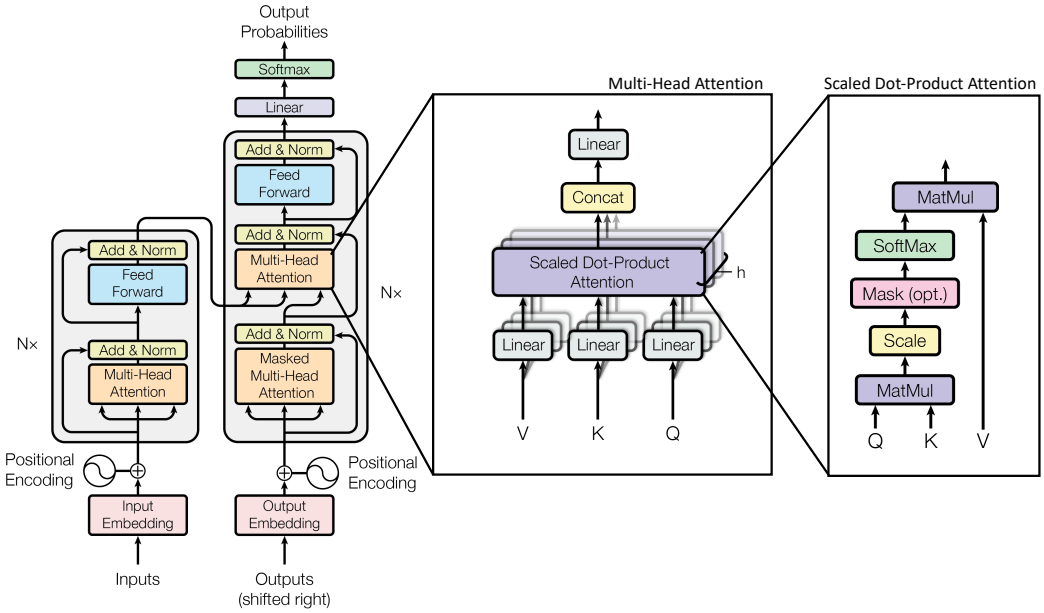

Figure 2.2: Overview of the transformer architecture. The input sequence is fed through N encoder layers, each consisting of one multi-head attention and a feed-forward network. The output sequence aggregates information from the input embeddings throughout the N decoder layers via the the second multi-head attention module. Schematics taken from the original paper [2].

2.4.1 Attention

The attention module is the core of the transformer architecture, and it is what enables the model to reason about contextual dependencies over sequences. The transformer's attention mechanism takes as input a sequence of queries, keys and values, $q, k, v \in \mathbb{R}^{l \times d_{model}}$, and outputs a sequence which is a weighted sum of the values. Here, l is the length of the input sequence and d_{model} is the hidden dimensionality of the architecture, usually set to 256. The queries, keys and values are linearly projected using unique projection matrices W_Q, W_K and W_V respectively before the attention is calculated as

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V, \tag{2.25}$$

where Q, K and V are the projected versions of the queries, keys and values respectively and d_k is the dimensionality of the keys. This is equivalent to the scaled

dot-product attention depicted in Figure 2.2. In attention, each element in the query sequence can attend to every element in the sequence of keys; this way, long-range dependencies can be modelled and allows the transformer to reason about the context in the sequence. An example of what the attention looks like in a model trained for NLP can be seen in Figure 2.3. Clearly, the model has learned some dependency in the phrase "making ... more difficult" and can reason about context.

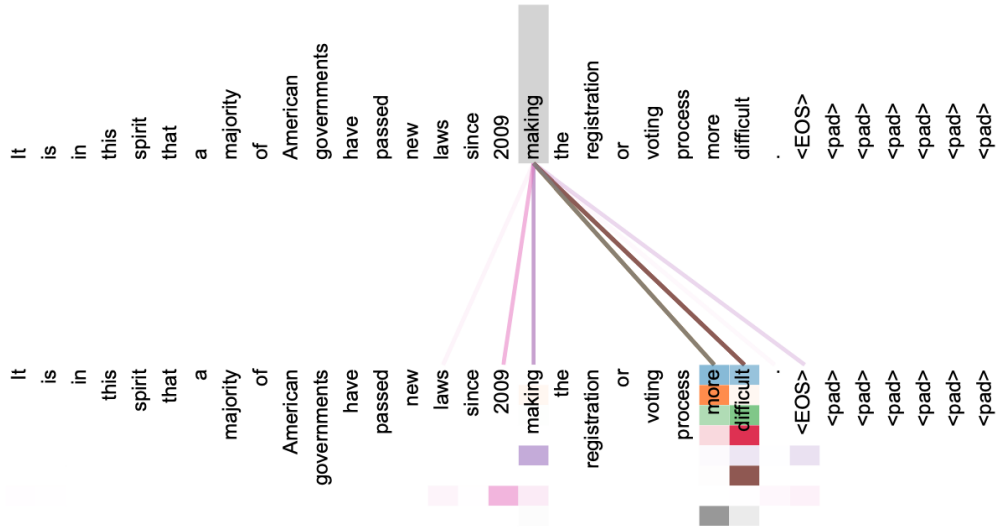

Figure 2.3: Visualization of attention map, borrowed from [2]. Line transparency reflect the amount of attention from the word "making" to all other words in the sentence. Different colors represent different heads.

2.4.2 Multi-head attention

The authors of [2] found it beneficial to, instead of using single-attention as explained above, performing attention several times in parallel, as seen in Figure 2.2. Using h heads, the multi-head attention projects the queries, keys and values h times with different, learned projections before concatenating the results and aggregating them by projecting them to suitable dimensions. The main advantage of multi-head attention is that it allows the model to reason about different types of contextual dependencies in each of the heads. Another advantage of the multi-head attention layer is that the attention in each head can be computed in parallel, allowing for efficient implementation using highly optimized matrix multiplications.

2.4.3 Encoder

The transformer encoder [2] is a permutation equivariant neural network architecture that maps a sequence of inputs $\mathbb{Z} = [z_0, \ldots, z_n]$ into a sequence of contextually aware embeddings $\mathbb{E} = [e_0, \ldots, e_n]$. The encoder itself is made up of N encoder layers where each encoder layer consist of a multi-head attention layer and feed-forward neural net and two skip connections, as shown in Figure 2.2. At the first encoder layer, the queries, keys and values are all set to be the input sequence $\mathbb{Z}$, and as we move up through the encoder layers, the queries, keys and values are set as the output of

the previous encoder layer. This means that the entire sequence can always attend to itself and is therefore referred to as self-attention.

2.4.4 Decoder

The transformer decoder [2] maps a sequence of initial output embeddings $\mathbb{O} = [o_0, \ldots, o_m]$ into a sequence of outputs $\mathbb{Y} = [y_0, \ldots, y_m]$ using the information aggregated by the encoder. Similarly to the encoder, the transformer decoder is made up of M identical decoder layers. However, each layer contains the multi-head attention, feed-forward neural network, and the skip connections we find in the encoder layer and contains yet another multi-head attention layer referred to as the cross-attention layer. In this layer, the keys and values are set to be the embeddings $\mathbb{E}$ computed by the encoder, while the queries are the propagated output embeddings from the preceding multi-head (self) attention layer.

2.4.5 Positional encoding

The transformer was initially developed for use in Natural Language Processing (NLP) tasks. In such a setting and many other sequence prediction tasks, the sequence order plays a crucial part in its meaning. For example, the sentences "I want my dog to eat" and "I want to eat my dog" have very different meanings even though the sequences are made up of identical words. However, as stated previously, the transformer is permutation equivariant, meaning that it has no notion of order and can not distinguish between the two sentences above. As this is undesirable, the authors of [2] proposed using a positional encoding that is generated based on the inputs place in the sequence. These positional encodings are then added to the input and output sequence, which allows the transformer to reason about where in the sequence this particular query is located. The authors proposed two different ways of generating these positional encodings. The first is to let the encoding be based on sine and cosine functions with different frequencies depending on the absolute position in the sequence, commonly denoted the Fourier frequencies in the literature. The other is to learn a positional embedding vector for each possible position during training. Note that the latter implies that one must set a maximum length of the input sequence, which is not a limitation when using sine and cosine positional embeddings.

2.4.6 Detection transformer

In [3], the authors provided an alteration of the original transformer architecture that enabled the model to perform object detection in images. When compared to the vanilla transformer in Figure 2.2, the Detection Transformer (DETR) architecture shown in Figure 2.4 display two major differences.

First, rather than adding the positional embedding to the input and output sequence prior to the encoder and decoder, in the DETR architecture, the positional embeddings are added to the sequence at each encoder and decoder layer. The authors showed that this substantially improved performance of their predictions. Secondly,

in the vanilla transformer, the output sequence is generated in an auto-regressive fashion. That is, it generates the output one item at a time and is fed all previously generated output items when generating the current one. Without any alterations of the model architecture, the authors of DETR introduced the concept of `object queries`. These are a small set of learned positional embeddings $\mathbb{O} \in \mathbb{R}^{n_q \times d_{model}}$ that accumulate valuable information as they are propagated through the decoder. Here, n_q is the number of positional embeddings and d_{model} is the hidden dimensionality, usually set to 256. By using object queries, the authors digress from the auto-regressive use of the decoder and instead formulate the object detection task as a set prediction problem that can be solved in one single forward pass of the model. Once propagated, each of the object queries is fed through two FFNs that predict either an object class or the `no object` class and a bounding box.

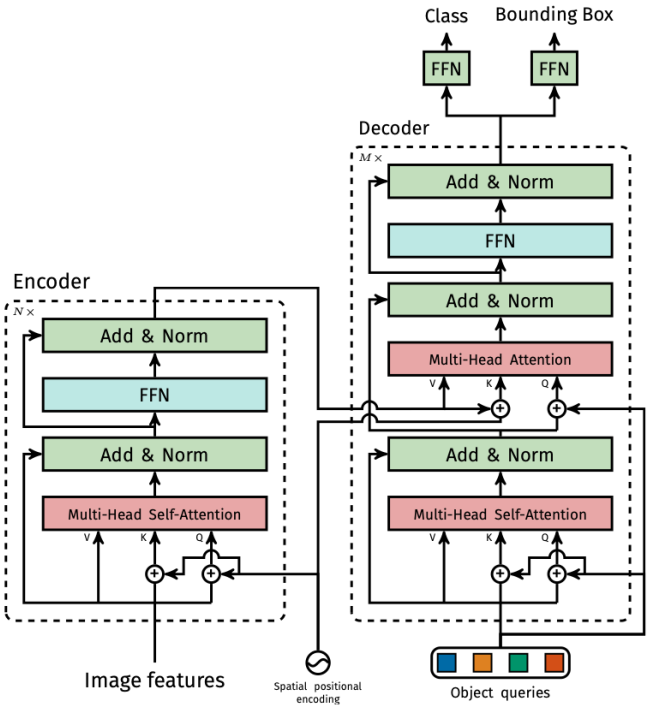

Figure 2.4: DETR original architecture, used for set predictions. Schematics taken from the original paper [3].

2.5 Contrastive learning

Contrastive learning is a common technique for representation learning, which is a task concerned with learning informative representations from data. These representations tend to improve the performance when used in downstream tasks, such as classification or detection. Contrastive learning has achieved state-of-the-art performance in unsupervised training of deep image models. However, recently methods to leverage it in a supervised fashion have been shown to be successful as well. In this thesis, it is utilized to learn sensible representations of embedded measurements, and thus a short introduction of the main concepts is given here.

Popular frameworks in the self-supervised domain include SimCLR [39, 40] and MoCo [41, 42], but there are also approaches leveraging annotated data [43]. Common for all these approaches is that they try to pull an anchor and "positive" samples towards each other in the embedding space and push the anchor away from "negative" samples. For unsupervised learning, the anchor is a sample from the minibatch. Simultaneously, a positive example often is an augmented version of the anchor, and a negative example is a randomly chosen sample from the same minibatch. In [43], they utilize existing labels and use samples from the same class as positives and samples from different classes as negatives. They formulate a contrastive loss function

$$\mathcal{L}^{cont} = \sum_{i \in I} \frac{-1}{|P(i)|} \sum_{p \in P(i)} \log \frac{\exp(\mathbf{e}_i \cdot \mathbf{e}_p / \tau)}{\sum_{a \in A(i)} \exp(\mathbf{e}_i \cdot \mathbf{e}_a / \tau)}, \tag{2.26}$$

where I is the set of indices in a minibatch, $P(i)$ is the set of indices in the minibatch that have the same label as i, and, $A(i) \equiv I \setminus \{i\}$. Further $\tau \in \mathcal{R}^+$ is a scalar parameter denoted temperature. Common choices are $\tau = 0.1$, as smaller values help training more than larger ones, but can be harder to train due to numerical instabilities [43]. Finally, $\mathbf{e}$ denotes the embeddings of the samples, with $\cdot$ being the inner dot product.

3

Methods

In this chapter, the methods used to solve the task of MOT using the transformer architecture are presented in-depth. Initially, we present the type of data used and how this data is collected. More specifically, we introduce the data-generation module created for this thesis and how the data is generated in a step-by-step fashion. Subsequently, we describe the loss functions used to train the models. This includes loss functions crucial to solving the task of MOT for both PMOT and EMOT as well as auxiliary losses to aid training. Mainly, we illustrate and explain the models we adopted and created to solve the task of MOT. This includes our adaptation of the DETR architecture and two novel model architectures: the Multi-Object Tracking Transformer (MOTT) and the Single-Trajectory Transformer (STRAT). Lastly, we present the methodology we have used to evaluate our models and our baselines. This evaluation framework utilizes the data-generation module and Monte-Carlo sampling module to obtain an average performance of each of the models and baselines.

3.1 Data generation

In many, if not all, deep learning applications, the amount of data plays a crucial role in obtaining models with good performance. This holds especially true for transformer architectures as they commonly require copious amounts of data during training [3, 20, 21] to achieve their competitive performance. Throughout this thesis, we have worked with synthetic data, partly to ensure that sufficient amounts of data were available when training our models. Furthermore, it is customary to evaluate using synthetic data in the MOT community, and it ensures that no modelling errors occur in the model-based baselines used for evaluation. That is, we created a module that can generate realistic scenarios, thus allowing us to generate an infinite amount of training data and give complete control over task complexity and provide full annotation of the data used. To achieve this, the data-generation module is highly parameterized, meaning many hyperparameters govern the complexity of the data it generates. A tabular overview of each of the hyperparameters and different hyperparameter configurations used during training and evaluation is shown in Appendix A.1. Furthermore, an overview of the data generation is shown in Figure 3.1, and the details of the module are further described below. It should be noted that the data-generator is designed to be in line with what is considered

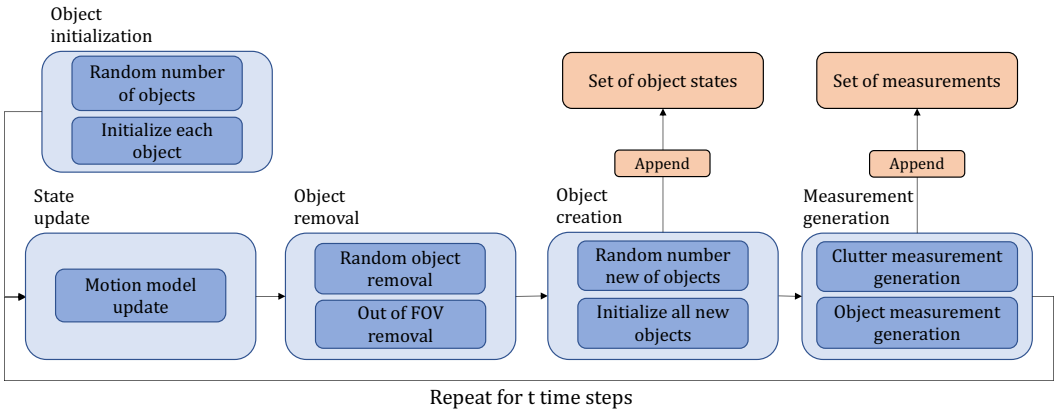

Figure 3.1: Overview of the data generation module. High-level description of how a single scenario is generated. After the object initialization, the state update, object removal, object creation and measurement generation are repeated for t time steps.

to be common practice within the model-based MOT community, meaning that all models, equations, and assumptions are commonly found in MOT literature [31]. While some design-choices are made, such as the use of a constant velocity motion model, all of these are made within the bounds of what is viewed as standard in the community.

3.1.1 Point objects

The data-generation module initializes a new scenario by sampling the number of starting objects, traditionally denoted birth objects, from a Poisson distribution

$$n_{\text{birth}} \sim \text{Pois}(\bar{n}_{\text{birth}}), \tag{3.1}$$

where $\bar{n}_{\text{birth}}$ is a hyperparameter of the data-generator. Furthermore, it also samples an initial state $\mathbf{x}_0 \in \mathbb{R}^4$ for each of the objects that are created. In PMOT the state of each object consist of a 2D position and velocity which are initially sampled from a uniform and a multivariate Gaussian distribution respectively

$$\mathbf{p}_0 \sim \mathcal{U}(\text{FOV}_{lb}, \text{FOV}_{ub}) \in \mathbb{R}^2, \tag{3.2a}$$

$$\mathbf{v}_0 \sim \mathcal{N}(\mu_{v0}, \sigma_{v0}^2 \mathbf{I}_2) \in \mathbb{R}^2 \tag{3.2b}$$

where FOV_{lb} and FOV_{ub} is the lower and upper bound of the Field of View (FOV) respectively and μ_{v0} and σ_{v0} are two hyperparameters that govern the average and standard deviation of the initial velocity. The FOV represent the area in which objects are considered to be alive and generate measurements.

After the initialization the data-generator will iteratively update the state of each object according to a motion model. Throughout this thesis a discretized constant

velocity motion model has been used

$$\mathbf{x}_k = \mathbf{I}_2 \otimes \begin{bmatrix} 1 & \Delta t \\ 0 & 1 \end{bmatrix} \mathbf{x}_{k-1} + \mathbf{q}, \tag{3.3}$$

where

$$\mathbf{q} \sim \mathcal{N}(\mathbf{0}_4, \mathbf{Q}), \tag{3.4a}$$

$$\mathbf{Q} = \mathbf{I}_2 \otimes \sigma_q^2 \begin{bmatrix} \Delta t^3/3 & \Delta t^2/2 \\ \Delta t^2/2 & \Delta t \end{bmatrix}. \tag{3.4b}$$

Here $\otimes$ is the Kronecker product, Δt is the size of the time step, and σ_q^2 is the process noise intensity, where the latter two both are tuneable hyperparameters of the data-generator.

After each state update step, objects that are outside the FOV are terminated. Furthermore, object deaths are simulated by potentially removing each object according to a Bernoulli distribution with mean p_{remove}, i.e., each object is terminated with a probability of p_{remove}.

At each time step, there is a chance that new objects are created. The number of new objects created at each time step is again sampled from a Poisson distribution, as in (3.1), but with a different hyperparameter n_{add}. Each of the new objects is distributed according to (3.2).

Subsequently, each object may give rise to a noisy measurement. The chance that a certain object yields a measurement follows a Bernoulli distribution with mean p_{measure}. If the object o is to generate a measurement $\mathbf{z}_k^o$ at time-instant k, it will be collected based on the object's true state at that time according to the following measurement model

$$\mathbf{z}_k^o \sim \begin{bmatrix} \mathbf{I}_2 & \mathbf{0}_2 \end{bmatrix} \mathbf{x}_k + \mathcal{N}(\mathbf{0}_2, \sigma_y^2 \mathbf{I}_2) \in \mathbb{R}^2, \tag{3.5}$$

where σ_y^2 is a parameter of the data-generator denoted measurement noise intensity. Note that in PMOT, each object may give rise to at most one measurement, as explained in Section 2.1.

By doing this for all objects that are currently alive we obtain a set of measurement that originate from objects $\mathbb{Z}_{\text{obj}}^k$ at the current time k. If we combine this with the set of clutter measurement $\mathbb{Z}_{\text{clutter}}^k$ we obtain the set of measurements $\mathbb{Z}^k$ at the current time step which is saved to later be used as input to our models and the baselines. The number of clutter measurements at each time step is sampled from a Poisson distribution with mean n_{clutter} and each of the measurements are sampled uniformly in the FOV

$$|\mathbb{Z}_{clutter}^k| \sim \text{Pois}(n_{\text{clutter}}), \tag{3.6a}$$

$$\mathbf{z} \sim \mathcal{U}(\text{FOV}_{lb}, \text{FOV}_{ub}), \ \forall \mathbf{z} \in \mathbb{Z}_{clutter}^k. \tag{3.6b}$$

By repeating this process iteratively for τ time steps, we can obtain a collection of measurements $\mathbb{Z} = \{\mathbb{Z}^1, \cdots, \mathbb{Z}^\tau\}$ and the corresponding object states for all objects

at all times $\mathbb{X} = \{\mathbb{X}^1, \cdots, \mathbb{X}^\tau\}$ which can be used during training and evaluation as input and ground-truths respectively.

3.1.2 Extended objects

While the general structure of the data-generator remains the same when generating data for EMOT, modifications are required to relax some of the assumptions made in PMOT. Firstly, in EMOT each object has a spatial extent included in the state vector. While there are several ways of parameterizing this extent, we chose to represent the extent as an ellipse parameterized by a 2×2 dimensional matrix. During initialization, the initial position and velocity is sampled according to (3.2), and the extent-state is sampled from the inverse-Wishart distribution as

$$\mathbf{e}_0 \sim \mathcal{W}^{-1}(\Psi, \nu) \in \mathbb{R}^{2\times 2}, \tag{3.7}$$

where both $\mathbf{e}_0$ and Ψ are 2×2 positive definite matrices and ν a real scalar. Here, both Ψ and ν are tuneable parameters of the data generation. As explained in Section 2.2.6, a positive define matrix is obtained by sampling from the inverse Wishart distribution. This allows us to interpret $\mathbf{e}_0$ as the covariance of a multivariate Gaussian, which is one way to represent ellipses, as shown in Section 2.1.3.

Note that throughout the iterative state updates, the extent state $\mathbf{e}_k \equiv \mathbf{e}_0$ of a particular object remains constant, meaning that the constant velocity motion model equation described in (3.3) still can be used to update the position and velocity for all objects. Furthermore, the termination of objects and the creation of new ones follow the same methodology as described in Section 3.1.1.

The main difference between EMOT and PMOT is the relaxation of the assumption that each object can at most give rise to one measurement per time step. To reflect this in the data-generation, each object has an average number of measurements it gives rise to. This number is sampled from the gamma distribution

$$\bar{n}^o_{\text{meas}} \sim \Gamma(\alpha, \beta) \in \mathbb{R}, \tag{3.8}$$

where α and β are denoted as the scale and rate parameter respectively and are both hyperparameters of the data-generation. Each time an object is to generate measurements, the actual number of measurements that are collected from that particular object is sampled from a Poisson distribution using the number sampled in (3.8)

$$n^o_{\text{meas}} \sim \text{Pois}(\bar{n}^o_{\text{meas}}), \tag{3.9}$$

Note that the number of measurements one object yields n^o_{meas} can vary with each time step while the average number $\bar{n}^o_{\text{meas}}$ is constant over all time steps for one object.

Since each object now has a spatial extent, the measurement model described in 3.5 has to be modified slightly to reflect this. While the measurement will still be centered around the object's true position, the covariance of the sample has to

change in order to account for the spatial extent. This is done by modifying the covariance matrix as follows

$$\Sigma = \sigma_h^2 \mathbf{I}_2 \mathbf{e}_0 + \sigma_y^2 \mathbf{I}_2, \tag{3.10}$$

where σ_h is a parameter of the data-generator denoted as the multiplicative noise parameter which regulates the spread of measurements. By setting $\sigma_h = 1/2$, it has been shown that this approximates uniform sampling of measurements across the object's extent [33]. This leads to the measurements being collected as

$$\mathbf{z}_k^o \sim \begin{bmatrix} \mathbf{I}_2 & \mathbf{0}_2 \end{bmatrix} \mathbf{x}_k + \mathcal{N}(\mathbf{0}_2, \Sigma) \in \mathbb{R}^2. \tag{3.11}$$

The set of measurements generated by object o at time step k can thus be written as

$$\mathbf{z}_k^o = \{\mathbf{z}_{k,i}^o\}_{i=1}^{n_{\text{meas}}^o}, \tag{3.12}$$

where n_{meas}^o is given by (3.9).

3.2 Primary loss functions

For DNNs to learn how to solve a task, it is crucial to provide them with suitable loss functions. Depending on the model architecture, the type of predictions and the task at hand, many potential loss functions are available. Note that the distinction between a primary and an auxiliary loss function is that a primary loss functions' main objective is to teach the model to perform the task of MOT. In contrast, an auxiliary loss supports the model to learn helpful information that can be used to solve the task. Throughout the project, a variety of primary loss functions have been utilized, which are introduced below.

3.2.1 Matching

Many of the created architectures infer a fixed-size set of N predictions, where N is larger than the number of ground truth targets. This poses a challenge, as one has to score these large number of predictions with respect to the ground truth. Inspired by [3], we use an optimal bipartite matching between predictions and ground truth and then optimize object-specific losses.

Let y denote the set of ground truth targets, and $\hat{y} = \{\hat{y}_i\}_{i=1}^N$ the set of N predictions. Assuming N is larger than the number of ground truth targets, y can be padded with instances of the no object class $\varnothing$, to be of equal size as $\hat{y}$. The bipartite matching between the two sets is found by searching for the permutation of N elements $\sigma \in \mathfrak{S}_N$ that minimizes

$$\hat{\sigma} = \arg\min_{\sigma \in \mathfrak{S}_N} \sum_i^N \mathcal{L}_{\text{match}}(y_i, \hat{y}_{\sigma(i)}), \tag{3.13}$$

where $\mathcal{L}_{\text{match}}(y_i, \hat{y}_{\sigma(i)})$ denotes a pair-wise *matching cost* between ground truth target y_i and the prediction with index $\sigma(i)$. The optimal permutation is found by using the

Hungarian algorithm, which is an optimization algorithm for assignment problems introduced in [44]. Depending on the downstream loss function, different matching costs can be utilized, and these are further described below.

One issue with this approach is that, while the object specific loss is differentiable, the matching itself is not. This is due to the discrete nature of the assignment problem, where algorithms to find the optimal assignment, such as the Hungarian algorithm, contain non-differentiable operations. The object specific loss applied based on the matching trains the network to improve the predictions, but does not convey any direct information to the neural network how the matching was done. Approaches to make the assignment in a differentiable manner exist and have shown to improve final performance, e.g., [45, 46], but are considered out-of-scope for this thesis.

3.2.2 Point object L1-loss

For point objects, the ground truth labels can be written as $y_i = \langle p_i, x_i \rangle$ where $p_i \in \{0, 1\}$ is the probability of existence (with $p_i = 0$ for $\varnothing$ objects) and x_i is the object state vector containing the position of the object. The prediction with index $\sigma(i)$ consists of an existence probability $\hat{p}_{\sigma(i)} \in [0, 1]$ and a predicted state vector $\hat{x}_{\sigma(i)}$. The matching cost is written as

$$\mathcal{L}_{\text{match}}(y_i, \hat{y}_{\sigma(i)}) = \|x_i - \hat{x}_{\sigma(i)}\|_1 - \hat{p}_{\sigma(i)}, \tag{3.14}$$

where $\|\cdot\|_1$ is the l_1 loss. This matching cost is used in (3.13) to obtain an optimal assignment $\hat{\sigma}$. Given the optimal matching, we can formulate a loss function for all matched pairs. Again inspired by [3], the point object L1-loss consists of a weighted binary cross-entropy loss for the existence probability and a l_1 loss for the state regression

$$\mathcal{L}_{\text{point}}(y, \hat{y}) = \sum_{i=1}^{N} \left[-\left(\omega p_i \log \hat{p}_{\hat{\sigma}(i)} + (1 - p_i) \log \hat{p}_{\hat{\sigma}(i)} \right) + p_i \|x_i - \hat{x}_{\hat{\sigma}(i)}\|_1 \right], \tag{3.15}$$

where ω is a weight term to account for class imbalance between true targets and padded $\varnothing$ targets. It is calculated as number of padded targets divided by number of true targets. Effectively, the loss function encourages the model to push predicted existence probabilities towards 1 if the prediction is matched with a true object, and zero if it is matched with an $\varnothing$ object. Similarly, for predictions matched with true objects, the loss is minimized by pushing predicted states towards true target states.

3.2.3 Gaussian-Wasserstein loss

For extended objects, the ground truth labels are written in the same way as for point objects, namely $y_i = \langle p_i, x_i \rangle$ where $p_i \in \{0, 1\}$ is the probability of existence (with $p_i = 0$ for $\varnothing$ objects) and x_i is the object state vector. However, here the state vector contains both the position and the extent of the object. The prediction with index $\sigma(i)$ consists of an existence probability $\hat{p}_{\sigma(i)} \in [0, 1]$ and a predicted state vector $\hat{x}_{\sigma(i)}$, again containing both position and extent. Assuming the object extent

is elliptic, it can be parameterized as a Gaussian, as described in Section 2.1.2. The position of the object is then interpreted as the Gaussian mean, while the remaining states describe the shape and thus the Gaussian covariance. The matching cost is written as

$$\mathcal{L}_{\text{match}}(y_i, \hat{y}_{\sigma(i)}) = d_{\text{GW}}(x_i, \hat{x}_{\sigma(i)}) - \hat{p}_{\sigma(i)}, \tag{3.16}$$

with d_{GW} being the Gaussian Wasserstein distance defined in (2.6). The matching cost is very similar to the one used for point objects in (3.14), with the only difference that the distance between ground truth and prediction is measured using the Gaussian Wasserstein metric instead of the l_1 norm. Now, using this matching cost in (3.13) the optimal assignment $\hat{\sigma}$ can be found. For this optimal matching, the loss function is defined as

$$\mathcal{L}_{\text{GW}}(y, \hat{y}) = \sum_{i=1}^{N} \left[-\left(\omega p_i \log \hat{p}_{\hat{\sigma}(i)} + (1 - p_i) \log \hat{p}_{\hat{\sigma}(i)} \right) + p_i d_{\text{GW}}(x_i - \hat{x}_{\hat{\sigma}(i)}) \right]. \tag{3.17}$$

Again, this loss is very similar to its point object counterpart, consisting of a binary cross-entropy loss and a state regression loss.

3.2.4 Extended object L1-loss

As for the previous sections, the ground truth labels are written as $y_i = \langle p_i, x_i \rangle$ where $p_i \in \{0, 1\}$ is the probability of existence (with $p_i = 0$ for $\varnothing$ objects) and x_i is the object state vector. While the object extent is still elliptic, the state vector is instead interpreted to consist of a position, ellipse semi-major and semi-minor axis, as well as ellipse rotation. The cost matrix and loss are formed in the same way as for regular point objects, i.e.,

$$\mathcal{L}_{\text{match}}(y_i, \hat{y}_{\sigma(i)}) = \|x_i - \hat{x}_{\sigma(i)}\|_1 - \hat{p}_{\sigma(i)}, \tag{3.18}$$

and

$$\mathcal{L}_{\text{extended}}(y, \hat{y}) = \sum_{i=1}^{N} \left[-\left(\omega p_i \log \hat{p}_{\hat{\sigma}(i)} + (1 - p_i) \log \hat{p}_{\hat{\sigma}(i)} \right) + p_i \|x_i - \hat{x}_{\hat{\sigma}(i)}\|_1 \right]. \tag{3.19}$$

While [26] argues the Gaussian Wasserstein metric to be the most suitable measure of distance between ellipses, the Wasserstein distance contains several matrix square roots, which can give rise to implementation and numerical issues, and thus can make it harder to compute reliably. The computation follows the Newton-Schulz implementation described in [47] and because of the approximations it introduces we chose to include the extended L1-loss to act as a base for comparison in subsequent evaluations.

3.2.5 Trajectory loss

While the trajectory loss is not novel in any way, minor modifications have to be made to account for loss over entire trajectories. Similar to previous sections, the

ground truth labels are written as $y_i^t = \langle p_i^t, x_i^t \rangle$ where $p_i^t \in \{0, 1\}$ is the probability of existence (with $p_i^t = 0$ for $\varnothing$ objects) and x_i^t is the object state vector. The predictions are also composed of an existence probability and a state vector $\hat{y}_i^t = \langle \hat{p}_i^t, \hat{x}_i^t \rangle$. Note that the added superscript t indicates what time step the ground truth label corresponds to, while the subscript i refers to which object it corresponds to as in previous sections.

This loss is trivial to use in the presence of only one object. In such a scenario, the prediction at each time step is simply matched with the ground truth corresponding to the same time step and then use any of the loss functions defined in (3.15), (3.17) or (3.19). However, in order to utilize this loss in the presence of multiple objects, one has to find a permutation $\hat{\sigma}$ that maps ground truth trajectories to predicted trajectories by minimizing a matching loss $\mathcal{L}_{\text{match}}^{traj}$. Unlike previous loss functions, the trajectory matching loss function has to take all time steps into account, meaning that the matching loss between a ground-truth and a prediction can be calculated as

$$\mathcal{L}_{\text{match}}^{traj}(y_i, \hat{y}_i) = \sum_{k=t-\tau}^{t} \mathcal{L}_{\text{match}}(y_i^k, \hat{y}_i^k), \tag{3.20}$$

where $\mathcal{L}_{\text{match}}(y_i^k, \hat{y}_i^k)$ could be any of the matching losses defined in (3.14), (3.16) or (3.18) above depending on the use-case.

Once the optimal permutation $\hat{\sigma}$ has been obtained the loss can be calculated similarly to previous sections, with the addition of a summation over all time steps

$$\mathcal{L}_{\text{traj}}(y, \hat{y}) = \sum_{i=1}^{N} \sum_{k=t-\tau}^{t} \mathcal{L}(y_i^k, \hat{y}_{\hat{\sigma}(i)}^k), \tag{3.21}$$

where $\mathcal{L}(\cdot)$ can be any of the loss functions defined in (3.15), (3.17) or (3.19).

3.3 Auxiliary loss functions

Training on auxiliary tasks often helps the training process and the generalization performance of the final model. It does so by leveraging information helpful for the task present in the inputs but not directly used by the primary loss chosen for training [48]. In MOT, this could be whether a measurement is clutter or not, or if it comes from the same object as another measurement. Although such information is helpful for solving the task, it is not directly used by the losses in (3.15), (3.17) or (3.19). Below we introduce two auxiliary loss functions that can be used to speed up and improve training.

3.3.1 Adjustment loss

As we have access to the complete annotation of the data during training, it is reasonably easy to create auxiliary loss functions to support training. One example of such an auxiliary loss created in this thesis is the adjustment loss function $\mathcal{L}_a$. Intuitively, this loss aims to adjust each measurement towards the ground truth

state from which it was sampled, thus actively trying to remove the effects of the measurement noise simulated during the data generation.

In more detail, let $\mathbf{z}$ be a measurement sampled from an objects' ground truth state $\mathbf{x}$ such that

$$\mathbf{z} \sim \mathcal{N}(\mathbf{x}, \sigma_y^2), \tag{3.22}$$

where σ_y^2 is the measurement noise intensity. Moreover, let $\hat{\delta}$ be an adjustment vector computed by feeding the measurement embedding $\mathbf{e}$ into a FFN. The adjustment loss then aims to minimize the L1-norm between the ground truth state and the sum of the adjustment vector and the measurement

$$\mathcal{L}'_a(\mathbf{e}, \mathbf{z}, \mathbf{x}) = \|(\mathrm{FFN}(\mathbf{e}) + \mathbf{z}) - \mathbf{x}\|_1. \tag{3.23}$$

The full adjustment loss is formed over an entire embedding sequence $\mathbb{E} = [\mathbf{e}_1, \cdots, \mathbf{e}_l]$ of length l, a measurement sequence $\mathbb{Z} = [\mathbf{z}_1, \cdots, \mathbf{z}_l]$ and the corresponding ground truth states $\mathbb{X} = [\mathbf{x}_1, \cdots, \mathbf{x}_l]$

$$\mathcal{L}_a = \sum_{i=1}^{l} \mathcal{L}'_a(\mathbf{e}_i, \mathbf{z}_i, \mathbf{x}_i). \tag{3.24}$$

Note that in the presence of clutter measurement in the measurement sequence, no corresponding ground truth state exist for the items labeled as clutter. The loss handles this by simply ignoring the items in the embedding and measurement sequence labeled as clutter. This is possible due to the complete annotation obtained from the data generation.

3.3.2 Contrastive loss

In order to steer the model into quickly learning how to perform the task of MOT, we use an idea inspired by Supervised Contrastive Learning [43] to create the contrastive loss $\mathcal{L}_c$, with the aid of object labels for each of the measurements. The idea is to encourage the encoder to generate embeddings of measurements that are similar for measurements coming from the same object but dissimilar for measurements coming from different objects. Clutter measurements are treated as coming all from the same dummy object (and therefore, their encoded representations should be similar to each other but dissimilar to the representations of all other measurements).

Concretely, for a given sequence of measurements $\mathbf{z}_{1:n}$, let b_i be the object from which measurement $\mathbf{z}_i$ came from, $i \in \mathbb{N}_n$ with $\mathbb{N}_n$ being the natural numbers ranging from 1 to n. We define $\mathbb{P}_i = \{j \in \mathbb{N}_n \mid j \neq i \ , \ b_i = b_j\}$, that is, $\mathbb{P}_i$ is the set of indices of measurements that came from the same object as $\mathbf{z}_i$. The auxiliary loss $\mathcal{L}_c$ is then defined as:

$$\mathcal{L}_c(\mathbf{u}_{1:n}, \mathbb{P}_{1:n}) = \sum_{i=1}^{n} \frac{-1}{|\mathbb{P}_i|} \sum_{i^+ \in \mathbb{P}_i} \log \frac{e^{\mathbf{u}_i^\top \mathbf{u}_{i^+}}}{\sum\limits_{j \in \mathbb{N}_n \setminus \{i\}} e^{\mathbf{u}_i^\top \mathbf{u}_j}} \tag{3.25}$$

where, $\mathbf{u}_{1:n}$ is a sequence of vectors computed by processing each $\mathbf{e}_i, i \in \mathbb{N}_n$ (embeddings of $\mathbf{z}_{1:n}$ computed by the encoder) with an FFN layer and normalizing to unit

length,

$$\mathbf{u}_i = \frac{\text{FFN}(\mathbf{e}_i)}{\|\text{FFN}(\mathbf{e}_i)\|_2}. \tag{3.26}$$

Minimizing this loss can be understood as making $\mathbf{u}_i^\top \mathbf{u}_j$ large when $\mathbf{z}_i$ and $\mathbf{z}_j$ are from the same object, but small when they are from different objects. Training on the sum of this auxiliary loss and the primary loss accelerated learning and improved the final performance of the trained models in all tasks considered in Chapter 4.

3.4 Transformer models

To solve the task of MOT, several different model architectures have been devised. This section aims to give an in-depth explanation of the models and their respective pipelines. Initially, we will present our adaptation of the DETR [3] architecture to the MOT problem. Subsequently, we will introduce the Multi-Object Tracking Transformer (MOTT), a novel model architecture utilizing the concepts of iterative refinement and scene-based object queries. Finally, we will introduce the Single-Trajectory Transformer (STRAT), another novel model architecture that tries to distinguish between true and false measurements to reduce the computational complexity of the model.

In the coming sections, all models take as input a measurement sequence $\mathbb{Z}$ and produce a prediction set $\mathbb{Y}$. While the contents of the prediction set vary between the models, the definition of the input remains the same. Here, a measurement sequence $\mathbb{Z}$ is a set of low dimensional measurements (containing position measurements) collected in a sliding window approach. That is, each measurement sequence is compromised of several independent measurement sets collected at different times

$$\mathbb{Z} = \{\mathbb{Z}^{t-\tau} \cup \cdots \cup \mathbb{Z}^t\}, \tag{3.27}$$

where $\mathbb{Z}^t$ is the set of measurement collected at the current time, and τ is the sliding window size.

3.4.1 DETR

In Figure 3.2 our adaptation of the DETR architecture to the MOT problem is shown. The low-dimensional measurement sequence is initially projected into a high-dimensional representation using a learnable linear projection layer

$$\mathcal{F} : \mathbb{R}^{l \times 2} \to \mathbb{R}^{l \times d_{model}}, \tag{3.28}$$

where d_{model} is a tuneable hyperparameter of the model architecture, commonly set to 256, and l is the length of the measurement sequence. It should be noted that each measurement is projected independently of each other. We hypothesize that using high dimensional embeddings gives the model more flexibility as each embedding can store more complex representations compared to when using low dimensional

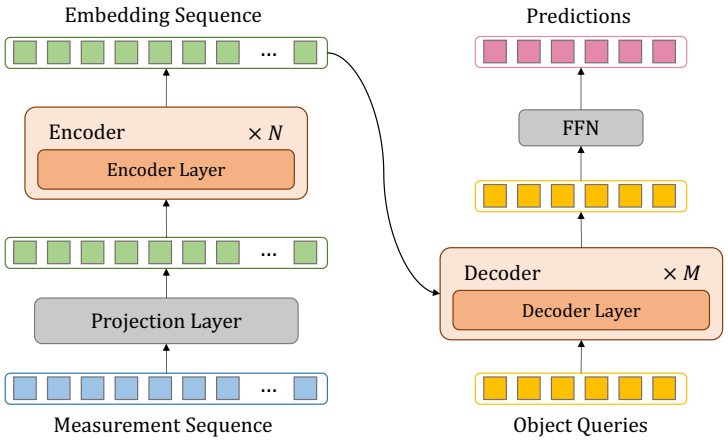

Figure 3.2: Schematic overview of our adaptation of the DETR architecture.

equivalents, thus enabling the model to reason about contextual dependencies in a more sophisticated fashion.

The projected sequence is then propagated through a DETR encoder as explained in Section 2.4.6, meaning that at each layer, a positional encoding is added to the sequence. However, as the location of each measurement in the sequence is irrelevant (as the measurement sets are unordered), we semantically change the positional encodings to what we call `temporal encodings`. Instead of adding an embedding depending on where in the sequence each item is located, we add an embedding based on at what time the corresponding measurement was sampled. The temporal encodings can be generated in different ways, as explained in Section 2.4.5, and here they are chosen to be learned vectors. This means that for each time step in the sliding window, a unique embedding is learned during training to help the model distinguish between measurements from different time steps.

Once propagated through the encoder, the initial low-dimensional measurement sequence $\mathbb{Z} \in \mathbb{R}^{l \times 2}$ has been transformed in to a high-dimensional contextually aware embedding sequence $\mathbb{E} \in \mathbb{R}^{l \times d_{model}}$. The embedding sequence $\mathbb{E}$ is used in the subsequent decoder, but can also be used to form an auxiliary contrastive loss objective, as explained in Section 3.3.2.

A sequence of n_q learned object queries $\mathbb{O} \in \mathbb{R}^{n_q \times d_{model}}$, see Section 2.4.6 for a full explanation, is fed to a DETR decoder. These aggregate information from the embedding sequence $\mathbb{E}$ as they are propagated throughout the decoder via the cross-attention module. Note that, while the number of queries n_q is small relative the length of the embedding sequence l it is assumed that the number of queries are always larger or equal to the number of objects alive in the scene n_{max} such that $n_{max} \leq n_q$. In practice, this can be enforced by setting n_q to the maximum number of ground-truths (potentially with an additional safety margin to account for differences during inference) found in any of the examples in the dataset or, as in our case, never generate data that has more than n_q objects alive at any time.

Finally, a state vector $\hat{x}^t \in \mathbb{R}^n$ and an existence probability $\hat{p}^t \in [0, 1]$ is predicted for each of the object queries by feeding them through two separate FFNs. Here, the dimensionality n of the predictions is set to match the dimensionality of the ground truths, which differs between point and extended object tracking. The existence probability indicate whether or not the associated state vector is valid or not, where $\hat{p} = 1$ indicates that the object exists. Thus, we obtain the set of predictions

$$\mathbb{Y} = \{\langle \hat{x}_i^t, \hat{p}_i^t \rangle\}_{i=1}^{n_q}, \tag{3.29}$$

which can be used to train the model using the loss functions described in Section 3.2 but also perform inference by thresholding $\hat{p}$ and considering all object that are above said threshold to be valid. Note that the superscript t indicates that the estimations are of the state vector at the final time step.

3.4.2 MOTT

A schematic overview of our novel model architecture, the Multi-Object Tracking Transformer (MOTT), is shown in Figure 3.3. The architecture is an extension of the DETR architecture, explained in Section 3.4.1, and introduce the concepts of iterative refinement and `scene-based object queries`. These concepts are inspired by the work in [49] and modified to suit the MOT setting.

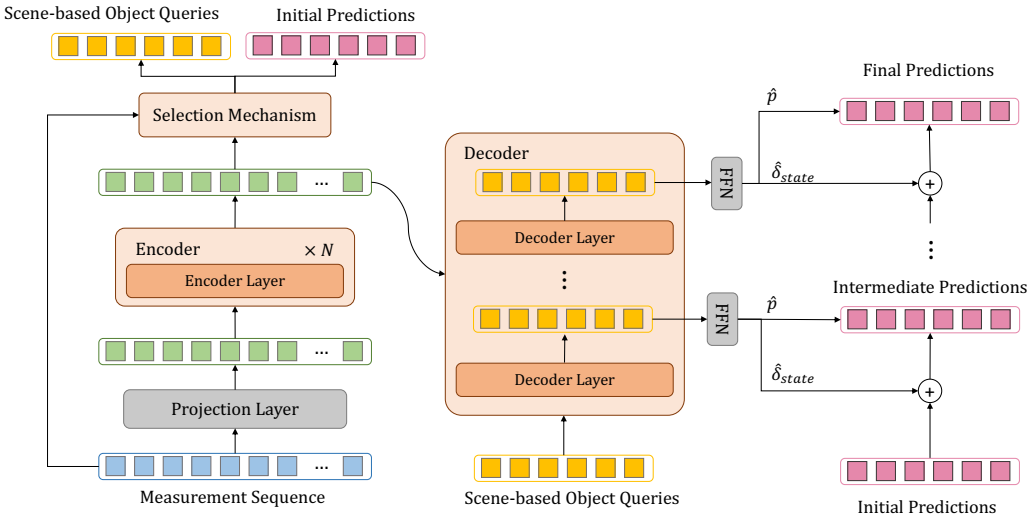

Figure 3.3: Schematic overview of the MOTT architecture. Note that following the selection mechanism, the scene-based object queries and initial predictions are used by the decoder.

The concept of scene-based object queries is fairly intuitive. In the DETR architecture, the n_q learned object queries have to be general enough to act as good initial embeddings for all possible scenarios. As the scenarios can vary quite extensively, this can be considered an inherent limitation of the architecture. This can be remedied by, instead of letting the object queries be learned, using a subset of the embedding sequence $\mathbb{E}$ computed by the encoder as object queries. By doing so, the

initial embeddings fed to the decoder depend on the specific scene, hence the name scene-based object queries.

The intuition behind the iterative refinement is that it is advantageous to iteratively update and refine the current state estimate rather than just giving a single estimation at the end of the decoder. By doing so, the transformer can adjust its estimates as it learns new information throughout the decoder. However, to be able to iteratively refine an estimate, an initial estimate must be provided. While this could be done in several ways, it seems natural to base an initial estimate from a measurement, meaning that the iterative refinement will aim to correct a raw measurement towards the ground truth state from which it was sampled.

As stated previously, the MOTT is an extension of the DETR architecture. The encoder part of the MOTT architecture follows the same structure as the DETR model. That is, a measurement sequence $\mathbb{Z} \in \mathbb{R}^{l \times 2}$ is fed to a projection layer (3.28) which is subsequently processed by the encoder to produce an embedding sequence $\mathbb{E} \in \mathbb{R}^{l \times d_{model}}$. Like for the DETR model, this sequence is later used in the decoder but can also be used to form an auxiliary contrastive loss objective, as explained in Section 3.3.2.

The selection mechanism is then fed the embedding sequence $\mathbb{E}$ and the original measurement sequence $\mathbb{Z}$ and produces the scene-based object queries $\mathbb{O} \in \mathbb{R}^{n_q \times d_{model}}$ and initial predictions $\mathbb{Y}_0 \in \mathbb{R}^{n_q \times 2}$ as shown in Figure 3.4. Here, the embedding sequence is passed to two prediction FFNs, where one produces an existence probability $\hat{p}$ and the other a refinement state vector $\hat{\delta}_{state}$ for each of the embeddings in the embedding sequence $\mathbb{E}$.

The sequence of existence probabilities $\mathbb{P} = \{\hat{p}_i\}_{i=1}^{l}$ is passed to a top-k operator, which simply returns the indices $\mathbb{I}$ of the k highest values in $\mathbb{P}$. To clarify, $\mathbb{I} = [i_1, \cdots, i_k]$ contains k elements, where i_1, i_2, i_3 etc. represent where in the sequence $\mathbb{P}$ the largest, second largest, third largest etc. values of $\mathbb{P}$ are located. The value of k is set to the number of object queries that are desired $k = n_q$. This process can be understood as selecting elements with the highest probability of existence as initial guesses for the decoder. Additionally, the refinement state vector sequence $\Delta_{state} = \{\delta_{state,i}\}_{i=1}^{l}$ is added to the measurement sequence $\mathbb{Z}$ giving the refined measurement sequence $\mathbb{Z}'$

$$\mathbb{Z}' = \{\mathbf{z}_i + \hat{\delta}_{state,i}\}_{i=1}^{l}, \tag{3.30}$$

where l again is the length of both the measurement sequence $\mathbb{Z}$ and the refinement state vector sequence Δ_{state}. Using the indices $\mathbb{I}$ obtained from the top-k operator, the scene-based object queries is then selected to be a subset of the embedding sequence $\mathbb{O} \subseteq \mathbb{E}$, and the initial predictions are selected to be a subset of the refined measurement vector sequence $\mathbb{Y}^0 \subseteq \mathbb{Z}'$.

While the structure of the decoder remains the same as in the DETR architecture, the way it is used is slightly different. Rather than simply making the predictions after the object queries have been propagated through all the decoder layers, as in the DETR architecture, here we employ the iterative refinement process after each

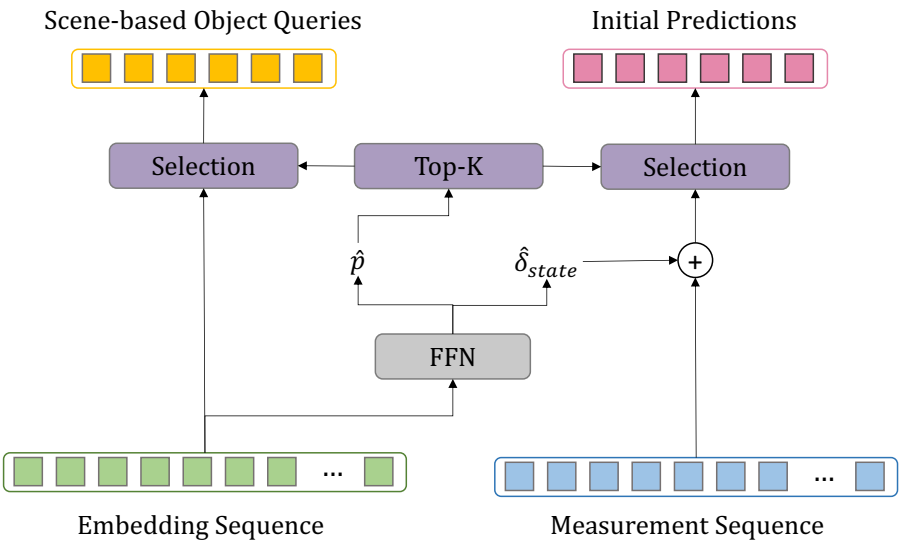

Figure 3.4: Schematic representation of the selection mechanism. Based on the predicted $\hat{p}$, the top-k operator finds the indices of the k largest elements. The corresponding embeddings and predictions and used as scene-based object queries and initial predictions, respectively.

of the layers. This entails that the scene-based object queries $\mathbb{O}^j$ at level j are fed through two prediction FFNs producing both an existence probability $\hat{p}^t$ and a refinement state vector $\hat{\delta}^t_{state}$ for each of the queries. To produce the intermediate prediction set $\mathbb{Y}^j$ the state estimate from the previous layer $\mathbb{Y}^{j-1}$ is summed with the current refinement state vector

$$\mathbb{Y}^j = \{\langle \hat{x}_i^{t,j}, \hat{p}_i^{t,j} \rangle\}_{i=1}^{n_q}, \tag{3.31}$$

where

$$\hat{x}_i^{t,j} = \hat{x}_i^{t,j-1} + \hat{\delta}_{state}^{t,j}. \tag{3.32}$$

Note here that the state estimation of each query $\hat{x}_i^t$ is updated iteratively while the existence probability $\hat{p}_i^t$ is computed independently at each layer. Furthermore, the FFNs at each layer do not share weights, meaning that they each have their set own unique weights that are learned during training.

Applying this process throughout the decoder yields $M+1$ prediction sets $\mathbb{Y}^0, \mathbb{Y}^1, \cdots, \mathbb{Y}^M$. Here $\mathbb{Y}^0$ is the prediction set produced by the selection mechanism, $\mathbb{Y}^{1,\cdots,M-1}$ are the intermediate prediction sets and $\mathbb{Y}^M$ is the final prediction set. Each of these can be trained in a supervised manner using the loss functions described in Section 3.2.

3.4.3 STRAT

The computational complexity of the transformers attention module is $O(n^2)$ w.r.t. the sequence length. This poses a problem as the sequence length increase substan-

tially when moving from point object to extended object tracking as each object can give rise to several measurements per time step. While it is fairly similar to the DETR architecture, our novel STRAT architecture described below was developed with this in mind. The architecture tries to decrease the sequence length by removing unnecessary items in the sequence, thus reducing the computational complexity. This can be compared to the gating performed in many filtering approaches, e.g., PMBM, to maintain a tractable computational load. While the architecture could, in theory, be generalized to work with multiple objects, the explanation below solely consider single object tracking. In Figure 3.5 a schematic overview of the proposed STRAT architecture is shown.

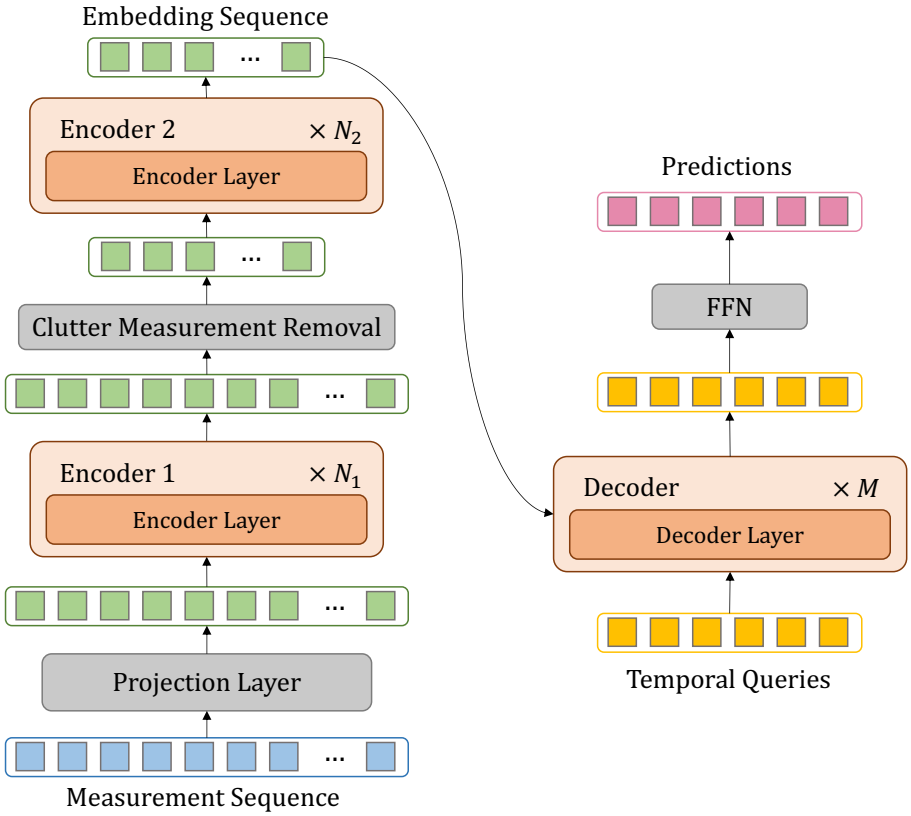

Figure 3.5: Schematic overview of the STRAT model. Following the first encoder, embedded measurements are discarded if they are predicted to be clutter, thereby reducing the sequence length.

Precisely as previous architectures, the STRAT architecture initially projects the measurement sequence $\mathbb{Z} \in \mathbb{R}^{l \times 2}$ into a higher dimension which is then fed through an encoder with N_1 layers to produce an embedding sequence $\mathbb{E} \in \mathbb{R}^{l \times d_{model}}$. The embedding sequence is then reduced by removing items classified to be clutter measurements. In the clutter measurement removal, each embedding is classified using a simple FFN producing a $\hat{p}_{clutter} \in [0, 1]$. As we have access to the full annotation obtained during the data generation, we know which measurements are clutter, thus this FFN can be trained using a simple binary cross-entropy loss function. All embeddings that get a clutter classification higher than a tuneable threshold

$$\hat{p}_{clutter} \geq p_{threshold} \tag{3.33}$$

are removed from $\mathbb{E}$, which leaves the reduced embedding sequence $\mathbb{E}' \in \mathbb{R}^{l' \times d_{model}}$, where $l' \leq l$. The reduced embedding sequence is subsequently propagated through a second encoder with N_2 layers before passed to the decoder.

In contrast to previous architectures, the STRAT model tries to estimate the object state throughout the sliding window (the object's trajectory) rather than only at the final time step. To do this, we again semantically change object queries to what we call `temporal queries`. Here, each temporal query is to aggregate information relevant to estimating the object's state at the corresponding time step. To clarify, the τ temporal queries are responsible for predicting the state of the object at each time step $x^{t-\tau}, \cdots, x^t$ in the sliding window. Note that we make a semantic distinction between the temporal queries used by the decoder and the additive temporal encodings fed to each layer of the encoder to highlight the novelty of the temporal queries. However, the temporal encodings used by the encoder are, in practice, the same feature vectors fed to the decoder as temporal queries.

Once the temporal queries are propagated through the decoder, an existence probability $\hat{p}$ and a state vector $\hat{x}$ is predicted for each of the queries using two separate FFNs. By doing so, we form the prediction set

$$\mathbb{Y} = \{\langle \hat{x}^i, \hat{p}^i \rangle\}_{i=t-\tau}^{t} \tag{3.34}$$

which contains one state vector and an existence probability for each of the time steps in the sliding window. Note that the method is applicable only to single object tracking and that the object id subscript is therefore left out. This prediction set can be used together with the ground truth trajectory of the object as well as the trajectory loss defined in 3.2.5 to train the model.

As mentioned previously, the architecture could, in theory, be extended to handle multiple objects by combining the concept of object and temporal queries into `trajectory queries`. Here each of the $\tau \times n_q$ trajectory queries would be associated with a specific object, determined by the object query, and a certain time step, determined by the temporal query. This would yield trajectory estimations for a maximum of n_q objects, but this line of research has not been explored during this thesis.

3.5 Evaluation

As in any development process, it is necessary to assess the performance of the developed method and compare them to what currently exists. To do this, we utilize the GOSPA metric, explained in Section 2.1.2, as our performance metric and compare the score obtained by our architectures to the score achieved by our baselines. These baselines are the model-based approaches explained in Section 2.3, which are currently considered to be state-of-the-art in MOT. The baseline performance is computed using code developed prior to this thesis[1]. During evaluation, the baselines are given an advantage as they are always provided with the correct models.

[1]We humbly thank Yuxuan Xia for the code and his assistance in making it work with our data.

While modelling errors are one of the drawbacks of model-based methods, by not introducing any modelling errors during evaluation we ensure a reliable comparison between our models and the state-of-the-art baselines.

All models created output a prediction set $\mathbb{Y}$, that is made up by items containing a state vector $\hat{x}$ and an existence probability $\hat{p}$. However, the prediction set may contain invalid objects, and to extract the object that are considered to be valid in $\mathbb{Y}$ a threshold on $\hat{p}$ must be applied. We can obtain the set of valid predictions as follows

$$\mathbb{Y}^{\text{valid}} = \{x_i | x_i \in \mathbb{Y}, \hat{p}_i > p_{\text{threshold}}\}, \tag{3.35}$$

where $p_{\text{threshold}}$ is a tuneable hyperparameter of the architecture, commonly set to $p_{\text{threshold}} = 0.9$. Now, the set of valid predictions $\mathbb{Y}^{\text{valid}}$, containing only predicted state vectors, can be used together with the set of ground truth object state vectors $\mathbb{X}$ to compute a GOSPA score and decomposition.

To obtain a statistically significant average performance measure, we perform a Monte-Carlo simulation and measure the average performance for all samples. Concretely, we generate a test-set containing $n = 1000$ evaluation examples using the data generator. While the data generation uses the same parameters as during training, we utilize random seeds to ensure that the data is previously unseen by the architectures. This test-set is then processed by our model and the baselines, yielding an average GOSPA score and decomposition for all evaluation examples, which gives an adequate basis for comparison.

3.6 Training and implementational details

The architectures were all developed in Pytorch. We make use of an ADAM optimizer with $\beta_1 = 0.9$ and $\beta_2 = 0.999$. We initially used a learning rate of $l_r = 0.0005$ and used a learning rate scheduler that reduced the learning rate with a factor $\alpha = 0.25$ every time the total loss did not reduce for $n = 20000$ gradient decent steps. The architectures were trained on NVIDIA V100 GPUs, and training times ranged from 24 to 72 hours per architecture, depending on the task complexity. During evaluation a cutoff distance of $c = 2$ and the order $p = 1$ was used to compute the performance in terms of GOSPA for all tasks.

4

Results

In this chapter, the transformer-based models are evaluated in the MOT task, and they are compared to state-of-the-art model-based approaches in terms of performance. The results are presented for tasks with increasing complexity to display a variety of comparisons. First, point object tracking with a single target is considered. Next, point object tracking with multiple targets is presented. Further, this is done for two different sets of data generation hyperparameters to adjust complexity within the same task structure. Subsequently, the results for extended object tracking with a single target are presented, again with two different sets of data generation hyperparameters. Finally, results displaying the inner workings of the transformer-based models are shown.

4.1 Point object tracking

For the point object tracking, three different tasks were considered. In the first task, the number of ground truth objects was limited to one, i.e., it is a single object tracking task. In the second task, multiple objects are in the scene. The third task is also considering multi-object tracking but with increased task complexity.

4.1.1 Point single-object tracking

For the point object tracking with a single object, all three transformer-based models were trained and evaluated. For all three models, the predictions in the decoder were trained using the point object L1-loss described in Section 3.2.2. This loss was also used to train the predictions from the encoder for the MOTT model. For the DETR and MOTT models, the sequence of embedded measurements produced by the encoder is further used during training by serving as input to the contrastive loss described in Section 3.3.2, which is added to the total loss.

One training example is used to visualize the task complexity and is shown in Figure 4.1. The complete configuration of the data generation hyperparameters for this task can be found in Appendix A.2.1. As we can see in the figure, the object follows a relatively linear motion over this timeframe, and the measurements are slightly noisy observations of the true object position. One should note that the object is not necessarily measured each time step but that a measurement is only collected

with a probability of 0.9. Furthermore, the scene is rather cluttered, with an average of 20 clutter measurements per time step. The main challenge for the models in this task is to discriminate between measurements from the object and clutter measurements and subsequently estimate the object's position based on estimated data associations.

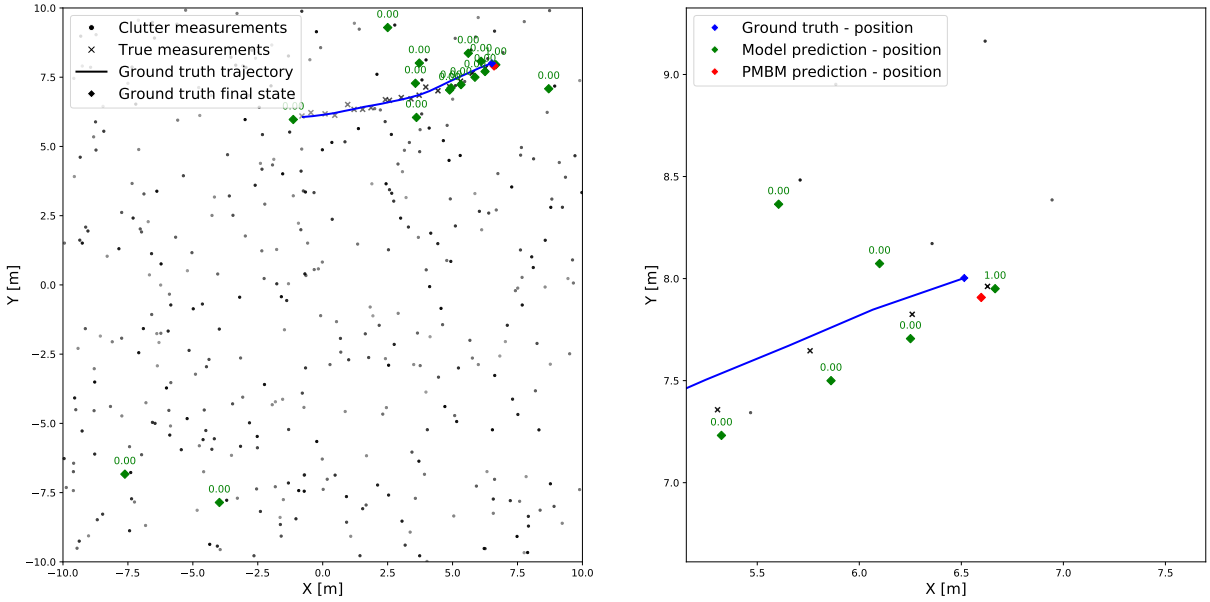

Figure 4.1: Example from the point object tracking task with a single target. The measurements are visualized as circles or crosses, depending on whether or not they are clutter measurements. The measurement's opacity reflects at which time they are collected, where darker equals most recent measurements. The object's trajectory is shown as a solid blue line, and the position at the final step is shown with a diamond. The MOTT predictions, along with their existence probabilities, are shown in green, while the PMBM estimates are shown in red.

The predictions, along with their existence probabilities, obtained from the MOTT architecture are also shown in Figure 4.1 together with the PMBM estimates. As seen in the figure, the MOTT model produces a fixed number of predictions and which ones are valid can be determined using the existence probabilities. The GOSPA score for this particular case was 0.1263 for the MOTT model and 0.1600 for the PMBM baseline, which can be compared to the average performance in Table 4.1.

For the performance evaluation, 1000 new unseen examples were generated using the same data generation hyperparameters. All models were then evaluated by computing their GOSPA scores for their predictions at the last time step. Their average scores for these 1000 examples and the average values of the decomposition are shown in Table 4.1.

As is evident from the GOSPA scores in Table 4.1, the PMBM filter achieves the lowest average scores across the evaluation data set. With a slightly higher GOSPA

Table 4.1: GOSPA score and decomposition for the DETR, MOTT and STRAT models, as well as the model-based methods PMBM and δ-GLMB for the point object tracking with single target. Lower scores equal higher performance.

Method	GOSPA	Localization	Misdetection	False detection
DETR	0.6834	0.6834	**0.0**	**0.0**
MOTT	0.1413	0.1363	**0.0**	0.005
STRAT	0.4462	0.4462	**0.0**	**0.0**
PMBM	**0.1286**	0.0936	0.035	**0.0**
δ-GLMB	0.1944	**0.0884**	0.09	0.016

score, the MOTT model is the best transformer-based architecture for this task and second-best in total. Inspecting the decomposition, it is evident that the transformer-based models all have a low count of misdetections and false detections compared to the PMBM and δ-GLMB filter. The main thing separating them in performance is their ability to localize the target correctly. In contrast, the model-based approaches achieve the lowest localization errors among the evaluated methods.

4.1.2 Point multi-object tracking - base case

For the first PMOT task, similar hyperparameters as for the single object equivalent were used, and a complete list can be found in Appendix A.2.2. While the parameters governing the motion of the individual targets and the measurement model are kept the same, the number of average starting objects and the chance for objects to appear and disappear have been adjusted. Specifically, the scene starts with four objects on average, and each time step $n \sim \text{Pois}(0.4)$ new objects appear. Further, each object has a probability of 0.05 to disappear between time steps. An example of one scene generated using these hyperparameters is shown in Figure 4.2 along with the predictions and existence probabilities obtained from the MOTT model and the estimates from the PMBM baseline. The GOSPA score for this scene is 1.2866 for the MOTT model and 1.2824 for the PMBM filter, which is close to the average performance shown in Table 4.2. While not many parameters changed compared to the single target version, this task is much harder since the models now have to estimate the number of objects alive and solve a larger data association problem.

For this task the DETR and MOTT model were trained and evaluated. As mentioned in Section 3.4.3, the STRAT architecture is not designed to handle multiple objects and is thus omitted here. Both transformer models were trained in the same fashion as in the single target task, namely using the L1-loss and the contrastive loss.

Again, 1000 new samples were generated to evaluate the model performance. The average GOSPA scores and decomposition can be seen in Table 4.2. For this task, the lowest average GOSPA score is achieved by the MOTT model, followed by the PMBM filter. While the MOTT model has a worse localization score, it outperforms

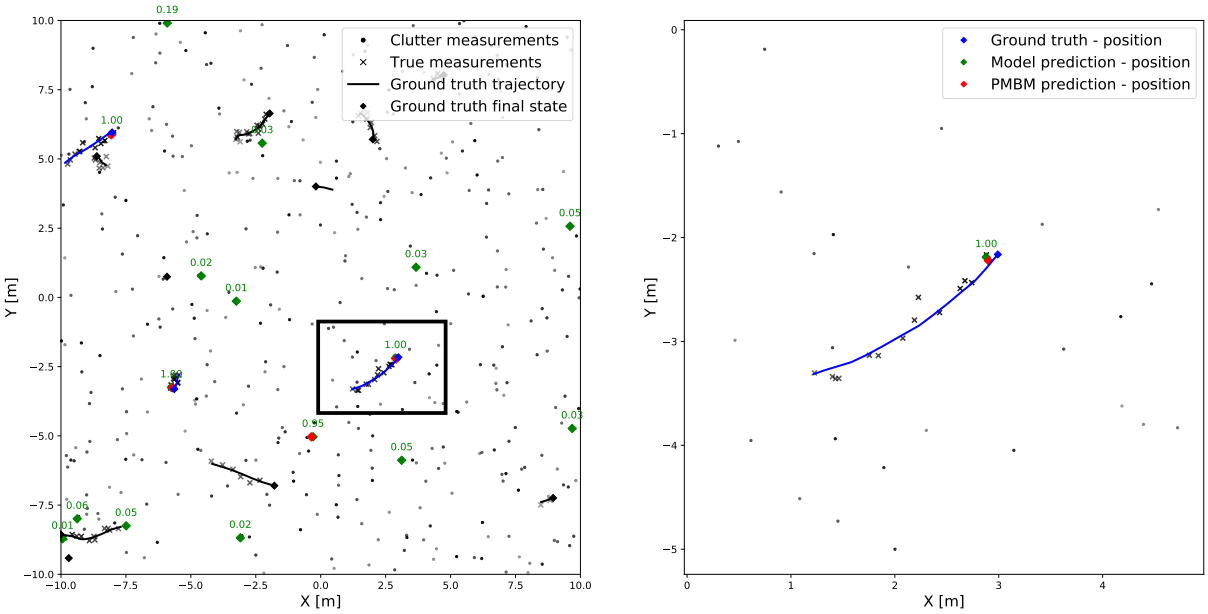

Figure 4.2: Point object tracking task with multiple targets. The measurements are visualized as circles or crosses depending on whether or not they are clutter detections. The measurement opacity reflects at which time they are collected where darker equals the most recent measurements. The trajectory of each object is shown as a solid line, and the state at the final time step is shown as a diamond. Here, blue lines correspond to objects alive at the final time step, while black ones represent objects that died in advance. The prediction and existence probabilities obtained from the MOTT model are shown in green, while the PMBM estimates are shown in red.

the PMBM filter in terms of misdetections and false detections. Even the DETR model, which has the worst overall GOSPA score, performs well in terms of misdetections and false detections when compared to the model-based methods. Finally, it is worth observing that the GOSPA scores for all models are higher than for the single target tracking task in Table 4.1, confirming the increase in task complexity.

Table 4.2: GOSPA score and decomposition for DETR and MOTT, as well as the model-based methods PMBM and δ-GLMB for the point object tracking with multiple targets, base case. Lower scores equal higher performance.

Method	GOSPA	Localization	Misdetection	False detection
DETR	3.475	2.820	0.530	0.125
MOTT	**1.118**	0.569	**0.459**	**0.09**
PMBM	1.2724	0.4474	0.6830	0.1420
δ-GLMB	1.9167	**0.0989**	1.2190	0.3030

4.1.3 Point multi-object tracking - high complexity

In comparison to the first PMOT task, the second task, defined by the hyperparameters in Appendix A.2.3, is considerably more complex. Here, the average number of objects present in the scene and the number of clutter detections per time step are increased. Furthermore, the process and measurement noise is increased while the probability that an object gives rise to a measurement is decreased. A training example with these hyperparameters is shown in Figure 4.3. The figure also shows the predictions and existence probabilities obtained from the MOTT model, and the PMBM estimates. The GOSPA score for this particular scene is 3.8890 for the MOTT model and 4.4991 for the PMBM baseline.

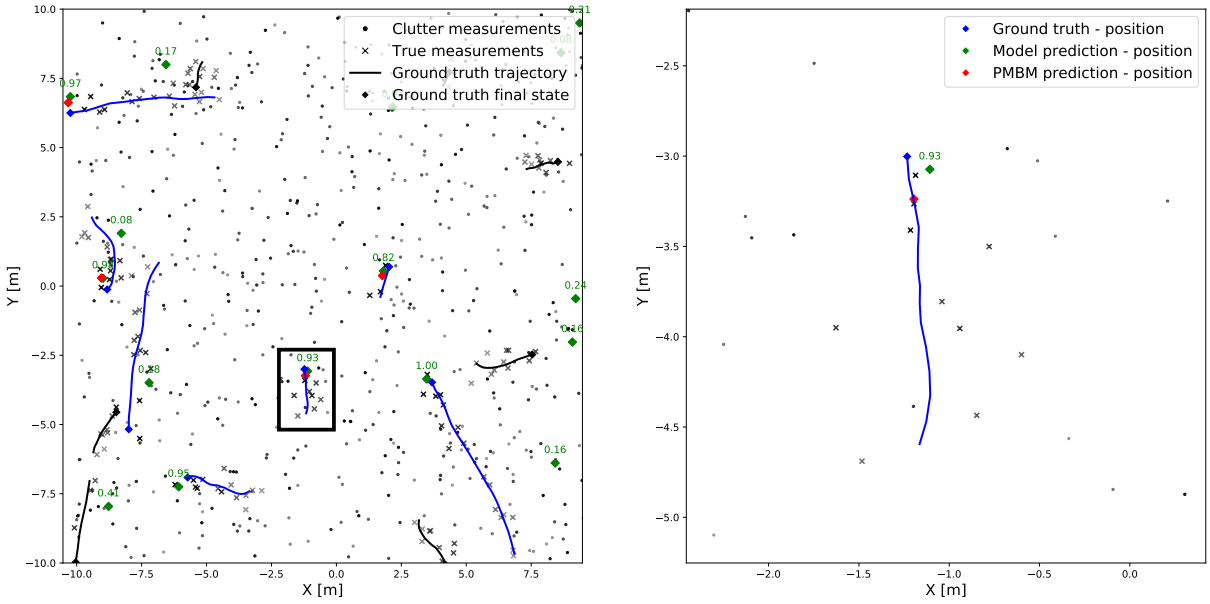

Figure 4.3: Point object tracking task with multiple targets. The measurements are visualized as circles or crosses depending on whether or not they are clutter detections. The measurement opacity reflects at which time they are collected where darker equals the most recent measurements. The trajectory of each object is shown as a solid line, and the state at the final time step is shown as a diamond. Here, blue lines correspond to objects alive at the final time step, while black ones represent objects that died in advance. The prediction and existence probabilities obtained from the MOTT model are shown in green, while the PMBM estimates are shown in red.

Just as for the base case, the DETR and MOTT architectures were trained while the STRAT model was omitted due to its inapplicability to scenes with multiple objects. Both models were trained using the L1-loss and the contrastive loss. The average performance of the trained models as well as the PMBM and δ-GLMB baselines is shown in Table 4.3.

As we can see from the table, the MOTT model outperforms all other methods with a significant margin. Similar to the results on previous tasks, it achieves a low score by having a low misdetection and false detection score. In terms of localization, it is

Table 4.3: GOSPA score and decomposition for DETR and MOTT, as well as the model-based methods PMBM and δ-GLMB for the point object tracking with multiple targets, high complexity. Lower scores equal higher performance.

Method	GOSPA	Localization	Misdetection	False detection
DETR	4.716	1.960	2.384	0.372
MOTT	**3.619**	1.231	**2.075**	0.313
PMBM	4.2239	0.7279	3.3510	**0.1450**
δ-GLMB	4.6396	**0.2891**	3.6860	0.3140

worse than the PMBM and δ-GLMB filters; however, these suffer from high scores in misdetections.

4.2 Extended object tracking

The evaluation of the extended object tracking functionality is split into two tasks with different complexity. Both tasks are concerned with only a single object, i.e., performing Extended Single-Object Tracking (ESOT). However, they differ in several hyperparameters as explained separately for each task below. The models, and the PMBM baseline, were compared using the GOSPA metric for extended objects as explained in Section 2.1.3. The δ-GLMB filter was not evaluated for these tasks since it was repeatedly outperformed by the PMBM filter for all point object tracking tasks, and a similar pattern was expected for the extended object tracking. The GOSPA score was calculated using the state estimates at the final time step, as the MOTT architecture cannot produce trajectory estimations.

Exactly as in the sections above, the MOTT model for extended objects was trained using the contrastive auxiliary loss $\mathcal{L}_c$ described in Section 3.3.2. However, in contrast to previous sections, it employs the Gaussian-Wasserstein loss $\mathcal{L}_{GW}$, described in Section 3.2.3 as its primary loss. The model is trained separately for each task.

Two separate STRAT architectures were trained and evaluated for each task. Both of them employ a binary cross-entropy loss function $\mathcal{L}_{bce}$ to train the clutter classification layer and both also utilize the adjustment loss $\mathcal{L}_a$, described in Section 3.3.1, as an auxiliary loss function. The only thing that separates the two models is the fact that one was trained using the Gaussian-Wasserstein loss $\mathcal{L}_{GW}$, described in Section 3.2.3, as its primary loss function. In contrast, the other model used the extended object L1-loss $\mathcal{L}_{\text{extended}}$ described in Section 3.19. As described in Section 3.2.3, while the Gaussian-Wasserstein distance is used to compute the GOSPA scores for extended objects, it involves operations that could lead to numerical instabilities in a deep learning setting. Thus its performance is compared to the more naïve approach of using the extended object L1-loss.

4.2.1 Extended single-object tracking - base case

The base case is defined by the hyperparameter configuration found in Appendix A.3.1. In this task, the object is always measured, the measurement and process noise are low, and almost no clutter detections are present. One scenario using this hyperparameter configuration is shown in Figure 4.4. As evident from the figure, the pattern of motion is fairly linear, and the measurements originating from the object are collected in close proximity to the actual position of the object. The distinction between clutter and true measurements is relatively easy, meaning that the main challenge is to estimate the state using the measurements. In addition, the figure shows the ground truth as a blue ellipse and the estimates obtained using the STRAT architecture and the PMBM baseline as green and red ellipses, respectively. The GOSPA score for this particular scenario was 0.0082 for the STRAT model and 0.0069 for the PMBM baseline, which can be compared with the average performance shown in Table 4.4.

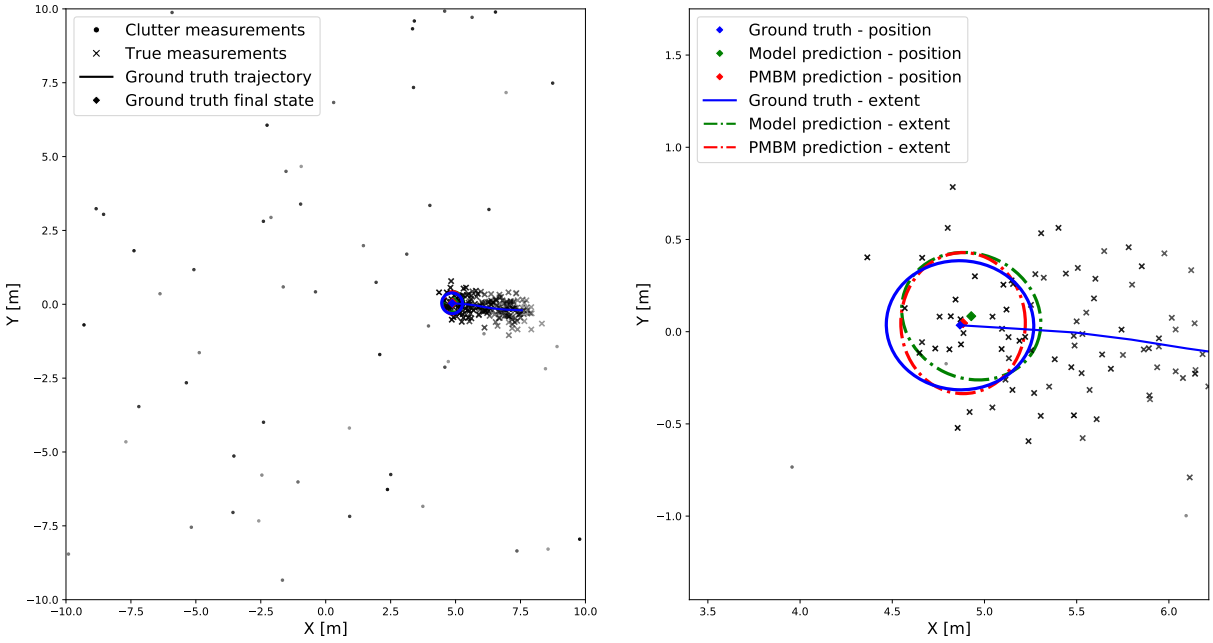

Figure 4.4: Extended object tracking task with a single target. The measurements are visualized as circles or crosses depending on whether or not they are clutter detections. The measurement opacity reflects at which time they are collected where darker equals the most recent measurements. The trajectory of the object is shown as a solid line. The object's extent is constant over the entire sequence and visualized as a blue ellipse at the final time step. Furthermore, the estimates obtained using the top-performing STRAT model and the PMBM filter are shown as green and red ellipses, respectively.

For each of the models and the PMBM baseline, a performance metric was obtained by computing the average GOSPA score over 1000 previously unseen examples. The GOSPA score was computed using the predictions/estimates at the final time step, and the examples were generated using the same hyperparameter configuration as

during training. The average performance for each model and baseline is shown in Table 4.4.

Table 4.4: Average GOSPA score and decomposition for the MOTT model, the two STRAT models as well as the PMBM filter for the extended object tracking with single target on the base case. Lower scores equal higher performance.

Method	GOSPA	Localization	Misdetection	False detection
MOTT	0.0385	0.0365	0.0010	0.0010
STRAT (L1)	0.04629	0.04629	**0.0**	**0.0**
STRAT (GW)	0.0098	0.0098	**0.0**	**0.0**
PMBM	**0.0069**	**0.0069**	**0.0**	**0.0**

As seen in the table, on this simple task, where the data associations are apparent, the PMBM baseline outperforms all of our transformer-based models. Moreover, the model trained with the Gaussian-Wasserstein primary loss achieves much better performance than the model trained using the extended L1-loss. Lastly, the MOTT architecture performs better than the STRAT architecture trained with the extended L1-loss but worse than the one trained with the Gaussian-Wasserstein.

4.2.2 Extended single-object tracking - high complexity

The high complexity task is defined by the hyperparameter configuration found in Appendix A.3.2. In this task, the object is not guaranteed to be measured, the measurement and process noise are significantly higher than in the base case, and we simulate a large number of clutter detections at each time step. One example generated using the hyperparameter configuration is shown in Figure 4.5. As in the previous task, the figure also shows the ground truth as a blue ellipse and the estimates obtained using the STRAT architecture and the PMBM baseline as green and red ellipses, respectively. The GOSPA score for this particular scenario was 0.02402 for the STRAT model and 0.03570 for the PMBM baseline, which can be compared with the average performance shown in Table 4.5.

The performance of each transformer-based model and the PMBM filter was computed in the same fashion as for the base case in Section 4.2.1. The average performance over the 1000 examples is shown in Table 4.5.

From the table, it can be seen that as the task complexity increased, the performance of both the PMBM baseline and all models and decreased. As in the less complex case, the STRAT architecture trained with the Gaussian-Wasserstein loss achieves better performance than the same architecture trained with the extended L1-loss. The STRAT architecture achieves the best performance among all the transformer-based architectures and, in contrast to the less complex base case, it also outperforms the PMBM filter. However, by inspecting the GOSPA decomposition, it is evident that the main difference in performance between the STRAT architecture and PMBM filter stems from misdetection. The STRAT architecture always assumes that there is one object present, while the PMBM filter does not,

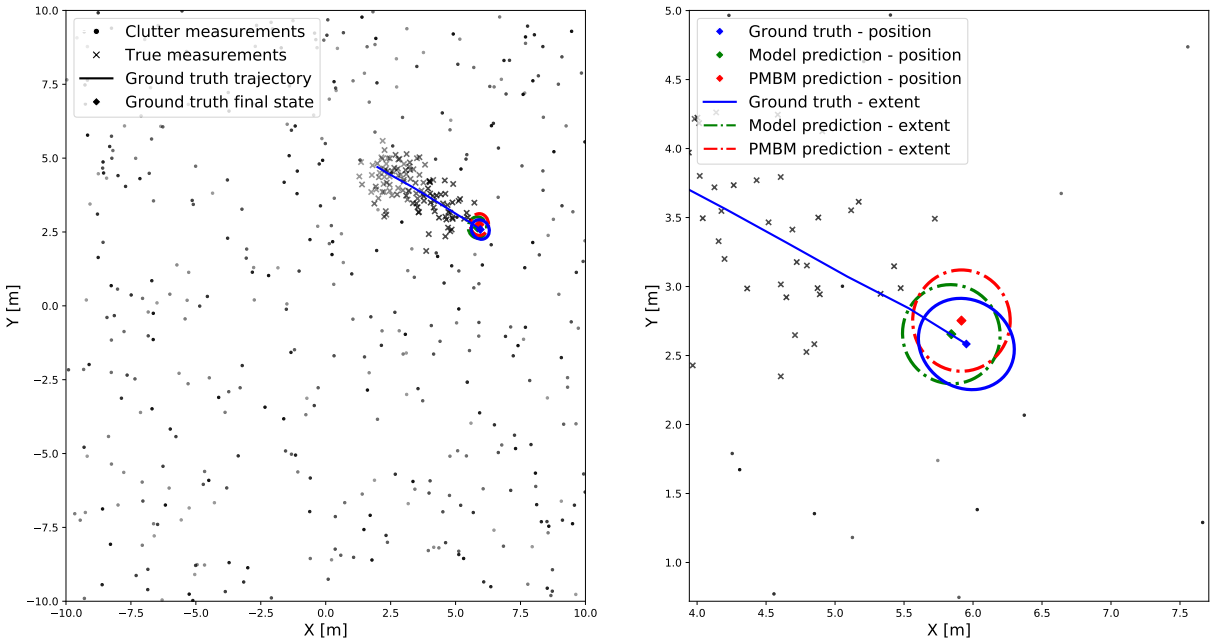

Figure 4.5: Extended object tracking task with a single target. The measurements are visualized as circles or crosses depending on whether or not they are clutter detections. The measurement opacity reflects at which time they are collected where darker equals the most recent measurements. The trajectory of each object is shown as a solid line. The objects' extent is constant over the entire sequence and visualized as a blue ellipse at the final time step. Furthermore, the estimates obtained using the top-performing STRAT model and the PMBM filter are shown as green and red ellipses, respectively.

making the comparison slightly biased in favor of the STRAT model. The localization error is marginally lower for the STRAT architecture, but this slight difference can not be considered statistically significant. Nevertheless, the STRAT architecture achieves performance on par with the PMBM filter, which is currently considered to be state-of-the-art in the field. Just as in the base case, the MOTT architecture's performance is located between the two STRAT architectures.

4.3 Transformer insights

Several interesting insights have been gained during the development and evaluation of the transformer-based architectures shown below. Firstly, we present results showing how the MOTT architecture reasons about data associations via its attention maps. Secondly, we show how the GOSPA score varies over the entire trajectory when using the STRAT architecture. Lastly, we present findings indicating the usefulness of the contrastive loss described in Section 3.3.2.

Table 4.5: Average GOSPA score and decomposition for the MOTT model, the two STRAT models as well as the PMBM filter for the extended object tracking with single target on the high complexity task. Lower scores equal higher performance.

Method	GOSPA	Localization	Misdetection	False detection
MOTT	0.0532	0.0532	**0.0**	**0.0**
STRAT (L1)	0.0602	0.0602	**0.0**	**0.0**
STRAT (GW)	**0.0237**	**0.0237**	**0.0**	**0.0**
PMBM	0.0356	0.0245	0.0110	**0.0**

4.3.1 Attention maps

All transformers-based architectures rely on the concept of attention explained in Section 2.4.1, which enable them to reason about contextual dependencies among a sequence of inputs. Simplified, for each item in the input sequence, it computes an attention weight for all other items in the sequence, meaning that for any item j, there exists a sequence of attention weights $\{a_i^j\}_{i=1}^l$, referred to as the attention map, for that particular item. Again, very simplified, the attention map describes how much attention that particular item gives to every other item in the input sequence, which is how it can reason about contextual dependencies.

In Figure 4.6, a scene from the PMOT base case, explained in Section 4.1.2, is shown along with the attention maps for three of the object queries individually. The attention maps are obtained by feeding the measurement sequence through a trained MOTT architecture. The attention maps are visualized such that each corresponding measurement in the attention map is plotted, and the opacity corresponds to its attention weight. Note that the attention weights are averaged over all decoder layers in the visualization.

From the figure, we can see that the object queries used for the final prediction of a specific object state have a highly focused attention map. Specifically, they mostly attend to embeddings related to measurement originating from a given object. Most measurement embeddings are deemed irrelevant and receive little to no attention from the object queries. Further, the attention is mostly on measurement embeddings from recent time steps, although measurements from many time steps are available.

In Figure 4.7 the attention maps after the first decoder layer is shown for two different objects and models. The scenario and color scheme used are the same as shown in Figure 4.6. The upper and bottom row shows the attention maps of the two objects when computed by a trained MOTT and DETR architecture, respectively. From the figure, it is clear that the DETR architecture pays attention more evenly across all measurements, while the MOTT model has highly focused attention maps only paying to a small number of measurements.

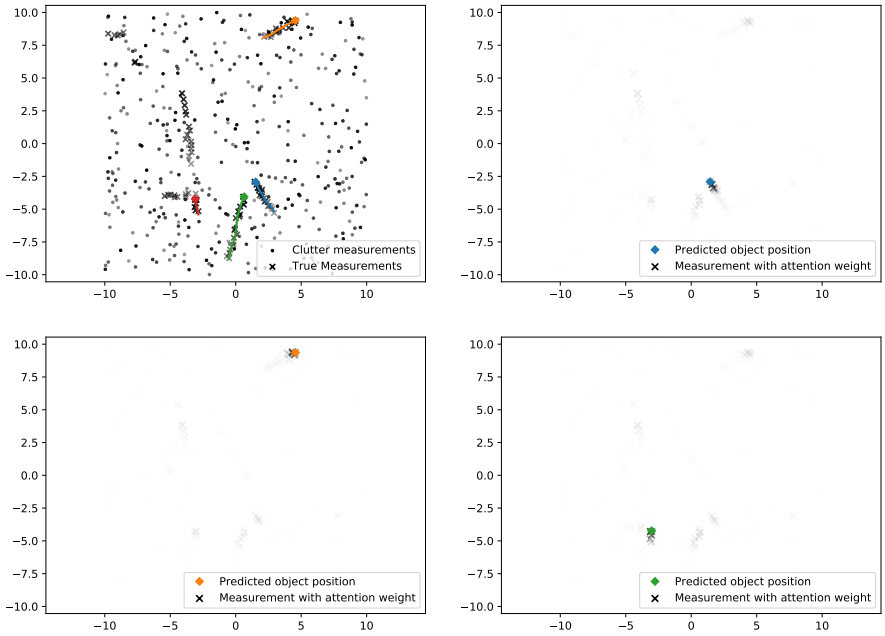

Figure 4.6: One training example (top-left) generated using the PMOT base case, defined in Section 4.1.2. The remaining three graphs show the attention maps of three object queries whose predictions were matched to the ground truth. The attention maps were computed by a trained MOTT architecture where the opacity of each measurement corresponds to the attention weight of that measurement.

4.3.2 Trajectory estimation

When using the STRAT architectures, we obtain a trajectory estimate, meaning that a state estimate for each time step in the sliding window is available. Using these estimates, the average performance, in terms of GOSPA, can be computed for the entire trajectory. In Figure 4.8, the GOSPA score has been computed for the entire trajectory and averaged over 1000 examples using two different tasks. The average performance was computed for the extended tasks described in Section 4.2.1 and 4.2.2 and are shown in the left and right parts of Figure 4.8 respectively.

By inspecting Figure 4.8, we can see the same behavior for both tasks, namely that the GOSPA score follows a U-shape over the trajectory. Starting at the beginning, we see a decrease in the GOSPA score as we move towards the center of the trajectory. However, as we move away from the center towards the end of the trajectory, we see the score increasing. To summarize, the architecture is better at predicting the states in the center of the trajectory than the initial and final states. These results are highly expected since the model can leverage both past and future measurements when performing the prediction in the center. However, when predicting the initial and final state estimate, the model can only leverage future and past measurements, respectively. By conditioning on both future and past measurements, the data associations for that particular time step become simpler, leading to more accurate state regression. This is very similar to what is known as fixed-lag smoother

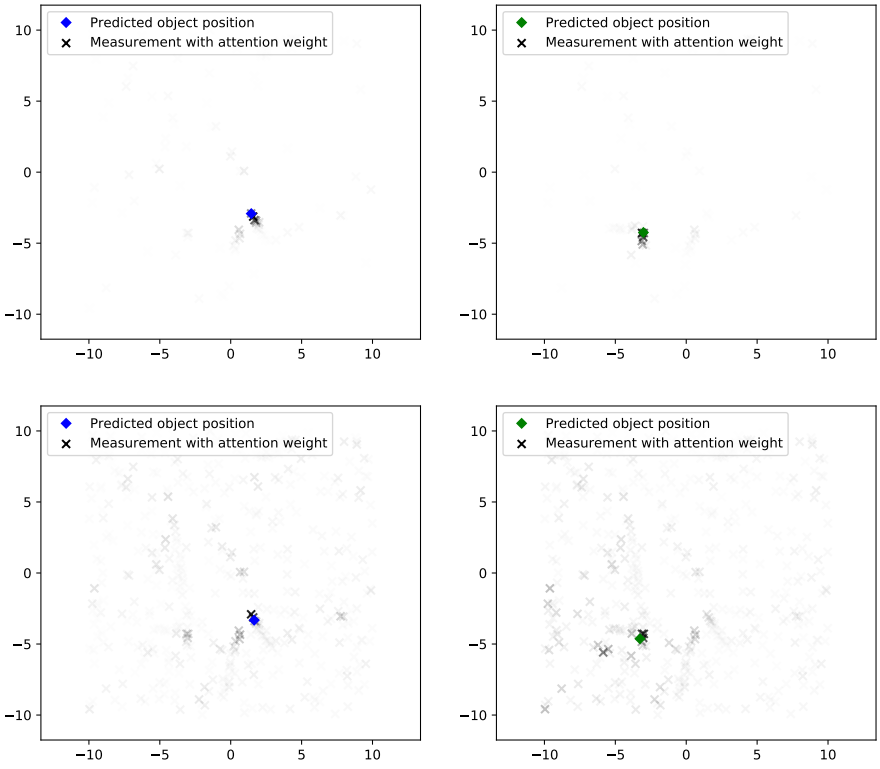

Figure 4.7: The attention map for two of the object queries in Figure 4.6 after the first decoder layer. The bottom row shows the attention map when computed using DETR model and the upper is computed using the MOTT model. The training example, objects and the color scheme is the same as in Figure 4.6.

in Bayesian filtering, where the objective is to estimate the lagged state $\hat{x}_{k-N|k}$ by leveraging on all measurement up to and including time k for a fixed number of lagged time steps N.

4.3.3 Contrastive learning

The main objective of the contrastive learning framework used in this thesis is to support the training by helping the encoder produce sensible embeddings. It does so by encouraging the encoder to produce similar embeddings if the corresponding measurements originate from the same object and dissimilar if they originate from different objects.

The embeddings produced by the encoder exist in a high-dimensional feature space and are therefore hard to interpret. However, using the t-distributed stochastic neighbor embedding (t-SNE) algorithm [50], it is possible to project these high-dimensional vectors into a 2d representation. In Figure 4.9 the result of using the t-SNE algorithm on one embedding sequence is shown. Here, each embedding has been transformed into a 2d representation to ease visualization. The blue scatter points represent embeddings corresponding to clutter measurement while the others

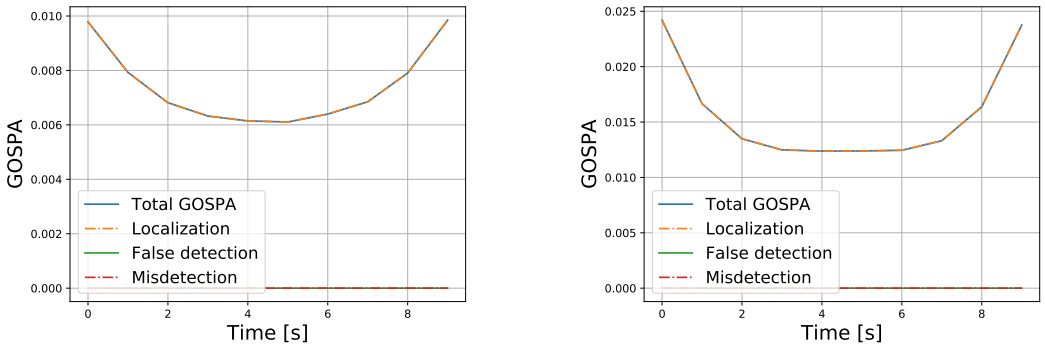

Figure 4.8: Average GOSPA score over the entire sliding window for 1000 examples. Obtained by performing trajectory estimation using the STRAT architecture. The left graph show the extended object base case while the right show the complex task.

are colored by object id.

As evident from the figure, the encoder has learned to produce embeddings such that it is possible to distinguish clutter measurements from true measurements reasonably well. Furthermore, the embeddings seem to hold information that makes it possible to cluster measurements originating from the same object close to each other, which is the objective of the contrastive loss explained in Section 3.3.2. Even though the t-SNE algorithm is stochastic, the results in Figure 4.9 indicate that contrastive loss helps the encoder learn to produce embeddings containing valuable information. While only one example is shown here, this behavior has been seen consistently over different scenes.

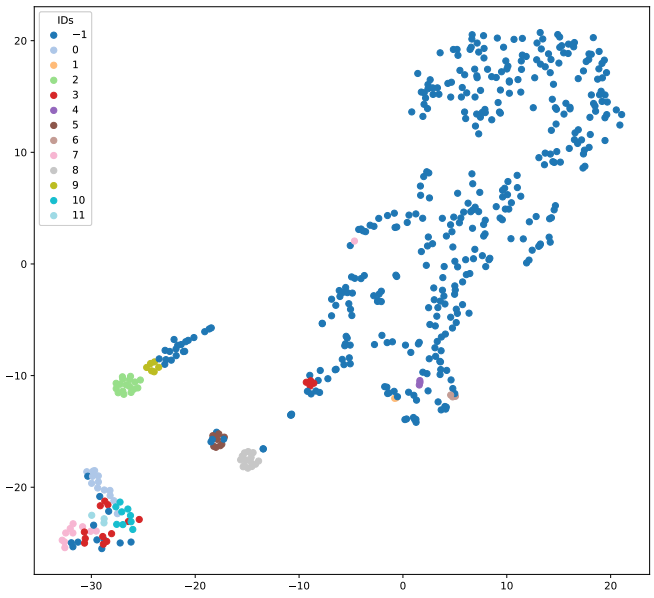

Figure 4.9: t-SNE 2D representation of a high-dimensional embedding sequence produced by an MOTT encoder. Here, blue scatter points (id = −1) correspond to clutter measurements while measurements originating from objects are colored by object id.

5

Discussion

In this chapter, the results of this thesis are discussed and analyzed. First, we highlight trends in the results and discuss some specific observations. The discussion of results is divided into point and extended object tracking, respectively. Subsequently, we reason about the thesis methodology, specifically the choice of using synthetic data. We analyze the implications of our design choices for both these topics and suggest directions for future research. Finally, we present several lines of futures work that were considered out of scope for this thesis.

5.1 Discussion of results

Here we discuss general trends present in the results and share observations that shed light upon the workings of our transformers. First, this is done focusing on point object tracking, followed by a similar discussion for extended object tracking and the differences for that task.

5.1.1 Discussion of point object tracking results

For the point object tracking, some common themes can be observed in the results across tasks. For one, there is a reoccurring order in the performance ranking amongst the transformer-based approaches. Further, the ranking between transformers and model-based approaches changes as task complexity increases. Lastly, the GOSPA decomposition seems to differ in similar ways between transformers and model-based approaches, even for different tasks. This section aims to dive deeper into these observations and discuss potential causes and implications for future research.

Transformers comparison. Looking at performance, it seems that for the point object tracking task, the best transformer-based model is MOTT followed by STRAT and DETR. For the single target task, the only thing setting them apart is their ability to estimate the target's position correctly. While STRAT was not evaluated for multiple targets, also in this setting, MOTT and DETR differ mainly in their localization scores. Thus we explore what it is, in the architectures design and their training scheme, resulting in this performance discrepancy.

Comparing the localization scores for the single target tracking in Table 4.1 shows

the STRAT model to clearly outperform DETR. However, by inspecting their architectures, one finds that, on a high level, they are quite similar in design. They both use the same number of encoder layers to embedded contextual information into the measurement embeddings, which then are used by a decoder with generic learned object queries. While the encoder in the DETR model is trained partly by the contrastive loss, the first encoder in the STRAT model is trained to discriminate between clutter measurements and measurements from objects. Other similarities are that the predictions from the decoder are trained using some L1-loss. For DETR this is only to predict position at the last time step, while STRAT does so for every time step in the current frame.

One hypothesis to explain their difference in localization scores is based on multitask learning. While having a similar structure, the STRAT model is trained to perform more tasks than the DETR model. Instead of only predicting the object states at the last time step, it is trained to predict them over the entire time horizon. We hypothesize that this conveys more information to the model about the structure in the input and how measurements relate to each other. The model is explicitly told where the object was located throughout the timeframe, which should help learning compared to only informing the model about the location at the last time step. Further, the adjustment loss $\mathcal{L}_a$ explained in Section 3.3.1 is used to train STRAT and implicitly gives information on which measurements originated from objects and what the true position of the object at that time was.

While STRAT might outperform DETR, it is further outperformed by MOTT. What is interesting is that MOTT utilizes the same training scheme as the DETR model; thus, this performance boost cannot stem from the training or loss function. Instead, to explain this behavior, we claim that the MOTT model has a suitable inductive bias built in for performing MOT, meaning reasonable assumption about the task structure have been incorporated into the model. Specifically, we argue that the scene-based object queries and iterative refinement are the main reasons for the improved localization.

We believe that the scene-based object queries enable the decoder to attend to relevant measurement embeddings in earlier decoder layers compared to general learned object queries. While the encoder might embed context such as potential data associations into the measurements, the decoder still has to aggregate and interpret said information. Aggregating such information should be more straightforward if the object queries already carry context information. An example displaying the more focused attention of scene-based object queries was further shown in Figure 4.7. Ultimately, this should enable the decoder to extract more valuable information and make better predictions. The predictions are further improved by utilizing iterative refinement. This allows the predictions to have a stronger connection to the current scene since their base value is the location of some measurement. Assuming MOTT selects recent measurements from object currently alive, predicting those measurement locations as positional estimates will yield the model a decent localization score. Hence this can be seen as a suitable inductive bias.

To summarize, techniques for creating high-performing transformers for MOT include the design of loss functions that convey relevant information and the use of domain knowledge when selecting model architecture. In this thesis, these two techniques have been used somewhat exclusively in the STRAT and MOTT models, respectively. For future work, we suggest combining these two into one uniform model and training scheme, with the potential to achieve even better performance.

Transformers vs. model-based. While the ranking amongst the transformers is constant across tasks, how they fare compared to model-based approaches is not. As tasks become increasingly complex, the general trend is that the performance of the transformers moves closer to that of the model-based methods. To be precise, the MOTT model is worse than the PMBM filter at single target tracking, on par for our base case with multiple objects, and substantially outperforms the filter for the complex PMOT task.

Inspecting the GOSPA decomposition further reveals that the model-based approaches and transformer models excel at minimizing different parts of the GOSPA score, with the former achieving lower localization scores and the latter lower misdetection and false detection scores. For the objects tracked by the model-based approaches, the states are estimated robustly and accurately using well-established filtering techniques. In contrast, the transformers are better at estimating the correct number of objects in the scene while being less confident in their true positions.

Splitting the MOT task into part data association and part filtering, we argue that the transformer is better equipped to solve the data association problem than, for instance, the PMBM filter. This is also motivated by the fact that when no modelling errors exist, most approximations are introduced in the PMBM filter when reducing the number of potential hypotheses and data associations. Given a correct data association, the filtering in the PMBM filter should provide Bayes-optimal state estimates. Further, we claim that as tasks become increasingly complex, it is mainly the data association problem that increases in difficulty. Increased noise intensity and more objects lead to an increased number of potential data associations. Thus, for complex tasks, limiting the number of hypotheses in the PMBM filter clearly reduces the quality of the approximations as more hypotheses must be discarded.

While it is hard to say with certainty what allows the transformers to perform data association, the Figures 4.6 and 4.9 allow some interpretability. From Figure 4.9 it is evident that the MOTT model learns to embed context into the measurements such that it is possible to cluster measurements from the same object. Further, Figure 4.6 shows that the attention indicates that the model is aware of which measurements are related. Since this is done using high-dimensional vectors, it is reasonable to assume that these data associations are not done in a hard way, i.e., there is no single hypothesis, but how measurements are related is done more fluidly. One interesting line of future work would be to harvest these soft data associations and combine them with the filtering techniques in the PMBM filter. Such an approach can utilize the better data associations from transformers in complex scenarios and improve final predictions by using filtering for state estimation. Further, this would

allow even greater insight into the model's internals, enabling some verification of correctness and robustness in the final predictions.

5.1.2 Discussion of extended object tracking results

Some of the observations from the point object tracking results hold true also for extended object tracking, e.g., the transformer models outperform the model-based approaches when task complexity is increased. However, other trends are no longer present, and the internal ranking amongst the transformers is different than for the point object tracking. The two main differences are that STRAT now outperforms the MOTT model and also achieves localization scores on par or better than the PMBM filter.

Explaining the fact that the STRAT model is worse than the MOTT model in the point object tracking case but better in the extended object tracking case is not easy. However, we hypothesize that the inductive bias introduced in the MOTT model is not as advantageous in the extended tracking case as in the point object tracking task. The measurements are sampled almost uniformly over the extent of the object, meaning that a considerable correction could potentially be needed to move that measurement to the center of the object. We believe that this could harm the model's performance and be viewed as a faulty inductive bias, which could be the reason for the reversed performance ordering.

We generate measurements in the extended object tracking task by sampling multiple measurements centered around the objects true position. An accurate position estimate can thus be obtained by simply averaging the measurements, provided the number of measurements is large enough. As discussed previously, the transformer architecture has been shown to excel in the task of data association. Therefore, if it can find relevant measurements and average the positional information stored within the corresponding embeddings, it can obtain good estimates. In both of the extended tasks, we generate on average 20 measurements per object per time step. We believe that this number is large enough to obtain a good estimate from averaging all relevant measurements and could be the reason for the improved localization score.

While both the STRAT and PMBM models provide trajectory estimates, they were compared based their estimations at the final time step. This was done because the MOTT architecture is not able to provide trajectories, and therefore, to enable comparisons between the models, this limitation was necessary. However, as a future line of work, it could prove interesting to compare the trajectory estimates provided by the STRAT model and the baseline based on the GOSPA metric generalized for trajectories developed in [51]. In this case, it would also be interesting to change the PMBM baseline to a improved variant of the PMBM filter specifically developed for trajectories [52, 53, 54], and see if the baseline performance can still be surpassed.

5.2 Discussion of methods

In order to enable a fair comparison between the model-based approaches and transformer-based models, it was decided to use only synthetic data for training and evaluation. As mentioned earlier, this erases modelling errors and provides cheap training data. However, it also raises the question of the applicability of the designed transformer models in real-world scenarios. This section discusses the implications of using only synthetic data for training transformers for MOT and what changes are needed to apply a transformer to real-world data.

First and foremost, to apply transformers to real-world data for kinematic MOT a suitable dataset is needed. Even if the dataset is not large enough for training, it should still provide a framework for evaluation and benchmarking. While datasets are common in the community for MOT in videos [7, 8, 9], no equivalent exists in the kinematic tracking community. Some multimodal datasets containing radar detections do exist [15] but have not received much attention from the filtering community. This makes it hard to compare methods for kinematic training as each new approach is free to choose the setting for evaluation. We believe that it is crucial for the research community to create such a dataset and agree on a set of performance metrics.

Assuming a suitable dataset for the evaluation would exist for kinematic MOT, this would pose several challenges for our existing transformer models. While our data generation module aims at generating somewhat realistic data, it is still based on models that only approximate reality. Thus, one potential challenge lies in whether the transformer can generalize to other data distributions, i.e., can the transformer be trained on synthetic data and perform well on real-world data. An obvious solution is to train the transformer directly on real-world data. However, as mentioned previously, the transformer architecture generally needs a large amount of data. One way to mitigate that would be to create some method to train using real and synthetic data jointly. Similar approaches exist in the literature [55] and have achieved new state-of-the-art results.

Furthermore, real-world data, assuming lidar or radar sensors, contains multiple measurements per object, thus being a EMOT task. This thesis only presents transformer models to solve PMOT and ESOT, thus requiring model adjustments to be able to solve EMOT. With the strong localization scores of STRAT for extended objects and the low GOSPA scores of MOTT in the multi-object tracking, we believe that combining them into a single model has the potential to reach state-of-the-art in EMOT. However, this was not possible within this thesis and is left for future research.

5.3 Future work

This section presents multiple lines of future research that were out of scope for this thesis. The suggestions have the potential to further improve transformer perfor-

mance in MOT and give additional insights into the inner workings of the transformer architecture.

5.3.1 Differentiable matching and GOSPA loss function

One issue with the used training methodology is that the matching procedure is non-differentiable, i.e., gradients to optimize the network cannot be computed through the matching. Thus the transformers are not given any direct information about how the matching is done based on their outputs. Further, all methods are trained with specific loss functions while being evaluated using a different metric, namely the GOSPA metric. This is because the GOSPA metric is not suited to be used as a loss function, also being non-differentiable.

In the video MOT community, similar issues exist, where the standard performance metrics are non-differentiable, and the training schemes rely on some matching procedure. In [46], they present differentiable proxies for both matching and metrics, allowing them to optimize their models directly for the relevant metrics. Specifically, a neural network is used to approximate the Hungarian algorithm, and its output can be used to formulate differentiable proxies of the performance metrics. Their approach improved the performance of multiple existing multi-object trackers. A similar approach was explored in this thesis, using a transformer encoder to approximate the Hungarian algorithm and creating a differentiable proxy for GOSPA. While the method suffered from instability during training and did not increase performance, further research could find ways to mitigate these issues.

5.3.2 Downsampling transformer

In both the STRAT and MOTT model, domain knowledge has been used to motivate architecture design choices. In STRAT, the sequence of measurement embeddings is first reduced based on the predictions of each measurement being clutter or not. Subsequently, that sequence is reduced by letting a set of object queries aggregate relevant information, where the number of object queries is based on the length of the time window. In MOTT, only a subset of the embedded measurements is used as object queries in the decoder, knowing that only some are relevant for final state predictions. These operations both reduce the models' computational complexity and aim to support the model to perform the task at hand.

Methods, where the sequence length is reduced gradually throughout the transformer, are an active field of research, see for example [56, 57, 58, 59]. Many approaches are inspired by the pooling operations used in convolutional neural networks. However, we believe more research is needed to find sound and well-motivated ways of reducing sequence lengths that are not domain-specific. This has the potential to improve the applicability of the transformer not only in MOT but many other fields.

5.3.3 Complete state prediction with uncertainty

One advantage of the model-based approaches is that they provide a complete state estimate and an indication of the uncertainty in the estimates. This could potentially be very useful in many downstream tasks, e.g., decision making. Currently, all transformer-based models described in this thesis only provide partial state predictions, e.g., no velocity estimates, and not any uncertainty estimates. We believe that extending the models to provide the complete state prediction and uncertainty estimates is an exciting line of research that should be evaluated further. Exploring this would, in theory, allow for the transformer-based models to fully replace model-based tracking algorithms even in a setting where the full state and uncertainty estimates are essential for other parts of the software stack. The uncertainty extension could potentially be done by simply letting another FFN provide these uncertainty estimates or by resorting to more advanced procedures, e.g., Monte-Carlo Dropout [60].

5.3.4 Trajectory transformer

While we argue that the single most interesting data provided by a tracking algorithm is the states of all object at the current time, some applications may require an algorithm to maintain identities of targets across time to form trajectories. Currently, all our models take a sequence of measurements and produce their estimates for that frame. This could be done iteratively; however, doing so does not provide any correlation between the estimates provided in the two separate frames. A naive approach to create this correlation, thus forming trajectories, would be to greedily assign estimates in one frame to estimates in the next frame via a pair-wise metric, e.g., distance. However, one can imagine that there exist ways of forming this correlation in a more sophisticated way. Several works [22, 61, 62, 63, 64] have successfully explored how to build this sort of correlation into the architecture of the model. Much inspiration could be taken from these approaches to extend our architectures further.

However, the iterative use of our model raises another concern. As we are already utilizing a sliding window approach, most of the measurements will be the same between two separate frames. To clarify, the first frame will have measurement collected from time $t = k - n$ to $t = k$. while the second will have measurements from $t = k-n+1$ to $t = k+1$. Intuitively, this seems wasteful as many computational resources are used to compute something very similar to what was computed in the previous frame. Trying to remedy this seemingly wasteful use of computational resources is another possible avenue of future work, potentially leading to a highly efficient tracking algorithm. Potentially, inspiration for such a model can be drawn from approaches such as [65].

6

Conclusion

Multi-object tracking is a key technology in enabling safe, autonomous vehicles of the future. The field is traditionally dominated by model-based Bayesian filtering approaches, which achieve the current state-of-the-art. However, these methods resort to approximations to remain computationally tractable in complex settings, leaving room for increased performance. In this thesis, we have explored the applicability of the transformer architecture to solve the multi-object tracking problem. In particular, we modified existing and designed novel, transformer-based models to estimate the set of object states, given a sequence of noisy measurements. Their performance was compared in terms of the GOSPA metric to current state-of-the-art approaches, namely the PMBM and δ-GLMB filters. This was done using synthetic data to remove any modelling errors and unlock unlimited amounts of easily generated training data.

Using the transformer models, we outperform the state-of-the-art filtering techniques on complex tasks while performing on par for simpler scenarios. Further, we show that the margin in performance gains increases with task complexity. This is attributed to lower counts of misdetections and false detections compared to the model-based approaches. Furthermore, the interpretability of the transformer architecture is utilized to highlight internal processes and behaviour. Finally, this thesis displays the potential of data-driven approaches in a regime dominated by modelling and filtering and shows that the flexible transformer architecture is applicable in yet another field.

Bibliography

[1] S. Pang and H. Radha, "Multi-object tracking using Poisson multi-Bernoulli mixture filtering for autonomous vehicles," *arXiv preprint arXiv:2103.07783*, 2021.

[2] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in neural information processing systems*, 2017, pp. 5998–6008.

[3] N. Carion, F. Massa, G. Synnaeve, N. Usunier, A. Kirillov, and S. Zagoruyko, "End-to-end object detection with transformers," *arXiv preprint arXiv:2005.12872*, 2020.

[4] (2019, Nov.) Safety in the automotive sector. [Online]. Available: https://ec.europa.eu/growth/sectors/automotive/safety_en

[5] W. Luo, J. Xing, A. Milan, X. Zhang, W. Liu, and T.-K. Kim, "Multiple object tracking: A literature review," *Artificial Intelligence*, p. 103448, 2020.

[6] A. Milan, L. Leal-Taixé, I. Reid, S. Roth, and K. Schindler, "Mot16: A benchmark for multi-object tracking," *arXiv preprint arXiv:1603.00831*, 2016.

[7] P. Dendorfer, H. Rezatofighi, A. Milan, J. Shi, D. Cremers, I. Reid, S. Roth, K. Schindler, and L. Leal-Taixé, "MOT20: A benchmark for multi object tracking in crowded scenes," *arXiv preprint arXiv:2003.09003*, 2020.

[8] P. Dendorfer, A. Osep, A. Milan, K. Schindler, D. Cremers, I. Reid, S. Roth, and L. Leal-Taixé, "MOTChallenge: A benchmark for single-camera multiple target tracking," *International Journal of Computer Vision*, vol. 129, no. 4, pp. 845–881, 2021.

[9] P. Voigtlaender, M. Krause, A. Osep, J. Luiten, B. B. G. Sekar, A. Geiger, and B. Leibe, "MOTS: Multi-object tracking and segmentation," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 7942–7951.

[10] Y. Xu, Y. Ban, G. Delorme, C. Gan, D. Rus, and X. Alameda-Pineda, "Transcenter: Transformers with dense queries for multiple-object tracking," *arXiv*

preprint arXiv:2103.15145, 2021.

[11] P. Bergmann, T. Meinhardt, and L. Leal-Taixe, "Tracking without bells and whistles," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2019, pp. 941–951.

[12] Á. F. García-Fernández, J. L. Williams, K. Granström, and L. Svensson, "Poisson multi-Bernoulli mixture filter: direct derivation and implementation," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 54, no. 4, pp. 1883–1901, 2018.

[13] B. Vo, B. Vo, and H. G. Hoang, "An efficient implementation of the generalized labeled multi-Bernoulli filter," *IEEE Transactions on Signal Processing*, vol. 65, no. 8, pp. 1975–1987, 2017.

[14] A. Geiger, P. Lenz, and R. Urtasun, "Are we ready for autonomous driving? the KITTI vision benchmark suite," in *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012.

[15] 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 preprint arXiv:1903.11027*, 2019.

[16] T. Yin, X. Zhou, and P. Krähenbühl, "Center-based 3d object detection and tracking," *arXiv preprint arXiv:2006.11275*, 2020.

[17] H.-k. Chiu, A. Prioletti, J. Li, and J. Bohg, "Probabilistic 3d multi-object tracking for autonomous driving," *arXiv preprint arXiv:2001.05673*, 2020.

[18] X. Weng, J. Wang, D. Held, and K. Kitani, "3d multi-object tracking: A baseline and new evaluation metrics," *arXiv preprint arXiv:1907.03961*, 2020.

[19] A. Kim, A. Ošep, and L. Leal-Taixé, "Eagermot: 3d multi-object tracking via sensor fusion," 2021.

[20] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.

[21] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell *et al.*, "Language models are few-shot learners," *arXiv preprint arXiv:2005.14165*, 2020.

[22] T. Meinhardt, A. Kirillov, L. Leal-Taixe, and C. Feichtenhofer, "Trackformer: Multi-object tracking with transformers," *arXiv preprint arXiv:2101.02702*, 2021.

[23] P. Chu, J. Wang, Q. You, H. Ling, and Z. Liu, "Spatial-temporal graph transformer for multiple object tracking," *arXiv preprint arXiv:2104.00194*, 2021.

[24] J. Pinto, G. Hess, W. Ljungbergh, Y. Xia, L. Svensson, and H. Wymeersch, "Next generation multitarget trackers: Random finite set methods vs transformer-based deep learning," 2021.

[25] A. S. Rahmathullah, A. F. Garcia-Fernandez, and L. Svensson, "Generalized optimal sub-pattern assignment metric," *2017 20th International Conference on Information Fusion (Fusion)*, 7 2017. [Online]. Available: http://dx.doi.org/10.23919/ICIF.2017.8009645

[26] S. Yang, M. Baum, and K. Granström, "Metrics for performance evaluation of elliptic extended object tracking methods," in *2016 IEEE International Conference on Multisensor Fusion and Integration for Intelligent Systems (MFI)*. IEEE, 2016, pp. 523–528.

[27] C. R. Givens, R. M. Shortt *et al.*, "A class of Wasserstein metrics for probability distributions." *The Michigan Mathematical Journal*, vol. 31, no. 2, pp. 231–240, 1984.

[28] S. Särkkä, *Bayesian filtering and smoothing*. Cambridge University Press, 2013, no. 3.

[29] J. L. Williams, "Marginal multi-Bernoulli filters: RFS derivation of MHT, JIPDA, and association-based MeMBer," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 51, no. 3, pp. 1664–1687, 2015.

[30] ——, "Hybrid Poisson and multi-bernoulli filters," in *2012 15th International Conference on Information Fusion*. IEEE, 2012, pp. 1103–1110.

[31] ChalmersX, "Multi-object tracking for automotive systems," https://www.edx.org/course/multi-object-tracking-for-automotive-systems, May 2021.

[32] K. Granström, M. Fatemi, and L. Svensson, "Poisson multi-Bernoulli mixture conjugate prior for multiple extended target filtering," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 56, no. 1, pp. 208–225, 2019.

[33] K. Granstrom, M. Baum, and S. Reuter, "Extended object tracking: Introduction, overview and applications," *arXiv preprint arXiv:1604.00970*, 2016.

[34] K. G. Murthy, "An algorithm for ranking all the assignments in order of increasing costs," *Operations research*, vol. 16, no. 3, pp. 682–687, 1968.

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

[36] B.-N. Vo, B.-T. Vo, and D. Phung, "Labeled random finite sets and the Bayes multi-target tracking filter," *IEEE Transactions on Signal Processing*, vol. 62, no. 24, pp. 6554–6567, 2014.

[37] B.-N. Vo, B.-T. Vo, and H. G. Hoang, "An efficient implementation of the generalized labeled multi-Bernoulli filter," *IEEE Transactions on Signal Processing*, vol. 65, no. 8, pp. 1975–1987, 2016.

[38] L. Chen, "An introduction to the generalized labeled multi-Bernoulli filter through MATLAB code," in *Signal Processing, Sensor/Information Fusion, and Target Recognition XXVII*, vol. 10646. International Society for Optics and Photonics, 2018, p. 1064604.

[39] T. Chen, S. Kornblith, M. Norouzi, and G. Hinton, "A simple framework for contrastive learning of visual representations," *arXiv preprint arXiv:2002.05709*, 2020.

[40] T. Chen, S. Kornblith, K. Swersky, M. Norouzi, and G. Hinton, "Big self-supervised models are strong semi-supervised learners," *arXiv preprint arXiv:2006.10029*, 2020.

[41] K. He, H. Fan, Y. Wu, S. Xie, and R. Girshick, "Momentum contrast for unsupervised visual representation learning," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 9729–9738.

[42] X. Chen, H. Fan, R. Girshick, and K. He, "Improved baselines with momentum contrastive learning," *arXiv preprint arXiv:2003.04297*, 2020.

[43] P. Khosla, P. Teterwak, C. Wang, A. Sarna, Y. Tian, P. Isola, A. Maschinot, C. Liu, and D. Krishnan, "Supervised contrastive learning," *arXiv preprint arXiv:2004.11362*, 2020.

[44] H. W. Kuhn, "The Hungarian method for the assignment problem," *Naval research logistics quarterly*, vol. 2, no. 1-2, pp. 83–97, 1955.

[45] S. Sun, N. Akhtar, H. Song, A. Mian, and M. Shah, "Deep affinity network for multiple object tracking," *IEEE transactions on pattern analysis and machine intelligence*, vol. 43, no. 1, pp. 104–119, 2019.

[46] Y. Xu, A. Osep, Y. Ban, R. Horaud, L. Leal-Taixé, and X. Alameda-Pineda, "How to train your deep multi-object tracker," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 6787–6796.

[47] S. Maji, "The matrix square root and its gradient," https://people.cs.umass.edu/~smaji/projects/matrix-sqrt/, May 2021.

[48] R. Caruana, "Multitask learning," *Machine Learning*, vol. 28, no. 1, pp. 41–75, 1997.

[49] X. Zhu, W. Su, L. Lu, B. Li, X. Wang, and J. Dai, "Deformable DETR: Deformable transformers for end-to-end object detection," *arXiv preprint arXiv:2010.04159*, 2020.

[50] L. Van der Maaten and G. Hinton, "Visualizing data using t-SNE." *Journal of machine learning research*, vol. 9, no. 11, 2008.

[51] 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 preprint arXiv:1605.01177*, 2016.

[52] K. Granström, L. Svensson, Y. Xia, J. Williams, and Á. F. García-Femández, "Poisson multi-Bernoulli mixture trackers: Continuity through random finite sets of trajectories," in *2018 21st International Conference on Information Fusion (FUSION)*. IEEE, 2018, pp. 1–5.

[53] Á. F. García-Fernández, L. Svensson, J. L. Williams, Y. Xia, and K. Granström, "Trajectory Poisson multi-Bernoulli filters," *IEEE Transactions on Signal Processing*, vol. 68, pp. 4933–4945, 2020.

[54] Y. Xia, K. Granström, L. Svensson, Á. F. García-Fernández, and J. L. Williams, "Extended target Poisson multi-Bernoulli mixture trackers based on sets of trajectories," in *2019 22th International Conference on Information Fusion (FUSION)*. IEEE, 2019, pp. 1–8.

[55] P. Tokmakov, J. Li, W. Burgard, and A. Gaidon, "Learning to track with object permanence," *arXiv preprint arXiv:2103.14258*, 2021.

[56] Z. Dai, G. Lai, Y. Yang, and Q. V. Le, "Funnel-transformer: Filtering out sequential redundancy for efficient language processing," *arXiv preprint arXiv:2006.03236*, 2020.

[57] B. Heo, S. Yun, D. Han, S. Chun, J. Choe, and S. J. Oh, "Rethinking spatial dimensions of vision transformers," *arXiv preprint arXiv:2103.16302*, 2021.

[58] Z. Pan, B. Zhuang, J. Liu, H. He, and J. Cai, "Scalable visual transformers with hierarchical pooling," *arXiv preprint arXiv:2103.10619*, 2021.

[59] S. Goyal, A. R. Choudhury, S. Raje, V. Chakaravarthy, Y. Sabharwal, and A. Verma, "PoWER-BERT: Accelerating BERT inference via progressive word-vector elimination," in *International Conference on Machine Learning*. PMLR, 2020, pp. 3690–3699.

[60] Y. Gal and Z. Ghahramani, "Dropout as a Bayesian approximation: Representing model uncertainty in deep learning," in *international conference on machine learning*. PMLR, 2016, pp. 1050–1059.

[61] W.-C. Hung, H. Kretzschmar, T.-Y. Lin, Y. Chai, R. Yu, M.-H. Yang, and D. Anguelov, "SoDA: Multi-object tracking with soft data association," *arXiv preprint arXiv:2008.07725*, 2020.

[62] P. Sun, Y. Jiang, R. Zhang, E. Xie, J. Cao, X. Hu, T. Kong, Z. Yuan, C. Wang, and P. Luo, "TransTrack: Multiple-object tracking with transformer," *arXiv*

preprint arXiv:2012.15460, 2020.

[63] A. Bewley, Z. Ge, L. Ott, F. Ramos, and B. Upcroft, "Simple online and realtime tracking," in *2016 IEEE International Conference on Image Processing (ICIP)*. IEEE, 2016, pp. 3464–3468.

[64] N. Wojke, A. Bewley, and D. Paulus, "Simple online and realtime tracking with a deep association metric," in *2017 IEEE international conference on image processing (ICIP)*. IEEE, 2017, pp. 3645–3649.

[65] A. Rangesh, P. Maheshwari, M. Gebre, S. Mhatre, V. Ramezani, and M. M. Trivedi, "TrackMPNN: A message passing graph neural architecture for multi-object tracking," *arXiv preprint arXiv:2101.04206*, 2021.

A

Data generation

A.1 Data generation hyperparameters

Table A.1: Summary of all the data generation hyperparameters used to control the complexity of the scenarios created.

Hyperparameter	**Explanation**
Data type	What kind of data generated. Potential values are either point object data or extended object data.
τ	The length of the sliding window. How many time steps are to be simulated.
Δt	The size of the time steps
$n_{\text{max\_objects}}$	The maximum number of objects that can be present in the scene.
$\bar{n}_{\text{birth}}$	The average number of objects initialized at the start of each example.
FOV_{ub}	The upper bound of the Field of View (FOV).
FOV_{lb}	The lower bound of the Field of View (FOV).
μ_{v0}	The mean velocity sampled for objects during initialization.
σ_{v0}	The standard deviation of the sampled velocity for objects during initialization.
σ_q^2	The process noise intensity. Controls how much noise is added to the constant velocity motion model.
p_{remove}	Probability of removing an object at any given time step.
n_{add}	The average number of new objects created at each time step.
p_{measure}	Probability that an object gives rise to measurement(s) at any time step.
σ_y^2	The measurement noise intensity. Controls how much noise is added to the true state vector during measurement generation.
n_{clutter}	Average number of clutter measurement appended to the true measurements at each time step.

Table A.2: Summary of the additional data generation hyperparameters used when generating data for extended objects.

Hyperparameter	**Explanation**
Ψ	2×2 scale matrix in the inverse Wishart distribution controlling the uncertainty in the object extent.
ν	Degrees of freedom in the inverse Wishart distribution. Controlling the uncertainty in the object extent.
α	Shape parameter in the gamma distribution. Controlling the number of measurement each object gives rise to at each timestep. Average number of measurements is α/β.
β	Scale parameter in the gamma distribution. Controlling the number of measurement each object gives rise to at each timestep. Average number of measurements is α/β.
σ_h^2	Multiplicative noise. Governs how measurements are generated based on the object extent.

A.2 Point object tracking tasks

A.2.1 Single object

Table A.3: Hyperparameter configuration for point single object tracking task.

Hyperparameter	**Value**
Data type	Point
τ	20
Δt	0.1 s
$n_{\text{max\_objects}}$	1
$\bar{n}_{\text{birth}}$	1
FOV_{ub}	10
FOV_{lb}	-10
μ_{v0}	[0,0]
σ_{v0}	$3\mathbf{I}_2$
σ_q^2	0.5
p_{remove}	0.0
n_{add}	0.0
p_{measure}	0.9
σ_y^2	0.1
n_{clutter}	20

A.2.2 Multi-object - base case

Table A.4: Hyperparameter configuration for point multi-object tracking task, base case.

Hyperparameter	**Value**
Data type	Point
τ	20
Δt	0.1 s
$n_{\text{max\_objects}}$	16
$\bar{n}_{\text{birth}}$	4
FOV_{ub}	10
FOV_{lb}	-10
μ_{v0}	[0,0]
σ_{v0}	$3\mathbf{I}_2$
σ_q^2	0.5
p_{remove}	0.05
n_{add}	0.4
p_{measure}	0.9
σ_y^2	0.1
n_{clutter}	20

A.2.3 Multi-object - high complexity

Table A.5: Hyperparameter configuration for point multi-object tracking task, complex task.

Hyperparameter	**Value**
Data type	Point
τ	20
Δt	0.1 s
$n_{\text{max\_objects}}$	16
$\bar{n}_{\text{birth}}$	6
FOV_{ub}	10
FOV_{lb}	-10
μ_{v0}	[0,0]
σ_{v0}	$3\mathbf{I}_2$
σ_q^2	0.9
p_{remove}	0.05
n_{add}	0.4
p_{measure}	0.8
σ_y^2	0.3
n_{clutter}	30

A.3 Extended object tracking tasks

A.3.1 Base case

Table A.6: Hyperparameter configuration for the base case in extended single object tracking.

Hyperparameter	**Value**
Data type	Extended
τ	10
Δt	0.1 s
$n_{\text{max\_objects}}$	1
$\bar{n}_{\text{birth}}$	1
FOV_{ub}	10
FOV_{lb}	-10
μ_{v0}	[0,0]
σ_{v0}	$10\mathbf{I}_2$
σ_q^2	0.1
p_{remove}	0.0
n_{add}	0.0
p_{measure}	1.0
σ_y^2	0.01
n_{clutter}	5
Ψ	$50\mathbf{I}_2$
ν	106
α	1000
β	50
σ_h^2	0.15

A.3.2 High complexity

Table A.7: Hyperparameter configuration for the high complexity task in extended single object tracking.

Hyperparameter	**Value**
Data type	Extended
τ	10
Δt	0.1 s
$n_{\text{max\_objects}}$	1
$\bar{n}_{\text{birth}}$	1
FOV_{ub}	10
FOV_{lb}	-10
μ_{v0}	[0,0]
σ_{v0}	$8\mathbf{I}_2$
σ_q^2	0.6
p_{remove}	0.0
n_{add}	0.0
p_{measure}	0.9
σ_y^2	0.15
n_{clutter}	40
Ψ	$50\mathbf{I}_2$
ν	106
α	1000
β	50
σ_h^2	0.25

B

Model configurations

B.1 DETR model configuration

Table B.1: Hyperparameter configuration for DETR model.

Hyperparameter	**Value**
Number of object queries	16
d_{model}	256
n_{heads} encoder	8
n_{layers} encoder	6
$d_{\text{feed-forward}}$ encoder	2048
p_{dropout} encoder	0.1
n_{heads} decoder	8
n_{layers} decoder	6
$d_{\text{feed-forward}}$ decoder	2048
p_{dropout} decoder	0.1
$d_{\text{feed-foward}}$ state prediction	128
$N_{\text{hidden layers}}$ state prediction	3

B.2 MOTT model configuration

Table B.2: Hyperparameter configuration for MOTT model.

Hyperparameter	**Value**
Number of object queries	16
d_{model}	256
n_{heads} encoder	8
n_{layers} encoder	6
$d_{\text{feed-forward}}$ encoder	2048
p_{dropout} encoder	0.1
n_{heads} decoder	8
n_{layers} decoder	6
$d_{\text{feed-forward}}$ decoder	2048
p_{dropout} decoder	0.1
$d_{\text{feed-foward}}$ state prediction	128
$N_{\text{hidden layers}}$ state prediction	1

B.3 STRAT model configuration

Table B.3: Hyperparameter configuration for STRAT model.

Hyperparameter	**Value**
Number of object queries	1
d_{model}	256
n_{heads} encoder 1	8
n_{layers} encoder 1	3
$d_{\text{feed-forward}}$ encoder 1	2048
p_{dropout} encoder 1	0.1
n_{heads} encoder 2	8
n_{layers} encoder 2	3
$d_{\text{feed-forward}}$ encoder 2	2048
p_{dropout} encoder 2	0.1
n_{heads} decoder	8
n_{layers} decoder	6
$d_{\text{feed-forward}}$ decoder	2048
p_{dropout} decoder	0.1
$d_{\text{feed-foward}}$ state prediction	128
$N_{\text{hidden layers}}$ state prediction	1
$p_{\text{threshold}}$ clutter removal	0.9

DEPARTMENT OF ELECTRICAL ENGINEERING
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden
www.chalmers.se

CHALMERS
UNIVERSITY OF TECHNOLOGY