CHALMERS
UNIVERSITY OF TECHNOLOGY

Deep Decentralized Multiple Object Tracking

Master's thesis in Systems, control and mechatronics

LECHI LI
CHEN DAI

Department of Electrical Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2022

Master's thesis 2022

Deep Decentralized Multiple Object Tracking

LECHI LI
CHEN DAI

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Electrical Engineering
Chalmers University of Technology
Gothenburg, Sweden 2022

Deep Decentralized Multiple Object Tracking
LECHI LI
CHEN DAI

Supervisor: Yuxuan Xia, Lennart Svensson, Department of Electrical Engineering
Examiner: Lennart Svensson, Department of Electrical Engineering

Master's Thesis 2022
Department of Electrical Engineering
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Typeset in LaTeX
Gothenburg, Sweden 2022

Deep Decentralized Multiple Object Tracking
LECHI LI
CHEN DAI

Department of Electrical Engineering
Chalmers University of Technology

Abstract

Multi-object tracking (MOT) tasks use noisy sensor measurements to estimate objects' states. MOT is widely used in many areas, including autonomous cars, surveillance systems, and multi-agent systems. In a multi-sensor setting, measurements from sensors located at different places are combined for improved accuracy. However, limited communication capacity or processing power sometimes makes it unfeasible to collect and process detections from all sensors jointly. One way to deal with this problem is using decentralized MOT, where object states (and possibly also their densities) are estimated from local sensors and fused to construct better estimation.

This thesis investigates the feasibility of using deep learning-based methods in a decentralized MOT context. Specially, we adapt recently proposed deep multi-object trackers based on the transformer architecture to fuse the trajectory estimates obtained from different local multi-object trackers and output states as well as uncertainties of objects at the current time step. We compare the performance of deep learning-based trackers with a state-of-the-art (SOTA) model-based Bayesian decentralized multi-object tracker in simulated scenarios with different sensor settings and parameters of local multi-object trackers. The simulation results show that learning-based models can outperform or match the performance of the SOTA method in our experimental scenarios.

Keywords: Decentralized multi-object tracking, Deep Learning, Transformers.

Acknowledgements

We thank Yuxuan Xia for being our supervisor. He gives us valuable advice, check our codes and correct our report. He is always there whenever we need him.

We thank Lennart Svensson for being our supervisor and examiner. Thank you for being part of our bi-weekly meetings, providing us with helpful feedback, and answering our questions.

Also, thank you for formulating this interesting thesis project.

In addition, we thank Chalmers Center for Computational Science and Engineering (C3SE) for providing us with the computational resources.

At last, We thank our families and friends for their support and love.

Lechi Li & Chen Dai, Gothenburg, June 2022

Contents

List of Figures

List of Tables

1
Introduction

Autonomous vehicles (AVs) are being developed to provide people with a safe, comfortable, and efficient driving experience [4]. They have the potential to reduce road accidents, minimize the efforts of human drivers, and save fuel [5, 6, 7]. One essential function, which enables vehicles to operate reliably, is called perception, and it aims to comprehend the vehicle's environment in real-time precisely. Specific tasks that fall under the category of perception are, for example, the detection of dynamic and static objects [4]. Similar to how humans sense surroundings using their eyes, AVs rely on different kinds of sensors (for example, cameras, LiDARs, and radars) for perceiving the environment and extracting information for later usage.

Multi-object tracking (MOT) tasks use noisy sensor measurements to estimate objects' states. MOT is widely used in many areas, including autonomous cars, surveillance systems, and multi-agent systems. In a multi-sensor setting, measurements from sensors located at different places are combined for improved accuracy. However, limited communication capacity or processing power sometimes makes it unfeasible to collect and process detections from all sensors jointly. One way to deal with this problem is using decentralized MOT, where object states (and possibly also their densities) are estimated from local sensors and fused to construct better estimates. An essential challenge of decentralized MOT is to leverage information obtained by independent tracking algorithms while ensuring that the unknown common information is not double-counted [8].

In a decentralized MOT setting, the model-based Bayesian filters are currently considered the state-of-the-art (SOTA), which can be classified into two groups [9]. The first group of filters aims at finding the optimal posterior density through computing the joint likelihood using all sensor measurements. Examples of algorithms in this group include the multi-sensor probability hypothesis density (PHD) filter [10], the multi-sensor generalized labeled multi-Bernoulli (GLMB) filter [11] and the multi-sensor multi-Bernoulli filter [12]. In practice, measurements are transmitted between sensors, which might be asynchronous and heterogeneous [9]. As a consequence, applying these algorithms directly are computationally intractable, and approximations are therefore usually required.

In contrast, the filters in the second group, which is the focus of this thesis, fuse the multi-object posterior densities yielded by the local sensors, where the SOTA is achieved by filters that use average approaches; namely, arithmetic average (AA) and geometric average (GA) [13, 14]. These filters are more robust and fault-tolerant

but typically yield sub-optimal solutions since the cross-correlation between densities has been omitted [8].

Deep learning (DL) based MOT trackers have shown unmatched performance in academia and industry in recent years. These trackers typically perform tracking using high-dimensional measurements, such as images and video sequences, which have been boosted due to the rapid increase in computational power. Typical examples include Trackformer [15] and TranTrack [16]. Besides tracking objects in images and videos, transformer-based multi-object trackers or multi-object tracking transformers (MOTTs) have displayed great performance on multiple point object tracking (MPOT) [2, 3, 17], using low-dimensional measurements, e.g., positions and velocities. These trackers are the main concern in this project, which were not trained and evaluated for decentralized tracking before this thesis.

In this thesis, we explore to what extent DL-based MOT trackers can outperform the current SOTA in the context of decentralized fusion. Specially, we apply MOTTs to solve decentralized MOT tasks using synthetic data. Besides producing promising results for various tasks, the MOTTs have many reasons for being suitable candidates to solve decentralized MOT tasks. First, the MOTTs treat MOT tasks as a set prediction problem, where they input a sequence of measurements and produce a sequence of estimates [2]. We can adapt this idea to the decentralized fusion and replace the input of MOTTs from a sequence of measurements with representations of sets of trajectories. In addition, MOTTs are tailored to the representations of the long-range patterns of trajectory estimates, allowing the model to make predictions after considering the information contained in each input element [2]. Nevertheless, to the best of our knowledge, using MOTTs for solving decentralized MOT tasks has not been explored, and the performance of transformers for decentralized MOT has not been compared to the performance of current SOTA model-based Bayesian solutions before this thesis. Therefore, we believe the investigation into this topic is necessary.

1.1 Objective

In this thesis, our objective is to use MOTTs [2, 3, 17] for solving decentralized MOT tasks. In particular, based on sets of trajectory estimates obtained from different local sensor systems over a certain period, the goal is to estimate object states as well as associated uncertainties at the current time using MOTTs. We modify the existing MOTTs to meet our requirements.

To study the performance of each model, we feed the models with datasets generated in different scenarios and with varying complexity to compare their performance with the SOTA model-based methods. The differences and complexities of the datasets differ in sensor locations, areas of overlapped sensor field-of-views (FoVs), and parameters of local multi-object trackers. In particular, in this thesis, we would like to solve the following two tasks:

- Modifying the existing transformer-based multi-object trackers [3, 17] to solve

different types of decentralized multi-objecting tasks.

- Compare the performance of the MOTTs with a SOTA model-based Bayesian filter using synthetic data.

1.2 Demarcation

In this thesis, we will work on synthetic data. Implementing the model-based Bayesian filters is considered outside the scope, and we instead use and modify existing ones to suit our needs. For the output from the network, we focus on estimating object states at the last time step instead of the estimation in an entire interval.

1.3 Thesis outlines and main contributions

In Chapter 2, the necessary background theory to understand the rest of the thesis is provided to the readers. The concepts used in decentralized multi-object tracking are defined. Next, we introduce a model-based filter used for data generation and the model-based decentralized fusion method used as our baseline. Lastly, we introduce the two MOTTs used in this thesis.

Chapter 3 describes the methods that we have used in this project. Firstly, we present how to generate the data and construct the dataset. Secondly, we show the preprocessing of the data set. At last, we present the evaluation methods.

The experimental evaluation is described in Chapter 4, where the experiments are done in two different scenarios, each with three different tasks. The results obtained from Chapter 4 are discussed and analyzed in Chapter 5, and we discuss future work and draw conclusions in Chapter 6.

This thesis work incorporates the following major contributions:

- Applying MOTT for decentralized filtering.

- Experimental comparison between the performance of model-based filter and MOTT in the context of decentralized fusion.

2
Theory

We introduce the related background theory for this thesis. We will first introduce decentralized multiple object tracking, including the basic concept of multiple object tracking, random finite sets with their densities, and the trajectory Poisson multi-Bernoulli filter as well as the SOTA model-based decentralized fusion method. In addition, we provide an overview of MOTT that we apply to decentralized fusion in this thesis. Lastly, we introduce a function that maps a scalar to a higher dimension.

2.1 Decentralized multiple object tracking

In this section, we introduce concepts and methods which arise in model-based decentralized multiple object tracking considered in this thesis.

2.1.1 Multiple object tracking

MOT is a perception task that involves processing measurements collected from multiple objects to estimate their current states. One major challenge in MOT tasks is called data association, which is caused by unknown relations between measurements and objects [18].

When the underlying multi-object models (see Section 2.1.3) and low-dimensional measurements are provided, the MOT tasks are typically solved using model-based Bayesian filters. For standard MOT models with object birth process modeled by a Poisson point process (PPP, see Section 2.1.5 as well as Section 2.1.2) [19], the Poisson multi-Bernoulli mixture (PMBM) filter based on random finite sets [19] can provide the closed-form solution [20], which, in theory, yields exact multi-object posterior without approximation. In practice, it is unrealistic to obtain a computationally tractable solution without approximation due to the data association problem [21]. Thus, these methods must resort to different approximation methods to reduce their computational complexity, inevitably leading to a deterioration of tracking performance.

In the past few decades, with the development of DL technology as well as the rapid increase in computational power, the DL-based trackers have emerged as attractive alternatives to the model-based Bayesian filters, especially in cases where measurements are of high dimension and the multi-object models are highly non-linear or even not available [22]. In particular, the MOTTs are trackers developed based on

the Transformer [23] and have been applied for solving point object tracking tasks, where they have achieved competitive performance compared with SOTA model-based Bayesian filters [3].

2.1.2 Random finite sets

In this work, we use sets to represent object states, measurements, and trajectories. By definition, a RFS is a random variable that outputs sets with a randomly finite number of random object states [2]. More specifically, given a RFS:

$$\mathbf{x} = \{x^1, x^2 ..., x^n\}, \tag{2.1}$$

the object state x^i, $i \in \mathcal{X}$ as well as the set cardinality can be random. In the MOT context, $\mathcal{X}$ is either object state space $\mathbb{R}^{n_x}$ or measurement space $\mathbb{R}^{n_z}$, where n_x and n_z are, respectively, the dimension of single-object state and single measurement. There are many advantages of using RFSs in MOT. For example, they facilitate modeling uncertainties in MOT problems, e.g., the appearing and disappearing of objects. In addition, they are potent tools for deriving Bayesian-optimal solutions in theory. Furthermore, they help develop metrics for performance evaluation.

RFSs are characterized by their probability density functions (PDFs). The PDF of a RFS captures the distribution over its cardinality and its elements (given the cardinality). We introduce a few vital RFSs and their distributions, which are the building blocks of many RFS-based MOT methods.

The Poisson RFS is a RFS with Poisson distributed set cardinality. In MOT, the object birth process and the generation of clutter measurements are commonly modeled by the Poisson RFS, the PDF of which is

$$p_{\mathbf{x}}^{\mathrm{PPP}}(\mathbf{x}) = e^{-\int \lambda(x')dx'} \prod_{x \in \mathbf{x}} \lambda(x), \tag{2.2}$$

where $\lambda(x)$ is the intensity function, and its integral $\int \lambda(x)dx$ gives the Poisson rate. In addition, the Poisson RFS is also call Poisson point process (PPP).

The Bernoulli RFS is a RFS with a Bernoulli distributed set cardinality. In MOT, the Bernoulli RFS is commonly used to model single-object detection in point object tracking and the single-object state with uncertain existence. Its PDF is of form

$$p_{\mathbf{x}}^{\mathrm{Ber}}(\mathbf{x}) = \begin{cases} 0, & \text{if} \quad |\mathbf{x}| > 1 \\ r p_x(x), & \text{if} \quad \mathbf{x} = \{x\} \\ 1 - r, & \text{if} \quad \mathbf{x} = \emptyset \end{cases} \tag{2.3}$$

Here, r and $p_x(x)$ represent the existence probability and a PDF describing the distribution over random element x conditioned on existence, respectively.

In MOT, Bernoulli RFSs can be used to model a single object, and multiple objects can be naturally represented through a multi-Bernoulli (MB) RFS. We assume that $\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_N$ are independent Bernoulli RFSs with corresponding PDFs

$p_{\mathbf{x}_1}(\mathbf{x}_1), p_{\mathbf{x}_2}(\mathbf{x}_2), ..., p_{\mathbf{x}_N}(\mathbf{x}_N)$, then the PDF of a MB RFS is:

$$p_{\mathbf{x}}^{\mathrm{MB}}(\mathbf{x}) = \sum_{\biguplus_{i=1}^{N} \mathbf{x}_i = \mathbf{x}} \prod_{j=1}^{N} p_{\mathbf{x}_j}(\mathbf{x}_j), \tag{2.4}$$

where $\uplus$ denotes the disjoint union.

A multi-Bernoulli mixture (MBM) RFS is made up of MB RFSs, and its PDF is:

$$p_{\mathbf{x}}^{\mathrm{MBM}}(\mathbf{x}) = \sum_{h=1}^{\mathcal{M}} w_h p_{\mathbf{x}}^{h}(\mathbf{x}), \tag{2.5}$$

with $\mathcal{M}$ mixture components. Further, $p_{\mathbf{x}}^{h}(\mathbf{x})$ is the PDF of the h-th MB RFS (2.4), and w_h is its corresponding weight, where $\sum_{h=1}^{\mathcal{M}} w_h = 1$.

The PMBM is made up of a PPP and an MBM RFS, and its PDF is given by:

$$p_{\mathbf{x}}^{\mathrm{PMBM}}(\mathbf{x}) = \sum_{\mathbf{u} \biguplus \mathbf{v} = \mathbf{x}} p_{\mathbf{u}}^{\mathrm{PPP}}(\mathbf{u}) p_{\mathbf{v}}^{\mathrm{MBM}}(\mathbf{v}), \tag{2.6}$$

where $\mathbf{u}$ represents the set of undetected objects with PDF of form (2.2), and $\mathbf{v}$ represents the set of detected objects with PDF of form (2.5).

2.1.3 Transition and measurement models for multiple objects

We introduce transition and measurement models for multiple point objects [19]. Concretely, the object state $x \in \mathbb{R}^{n_x}$ evolves using an independent and identically distributed Markovian process with $\pi(x_{t+1}|x_t)$ being single-object transition density, where x_{t+1} and x_t denote object state vectors for time $t+1$ and t, respectively. Each $x \in \mathbf{x}_t$ has probability $p^S(x)$ to survive. In addition, new object birth process is modeled by a PPP with intensity $\lambda^{\mathrm{birth}}(x)$.

We proceed to describe the measurement model for multiple point objects. For every $x \in \mathbf{x}_t$, if it is detected with probability $p^D(x)$, we then use $g(z|x)$ to generate a measurement; Otherwise it results in a misdetection with probability $1 - p^D(x)$. Furthermore, measurements comprise object-generated measurements z and clutter measurements, where a PPP models the latter ones with intensity $\lambda^{\mathrm{clutter}}$. Since we are concerned with point objects in the thesis, we assume that at most one measurement is allowed to be generated by at most one object and vice versa.

2.1.4 The Set of trajectories

We adapt the trajectory state model proposed in [20, 24, 25]: the states of the trajectory are represented by

$$X = (t_s, x^{1:\kappa}), \tag{2.7}$$

where t_s, κ, $x^{1:\kappa}$, respectively, represent the trajectory's start time, the trajectory's length, and a sequence of object state vectors

$$x^1, x^2, \cdots, x^{\kappa-1}, x^{\kappa}. \tag{2.8}$$

This gives a single trajectory that starts from time step t_s and ends at time step $\kappa + t_s - 1$, where $x \in \mathbb{R}^4$. Further, let t be the current time, we are mainly interested in trajectories that have existed up to t, with state space defined as [20, 24, 25]

$$\mathcal{T}_t = \biguplus_{(t_s,\kappa)\in I_t} \{t_s\} \times \mathbb{R}^{n_x \kappa}, \tag{2.9}$$

where $\biguplus$ is the disjoint union operator and

$$I_t = \{(t_s, \kappa) \mid t_s \in \{0, \cdots, t\}, \kappa \in \{1, \cdots, t - t_s + 1\}\}. \tag{2.10}$$

By collecting a finite number of trajectories in space $\mathcal{T}_t$, we obtain a set of trajectory $\mathbf{X}_t$ at time t, and collecting all possible finite sets of trajectories in space $\mathcal{T}_t$ results in a space $\mathcal{F}(\mathcal{T}_t)$ [24, 25], where $\mathbf{X}_t \in \mathcal{F}(\mathcal{T}_t)$. Further, we can define integration over a single trajectory and a set of trajectories [1]. Concretely, let $f(\cdot)$ be a real-valued function on $\mathcal{T}_t$, we denote $\int f(X)dX$ as the integration over a single trajectory [26, 20]. If $f(\cdot)$ is non-negative and $\int f(X)dX = 1$, then $f(\cdot)$ is a single trajectory density of X.

Furthermore, if $f(\cdot)$ is real-valued but defined on $\mathcal{F}(\mathcal{T}_k)$, we denote $\int f(\mathbf{X})\delta\mathbf{X}$ as the integration over a set of trajectories [20]. Similarly, if $f(\cdot)$ is non-negative and $\int f(\mathbf{X})\delta\mathbf{X} = 1$, then f is a density of a RFS of trajectories [20].

Finally, if each trajectory in $\mathbf{X}_t$ either ends as t or before t, we call $\mathbf{X}_t$ the set of all trajectory [20] in the remaining parts of this thesis.

2.1.5 Trajectory RFS

The RFSs in Section 2.1.2 are also called target RFSs, where we are mainly concerned with object states [25]. In this section, we instead introduce trajectory RFS and instead focus on the set of trajectories. In what follows, we introduce some important trajectory RFSs, which are generalizations of target RFSs introduced in Section 2.1.2.

The density of a trajectory Poisson RFS (or PPP) is [24]

$$f^{\text{PPP}}(\mathbf{X}) = e^{-\int D(X')dX'} \prod_{X\in\mathbf{X}} D(X), \tag{2.11}$$

where $D(\cdot)$ is its intensity function.

The density of a trajectory Bernoulli RFS is [24]

$$f^{\text{Ber}}(\mathbf{X}) = \begin{cases} 0, & \text{if} \quad |\mathbf{X}| > 1 \\ rf(X), & \text{if} \quad \mathbf{X} = \{X\} \\ 1 - r, & \text{if} \quad \mathbf{X} = \emptyset \end{cases}, \tag{2.12}$$

[1] The mathematical details of the integration can be found in [20].

where r is the existence probability and $f(\cdot)$ is the density of a single trajectory .

The trajectory MB RFS is made up of multiple trajectory Bernoulli RFSs, and it has a density [24]

$$f^{\mathrm{MB}}(\mathbf{X}) = \sum_{\biguplus_{i=1}^{N} X_i = X} \prod_{i=1}^{N} . f_i^{\mathrm{Ber}}(X_i), \tag{2.13}$$

The trajectory MBM RFS is made up of trajectory MB RFS densities, and it has a density [24, 20]

$$p^{\mathrm{MBM}}(\mathbf{X}) = \sum_{h=1}^{\mathcal{H}} w_h f_h^{\mathrm{MB}}(\mathbf{X}), \tag{2.14}$$

where $\sum_{h=1}^{\mathcal{H}} w_h = 1$.

Finally, the trajectory PMBM RFS has a density [20, 8]

$$f^{\mathrm{PMBM}}(\mathbf{X}) = \sum_{\mathbf{U} \biguplus \mathbf{V} = \mathbf{X}} f^{\mathrm{PPP}}(\mathbf{U}) f^{\mathrm{MBM}}(\mathbf{V}), \tag{2.15}$$

which comprises a trajectory PPP and a trajectory MBM RFS.

2.1.6 Multi-trajectory transition and measurement model

Similar to Section 2.1.3 where we have introduced the Bayesian models for objects, we briefly mention the Bayesian models for the set $\mathbf{X}$ of all trajectories. Each trajectory $X \in \mathbf{X}$ evolves using the Markov system with density $\pi'(\cdot \mid X)$ described in [20]. Newborn trajectories are modeled using the trajectory Possion RFS with $D_{\mathrm{birth}}(X)$ as intensity function, and each trajectory $X \in \mathbf{X}$ remain in $\mathbf{X}$ with a survival probability $P^S(X)$ [20].

For each trajectory $X \in \mathbf{X}$, if it is detected with probability $p^D(X)$, then a measurement is generated using $l(\cdot|X)$; Otherwise it results in a misdetection with probability $1 - p^D(X)$. It is worth noting that the functions introduced in this section so far are defined on a single trajectory X instead of sets of states in Section 2.1.3 despite the overall structures being similar. Furthermore, The clutter model is the same as defined in Section 2.1.3.

2.1.7 Hypotheses and track in MOT

We introduce the concept of local hypothesis, global hypothesis, and track as they arise in deriving the filter used in this thesis. Definitions 1-3 are taken from [1].

Definition 1 (Global Hypothesis) *Given several potential objects along with a set Z contains measurements collected up to the current time, a global hypothesis is a way to divide Z into subsets and assign each subset to a particular object by*

assuming the subset of measurements is associated with that object.

Definition 2 (Local Hypothesis) *A local hypothesis is a subset of measurements such that each measurement is assigned to the same object by assuming that all the measurements within that set are associated with that object.*

First of all, the global hypothesizes are made up of local hypotheses, where each local hypothesis contains the information of measurements that are associated with a particular object under the hypothesis, the associated hypothesis weight, and the Bernoulli distribution of the form (2.12) associated with the hypothesis, parameterized by the existence probability and the PDF [1].

Definition 3 (Track) *A track is a set of local hypotheses related to an object since it was detected, which contains information on associating measurements to that object in different possible ways.*

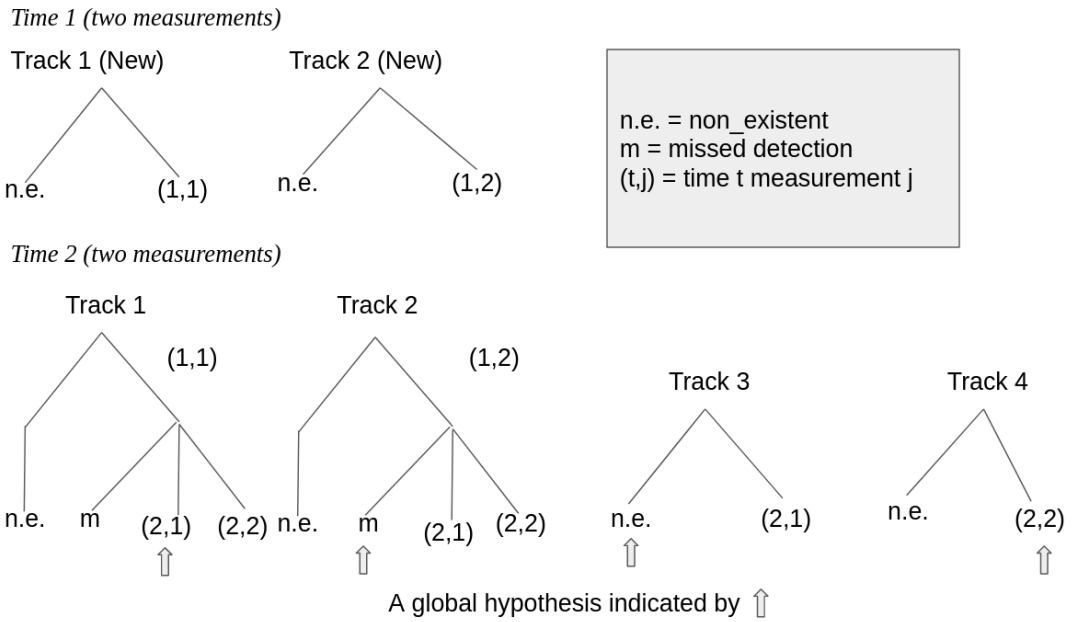

Figure 2.1: The illustration of an example of local hypothesis, global hypothesis and track. The figure is inspired by [1, Figure 1].

We use a small example to combine the concepts introduced in this section, shown in Figure 2.1. We start with 2 measurements time step 1, each of which gives rise to a new track by updating the PPP of unknown objects [1]. As shown in the figure, each new track contains two hypotheses:

- Non-existence (n.e. for short) hypothesis: Under this hypothesis, the measurement follows one of the previous tracks, so the corresponding new track will not be associated with this new measurement.
- (t, j) hypothesis: Under this hypothesis, the measurement follows new track.

Here, t denotes the time step while j is the the measurement index, where $t \in \{1, 2\}$ and $j \in \{1, 2\}$.

For time step 2, we also assume that two measurements are received. Again, they give rise to two new tracks, which behave similarly to track 1 and track 2 at time 1. The $(1, 1)$ hypothesis of track 1 and $(1, 2)$ hypothesis of track 2 all branch into 3 cases: one for missed detection and the others for the new measurements. The n.e. hypothesis at time 2 is not branching. A sample of a global hypothesis is shown in Figure 2.1, where every measurement in every step is used precisely once [1].

2.1.8 Trajectory Poisson multi-Bernoulli (TPMB) filter

We apply the TPMB filter to process the measurements collected by the associated sensor and output trajectory estimates. Concretely, let t be the current time, given a set of measurements $\mathbf{z_t} = \{z_t^1, \cdots, z_t^{m_t}\}$, we would like to find the TPMB posterior density of $\mathbf{X}_t$ (see Section 2.1.4 for the meaning of $\mathbf{X}_t$) and extract trajectory estimates from the posterior.

Now, let $t' \in \{t, t+1\}$, the TPMB density is denoted as [20]

$$f_{t'|t}\left(\mathbf{X}_{t'}\right) = \sum_{\uplus_{l=1}^{t'|t}\mathbf{X}^l \uplus \mathbf{Y} = \mathbf{X}_{t'}} f_{t'|t}^{\mathrm{PPP}}(\mathbf{Y}) \prod_{i=1}^{n_{t'|t}} \left[f_{t'|t}^{\mathrm{Ber},i}\left(\mathbf{X}^i\right)\right]. \tag{2.16}$$

Here, $f_{t'|t}^{\mathrm{PPP}}$ is the density of trajectory Poisson RFS defined in (2.11), specified by the intensity $D_{t'|t}(\cdot)$. In addition, $f_{t'|t}^{\mathrm{Ber},i}$ is the i-th Bernoulli component of the MB component in (2.16), and there are $n_{t'|t}$ such components in total. Further, $f_{t'|t}^{\mathrm{Ber},i}$ is specified by the existence probabilities $r_{t'|t}^i$ and the single trajectory density $f_{t'|t}^i$ in (2.12), where $i \in n_{t'|t}$.

The TPMB filtering recursion for sets of all trajectories comprises a TPMB prediction followedby a TPMB update. In particular, TPMB prediction step and TPMB update step introduced in this section are based on **Lemma 4** and **Lemma 5** in [20]. In each recursion, we will start with a PMB density (see (2.16)), where the resulting density after a prediction step is a PMB and the resulting density after an update step a TPMBM (see equation (2.15)). Concretely, for the prediction step at time $t-1$, we assume a TPMB density defined the same way as (2.16), specified by parameters:

- $D_{t-1|t-1}$: the intensity of the trajectory Poisson RFS.
- $n_{t-1|t-1}$: the number of Bernoulli components.
- $r_{t-1|t-1}^i$: the existence probabilities of $f_{t-1|t-1}^{\mathrm{Ber},i}$.
- $f_{t-1|t-1}^i$: the single trajectory density of $f_{t-1|t-1}^{\mathrm{Ber},i}$.

The prediction step updates the above parameters as follow [20]:

$$D_{t|t-1}(X) = D_t^{\text{birth}}(X) + \int D_{t-1|t-1}(X)\pi_t'(X \mid \cdot)P^S(\cdot)dX \tag{2.17}$$

$$n_{t|t-1} = n_{t-1|t-1} \tag{2.18}$$

$$r_{t|t-1}^i = r_{t-1|t-1}^i \int f_{t-1|t-1}^i(X)P^S(X)dX \tag{2.19}$$

$$f_{t|t-1}^i(X) = \frac{\int f_{t-1|t-1}^i(X)\pi_t'(X \mid \cdot)P^S(\cdot)dX}{\int f_{t-1|t-1}^i(X)P^S(X)dX} \tag{2.20}$$

where $D_k^{\text{birth}}(\cdot)$, $\pi_t'(\cdot)$ and $P^S(\cdot)$ follow the definition in Section 2.1.6 and X a single trajectory.

For the update step at time t, given a set of measurements $\mathbf{z_t} = \{z_t^1, \cdots, z_t^{m_t}\}$ received from a sensor at time step t as well as the predicted TPMB density specified by updated parameters of form (2.18)-(2.20), the resulting density after the update is a trajectory TPMBM [20], which is made up of a trajectory PPP RFS and a trajectory MBM RFS.

The intensity of the trajectory PPP and the number of Bernoulli components are updated by

$$D_{t|t}(X) = D_{t|t-1}(X)\left(1 - P_k^D(X)\right) \tag{2.21}$$

$$n_{t|t} = m_t + n_{t|t-1} \tag{2.22}$$

where $P_t^D(X)$ is also defined in Section 2.1.6. For each Bernoulli component $f_{t|t-1}^{\text{Ber},i}(\cdot)$ in (2.16), $i \in \{1, \cdots, n_{t|t-1}\}$, there are m_t+1 local hypotheses [20], where m_t of them correspond to associating the component to m_t different measurements, and we plus one as we also consider the misdetection hypothesis. Concretely, if the Bernoulli component i is under the misdetection hypothesis , then the updated parameters are:

$$w_{t|t}^{i,1} = 1 - r_{t|t-1}^i \int f_{t|t-1}^i(X)P_t^D(X)dX \tag{2.23}$$

$$r_{t|t}^{i,1} = \frac{r_{t|t-1}^i \int f_{t|t-1}^i(X)(1 - P_t^D(X))dX}{1 - r_{t|t-1}^i \int f_{t|t-1}^i(X)P_t^D(X)dX} \tag{2.24}$$

$$f_{t|t}^{i,1}(X) = \frac{\left(1 - P_t^D(X)\right) f_{t|t-1}^i(X)}{\int f_{t|t-1}^i(X)(1 - P_t^D(X))dX} \tag{2.25}$$

If the Bernoulli component i is associated with measurement z_t^j, $j \in \{1, \cdots, n_{t|t-1}\}$, then the updated parameters are:

$$w_{t|t}^{i,1+j} = r_{t|t-1}^i \int f_{t|t-1}^i(X)l\left(z_t^j \mid \cdot\right) P_t^D(\cdot)dX \tag{2.26}$$

$$r_{t|t}^{i,1+j} = 1 \tag{2.27}$$

$$f_{t|t}^{i,1+j}(X) = \frac{l\left(z_t^j \mid X\right) P_t^D(X) f_{t|t-1}^i(X)}{\int f_{t|t-1}^i(X) l\left(z_t^j \mid \cdot\right) P_t^D(\cdot) dX} \tag{2.28}$$

Here, $w_{t|t}^{i,1}$ and $w_{t|t}^{i,1+j}$ are the associated hypothesis weights, and $l(\cdot)$ is the measurement model defined in Section 2.1.6.

New measurements give rise to new Bernoulli component (or new track) with indice i, $i \in \{n_{t|t-1}+1, \cdots, n_{t|t-1}+m_t\}$. In particular, if measurement z_t^j, $j \in \{1, \cdots, m_t\}$ give rises to component i, there are 2 local hypotheses:

$$w_{t|t}^{i,1} = 1, r_{t|t}^{i,1} = 0 \tag{2.29}$$

and

$$w_{t|t}^{i,2} = \lambda^C\left(z_t^j\right) + \int D_{t|t-1}, l\left(z_t^j \mid \cdot\right) P_t^D(\cdot) dX \tag{2.30}$$

$$r_{t|t}^{i,2} = \frac{\int D_{t|t-1}(X) l\left(z_t^j \mid \cdot\right) P_t^D(\cdot) dX}{\lambda^C\left(z_t^j\right) + \int D_{t|t-1}(X) l\left(z_t^j \mid \cdot\right) P_t^D(\cdot) dX} \tag{2.31}$$

$$f_{t|t}^{i,2}(X) = \frac{l\left(z_t^j \mid X\right) P_t^D(X) D_{t|t-1}(X)}{\int D_{t|t-1}(X) l\left(z_t^j \mid \cdot\right) P_t^D(X\cdot) dX} \tag{2.32}$$

$$\tag{2.33}$$

where $\lambda^C(\cdot)$ is the clutter intensity and (2.29) refers to the non-existence hypothesis introduced in Section 2.1.7.

As the updated density is a TPMBM, the best TPMB fit of a TPMBM is obtained by minimizing the Kullback-Leibler divergence (KLD) on an augmented space [20]. Then, from the resulting density, we can extract trajectory estimates from the Bernoulli component with an existence probability larger than a predefined threshold. In this thesis, we use the TPMB filter for Gaussian densities, and the resulting TPMB density consists of the start time of the trajectory, the probability if the trajectory ends at a particular time, and the mean as well as the covariance matrix of the trajectory conditional on terminating on that specific time. Further, we note that the PMB density for the objects' states at the current time can be obtained by marginalizing out previous object states from the TPMB density.

2.1.9 Model-based decentralized fusion

In decentralized fusion, object state estimates and their densities from local trackers are fused in a global filter to construct a better estimation. A significant challenge here is to leverage information obtained by independent trackers while guaranteeing that the unknown common information is not double-counted.

In our thesis, the model-based decentralized fusion is formulated as combining N_s PMB posterior densities $f^{\mathrm{PMB},1}(\mathbf{x}|\mathbf{z}^1), \cdots, f^{\mathrm{PMB},N_s}(\mathbf{x}|\mathbf{z}^{N_s})$, obtained from N_s independent TPMB filters, to get a global PMB posterior $\overline{f}^{\mathrm{PMB}}_{\mathrm{global}}(\mathbf{x}|(\mathbf{z}^1, \ldots, \mathbf{z}^{N_s}))$ over

the set of current objects, with unknown prior densities. Here, $\mathbf{z}^i$ is the set of measurements from sensor i, $i \in \{1, \cdots, N_s\}$. The problem is tackled by minimizing the Kullback-Leibler average (KLA) between $f^{\text{PMB}}_{\text{global}}(\mathbf{x}|(\mathbf{z}^1, \ldots, \mathbf{z}^{N_s}))$ and $f^{\text{PMB},i}(\mathbf{x}|\mathbf{z}^i)$ for all $i \in \{1, \cdots, N_s\}$ [8]:

$$\bar{f}^{\text{PMB}}_{\text{global}} = \arg \inf_{f^{\text{PMB}}_{\text{global}}} \sum_{s=1}^{N_s} w_s \text{KLD}\left(f^{\text{PMB}}_{\text{global}} \| f^{\text{PMB},s}\right), \tag{2.34}$$

where $w_s \in [0, 1]$ such that $\sum_s w_s = 1$, and KLD stands for the Kullback-Leibler divergence, defined as:

$$\text{KLD}\left(f^{\text{PMB}}_{\text{global}} \| f^{\text{PMB},s}\right) = \int f^{\text{PMB}}_{\text{global}}\left(\mathbf{x} \mid (\mathbf{z}^1, \ldots, \mathbf{z}^{N_s})\right) \log \frac{f^{\text{PMB}}_{\text{global}}\left(\mathbf{x} \mid (\mathbf{z}^1, \ldots, \mathbf{z}^{N_s})\right)}{f^{\text{PMB},s}\left(\mathbf{x} \mid \mathbf{z}^s\right)} \delta\mathbf{x}. \tag{2.35}$$

One challenge for fusing PMB densities is that sensors may have different FoVs. For instance, an object may be in the FoV of one sensor but outside the other one, where we can model the former and latter cases using Bernoulli RFS and PPP, respectively. Consequently, we are facing a situation of fusing a Bernoulli with partial PPP intensity, resulting in incorrect solutions when minimizing the KLA [8]. One way to deal with this is by dividing the PPP intensity of each PMB density into many independent PPP intensities (with their sum equal to the original intensity), such that the total number of components (PPP intensities + Bernoulli components) is the same for all PMB densities. We assume there are K such components, then the PMB density (2.16) can be written as

$$f^{\text{PMB},s}(\mathbf{x}) = \sum_{\cup_{i=1}^{K} \mathbf{x}^i = \mathbf{x}} \prod_{i=1}^{K} f^i_s\left(x^i\right), \tag{2.36}$$

where the condition on measurements is dropped for simplicity and f^i_s is the i-th component of $f^{\text{PMB},s}$, which can be either a PDF of PPP (2.11) or a Bernouli RFS (2.12). In this case, we can match any component $f^i_s(\cdot)$ of the PMB density $f^{\text{PMB},s}$ from sensor s to any other component $f^j_{s'}(\cdot)$ of the PMB density $f^{\text{PMB},s'}$ from sensor s' [8], where $i, j \in \{1, \cdots, K\}$ and $s, s' \in \{1, \cdots, N_s\}$.

To perform the decentralized fusion in practice, we can assume that the fused density $\overline{f}^{\text{PMB}}_{\text{global}}(\mathbf{x})$ also has form (2.36)

$$\overline{f}^{\text{PMB}}_{\text{global}}(\mathbf{X}) = \sum_{\cup_{i=1}^{K} \mathbf{x}^i = \mathbf{x}} \prod_{i=1}^{K} \overline{f}^i\left(x^i\right), \tag{2.37}$$

where $\overline{f}^i(\cdot)$ is its i-th fused components, $i \in \{1, \cdots, K\}$. The KLA (2.34) can be approximated by minimizing

$$\sum_{s=1}^{N_s} w_s \left[\sum_{i=1}^{K} \text{KLD}\left(\bar{f}^i(\mathbf{x}) || f^{\sigma^*_s(i)}_s(\mathbf{x})\right)\right], \tag{2.38}$$

where $\bar{f}^i(\mathbf{x})$ and $f^{\sigma^*_s(i)}_s$ are the i-th and $\sigma^*_s(i)$-th parts of $\overline{f}^{\text{PMB}}_{\text{global}}$ and $f^{\text{PMB},s}$, respectively. Here, both i and $\sigma^*_s(i)$ belong to $\{1, \cdots, K\}$. In addition, $\sigma^*_s()$ is an optimal

permutation of the component index set $\{1, \cdots, K\}$ of density $f^{\mathrm{PMB},s}$, such that $\bar{f}^i$ and $f_s^{\sigma_s^*(i)}$ are matched if they are similar in principle of the KLD [8], where the permutation can be defined as a function:

$$\sigma : i \in \{1, \cdots, K\} \to j \in \{1, \cdots, K\}, \tag{2.39}$$

where each i can be assigned at most once. The optimal permutation is to reduce the computational complexity caused by dividing the PPP intensities. The optimal permutation is also called the best possible fusion map [8], which is typically formulated as an optimal assignment problem and solved by the Munkres algorithm [27].

The solution to (2.38) is computed for each component $\bar{f}^i$ of $\overline{f}^{\mathrm{PMB}}_{\mathrm{global}}$ [8]:

$$\bar{f}^i(\mathbf{X}) = \frac{\prod_{s=1}^{N_s} \left(f_s^{\sigma_s^*(i)}(\mathbf{X}) \right)^{w_s}}{\int \prod_{s=1}^{N_s} \left(f_s^{\sigma_s^*(i)}(\mathbf{X}) \right)^{w_s} \delta\mathbf{X}}. \tag{2.40}$$

After obtaining the K components of $\overline{f}^{\mathrm{PMB}}_{\mathrm{global}}$, all the PPP components are merged to one PPP, and any Bernoulli component with low existence probability will be recycled [8].

2.2 Multi-object tracking transformers

The multi-object tracking transformers (MOTTs) are transformer-based trackers that aim to solve MOT tasks. In the upcoming sections, we will mainly introduce two MOTTs: MT3 [2, 3] and MT3v2 [17], after a brief review of the Transformer [23].

2.2.1 Transformer

The Transformer [23] was initially developed to solve machine translation tasks. The overall structure of the Transformer (c.f. Figure 2.2) is made up of an encoder and a decoder. The input data is transfomed to a new representation by the encoder via the attention mechanism. At the same time, the decoder uses the incorporated contextual information [28] from the representation to generate an output sequence.

2.2.2 The attention mechanism

Motivated by human biological systems, the attention mechanism is the critical component in constructing the Transformer model. It allows the model to concentrate on the distinctive parts when dealing with large amounts of information [29] and model the long-range dependencies [2]. Loosely put, the attention mechanism can be viewed as a retrieval system, which stores a set of values with associated keys. We then query the key and output the value. Further, every query element can interact with every key, allowing the Transformer to infer the context of the input [3]. Regarding the attention mechanism, a weight calculated by finding the similarity between a query and the corresponding key is assigned to each value. we multiply

values with their weights and sum the results to obtain the finial output.

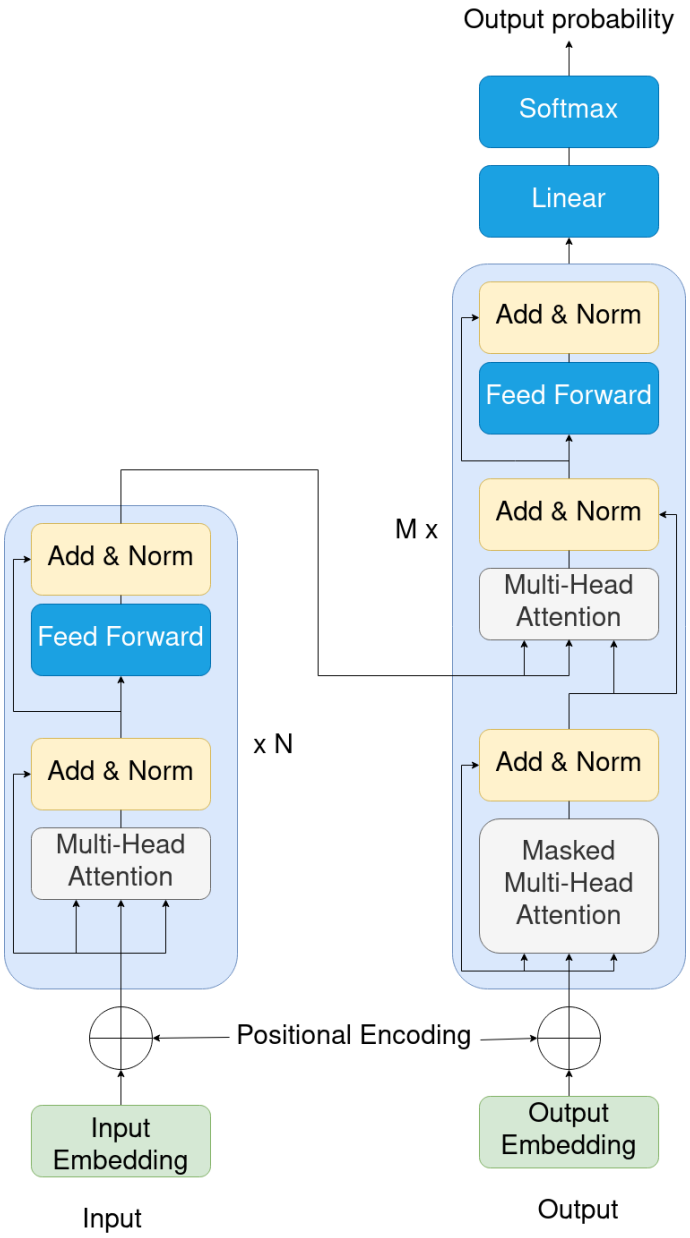

Figure 2.2: The structure of the Transformer is made up of an encoder and a decoder. The encoder consists of N identical layers. Every encoder layer comprises a multi-head attention sub-layer, an FFN, residual connection, and normalization layers. The decoder also has M identical layers, each of which has a similar structure as the encoder layer but with another multi-head attention module inserted.

Concretely, given a sequence of input vectors, we map each vector to the query vector q, the key vector k, and the value vector v using a linear transformation. We then construct three different matrices using the transformed vectors, Q, K, and V. Next, we compute attention by multiplying Q with K, then scaling the results.

Subsequently, we pass the scaled results to a softmax function, which is defined as

$$\text{softmax}(s_j) = \frac{e^{s_j}}{\sum_{i=1}^{n} e^{s_i}}, j \in \{1, \cdots n\}. \tag{2.41}$$

At last, we multiply the result from the softmax function with V to obtain the attention. The above calculation refers to "Scaled Dot-Product Attention", shown on the right of Figure 2.3, and we perform scaling to avoid the softmax function generating minimal gradients [23].

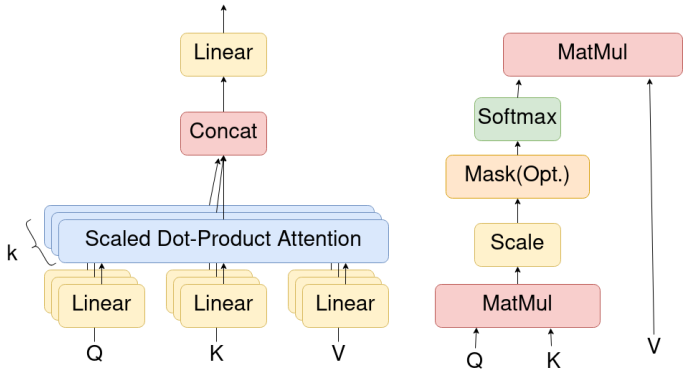

Figure 2.3: The multi-head attention and scaled dot-product attention.

In practice, the scaled dot-product attention is usually performed several times in parallel by a multi-head attention process to boost the performance, shown in the left of Figure 2.3. Suppose we have h heads and a sequence of transformed input vectors. For each head $i \in \{1, \cdots, h\}$, we pack the inputs to Q, K, and V, and perform the scaled dot-product attention at each head. Next, we concatenate all the results obtained from different heads and reduce the final result's dimensions by multiplying it with a matrix of suitable dimensions [23]. Applying multi-head attention can widen our ability to concentrate on one or more specific locations in the sequence without affecting the attention on other equivalently important locations in the meantime [28]. In practice, by using efficient matrix multiplication code, the multi-head attention can be implemented in a faster and space-saving manner[23].

2.2.3 The encoder and decoder architecture

The Transformer consists of an encoder and a decoder. The encoder has N identical encoder layers and encodes the input to a new representation. Every encoder layer comprises a multi-head attention sub-layer, an FFN, residual connection, and normalization layers. All these components are connected as shown on the left of Figure 2.2. As the previous layer's output is fed to the subsequent layer, the output of the encoder can represent all the relevant dependencies of the original input sequence.

The decoder also has a stack of M identical layers, and it maps the initial output embeddings to the output sequence based on the representation from the encoder. Each decoder layer has a similar structure as the encoder layer but with another

multi-head attention module inserted (c.f. Figure 2.2), and it computes the multi-head attention on representations obtained from the encoder. Similarly, the residual connections and layer normalization are employed in each decoder layer.

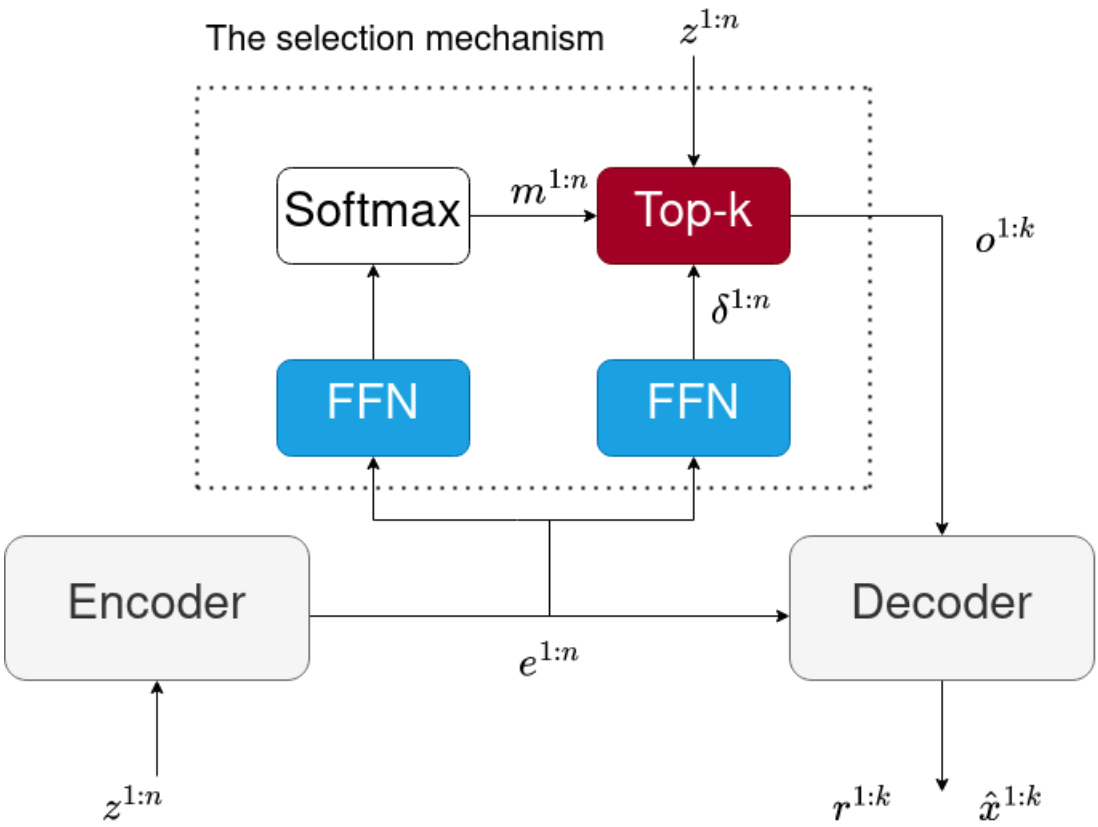

Figure 2.4: The structure of the MT3 is made up of an encoder, a selection mechanism, and a decoder. The encoder takes the measurement sequence $z^{1:n}$ and outputs new representations $e^{1:n}$ just like the transformer, which is fed to a selection mechanism to produce the object queries $o^{1:k}$ and initial predictions $\tilde{z}^{1:k}$. Next, based the above information, the decoder predicts the existence probabilities $p^{1:k}$ and object states $\hat{x}_T^{1:k}$ at time T.

2.2.4 Positional encoding

As stated early, the Transformer was originally developed to solve tasks within the field of NLP. In such a setting, the inputs to the Transformer are often sentences in different languages, and changing the order of words in a sentence may affect the sentence meaning significantly. However, the Transformer is permutation-invariant, meaning that it is not sensitive to the word order of input sentences. To use the order information contained in the input sequence, the authors in [23] proposed positional encoding, which is a function denoted as $\phi(\cdot)$. It maps the index i of a vector c^i of the sequence of vectors $c^{1:n}$ to a unique embedding vector ϵ^i:

$$\epsilon^i = \phi(i), \tag{2.42}$$

where ϵ^i typically has the same shape as the vector c_i such that they can (and will) be added before feeding them to the network, allowing the model to acquire the order information associated with the input. The positional encoding can be either absolute or learnable, where the authors in [23] found these two versions produced nearly identical results.

2.2.5 MT3

The MT3 is a transformer-based deep neural network that aims to solve the MOT tasks and an overview of its structure is shown in Figure 2.4. The MT3 takes a sequence of measurements $z^{1:n}$ as inputs and processes them by a Transformer architecture. The encoder first maps the measurements $z^{1:n}$ to a new representations (also called embeddings) $e^{1:n}$. We then feed these representations $e^{1:n}$, together with initial predictions $\tilde{z}^{1:k}$ (see Section 2.2.8) as well as the object queries $o^{1:k}$ (see Section 2.2.7) produced by a selection mechanism (see Section 2.2.8), to a decoder to predict the object positions $\hat{x}_T^{1:k}$ and existence probability $p^{1:k}$ at the time step T.

2.2.6 Preprocessing

Data prepossessing is a process in the machine learning task of preparing the raw data in a format such that it is suitable for the network. In training MT3, three prepossessing methods are applied. First, each z^i of $z^{1:n}$ is normalized to be within $[0, 1]$ according to the given FoVs information. Next, each measurement vector z^i is projected linearly into a higher dimensional vector z'. Lastly, the time step indices are used to generate the positional encodings. It is done by implementing a learnable temporal encoder, which maps the time step index $j \in \{1, ..., t\}$ to its embedding vector ϵ^j, which will then serve as the positional encodings.

2.2.7 Object queries

The authors of MT3 made use of a collection of random initialized learnable positional encodings $\mathbb{Q} \in \mathbb{R}^{n_q \times n_m}$, called object queries [30], as the input to the MT3 decoder to make its predictions. Here, n_q is the number of object queries, and n_m is a model parameter, commonly chosen to be 256. At each decoder layer, object queries are refined and acquire useful knowledge from the encoder output during training [2]. Further, the authors in [30] claim that the model tends to reuse a given object query to predict objects in a given area.

2.2.8 The Selection mechanism and the iterative refinement

The MT3 has two processes to improve its performance: the selection mechanism and the iterative refinement of the state. For the selection mechanism, we start with the measurements $z^{1:n}$, and among all these measurements, n_q of them are selected as the starting points, which the decoder will use later. Concretely, given the embedding sequence $e^{1:n}$, a score m_i is computed for each embedding e^i by :

$$m_i = \text{softmax}(\text{FFN}(e^i)), \ i \in \{1, \dots, n\}. \tag{2.43}$$

We then sort the scores $m_{1:n}$ in the descending order:

$$m_{r_1} \geq m_{r_2} \geq \cdots \geq m_{r_i} \geq \cdots \geq m_{r_{n-1}} \geq m_{r_n}, \tag{2.44}$$

where $r_i \in \{1, \ldots, n\}$ is the associated index with score m_{r_i} after sorting. Next, We pick n_q measurements $z^{r_1:r_{n_q}}$ according to the first n_q highest scores $m_{r_1:r_{n_q}}$ and their corresponding indexes $r_1 : r_{n_q}$ by the above inequality (2.44).

For more flexibility, an adjustment δ_{r_i} is calculated by $\delta_{r_i} = \text{FFN}(e^{r_i})$ and added with the selected measurement z_{r_i} to produce the initial measurements $\tilde{z}^{r_i}$ for the decoder:

$$\tilde{z}^{r_i} = z^{r_i} + \delta^{r_i}. \tag{2.45}$$

The object queries $o^{1:n_q}$ as well as decoder positional encodings $\epsilon^{1:n_q}$ can be obtained by passing $e^{r_1:r_{n_q}}$ to two different FFNs. Instead of predicting the estimates directly, we use the initial measurements $\tilde{z}^{r_1:r_{n_Q}}$ obtained by the above selection mechanism and treat them as the initial estimates. We then keep refining them with additional adjustments, and this process is called iterative refinement of state. Concretely, at each decoder layer l (assuming there are L layers in total), we feed the outputs from the last layer to an FFN to predict an offset $\mathbf{\Delta}^{i,l}$, and we sum all the offsets with the initial estimates to produce the estimates:

$$\hat{x}_T^{i,L} = \tilde{z}^i + \sum_{l=1}^{L} \mathbf{\Delta}^{i,l}. \tag{2.46}$$

One should note that only the states (positions) here are iteratively refined while the existence probabilities at decoder layer l, however, are directly computed by passing $\hat{x}_T^{i,l}$ to an FFN:

$$p^{i,l} = \text{FFN}(\hat{x}_T^{i,l}), \tag{2.47}$$

where $\hat{x}_T^{i,l}$ is the output of decoder layer l.

2.2.9 Loss function

The MT3 utilizes a set prediction loss similar to [30], where it first seeks an optimal permutation of the estimates (MT3 output) such that the estimates and targets are matched in the principle of the lowest matching loss and then optimizes object-specific losses. To illustrate, in Figure 2.5, 5 objects together with their indices are shown in the circle, and the same number of estimates are shown in triangular. The numbers are their associated indexes. Assume the arrows indicate the correct matching, and a permutation of the estimates is performed so that the objects and estimates can be grouped as pairs as shown in Figure 2.5. For example, we can compute the $L2-$norm based on the associated object-estimate pairs.

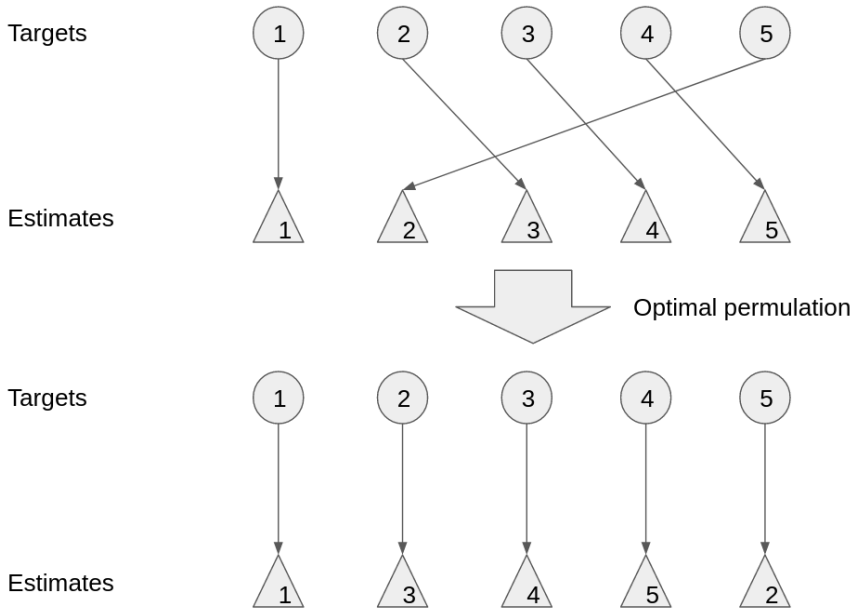

Figure 2.5: An example of matching. We have five objects (shown in circles) and five estimates (shown in triangular). The numbers within them are indexes. The arrows indicate the real matchings. We search for an optimal permutation (1,3,4,5,2 shown in this example) with the lowest costs.

When performing the matching in practice, we may assume the same number of estimates and ground truth since MT3 infers a fixed number of n_q estimates (n_q is the number of object queries, see Section 2.2.7), which is typically more than the number of ground truth. Therefore, if the estimates at time T in layer l are:

$$\boldsymbol{\varpi}^{1:n_q,l} = (\hat{x}_T^{1,l}, \cdots, \hat{x}_T^{n_q,l}) \tag{2.48}$$

and the ground-truth object states are:

$$\boldsymbol{\varrho}^{1:|\mathbb{X}^T|} = (x_T^1, \cdots, x_T^{|\mathbb{X}^\mathbb{T}|}), \tag{2.49}$$

we first pad the ground-truth $\boldsymbol{\varrho}^{1:|\mathbb{X}^T|}$ with $\emptyset$ to increase its cardinality to n_q (such that both the estimates and objects now have the same number of elements). Next, we search for a permutation σ_*^l that gives the minimum matching cost:

$$\sigma_*^l = \arg\min \sum_{i=1}^{n}{}_q \Phi(\boldsymbol{\varpi}^{i,l}, \boldsymbol{\varrho}^{\sigma(i)}), \tag{2.50}$$

where σ is defined the same way as (2.39) in Section 2.1.9 and $\Phi(\cdot)$ is the matching loss, defined as [3]:

$$\Phi(\boldsymbol{\varpi}^{i,l}, \boldsymbol{\varrho}^{\sigma(i)}) = \mathbb{1}_{\boldsymbol{\varpi}^{i,l}=\emptyset}(\|\hat{x}_T^{i,l} - x_T^i\| - p^{i,l}), \tag{2.51}$$

and $p^{i,l}$ is the corresponding existence probability of estimate $\hat{x}_T^{i,l}$. Here, $\mathbb{1}_{\boldsymbol{\varpi}^{i,l}=\emptyset}$ is an indicator function of form

$$\mathbb{1}_{\boldsymbol{\varpi}^{i,l}=\emptyset} = \begin{cases} 1, & \text{if} \quad \boldsymbol{\varpi}^{i,l} = \emptyset \\ 0, & \text{if} \quad |\boldsymbol{\varpi}^{i,l}| > 0 \end{cases}. \tag{2.52}$$

The optimal permutation is found by reformulating the problem as a 2D assignment problem, which is solved by, e.g., Munkres Algorithm [27]. After the optimal matching, the final loss for training is defined as [3]:

$$\mathcal{L}_T(\boldsymbol{\varpi}^{1:n_q,1:M}, \boldsymbol{\varrho}^{1:n_q}) = \sum_{l=1}^{M} \sum_{i=1}^{n_q} \Psi(\boldsymbol{\varpi}^{i,l}, \boldsymbol{\varrho}^{\sigma_*^l(i)}) \tag{2.53}$$

where $\boldsymbol{\varrho}^{1:n_q}$ is the ground-truth after padding and Ψ is defined as [3]:

$$\Psi(\boldsymbol{\varpi}^{i,l}, \boldsymbol{\varrho}^{j}) \;=\; (1 - \mathbb{1}_{\boldsymbol{\varrho}^{j}=\emptyset})(\|\hat{x}_T^{i,l} - x_T^i\| - \log(p^{i,l})) - \log(1 - p^{i,l})\mathbb{1}_{\boldsymbol{\varrho}^{j}=\emptyset}, \tag{2.54}$$

where $\mathbb{1}_{\boldsymbol{\varrho}^{j}}$ is similar to (2.52). As can be seen from equation (2.53), instead of just using the outputs from the finial decoder layer to compute the loss, outputs from all M decoder layers have made contributions, which has been shown to accelerate learning [3, 17].

2.2.10 MT3v2

MT3v2 is a variant of MT3 and there are two major differences between them. First of all, MT3 predicts the estimated positions of objects and their existence probabilities, $(\hat{x}_T^{1:n_q,l}, p^{1:n_q,l})$, at the current time T, while MT3v2 outputs estimated positions, velocities, covariance matrices and existence probabilities at the current time T, denoted as $\beta^{1:n_q,l} = (\mu_T^{1:n_q,l}, \Sigma_T^{1:n_q,l}, p^{1:n_q,l})$ (These are n_q Bernoulli components which can be regarded as parameters specifying a MB density (2.4) with Gaussian spatial distributions).

Concretely, for $i \in \{1, \cdots, n_q\}$, the estimated states $\mu_T^{i,l}$ (positions and velocities) are iteratively refined according to equation 2.46:

$$\mu_T^{i,l} = \mu_T^{i,0} + \sum_{l=1}^{M} \boldsymbol{\Delta}^{i,l} \tag{2.55}$$

through M decoder layers. Moreover, existence probability $p^{i,l}$ is computed by feeding $\mu_T^{i,l}$ to an FFN, then passing the result to a sigmoid layer. The covariance matrix $\Sigma_T^{i,l}$ is a diagonal matrix, and computed by the following equation [17]:

$$\Sigma_T^{i,l} = \boldsymbol{F}(\varkappa(\mathrm{FFN}(\mu_T^{i,l}))), \tag{2.56}$$

where $\varkappa$ is defined as

$$\varkappa(x) = \log(e^x + 1), \tag{2.57}$$

and $\boldsymbol{F}$ is a function that maps a vector of $\mathbb{R}^n$ to a diagonal matrix of $\mathbb{R}^{n \times n}$ (the elements contained in that vector will be the entries along the matrix diagonal).

The second difference is that the loss function of MT3v2 is based on the negative-log likelihood (NLL) [31] (see Section 3.4.2). Concretely, given MT3v2 predictions $\boldsymbol{\varpi}^{1:n_q,1:M} = (\beta^{1:n_q,1}, \cdots, \beta^{1:n_q,M})$ of all decoder layers and the ground-truth object states $\boldsymbol{\varrho}^{1:n} = (x^1, \cdots, x^n)$, we have the training loss:

$$\mathcal{L}_T(\boldsymbol{\varpi}^{1:n_q,1:M}, \boldsymbol{\varrho}^{1:n}) = -\sum_{l=1}^{M} \log f^{\mathrm{MB},l}(\boldsymbol{\varrho}^{1:n}), \tag{2.58}$$

where $f^{\mathrm{MB},l}(\cdot)$ is a multi-Bernoulli density of form (2.4) parameterized by $\beta^{1:n_q,l}$, $l \in \{1, \cdots, M\}$. However, if all possible data associations of the ground-truth and the MB components are included in the computation of the NLL loss (2.58), it would become intractable, and approximation of the MB density $\log f^{\mathrm{MB},l}(x^{1:m})$ is therefore needed [17]. Similar to MT3, we first pad the ground-truth $\varrho^{1:n}$ with $\emptyset$ to obtain $\varrho^{1:n_q}$ (such that the cardinality of MB components at layer l, $\beta^{1:n_q,l}$, and that of the ground-truth are the same), followed by matching the ground-truth with the MB components of the predictions. This is done by finding an optimal permutation of the ground truth such that the ground truth and the MB components are matched in the principle of the lowest matching loss, which is similar to what we have done in constructing the loss function for MT3 (see Section 2.2.9).

The optimal permutation of the ground-truth σ_*^l is also considered the most likely way of associating the ground-truth with MB components at decoder layer l, which is defined the same way in (2.39). The MB density $\log f^{\mathrm{MB},l}(\boldsymbol{\varrho}^{1:n})$ can now be approximated by

$$\log f^{\mathrm{MB},l}(\boldsymbol{\varrho}^{1:n}) \approx \sum_{i=1}^{n} \log \varphi^{i,l}(\boldsymbol{\varrho}^{\sigma_*^l(i)}), \tag{2.59}$$

where $\varphi^{i,l}(\cdot)$ in (2.59) is given by:

$$\varphi^{i,l}(\boldsymbol{\varrho}^j) = \mathbb{1}_{\varrho^j \neq \emptyset} \log \frac{\mathcal{N}(\tilde{x}^j; \mu^{i,l}, \Sigma^{i,l})}{p^{i,l}} + (1 - \mathbb{1}_{\boldsymbol{\varrho}^j \neq \emptyset}) \log(1 - p^{i,l}) \tag{2.60}$$

and $\mathbb{1}_{\boldsymbol{\varrho}^j \neq \emptyset}$ is an indicator function such that

$$\mathbb{1}_{\boldsymbol{\varrho}^j \neq \emptyset} = \begin{cases} 1, & \text{if} \quad \boldsymbol{\varrho}^j \neq \emptyset \\ 0, & \text{otherwise} \end{cases}. \tag{2.61}$$

2.3 Mapping vectors to a higher dimension

In this section, we introduce a function γ that maps a real number p to a vector of dimension $2L$:

$$\gamma(p) = (\sin(2\pi p^0), \cos(2\pi p^0), \cdots, \sin(2\pi p^{L-1}), \cos(2\pi p^{L-1})) \in \mathbb{R}^{2L}, \tag{2.62}$$

This function was used in [32] to increase the input vectors' dimensions before the neural network uses them and prevents the network from being biased toward learning low-frequency functions [33]. The state representation γ will serve as a comparison in the later experiments.

3

Methods

3.1 Problem formulation

We consider a scenario with N_s sensors located in different positions with partially overlapped FoVs. They work together to surveil an environment in which objects move for 10 time steps starting from time step 1, where $T = 10$ is the current time. The multi-object dynamic and measurement models follow the models described in Section 2.1.3 with nearly constant velocity motion model and linear Gaussian measurement model. Further, we denote $\mathbf{x}_T$ as the collection of object states at T. For each sensor in the scene, it collects and processes the measurements using a TPMB filter (see Section 2.1.8) to track moving objects in its FoV. Concretely, the set of measurements collected by sensor i for all objects that have been presented in its FoV from time 1 to T is denoted by $\mathbf{z}^i$, which is then fed to a TPMB filter, outputting a set of trajectory estimates up to the current time, denoted as $\mathbf{E}_T^i$, $i \in \{1, \cdots, N_s\}$.

For each $\mathbf{E}_T^i$, $i \in \{1, \cdots, N_s\}$, it contains the following information:

- Total number of trajectory estimates: $N^i_{\text{trajectory}}$.
- The existence probability of the trajectory estimate j: $p^{j,i}_{\text{existence}}$.
- The alive probability of trajectory estimate j at time T conditioned on existence: $p^{j,i}_{\text{alive}}$.
- The start time of trajectory estimate j: $t^{j,i}_{\text{start}}$.
- The length of trajectory estimate j: L^i_j.
- The states and their covariance matrixes of trajectory estimate j:

$$\{x_t^{j,i}\}_{t=t^{j,i}_{\text{start}}}^{t^{j,i}_{\text{start}}+L^i_j-1} \in \mathbb{R}^4 \quad \text{and} \quad \{C_t^{j,i}\}_{t=t^{j,i}_{\text{start}}}^{t^{j,i}_{\text{start}}+L^i_j-1} \in \mathbb{R}^{4\times 4}, \tag{3.1}$$

where $j \in \{1, \cdots, N^i_{\text{trajectory}}\}$.

In this project, we use a sliding window approach [3] to estimate $\mathbf{x}_T$ based on the knowledge of sets of trajectory estimates $\{\mathbf{E}_T^1, \cdots, \mathbf{E}_T^{N_s}\}$. Concretely, we would like to apply MOTTs to fuse the trajectory estimates $\{\mathbf{E}_T^1, \cdots, \mathbf{E}_T^{N_s}\}$ extracted from different TPMB filtering densities at different sensors to obtain parameters that describe the density representation of $\mathbf{x}_T$.

3.2 Data Generation

We show how data MOTTs will use is generated. This includes generating object trajectories, measurements, and trajectory estimates. Further, we introduce how trajectory estimates are prepossessed such that they can be fed into MOTTs.

3.2.1 Object trajectories generation

This thesis considers object moves in a scene for 10-time steps. Initially, we assume there are n_{initial} objects in the scene and sample newborn objects from the Poisson distribution :

$$n_{\text{birth}} \sim \text{Poisson}(\lambda^{\text{birth}}), \tag{3.2}$$

where λ_{birth} is the Poisson birth rate. Next, each of these $n_{\text{initial}} + n_{\text{birth}}$ objects is initialized with a state vector $x_1^i \in \mathbb{R}^4$ by sampling from:

$$x_1^i \sim \mathcal{N}(\mu_{\text{birth}}, \sigma^2_{\text{birth}}), \tag{3.3}$$

where the mean $\mu_{\text{birth}} \in \mathbb{R}^4$ and covariance $\sigma^2_{\text{birth}} \in \mathbb{R}^{4\times4}$ are hyper-parameters specified in Section A.1; 1 indexes the time step 1 and $i \in \{1, \cdots, n_{\text{initial}} + n_{\text{birth}}\}$. In addition, we have $x_1^i = [p_1^i, v_1^i]$, where p_1^i represents the 2D position and v_1^i represent the 2D velocity. If any sampled state is not inside the sensors' FoVs, it is removed. The object birth process (3.2) and its initialization (3.3) is repeated for all time steps.

Assuming we are at time step $t \in \{1, \cdots, 10\}$ with the above processes have been performed. Given an object state vector x_t (x_t^i is not used here since the object may be newborn and does not belong to any existing object in the scene), a number r_{survival} is sampled according to:

$$r_{\text{survival}} \sim \text{Uniform}([0, 1]), \tag{3.4}$$

and if $r_{\text{survival}} > P^S$, the object will not be further processed in the remaining time steps; otherwise we update the object state for time step $t + 1$ according to the nearly constant velocity motion model:

$$x_{t+1} \sim \mathcal{N}(Fx_t, Q) \in \mathbb{R}^4. \tag{3.5}$$

Here, $F \in \mathbb{R}^{4\times4}$ and $Q \in \mathbb{R}^{4\times4}$ are defined as

$$F = \begin{bmatrix} \mathbf{I_2} & \mathbf{I_2}\Delta_t \\ \mathbf{0} & \mathbf{I_2} \end{bmatrix}, \tag{3.6}$$

$$Q = \sigma_q^2 \begin{bmatrix} \mathbf{I_2}\frac{\Delta_t^3}{3} & \mathbf{I_2}\frac{\Delta_t^3}{3} \\ \mathbf{I_2}\frac{\Delta_t^2}{2} & \mathbf{I_2} \end{bmatrix}, \tag{3.7}$$

where Δ_t is the scanning time interval, $\mathbf{I_2}$ is a 2D identity matrix, and we denote σ_q^2 as the variance of the motion noise. The updated state x_{t+1} will be kept after

checking it is inside the sensors' FOV.

Repeating the above process for that object for the remaining time steps, and collecting all the states results in a single set of trajectory for that object: $X = \{x_j\}_{j=t}^{T'}$, where $T' \leq 10$ and $t \in \{1, \cdots, 10\}$. Further, after repeating the above progress for all time steps and all objects, we obtain a set of object trajectories, denoted as $\mathbf{X} = \{X^1, \cdots, X^l\}$, where l is the number of trajectories in $\mathbf{X}$ and there are l objects in the scene. At last, we present the pseudocode for object trajectory generation in Algorithm 1.

Algorithm 1 An algorithm for generating object trajectory

Start with n_{initial} objects and create n_{initial} empty sets to store trajectories created by these objects.

for $t \leftarrow 1$ to 10 **do**
  Generate n_{birth} new born objects using (3.2) and create n_{birth} empty sets to store trajectories created by these new objects.

  Initialize new born objects using (3.3).

  for each object **do**
    if object is inside FoV and is going to survive **then**
      Update object state according to (3.5).
      if updated object state is inside FoV **then**
        Append updated object state to the set.
      else
        Move to the next iteration and the object will not be considered in the remaining interations.
      end if
    else
      Move to the next iteration and the object will not be considered in the remaining interations.
    end if
  end for
end for

3.2.2 Measurements generation

This section introduces how the measurements are generated for a single sensor and applied to all other sensors. For $X \in \mathbf{X}$, we generate measurement for every $x \in X$ with probability $p^D(x)$. Concretely, a number $r_{\text{detection}}$ is sampled according to:

$$r_{\text{detect}} \sim \text{Uniform}([0, 1]), \tag{3.8}$$

and if $r_{\text{survival}} \leq P^D(x)$, a measurement z is generated for x according to the measurement model

$$z \sim \mathcal{N}(Hx; R) \tag{3.9}$$

with H and R defined as

$$H = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}, \tag{3.10}$$

$$R = \sigma_z^2 \mathbf{I_2}, \tag{3.11}$$

where σ_z^2 is the variance of the measurement noise.

Clutter measurements are generated independently from the above object measurements generation, and they are sampled from a PPP at each time step $t \in \{1, \cdots, 10\}$. Firstly, the number of clutter measurements at t is determined by:

$$n_t^{\text{clutter}} \sim \text{Poisson}(\lambda^{\text{clutter}}), \tag{3.12}$$

where λ^{clutter} is the clutter rate. Next, we uniformly generate n_t^{clutter} clutter measurements using the area of FoV:

$$z_t^{\text{clutter},i} \sim \text{Uniform(FoV)}. \tag{3.13}$$

Algorithm 2 An algorithm for generating measurements for each sensor

Assume there are N_s sensors and $n_{\text{trajectory}}$ object trajectories generated using Algorithm 1, we create N_s empty sets to store measurements for sensors.

for sensor from 1 to N_s **do**
  for trajectory from 1 to $n_{\text{trajectory}}$ **do**
    for each state in this trajectory **do**
      if object is inside the sensor FoV and detected **then**
        Generate one measurement according to (3.9).
        Append the measurement to the measurement set of the corresponding sensor.
      end if
    end for
  end for
  for $t \leftarrow 1$ to 10 **do**
    Generate the number of clutter n_{clutter} according to (3.12).
    Generate n_{clutter} clutter measurements with the sensor's FoV according to (3.13).
    Append the clutter measurements to the measurement set of the corresponding sensor.
  end for
end for

where $i \in \{1, \cdots, n_t^{\text{clutter}}\}$. We then collect these clutter measurements into

$$\mathbf{z}_t^{\text{clutter}} = \{z_t^{\text{clutter},1}, \cdots, z_t^{\text{clutter},n_{\text{clutter}}^t}\}. \tag{3.14}$$

Along with the objects generated measurements at t, $\mathbf{z}_t^{\text{object}}$, measurements at t is given by

$$\mathbf{z}_t = \mathbf{z}_t^{\text{clutter}} \bigcup \mathbf{z}_t^{\text{object}}. \tag{3.15}$$

After repeating the above process for all time steps, what we obtain is a sequence of T sets of measurements

$$(\mathbf{z}_1, \cdots, \mathbf{z}_T). \tag{3.16}$$

The pseudocode of the measurement generation for each sensor is given in Algorithm 2.

3.2.3 Filtering and dataset construction

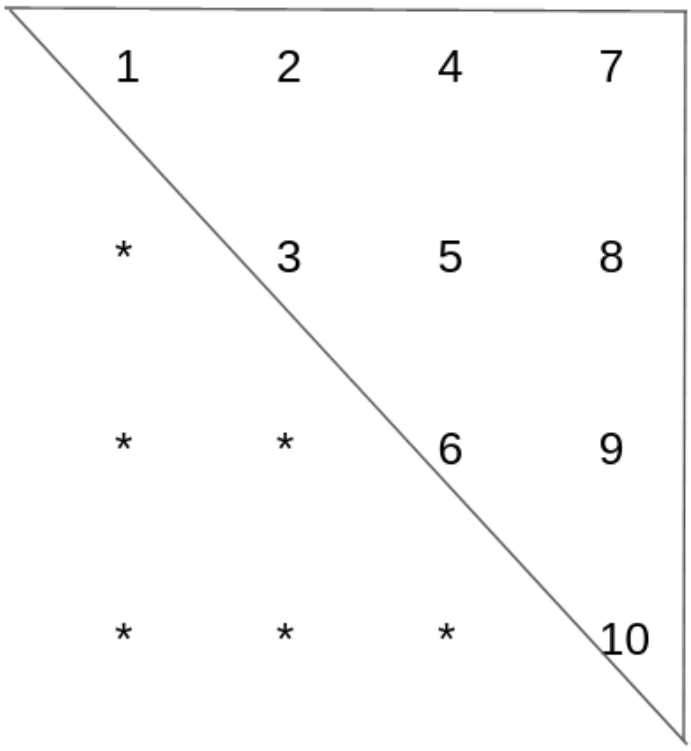

Figure 3.1: The MASK function selects the elements in the upper-triangular area of a square matrix and rearranges the selected elements into a vector. The number in the figure indicates the order of the rearrangement.

Explained in Section 3.1, the TPMB filter inputs measurements and outputs a set of trajectory estimates. The implementation of the tracking algorithm [1] for each sensor is considered outside the scope of this thesis and is therefore left out.

Also, in Section 3.1, we have introduced the components and information contained in the set of trajectory estimates. However, they can not be directly used by the Transformer as it only accepts a sequence of vectors or scalars as input. To solve this issue, we show how to transform N_s sets of trajectory estimates $\{\mathbf{E}_T^1, \cdots, \mathbf{E}_T^{N_s}\}$ to a sequence of vectors. In addition, the information contained in each set of trajectory estimates follows the same pattern and notations described in Section 3.1. Specifically, we use a mask function:

$$\text{MASK} : W \in \mathbb{R}^{4\times4} \rightarrow w \in \mathbb{R}^{10}, \tag{3.17}$$

which selects the entries along and above the main diagonal of a matrix $W \in \mathbb{R}^{4\times4}$ (shown in Figure 3.1) and then arrange them to a vector $w \in \mathbb{R}^{10}$. The detail implementation is given in Algorithm 3.

Algorithm 3 Transforming sets of trajectory estimates to a sequence of vectors

Initialize a cell of cells: $\boldsymbol{\Upsilon} = (\Upsilon^1, \cdots, \Upsilon^T)$, where each cell Υ^n, $n \in \{1, \cdots, T\}$ can store the transformed vectors. Accessing a specific cell t is denoted by Υ^t.

for $n \leftarrow 1$ to N_s **do**
  Get the number of trajectories $N^n_{\text{trajectory}}$.
  for $j \leftarrow 1$ to $N^n_{\text{trajectory}}$ **do**
    Get the existence probability: $p^{j,n}_{\text{existence}}$.
    Get the alive probability: $p^{j,n}_{\text{alive}}$.
    for $t \leftarrow t^{j,n}_{\text{start}}$ to $L^n_j + t^{j,n}_{\text{start}} - 1$ **do**
      Get the states: $x^{j,n}_t$.
      Get the covariance: $C^{j,n}_t$.
      Apply the MASK function : $\text{MASK}(C^{j,n}_t) = w^{j,n}_t$. ▷ (1)
      Append vector $[x^{j,n}_t; w^{j,n}_t; p^{j,n}_{\text{existence}} \times p^{j,n}_{\text{alive}}; t; j; n]$ to Υ^t. ▷ (2)
    end for
  end for
end for
for $t \leftarrow 1$ to T **do**
  Randomly permute the vectors in Υ^t.
end for
Concatenate all the vectors in $\boldsymbol{\Upsilon}$ to formulate a sequence of vectors $\{d'\}_1^l$, where l is the length of the object and $l = \sum_{n=1}^{N_s} \sum_{j=1}^{N^n_{\text{trajectory}}} L^n_j$.

The principle of Algorithm 3 is by formulating estimated states (and their accompanying information) into vectors and grouping them according to the time steps

[1] The MATLAB implementation in https://github.com/Agarciafernandez/MTT/tree/master/TPMBM%20filter is used

$\{1, \cdots, T\}$. In (1), applying the MASK function also reduce the training data size. This is because the covariance matrixes are is positive-symmetric-definite, such that $w_t^{j,n}$ has captured sufficient information of $C_t^{j,n}$. Inspired by [34] and [3], in (2) we denote $d' = [d; t; j; n] = [x; w; p'; t; j; n] \in \mathbb{R}^{18}$ (indexes omitted for simplification), where

$$d = [x; w; p'] \in \mathbb{R}^{15}, p' = p_{\text{existence}} \times p_{\text{alive}}, \tag{3.18}$$

and $t, j, n \in \mathbb{R}$ are time step, the trajectory ID and the sensor ID, respectively. We further refer vector $[w; p'] \in \mathbb{R}^{11}$ as uncertainty and study how it influence the model's performance in Section 4.2.

3.3 Transformer models

This section explains our adaptation of MT3 and MT3v2 to the decentralized MOT problem. First, we introduce an optional state representation technique. Subsequently, we will present how the training data is encoded. In the coming sections, MT3 and MT3v2 input a sequence of vectors obtained from Algorithm 3 and output a prediction set ϖ. Although the information contained in the prediction set of MT3 is different from that of MT3v2 (see Section 2.2.5 and 2.2.10 for the output of MT3 and MT3v2), the definition of the input remains the same.

3.3.1 State representation

In MT3 and MT3v2, each vector d (3.18) of the vector d' of the input sequences is independently transformed to a higher dimensional source vector u_{source} through a linear projection before passing them to the network:

$$\mathcal{P} : d \in \mathbb{R}^{15} \rightarrow u_{\text{source}} \in \mathbb{R}^{d_{\text{model}}}, \tag{3.19}$$

where d_{model} is a hyper-parameter and set to be 256. It is claimed in [2] that increasing the input vectors' dimensions allows the transformer-based model to store more complex representations. In addition, we introduce another way of performing state representation:

$$\gamma : d \in \mathbb{R}^{15} \rightarrow u_{\text{temp}} \in \mathbb{R}^{30L}, \quad \mathcal{P} : u_{\text{temp}} \in \mathbb{R}^{30L} \rightarrow u_{\text{source}} \in \mathbb{R}^{d_{\text{model}}}. \tag{3.20}$$

Here, we first map each vector to dimension $\mathbb{R}^{30L}$ using the function γ introduced in Section 2.3 and then apply projection $\mathcal{P}$. Besides increasing the capability of complex representation, applying function γ in the network prevents the deep network from being biased towards learning low-frequency functions [33]. The state representation γ will serve as a comparison in the experiments in Section 4.2.

3.3.2 Input embedding and positional encoding

In this section, we introduce how we manage the input sequence obtained by Algorithm 3, which is denoted as $\{d'\}_1^l$. Explained in Section 2.2.4, the positional encoding is vital in the development of Transformer, which is also true for the MOTTs.

We define three temporal encoders [2] : $\phi^{\text{Time}} : \mathbb{R} \to \mathbb{R}^{d_{\text{model}}}$, $\phi^{\text{Trajectory}} : \mathbb{R} \to \mathbb{R}^{d_{\text{model}}}$ and $\phi^{\text{Sensor}} : \mathbb{R} \to \mathbb{R}^{d_{\text{model}}}$, and use them to encode the time step (t), the trajectory ID (j) and the sensor ID (n), contained in each d' respectively. The outputs from these encoders are added together to be the positional encoding for the associated vector d of d' (see (3.18), d contains the remaining elements of d' after t, j, n have been used for producing the positional encodings):

$$u'_{\text{source}} = u_{\text{source}} + \phi^{\text{Time}}(t) + \phi^{\text{Trajectory}}(j) + \phi^{\text{Sensor}}(n) \in \mathbb{R}^{d_{\text{model}}}, \tag{3.21}$$

where u_{source} is obtained by 3.19. The sequence $\{u'_{i,\text{source}}\}_1^l$ is then fed to the MOTTs' encoders to generate an embedding.

3.4 Evaluation

In our development, we utilise the the generalized optimal sub-pattern assignment (GOSPA) [35] as well as the NLL [31] as our performance measure. Especially, GOSPA is applied to the SOTA method, MT3, and MT3v2, while NLL is applied to the SOTA method based on the model-based fusion method introduced in Section 2.1.9, and MT3v2. During the evaluation, the SOTA method is given an advantage as they can access the correct models.

Different fusion methods can have different outputs. The output of the MT3 contains only the predicted object 2D positions and their corresponding existence probabilities. In contrast, MT3v2 and the SOTA method outputs the PMB density [2] (see Section 2.2.10), where each Bernoulli component contains estimated positions, velocities, covariance matrix, and existence probabilities. Unlike the NLL metric, which uses the entire PMB density, the GOSPA metric only uses the valid predicted states, which refer to states with associated existence probabilities higher than a certain threshold. Concretely, let $\boldsymbol{\varpi}_{\text{valid}}^{\text{MT3}}$ be set of MT3 predicted positions, $\boldsymbol{\varpi}_{\text{valid}}^{\text{MT3v2}}$ be the set of MT3v2 predicted states (the states are made up of 2D positions and velocities), and $\boldsymbol{\varpi}_{\text{valid}}^{\text{SOTA}}$ be the set of SOTA predicted states (the states are made up of 2D positions and velocities), the valid predicted states for these models are formulate as:

$$\boldsymbol{\varpi}_{\text{valid}}^{\text{MT3}} = \{\tilde{x}_i | \tilde{x}_i \in \boldsymbol{\varpi}^{\text{MT3}}, p_i^{\text{MT3}} > p_{\text{threshold}}^{\text{MOTTs}}\} \tag{3.22}$$

$$\boldsymbol{\varpi}_{\text{valid}}^{\text{MT3v2}} = \{\tilde{x}_i | \tilde{x}_i \in \boldsymbol{\varpi}^{\text{MT3v2}}, p_i^{\text{MT3v2}} > p_{\text{threshold}}^{\text{MOTTs}}\} \tag{3.23}$$

$$\boldsymbol{\varpi}_{\text{valid}}^{\text{SOTA}} = \{\tilde{x}_i | \tilde{x}_i \in \boldsymbol{\varpi}^{\text{SOTA}}, p_i^{\text{SOTA}} > p_{\text{threshold}}^{\text{SOTA}}\} \tag{3.24}$$

respectively, where $p_{\text{threshold}}^{\text{MOTTs}}$ and $p_{\text{threshold}}^{\text{SOTA}}$ are hyperparameters of the architecture, set to be 0.75 and 0.5. Now, the sets of valid predictions and the ground truth can be used by the GOSPA metric. Note that if the SOTA method is compared with MT3, then it means that we are only interested in comparing 2D positions. The set $\boldsymbol{\varrho}$ should only contain 2D position vectors.

[2]The PPP component for MT3v2 is added in the same way in [17]

3.4.1 Generalized Optimal Sub-Pattern Assignment

We denote $\boldsymbol{\varrho} = \{x^1, \cdots, x^k\}$ as the set of ground-truth and $\boldsymbol{\varpi} = \{\tilde{x}^1, \cdots, \tilde{x}^l\}$ as the set of estimations, then GOSPA searches for an optimal assignment A between $\{1, \cdots, |\boldsymbol{\varrho}|\}$ and $\{1, \cdots, |\boldsymbol{\varpi}|\}$ over all possible assignment sets Γ, where $A \in \{1, \cdots, |\boldsymbol{\varrho}|\} \times \{1, \cdots, |\boldsymbol{\varpi}|\}$, such that the distance between these two sets :

$$d_p^{(c,\alpha)}(\boldsymbol{\varpi}, \boldsymbol{\varrho}) = \left[\min_{A \in \Gamma} \sum_{(i,j) \in A} d(\boldsymbol{\varpi}^i, \boldsymbol{\varrho}^j)^p + \frac{c^p}{\alpha}(|\boldsymbol{\varrho}| + |\boldsymbol{\varpi}| - 2|A|) \right]^{\frac{1}{p}}, \tag{3.25}$$

is minimized. Each element in $\{1, \cdots, |\boldsymbol{\varpi}|\}$ (or in $\{1, \cdots, |\boldsymbol{\varrho}|\}$) can not be assigned more than once. The scalar p controls the level at which the outlier is penalized. The parameter c, along with α, affects the maximum allowable localization error and error caused by cardinality mismatch. The distance $d(x^i, \tilde{x}^j)^p$ is typically chosen to be the L2-norm between x^i and $\tilde{x}^j$. Additionally, the GOSPA scores can be divided into three parts:

- The localization errors: $\sum_{(i,j) \in A} d(\boldsymbol{\varpi}^i, \boldsymbol{\varrho}^j)^p$.
- The false detection errors: $\frac{c^p}{\alpha}(|\boldsymbol{\varpi}| - |A|)$.
- The miss detection errors: $\frac{c^p}{\alpha}(|\boldsymbol{\varrho}| - |A|)$.

The number c^p/α is the cost for every missed or false target left unassigned.

3.4.2 Negative Log-Likelihood

The NLL is an uncertainty-aware performance measure that evaluates how well the MOT posteriors explain the ground-truth [31]. Compared to GOSPA, NLL considers uncertainty information, and no hyper-parameter is needed. Let f^M be the posterior density of the estimates from the MOT algorithm M. The NLL is defined as:

$$\text{NLL}(\boldsymbol{\varrho}, f^M) = \log f^M(\boldsymbol{\varrho}) \tag{3.26}$$

For different MOT algorithms, the NLL is computed differently. In this thesis, we focus on NLL for PMB posterior densities. Since directly computing NLL for PMB densities is computationally expensive [17], we use the algorithm in [31] as an approximation:

$$\begin{aligned} \text{NLL}(\boldsymbol{\varrho}, f^{PMB}) \approx \min_{A \in \Gamma} \sum_{(i,j) \in A} -\log(r^i p^i(\boldsymbol{\varrho}^j)) - \sum_{i \in \mathbb{F}(A)} \log(1 - p^i) \\ + \int \lambda(\boldsymbol{\varrho}') d\boldsymbol{\varrho}' - \sum_{i \in \mathbb{M}(A)} \log \lambda(\boldsymbol{\varrho}^j). \end{aligned} \tag{3.27}$$

We explains the symbols in (3.27) as follow:

- λ is the PPP intensity function of the predicted PMB density f^{PMB}.
- r^i and p^i specifies the i-th Bernoulli components (see (2.3)) of f^{PMB}.
- A and Γ are defined the same way in GOSPA (see Section 3.4.1).

- Set $\mathbb{M}(A)$ contains the indices of the ground-truch that are not assigned to any element in $\boldsymbol{\varpi}$.

- Set $\mathbb{F}(A)$ contains the indices of the components estimates that are not assigned to any element in $\boldsymbol{\varrho}$.

Similar to GOSPA, the NLL errors above can also be divided into three parts:
- The localization errors: $\sum_{(i,j)\in A} -\log(r^i p^i(x^j))$.
- The false detection errors: $-\sum_{i\in\mathbb{F}(A)} \log(1-p^i)$.
- The miss detection errors: $\int \lambda(x')dx' - \sum_{i\in\mathbb{M}(A)} \log \lambda(x^j)$.

3.5 Training and implementation detail

The data generator is programmed in MATLAB, and the other components are developed in PyTorch. We collect 100k groups of data for training and 25000 for validation for each task, where each group contains 32 batches of sets of trajectories (batch size: 32). For evaluation, we use a test set containing 1000 groups of data and change the batch size to 1. Additionally, we use different seeds when collecting different groups of data (one seed for one group).

For training, we train the MOTTs for a certain number of epochs (2 and 4 epochs in our simulations, meaning 200k gradient steps and 400k gradient steps, respectively). Whenever the training of an epoch is finished, we shuffle the data and send it back to the MOTTs. In addition, we set an ADAM optimizer with 0.00005 to be the initial learning rate and the reduce-learning-rate patience to be 50000 gradient steps. The learning rate will decrease by 1/4 if the total training losses do not decrease for 50000 gradient steps. Further, we use NVIDIA A40 GPUs to train MOTTs.

In evaluation, we choose $c = 2$, $p = 1$ and $\alpha = 2$ for computing GOSPA in all tasks. In addition, we use the same model configurations for MT3 and MT3v2 in there original papers [3, 17].

4

Simulations and Results

In this chapter, MOTTs are tested for different tasks in different scenarios, and their performances are compared to the results obtained from the SOTA via GOSPA and NLL. The simulations contain two different scenarios where sensors are placed differently. We will start by introducing our experiential setting and then present the results. Lastly, the attention maps of sampled input are shown to give an insight into the working mechanism of transformers.

4.1 Simulation settings

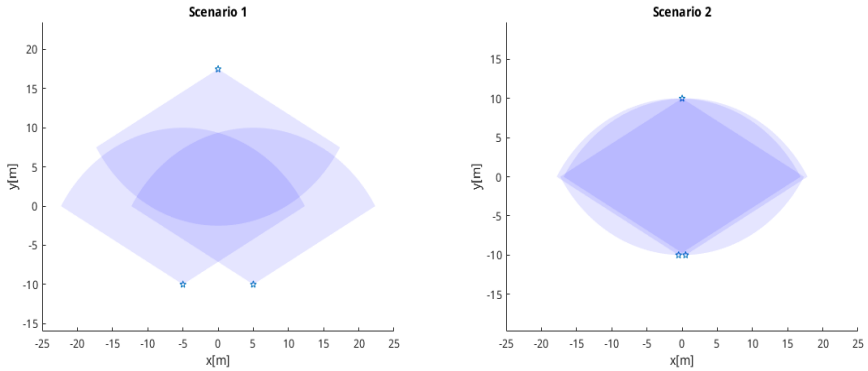

Figure 4.1: An illustration of scenarios 1 and 2. Each scenario has three sensors, and each sensor has a fan-shaped FoV with a bearing size of $2/(3\pi)$ and a radius of 20m.

The simulations are mainly conducted in two different scenarios, shown in Figure 4.1. In each scenario, there are three sensors, each of which has a fan-shaped FoV with a bearing size of $2/(3\pi)$ and a radius of 20m. Furthermore, the overlapped area of FOVs in scenario 2 is more significant than in scenario 1. In scenario 1, objects may be observed by all three sensors or partially observed by some. Also, the objects may move from a region with fewer overlapped FoVs to a region with more overlapped FoVs, and vice versa.

For each scenario, we define three tasks where task 1 is a baseline of comparison with the other two tasks. The parameters set for the different tasks can be found in Appendix A. In task 2, we increase the noise level of the measurements by adjusting

the noise covariance from $0.01\mathbf{I}_2$ to $0.1\mathbf{I}_2$ to investigate the effect of noise. In task 3, we lower the estimation existence threshold from 0.5 to 0.001 in local MOT filters, allowing them to produce more trajectory estimates. All tasks are trained using the same seed in PyTorch.

4.2 Results

We have summarised the results in Table 4.1-4.4 in Section 4.2.1 and Section 4.2.2. They are GOSPA errors and NLL, as well as their decompositions for tasks in different scenarios when using different algorithms (or models). Additionally, to allow us to present the result more efficiently, we have introduced a term called 'Factor' in each of the tables, and we make the following statements to help our reader to understand our experimental settings:

- The number of gradients: Whenever '200k' or '400k' is observed in Table 4.1-4.4, '200k' means the model of that particular experiment was trained for 200k gradient steps and '400k' was for 400 gradient steps.

- The uncertainty: If '▲' is observed in Table 4.1-4.4 under the 'Factor' column, it means that the uncertainty vector $[w; p'] \in \mathbb{R}^{11}$ is not included in the input during training in that particular experiment. The model was trained for 400k and the explanation of uncertainty $[w; p'] \in \mathbb{R}^{11}$ can be found in Section 3.2.3.

- State representation: If 'γ ' is observed in Table 4.1-4.4 under the 'Factor' column, it means that the state representation function γ is applied in the model. The model was trained for 200k, and the interpretation of the γ function can be found in Section 2.3 and 3.3.1.

- If '♦' is shown under the 'Factor' in Table 4.1-4.4, it means that the uncertainty is included and the γ function is not used. We refer to such cases as our baselines.

Further discussion of the results will be made in the next chapter.

4.2.1 GOSPA scores and their decompositions

Task	Model	Factor	Gradient Steps	Avg	Localization	Missed	False
1	SOTA			1.1232	**0.11522**	0.051	0.957
	MT3	♦	200k	0.47645	0.28445	0.024	0.168
	MT3	γ	200k	**0.45286**	0.27186	0.027	0.154
	MT3	♦	400k	0.45883	0.26983	**0.022**	0.167
	MT3	▲	400k	0.54033	0.27933	0.122	**0.139**
	SOTA			1.6176	**0.59555**	0.058	0.964
	MT3v2	♦	200k	1.58	1.022	0.056	0.502
	MT3v2	γ	200k	1.465	0.91602	0.066	0.483
	MT3v2	♦	400k	**1.3729**	0.95886	**0.051**	0.363
	MT3v2	▲	400k	1.4451	0.94711	0.152	**0.346**
2	SOTA			1.4936	**0.27257**	**0.024**	1.197
	MT3	♦	200k	0.8774	0.61044	0.074	0.193
	MT3	γ	200k	0.90222	0.63222	0.081	0.189
	MT3	♦	400k	**0.84616**	0.59516	0.068	**0.183**
	MT3	▲	400k	1.0274	0.64436	0.145	0.238
	SOTA			2.11	**0.82101**	**0.058**	1.231
	MT3v2	♦	200k	2.077	1.492	0.167	0.418
	MT3v2	γ	400k	2.061	1.514	0.153	0.394
	MT3v2	♦	400k	**2.0188**	1.4918	0.149	**0.378**
	MT3v2	▲	400k	2.1285	1.4755	0.205	0.448
3	SOTA			1.1232	**0.11522**	0.051	0.957
	MT3	♦	200k	0.41005	0.26205	**0.041**	0.107
	MT3	γ	200k	0.45418	0.29018	0.054	0.11
	MT3	♦	400k	**0.38915**	0.27015	0.044	**0.075**
	MT3	▲	400k	0.47281	0.27281	0.119	0.081
	SOTA			1.6176	**0.59555**	0.058	0.964
	MT3v2	♦	200k	1.4171	0.91207	0.064	0.441
	MT3v2	γ	200k	1.4418	0.95276	**0.049**	0.44
	MT3v2	♦	400k	**1.3782**	0.95317	0.057	**0.368**
	MT3v2	▲	400k	1.5037	0.87866	0.142	0.483

Table 4.1: GOSPA scores as well as their decompositions for tasks in scenario 1.

Task	Model	Factor	Gradient Steps	Avg	Localization	Missed	False
1	SOTA			0.85965	**0.12865**	0.068	0.663
	MT3	♦	200k	0.40225	0.25825	0.021	0.123
	MT3	γ	200k	0.46228	0.30128	0.023	0.138
	MT3	♦	400k	**0.38083**	0.24983	**0.015**	0.116
	MT3	▲	400k	0.47715	0.25715	0.11	**0.11**
	SOTA			1.4404	**0.69338**	0.076	0.671
	MT3v2	♦	200k	2.0018	0.86779	0.068	1.066
	MT3v2	γ	200k	1.5839	1.0059	0.07	0.508
	MT3v2	♦	400k	**1.413**	1.04	**0.031**	**0.342**
	MT3v2	▲	400k	1.4698	0.98879	0.13	0.351
2	SOTA			1.2323	**0.34128**	0.054	0.837
	MT3	♦	200k	0.9539	0.6739	0.055	0.225
	MT3	γ	200k	0.92822	0.71622	**0.048**	0.164
	MT3	♦	400k	**0.8323**	0.6283	0.059	**0.145**
	MT3	▲	400k	0.87065	0.59865	0.109	0.163
	SOTA			2.0073	**1.0283**	**0.098**	0.881
	MT3v2	♦	200k	2.0235	1.5175	0.167	0.399
	MT3v2	γ	200k	2.0422	1.5032	0.143	0.396
	MT3v2	♦	400k	**1.9328**	1.4708	0.154	**0.308**
	MT3v2	▲	400k	2.1303	1.5073	0.236	0.387
3	SOTA			0.85965	**0.12865**	0.068	0.663
	MT3	♦	200k	0.35883	0.27583	0.03	0.053
	MT3	γ	200k	0.34285	0.27285	**0.024**	0.046
	MT3	♦	400k	**0.34137**	0.27337	0.029	**0.039**
	MT3	▲	400k	0.45746	0.27346	0.124	0.06
	SOTA			1.4404	**0.69338**	0.076	0.671
	MT3v2	♦	200k	1.4129	0.96292	0.068	0.382
	MT3v2	γ	200k	1.7539	1.0049	0.073	0.676
	MT3v2	♦	400k	**1.3702**	0.97619	**0.052**	**0.342**
	MT3v2	▲	400k	1.4682	0.8762	0.132	0.46

Table 4.2: GOSPA scores as well as their decompositions for tasks in scenario 2.

4.2.2 NLL scores and their decompositions

Task	model	Factor	Gradient Steps	Avg	Localization	Missed	False
	SOTA			12.207	11.7659	0.3199	**0.1212**
	MT3v2	♦	200k	4.5409	3.632	0.5322	0.3768
1	MT3v2	γ	200k	4.9339	4.0376	0.4866	0.4098
	MT3v2	♦	400k	**1.1945**	**0.5716**	**0.3041**	0.3189
	MT3v2	▲	400k	3.6178	2.6219	0.3193	0.6766
	SOTA			11.2766	10.8198	**0.1018**	**0.3549**
	MT3v2	♦	200k	7.0909	6.3073	0.3193	0.4642
2	MT3v2	γ	200k	6.601	5.8943	0.3345	0.3721
	MT3v2	♦	400k	**6.1338**	**5.4745**	0.2889	0.3703
	MT3v2	▲	400k	7.8533	6.907	0.3401	0.6422
	SOTA			12.207	11.7659	0.3199	**0.1212**
	MT3v2	♦	200k	3.0852	2.3222	0.3041	0.4588
3	MT3v2	γ	200k	4.1651	3.3202	0.4258	0.4192
	MT3v2	♦	400k	**2.0797**	**1.4439**	**0.1977**	0.4382
	MT3v2	▲	400k	6.0556	4.837	0.441	0.7777

Table 4.3: NLL scores as well as their decompositions for tasks in scenario 1.

Task	Model	Factor	Gradient Steps	Avg	Localization	Missed	False
	SOTA			2.6947	2.1025	0.2525	0.3398
	MT3v2	♦	200k	12.8988	11.2651	1.0642	0.5696
1	MT3v2	γ	200k	6.7315	5.8914	0.3497	0.4903
	MT3v2	♦	400k	**1.8329**	**1.1692**	0.4105	**0.2532**
	MT3v2	▲	400k	3.7876	2.9973	**0.2129**	0.5773
	SOTA			7.2565	6.2315	**0.0825**	0.9424
	MT3v2	♦	200k	5.5272	4.8636	0.2889	0.3747
2	MT3v2	γ	200k	6.8235	6.0969	0.3953	0.3312
	MT3v2	♦	400k	**4.552**	**3.8982**	0.3649	**0.2889**
	MT3v2	▲	400k	7.1988	6.3318	0.2585	0.6085
	SOTA			2.6947	2.1025	**0.2525**	**0.3398**
	MT3v2	♦	200k	4.1164	3.2912	0.2889	0.5363
3	MT3v2	γ	200k	8.4787	7.403	0.6082	0.4675
	MT3v2	♦	400k	**2.2081**	**1.5663**	0.2737	0.3681
	MT3v2	▲	400k	6.2747	5.0727	0.3953	0.8076

Table 4.4: NLL scores as well as their decompositions for tasks in scenario 2.

4.3 Sample plots

A sample plot from scenario 1 has been shown in Figure 4.2. To have a closer look at the results, a selected region of Figure 4.2 have been plotted in Figure 4.3. In

this sample, the MT3 has generated five predictions, where the predicted positions are close to the objects' true positions at the current time. In contrast, the SOTA methods yield three predictions that can track but miss two objects. In addition, the training and validation losses for training the model in this sample have been plotted in Figure 4.4. It is observed that the losses drop quirky and reach 5 by the gradient step 10k. The values of losses fluctuate with a small range for the remaining gradients.

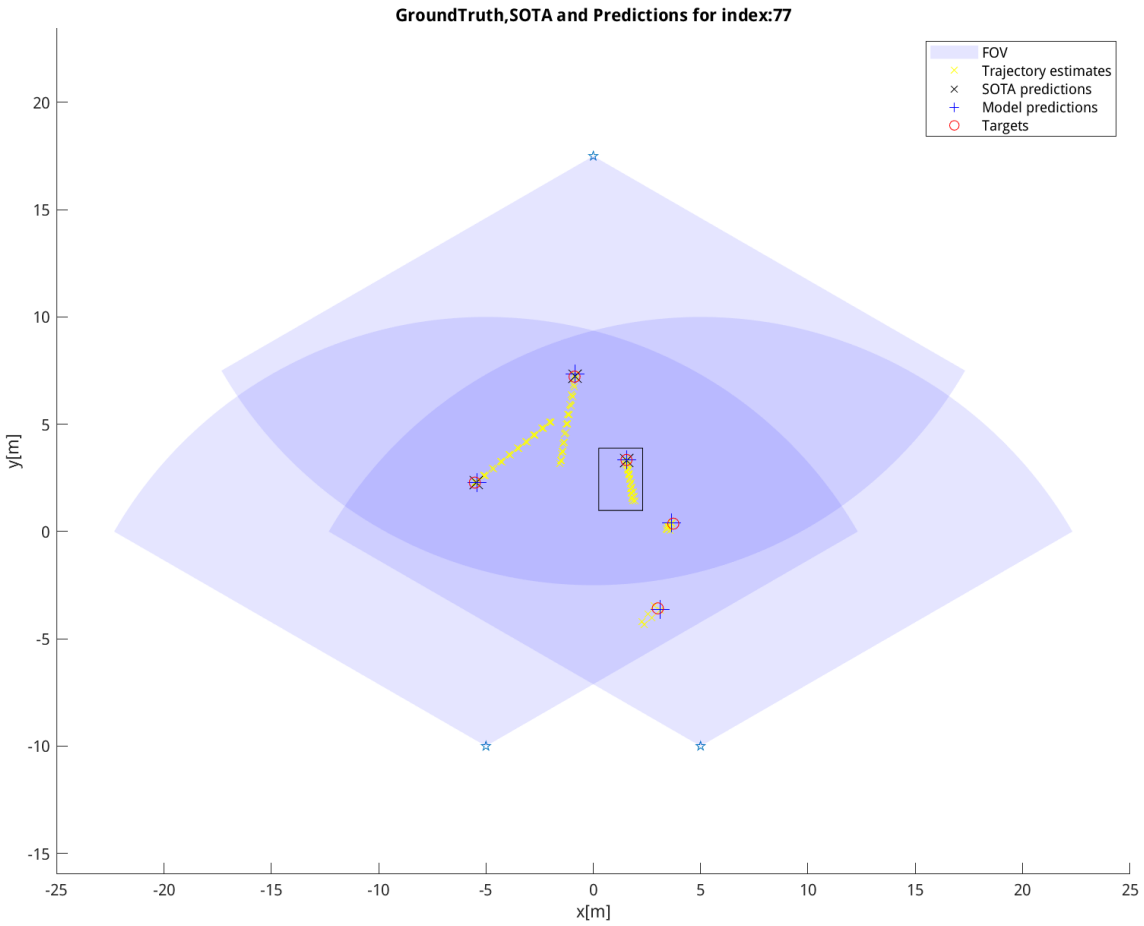

Figure 4.2: A sample plot from the test data set. The index is 77. The FoV, trajectory estimates, SOTA predictions, model predictions, and objects are shown in the figure. The black rectangular indicates a selected region, shown in Figure 4.3.

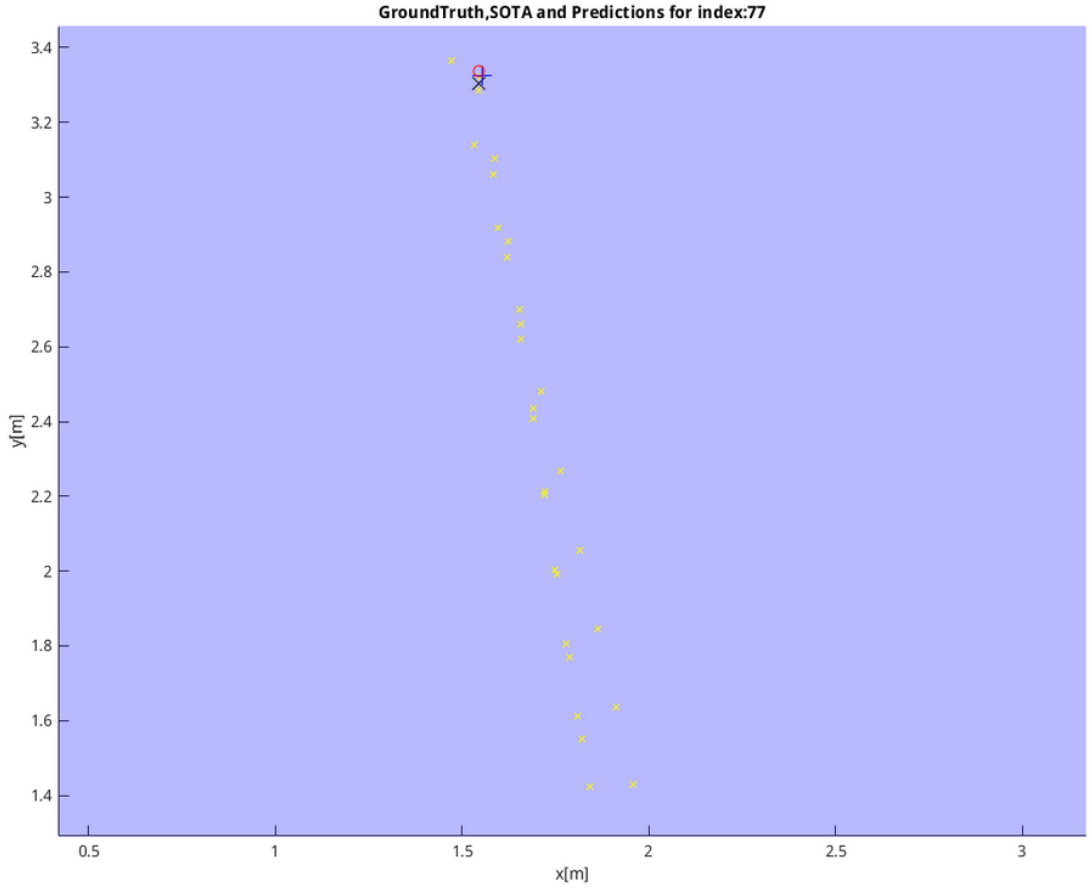

Figure 4.3: The selected region of the above Figure 4.2.

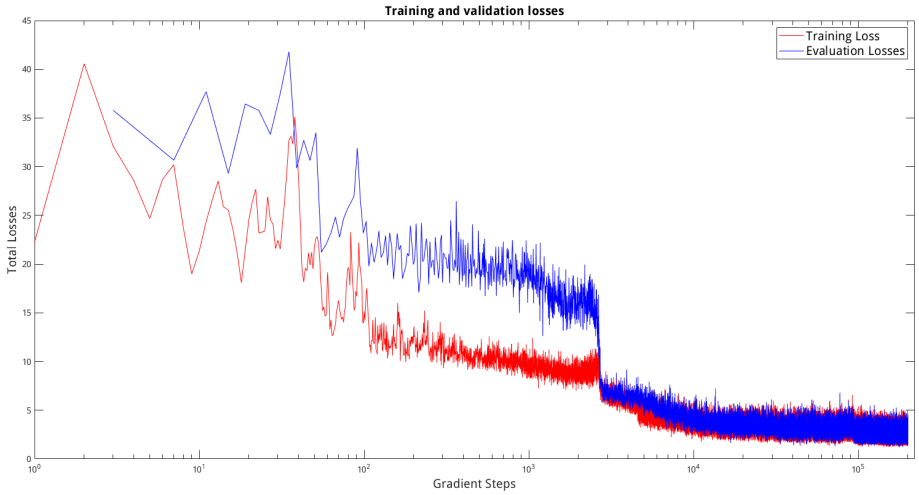

Figure 4.4: The training and validation losses for training MT3 in 200k gradient steps for task 1 in scenario 1 with state representation. The learning rate reduces from 0.0005 to 0.0000125 at gradient steps 94285 and 0.000003125 at gradient steps 193106.

4.4 Attention maps

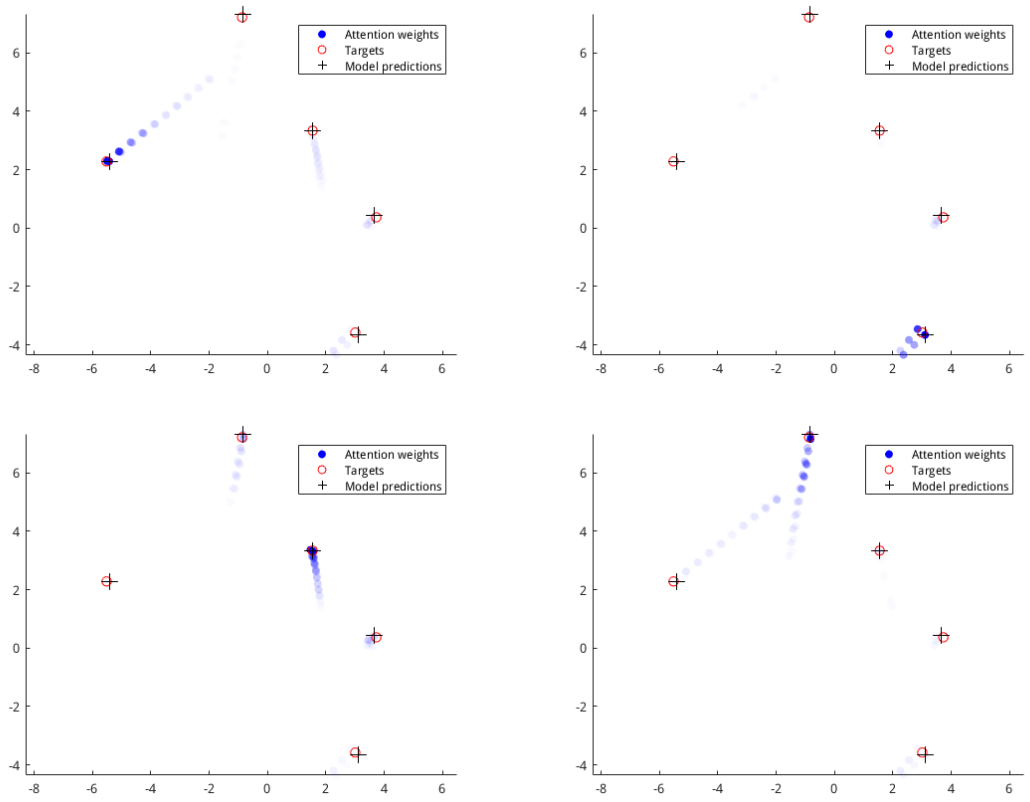

Figure 4.5: The figure is inspired by [2, 3]. The attention maps of 4 trajectories of a selected sample are shown. The attention weights are shown in blue-filled circles. The values of the weights determine their transparency: the more opaque a circle is, the higher the attention weight is. Further, objects and MOTT's predictions are also plotted.

Given a sequence of input elements, the attention map is a sequence of scalars computed by the attention mechanism, each of which shows the amount of attention that the associated element paid to other individual elements [2].

To illustrate, the attention maps of 4 trajectories of a selected sample are shown in Figure 4.5. The parameter setting for the scene is scenario 1, task 1, and the attention maps are obtained by passing a sequence of trajectory estimates to the MT3 model, which has been trained for 200k gradient steps with state representation. Also, the attention maps are plotted with their trajectory estimates in blue-filled circles, with the opacity indicating the value of the attention weights (the more opaque a circle is, the higher the attention weight is). Additionally, for a straightforward plot, we omit some elements in the plot and keep those only shown in the legend.

As shown in Figure 4.5, when the model predicts a specific object, it mainly focuses on the trajectory estimates from which that object originates, while other trajectory estimates have relatively small attention weights. Moreover, trajectory estimates from recent time steps receive greater attention than others. The pattern of the attention maps here is very similar to the case where MT3 is applied for solving single sensor MOT tasks [2].

5
Discussion

Different results obtained from the last chapter will be further discussed here. Next, we perform an ablation study, followed by a discussion of our method. Lastly, we point out some future work.

5.1 Discussion of the main results

We mainly compare 400k-baselines and the SOTA method in section 4.2. Concretely, we study how the results vary between models, tasks, and scenarios.

5.1.1 Results comparison between models

The MT3 predicts objects' states (positions) and their corresponding existence probabilities. The corresponding existence probabilities are only used to extract the predictions with the existence probabilities higher than a certain threshold. Together with the object positions, these predictions are involved in the computations of the GOSPA errors for the MT3. Although a similar process is performed when computing the GOSPA errors for MT3v2, unlike MT3, the predicted states of MT3v2 contain not only the positions but also the velocities (the object states now also involve velocities). As the dimension of objects and predicted states increases, the average GOSPA errors of MT3v2 are higher than the average GOSPA errors of MT3 in our simulations. Furthermore, the MOTTs' 400k-baselines outperform the SOTA method in all the tasks judging from the average GOSPA and NLL errors. Inspecting the decompositions of GOSPA and NLL scores, the SOTA method has the lowest GOSPA localization errors in all tasks, while the 400k-baselines achieve the lowest NLL localization errors in all tasks. Lastly, the average false errors of the 400k-baselines are lower than that of the SOTA method in all tasks.

5.1.2 Results comparison between tasks and scenarios

Based on the results from Section 4.2, increasing the noise level in data generation can deteriorate models' performance. In each scenario, it is observed that MOTTs achieve the highest GOSPA and NLL errors for task 2 among the other two tasks. In task 3, the number of trajectory estimates is increased. With more information, MT3 and MT3v2 achieve lower GOSPA errors than task 1. However, such a trend is not observed in the NLL errors. Instead, the MT3v2 obtains the lowest average

NLL errors in task 1. At last, it is worth noticing that increasing the number of trajectory estimates does not affect the SOTA method's performance in task 3, as its GOSPA and NLL errors in task 3 are the same as those in task 1.

In addition, all the models, in general, perform better in scenario 2 than in scenario 1. We hypothesize that this is because the complexity of the data generated in the first scenario is lower than its counterpart, as objects in scenario 1 may be partially observed or move from a region with fewer overlapped FoVs to a region with more overlapped FoVs, and vice versa.

5.2 Ablation Study

We study how increasing the number of gradient steps, including uncertainty during training and applying state representation may affect the models' performance. Moreover, unless otherwise specified, in Section 5.2.1, only 200k-baselines and 400k-baselines from Tables 4.1-4.4 are compared; in Section 5.2.1, only 400k-baselines and cases with '♦' from Tables 4.1-4.4 are compared; in Section 5.2.3, only 200k-baselines and cases with 'γ' from Tables 4.1-4.4 are compared.

5.2.1 Gradient steps

In general, it is observed that MOTTs achieve lower GOSPA and NLL errors when the number of gradient steps is increased from 200k to 400k, indicating an improvement in the models' performance. For the GOSPA metric, the MOTTs can achieve the lowest GOSPA average errors among other results with different experimental conditions besides the 400k-baseline of MT3 for task1 in scenario 1, with a slightly higher GOSPA score than that for simulation using an MT3 model with state representation trained for 200k gradient steps. In addition, increasing the number of gradient steps can also improve most of the average decompositions errors of the GOSPA and NLL.

5.2.2 Uncertainty

As seen from Section 4.2, including the uncertainty in the input improves the models' performance, allowing them to achieve lower GOSPA and NLL errors. Additionally, it is worth observing that almost all the decompositions of GOSPA and NLL, when computed using models with uncertainty included during training, are better than their counterparts. Based on the above observation, we hypothesize that including uncertainty in the input allows models to gain more information about the input, and the models can efficiently learn to use it, resulting in better performance.

5.2.3 State representation

As is evident from section 4.2, the performance of the models varies between tasks when γ is used. Inspecting the decompositions, applying γ does not explicitly influence a particular decomposition error. Since increasing the number of gradients steps can lead to a more stable improvement, we hypothesize that including γ in MT3 and MT3v2 might not be necessary.

5.3 Discussion of the method

In this thesis, we work on synthetic data, which reduces the modeling errors and allows us to control the complexity of the data set easily [2]. However, it also raises the question: can we apply the MOTTs in real-world scenarios for decentralized MOT? An answer to this is to train the MOTTs directly on a dataset collected in the real world. Such a dataset should contain trajectory estimates obtained from different sensors. Unfortunately, to our knowledge, such a data set does not exist. Also, training MOTTs typically requires a large amount of data, and many factors can affect the size of the dataset, such as how many sensors are considered and how many initial objects we assume at the beginning. Further, sensors such as Li-DAR may produce more than one detection, resulting in MEOT problems. A big challenge in constructing the MEOT dataset is that it would take much more time than MPOT since MEOT methods can typically be much slower than POT methods.

5.4 Future work

We discuss possible future work in this section.

5.4.1 Trajectory generation

So far, much attention has been paid to the tracking algorithm's ability to predict the states at the last step. We argue that in some cases, trajectory estimations must be compared with the whole objects' trajectories to answer trajectory-related questions. During our experiments, we have tried to generate trajectories by feeding trajectory estimates to MOTTs in a sliding window manner, but the result is not ideal. One possible reason might be that the MOTTs are not good at learning frame-by-frame data associations in decentralized content. One possible approach to handle this might be combining the MOTTs with the Trackformer [15]. Concretely, TrackFormer uses track queries [15] to achieve frame-to-frame data associations. The track queries integrate spatial information of objects, and the Transformer decoder adjusts the tracker queries between frames. The output embeddings are used at each frame as track queries for the next frame, achieving frame-to-frame data associations. By combining the MOTTs with the Trackformer, the detection and tracking might be performed in a unified way.

5.4.2 Moving sensors

In our thesis, the sensors' locations are fixed, meaning their positions are known and do not change with time. In practice, sensors are sometimes expected to move in the scene to know the detection area better. It is interesting to add sensor motion in future work, assuming that the sensor locations are known but time-varying. In that case, more information may need to be encoded, e.g., sensor positions and their poses.

5.4.3 Decentralized multiple extended object tracking

As mentioned, we focus on decentralized MOT for point objects, and if we want to apply MOTTs based on real-world data, some problems may arise since it might contain information for extended objects. A transformer-based model called STRAT [2] already exists developed for solving single extended object tracking. Similar to MOTTs, we believe that STRAT can also be applied to solving decentralized extended object tracking with little adjustments, provided that a suitable data set exists or can be simulated. This also requires the change of the input size and the tracking method at each local sensor, which can be further studied in future work.

6
Conclusion

Multi-object tracking involves the process of locating multiple objects using noisy sensor measurements. MOT is widely used in many areas, including autonomous cars, surveillance systems, and multi-agent systems. In a multi-sensor setting, measurements from sensors located at different places are combined for improved accuracy. However, limited communication capacity or processing power sometimes makes it unfeasible to jointly collect and process detections from all sensors. One way to deal with this problem is using decentralized Multi-object Tracking, where object states (and possibly their densities) are estimated from local sensors and fused to construct better estimation. In this thesis, we have investigated the possibility of using deep-learning-based methods to realize decentralized multi-object tracking. More specifically, we have modified and applied two novel transformer-based trackers to solve decentralized multi-object tracking tasks using synthetic data, namely, MT3 and MT3v2. Further, given the sets of trajectory estimations from different local trackers, we focused on estimating parameters that describe the object states of the current time.

The models' performance was compared in terms of GOSPA and NLL scores with the SOTA Bayesian decentralized filters in simulated scenarios with different sensor settings and parameters of local multi-object trackers. The evaluation results show that MT3 and MT3v2 outperform SOTA judging from the average GOSPA and NLL scores in our simulation settings. Further, we showed that including uncertainty during training can improve the models' performance. Finally, this thesis displays the potential of using deep-learning-based methods for decentralized multi-object tracking tasks and shows that the transformer-based trackers are highly competitive compared with the SOTA trackers.

Bibliography

[1] Jason L Williams. Marginal multi-Bernoulli filters: RFS derivation of MHT, JIPDA, and association-based MeMBer. *IEEE Transactions on Aerospace and Electronic Systems*, 51(3):1664–1687, 2015.

[2] Georg Hess and William Ljungbergh. Transforming the field of multi-object tracking. 2021.

[3] Juliano Pinto, Georg Hess, William Ljungbergh, Yuxuan Xia, Lennart Svensson, and Henk Wymeersch. Next generation multitarget trackers: Random finite set methods vs transformer-based deep learning. In *2021 IEEE 24th International Conference on Information Fusion (FUSION)*, pages 1–8. IEEE, 2021.

[4] Ekim Yurtsever, Jacob Lambert, Alexander Carballo, and Kazuya Takeda. A survey of autonomous driving: Common practices and emerging technologies. *IEEE access*, 8:58443–58469, 2020.

[5] Daniel J Fagnant and Kara Kockelman. Preparing a nation for autonomous vehicles: opportunities, barriers and policy recommendations. *Transportation Research Part A: Policy and Practice*, 77:167–181, 2015.

[6] Philip Koopman and Michael Wagner. Autonomous vehicle safety: An interdisciplinary challenge. *IEEE Intelligent Transportation Systems Magazine*, 9(1):90–96, 2017.

[7] Khan Muhammad, Amin Ullah, Jaime Lloret, Javier Del Ser, and Victor Hugo C de Albuquerque. Deep learning for safe autonomous driving: Current challenges and future directions. *IEEE Transactions on Intelligent Transportation Systems*, 22(7):4316–4336, 2020.

[8] Markus Fröhle, Karl Granström, and Henk Wymeersch. Decentralized Poisson multi-Bernoulli filtering for vehicle tracking. *IEEE Access*, 8:126414–126427, 2020.

[9] Kai Da, Tiancheng Li, Yongfeng Zhu, Hongqi Fan, and Qiang Fu. Recent advances in multisensor multitarget tracking using random finite set. *Frontiers of Information Technology & Electronic Engineering*, 22(1):5–24, 2021.

[10] Emmanuel Delande, Emmanuel Duflos, Philippe Vanheeghe, and Dominique Heurguier. Multi-sensor PHD: Construction and implementation by space partitioning. In *2011 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 3632–3635. IEEE, 2011.

[11] Ba-Ngu Vo and Ba-Tuong Vo. A multi-scan labeled random finite set model for multi-object state estimation. *IEEE Transactions on signal processing*, 67(19):4948–4963, 2019.

[12] Jin Wei and Xi Zhang. Sensor self-organization for mobile multi-target tracking in decentralized wireless sensor networks. In *2010 IEEE Wireless Communication and Networking Conference*, pages 1–6. IEEE, 2010.

[13] Simon J Julier. Fusion without independence. *In Proc. IET Seminar on Target Tracking and Data Fusion: Algorithms and Applications*, 2008.

[14] Tim Bailey, Simon Julier, and Gabriel Agamennoni. On conservative fusion of information with unknown non-Gaussian dependence. In *2012 15th International Conference on Information Fusion*, pages 1876–1883. IEEE, 2012.

[15] Tim Meinhardt, Alexander Kirillov, Laura Leal-Taixe, and Christoph Feichtenhofer. Trackformer: Multi-object tracking with transformers. *arXiv preprint arXiv:2101.02702*, 2021.

[16] Peize Sun, Jinkun Cao, Yi Jiang, Rufeng Zhang, Enze Xie, Zehuan Yuan, Changhu Wang, and Ping Luo. Transtrack: Multiple object tracking with transformer. *arXiv preprint arXiv:2012.15460*, 2020.

[17] Juliano Pinto, Georg Hess, William Ljungbergh, Yuxuan Xia, Henk Wymeersch, and Lennart Svensson. Can deep learning be applied to model-based multi-object tracking? *arXiv preprint arXiv:2202.07909*, 2022.

[18] Yuxuan Xia, Karl Granström, Lennart Svensson, and Ángel F García-Fernández. Performance evaluation of multi-Bernoulli conjugate priors for multi-target filtering. In *2017 20th International Conference on Information Fusion (Fusion)*, pages 1–8. IEEE, 2017.

[19] Ronald PS Mahler. *Statistical multisource-multitarget information fusion*, volume 685. Artech House Norwood, MA, USA, 2007.

[20] Ángel F García-Fernández, Lennart Svensson, Jason L Williams, Yuxuan Xia, and Karl Granström. Trajectory Poisson multi-Bernoulli filters. *IEEE Transactions on Signal Processing*, 68:4933–4945, 2020.

[21] Ba-Ngu Vo, Ba-Tuong Vo, and Hung Gia Hoang. An efficient implementation of the generalized labeled multi-Bernoulli filter. *IEEE Transactions on Signal Processing*, 65(8):1975–1987, 2016.

[22] Patrick Dendorfer, Aljosa Osep, Anton Milan, Konrad Schindler, Daniel Cremers, Ian Reid, Stefan Roth, and Laura Leal-Taixé. Motchallenge: A benchmark for single-camera multiple target tracking. *International Journal of Computer Vision*, 129(4):845–881, 2021.

[23] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[24] Yuxuan Xia, Karl Granström, Lennart Svensson, Ángel F García-Fernández, and Jason L Williams. Extended target Poisson multi-Bernoulli mixture trackers based on sets of trajectories. In *2019 22th International Conference on Information Fusion (FUSION)*, pages 1–8. IEEE, 2019.

[25] Karl Granström, Lennart Svensson, Yuxuan Xia, Jason Williams, and Ángel F García-Femández. Poisson multi-Bernoulli mixture trackers: Continuity through random finite sets of trajectories. In *2018 21st International Conference on Information Fusion (FUSION)*, pages 1–5. IEEE, 2018.

[26] Ángel F García-Fernández, Lennart Svensson, and Mark R Morelande. Multiple target tracking based on sets of trajectories. *IEEE Transactions on Aerospace and Electronic Systems*, 56(3):1685–1707, 2019.

[27] Harold W Kuhn. The Hungarian method for the assignment problem. *Naval research logistics quarterly*, 2(1-2):83–97, 1955.

[28] Kai Han, Yunhe Wang, Hanting Chen, Xinghao Chen, Jianyuan Guo, Zhenhua Liu, Yehui Tang, An Xiao, Chunjing Xu, Yixing Xu, et al. A survey on visual transformer. *arXiv e-prints*, pages arXiv–2012, 2020.

[29] Zhaoyang Niu, Guoqiang Zhong, and Hui Yu. A review on the attention mechanism of deep learning. *Neurocomputing*, 452:48–62, 2021.

[30] Nicolas Carion, Francisco Massa, Gabriel Synnaeve, Nicolas Usunier, Alexander Kirillov, and Sergey Zagoruyko. End-to-end object detection with transformers. In *European conference on computer vision*, pages 213–229. Springer, 2020.

[31] Juliano Pinto, Yuxuan Xia, Lennart Svensson, and Henk Wymeersch. An uncertainty-aware performance measure for multi-object tracking. *IEEE Signal Processing Letters*, 28:1689–1693, 2021.

[32] Ben Mildenhall, Pratul P Srinivasan, Matthew Tancik, Jonathan T Barron, Ravi Ramamoorthi, and Ren Ng. Nerf: Representing scenes as neural radiance fields for view synthesis. In *European conference on computer vision*, pages 405–421. Springer, 2020.

[33] Nasim Rahaman, Aristide Baratin, Devansh Arpit, Felix Draxler, Min Lin, Fred Hamprecht, Yoshua Bengio, and Aaron Courville. On the spectral bias of neural networks. In *International Conference on Machine Learning*, pages 5301–5310. PMLR, 2019.

[34] Zhicheng Zhou, Cheng Zhao, Daniel Adolfsson, Songzhi Su, Yang Gao, Tom Duckett, and Li Sun. Ndt-transformer: Large-scale 3d point cloud localisation using the normal distribution transform representation. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5654–5660. IEEE, 2021.

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

A

Data generation

A.1 Motion and measurement model hyperparameters

Hyperparameter	Explanation	Value
N_s	Number of sensors	3
T	Time steps of simulation	10
Δ_t	Sampling period	0.1s
λ^{birth}	Poisson birth rate	0.1
μ_{v0}	Gaussian mean to initialize new born target states	[0;0;0;0]
σ^2_{v0}	Gaussian covariance to initialize new born target states	diag([10 10 5 5])
p^S	Survival probability	0.95
σ^2_q	Noise intensity of motion model	0.5
$n_{initial}$	Number of initial targets	3
σ^2_z	Measurement noise intensity	0.01 for task 1, 3 0.1 for task 2
p^D	Detection probability	0.9
λ^{clutter}	Poisson clutter intensity	0.0012

Table A.1: Summary of hyperparameters for the data generation

A.2 Local filter parameters

Variable	Value
T_pruning	0.001
T_pruningPois	0.00001
Nhyp_max	100
gating_threshold	20
T_alive	0.0001
existence_estimation_threshold	0.5 for task 1,2/ 0.001 for task 3

Table A.2: Summary of TPMB filter parameters.

A.3 Scenario configuration

Scenario	**Cartesian sensor position**	**Polar sensor position**
1	$((5,-10),(-5,-10),(0,17.5))$	$(-\frac{\pi}{2},-\frac{\pi}{2},\frac{\pi}{2})$
2	$((-0.5,-10),(0.5,-10),(0,10))$	$(-\frac{\pi}{2},-\frac{\pi}{2},\frac{\pi}{2})$

Table A.3: Summary of sensor information.