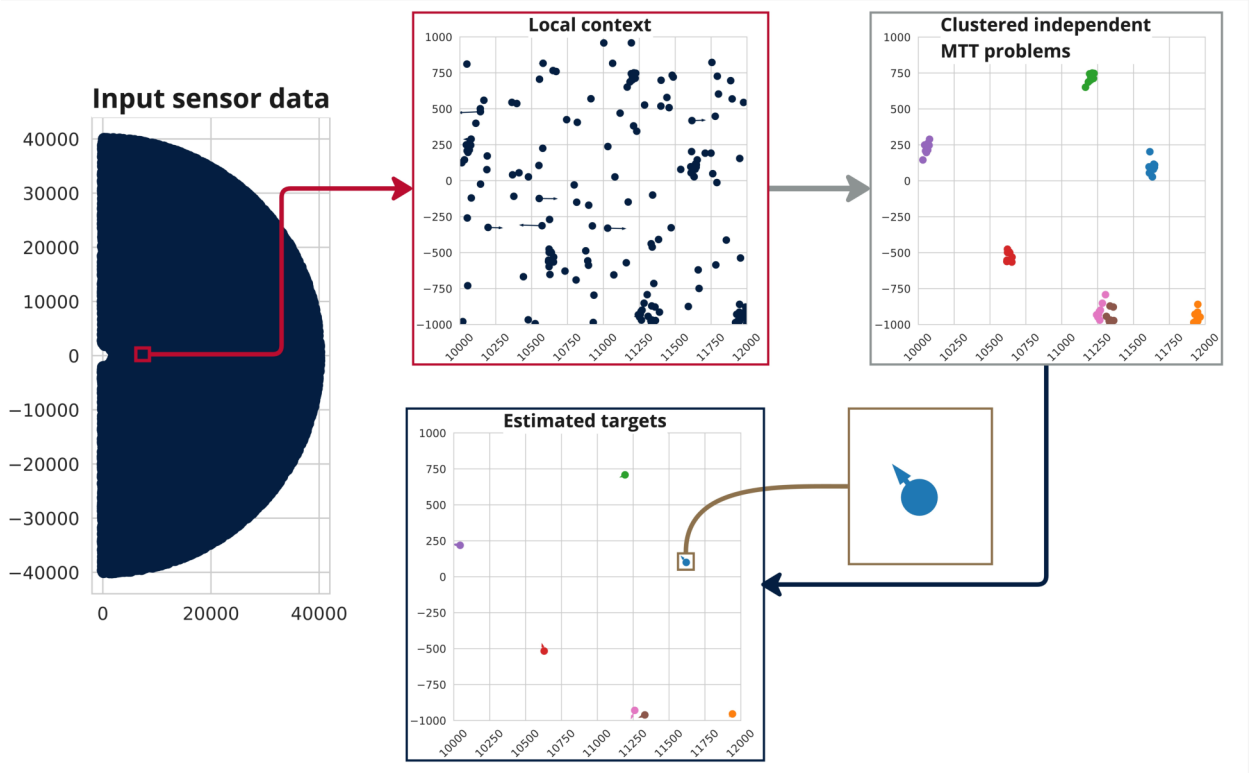
CHALMERS
UNIVERSITY OF TECHNOLOGY



Large-Scale Transformer-Based Multi-Target Tracking

Extending transformer-based MTT methods to an air-surveillance context

Master's Thesis in Engineering Mathematics and Computational Science

OLIVER SPJUTH

DEPARTMENT OF MATHEMATICAL SCIENCES

CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2024
www.chalmers.se

Master's thesis 2024

Large-Scale Transformer-Based Multi-Target Tracking

Extending transformer-based MTT methods to an air-surveillance context

OLIVER SPJUTH

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Mathematical Sciences
Division of Applied Mathematics and Statistics
Chalmers University of Technology
Gothenburg, Sweden 2024

Large-Scale Transformer-Based Multi-Target Tracking
Extending transformer-based MTT methods to an air-surveillance context
OLIVER SPJUTH

Supervisor: Adam Andersson, Department of Mathematical Sciences and Saab AB
Supervisor: Benjamin Svedung Wettervik, Saab AB
Examiner: Adam Andersson, Department of Mathematical Sciences

Master's Thesis 2024
Department of Mathematical Sciences
Division of Applied Mathematics and Statistics
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: An illustration of the different modules proposed in this thesis

Typeset in LaTeX
Printed by Chalmers Reproservice
Gothenburg, Sweden 2024

Large-Scale Transformer-Based Multi-Target Tracking
Extending transformer-based MTT methods to an air-surveillance context
OLIVER SPJUTH
Department of Mathematical Sciences
Chalmers University of Technology

Abstract

In military surveillance, radar-based tracking of objects is essential. The growing use of small-scale drones, as seen in the Russia-Ukraine war, necessitates tracking at low speeds. At these speeds, birds are also detected and the number of false detections increases, making the already complex Multi-Target Tracking (MTT) problem more challenging. Recent advances in machine learning, particularly the transformer architecture, present new opportunities to address these challenges, making it valuable to explore their application in air surveillance contexts.

Although transformers have shown promise in related fields such as automotive radar, adapting them to air surveillance presents specific hurdles. These include managing the quadratic scaling of attention as the number of detections increases, ensuring accurate state estimation across large continuous areas, and simultaneously estimating a large number of targets.

To address these challenges, a four-module pipeline was developed. The first module reduced attention complexity by generating local contexts of detections for parallel processing. This was followed by a transformer-encoder-based filter designed to eliminate false detections (FDF). Next, the original problem was partitioned into independent subproblems using a graph-based clustering approach. One suggested implementation utilized the attention scores from the FDF to construct edges between detections (nodes). The Leiden algorithm, a community detection algorithm, was then applied to identify clusters of related detections. These clusters were subsequently processed in parallel by the final transformer-based MTT module.

This approach significantly reduced the initial memory demands of attention from approximately 320 GB to 1.6 GB while maintaining performance across the pipeline. The false detection filter achieved a balanced accuracy and F1 score of 99%, effectively reducing the problem complexity. The attention-score-based partitioning method accurately identified subproblems that were predominantly optimal (single-target) or near-optimal.

When evaluated using MTT metrics, the pipeline employing the attention-score-based partitioning method demonstrated promising results, with few missed or false detections and a total inference time of approximately 0.5 seconds for over 100,000 detections. The system scaled effectively with increased complexity and adapted well to varying conditions.

Keywords: transformer, multiple target tracking, data association, leiden algorithm, clustering, attention, radar tracking, radar, graph clustering

Acknowledgements

First and foremost, I would like to express my sincere gratitude to Saab for providing me with the opportunity to pursue this thesis. I am especially thankful to my advisors, Benjamin Svedung Wettervik and Adam Andersson, for their support throughout this process. Their willingness to discuss the challenges I encountered, coupled with their insightful feedback during the writing of this report, was instrumental in shaping the outcome of this work.

This thesis became a significant part of my life and extended beyond the timeline I initially envisioned. The journey was not without its challenges, but throughout this process, I was fortunate to have the constant support of my partner, Matilda. Without her, this thesis may not have been completed.

Oliver Spjuth, Gothenburg, December 2024

List of Acronyms

Below is the list of acronyms that have been used throughout this thesis, listed in alphabetical order:

AMI	Adjusted Mutual Information
ARI	Adjusted Rand Index
ASP	Attention Score Pipeline
EPP	Elliptical Proximity Pipeline
FDF	False Detection Filter
FFN	Feedforward Neural Network
FOV	Field of View
GOSPA	Generalized Optimal Subpattern Assignment
LCG	Local Context Generation
MB RFS	Multi-Bernoulli Random Finite Set
MHT	Multi-Hypothesis Tracker
MOT	Multiple Object Tracking (used interchangeably with MTT)
MT3LS	Multi-Target Tracking Transformer Large Scale
MT3v2	Multi-Target Tracking Transformer version 2
MTT	Multiple Target Tracking (used interchangeably with MOT)
NLL	Negative Log-Likelihood
NMI	Normalized Mutual Information
PNP	Probabilistic Normalization Pipeline
RFS	Random Finite Set
ReLU	Rectified Linear Unit
RI	Rand Index
SRRP	Statistical Range-Rate Pipeline
SOTA	State of the Art

Contents

1
Introduction

Object tracking, particularly multiple object tracking (MOT), also known as multiple target tracking (MTT), is a crucial element in various decision-making processes, from everyday activities, such as crossing the road, to more complex scenarios, including orchestrating air traffic or neutralizing missile threats. MOT plays a vital role in forming accurate situational awareness that aids in safe and effective decision-making.

Today, MOT applications make use of data from a wide range of different sources, e.g. radar, lidar, and cameras. The primary focus of this thesis is on the application of MOT within a military context, specifically radar surveillance of vast areas.

1.1 Background and Motivation

In the military domain, creating accurate situational awareness is a matter of life and death. Consequently, countries have invested heavily in the research and development of radar systems, especially during the pre-World War II era, which eventually led to the advancement of modern radar technologies [14].

Radars work by transmitting electromagnetic waves, receiving the reflected waves, and processing the information from these reflections to detect objects of interest [14]. Since these waves are reflected not only by targets of interest, but also by backgrounds such as the ground and even rain, radar systems have to perform filtering to identify detections corresponding to true targets [13]. However, this filtering can result in both detections from targets of interest being filtered out, and also detections from non-targets being present, the first known as missed detections and the latter being known as false detections (or false alarms) [13]. One challenge in radar data processing is to accurately distinguish between true targets and false detections. A related challenge is resolving the data association ambiguity: determining which detections correspond to which objects, here we also include the challenge of identifying objects in the first place.

Radars have from the start had a number of different functions, for example, during World War II, one of the most important uses of radar was the detection of air raids [14]. Many applications primarily involve the identification of objects, their velocities, and trajectories. Subsequent analysis of these data facilitates various applications, such as determining the nature of the object, its origin, and potential

destinations. For example, recognizing an object as a missile and calculating its trajectory from inception to termination can provide critical information in a combat scenario. Differentiating between similar targets, such as birds and drones that travel at comparable speeds, is another important application, as witnessed by the Russia-Ukraine war [42].

One of the most critical uses of radar technology in modern times is surveillance. Whereas early surveillance radar systems required manual tracking of objects [20] [3], modern radar systems employ digital processing to automatically analyze radar data [20]. In this context, MOT algorithms are essential to estimate the number and states of multiple objects over time, even when these detections are contaminated with noise and false detections.

1.2 Problem Definition

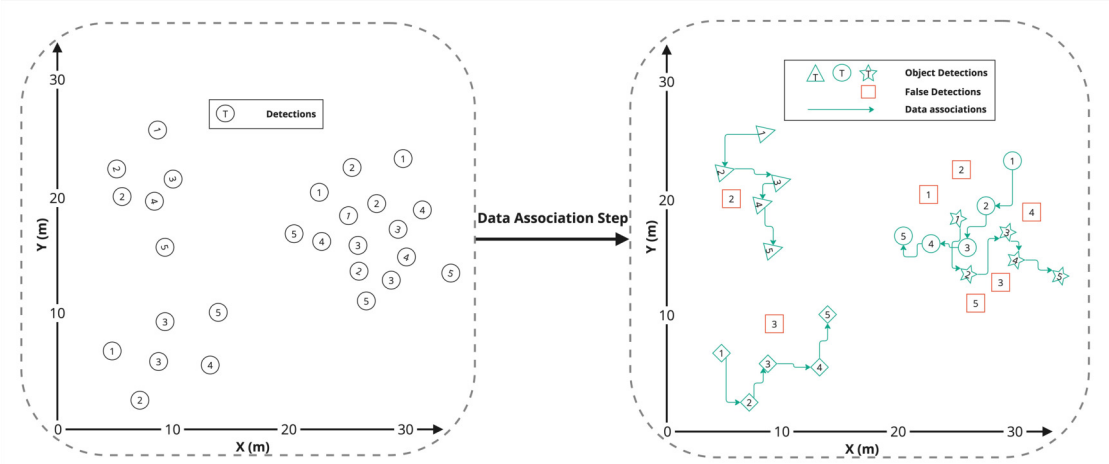


Figure 1.1: An illustration of data association in MTT. The left graph contains detections before associations are made, with the number inside the shapes representing the time of each detection. On the right side, one possible data association is shown. The shapes with green outlines are deemed to be originating from objects, where shapes differentiate the different objects, and lines connect object detections. Shapes with red outlines are deemed to be false detections.

Military radar systems must track multiple objects simultaneously, including aircraft, drones, and even birds, leading to a complex data association problem that needs to be efficiently solved. This includes both the process of identifying targets and associating targets with detections. Given target-detection associations, the state of the targets can be estimated. The data association problem is illustrated on a small scale in Figure 1.1. Although classical approaches such as the Multi-Hypothesis Tracker (MHT) [4] and more recent Random Finite Set (RFS) methods [17] are often employed to solve the MOT problem, these methods often struggle to balance accuracy and computational efficiency in environments with dense targets and high clutter intensity. As a result, exploring alternative approaches is of signif-

icant interest.

Following the introduction of the transformer architecture and its core mechanism, *attention*, in the influential paper *Attention is All You Need* [27], transformer-based models initially revolutionized natural language processing (NLP) tasks and have since rapidly advanced across numerous fields. Models such as Detection Transformer (DETR) [31], Deformable DETR [35], and TrackFormer [36] have demonstrated impressive capabilities in handling MOT tasks, particularly in video applications.

This thesis is inspired by the success of the Multi-Target Tracking Transformer v2 (MT3v2) model developed at Chalmers University of Technology. MT3v2 was specifically designed to process radar point clouds and generate filter estimates that are directly comparable to RFS models. The study demonstrated promising results for MT3v2 by comparing it with current state-of-the-art (SOTA) RFS-based methods [37]. In some scenarios, MT3v2 performed better than the SOTA RFS models in the Generalized Optimal Sub-Pattern Assignment (GOSPA) metric [26] [37]. Furthermore, the model exhibited faster inference times compared to the SOTA models [37].

The MT3v2 model was tested in scenarios similar to those of an automotive radar with a couple of hundred detections and a circular sector FOV with a radius of 15 meters [37].

1.3 Aim and Scope

In contrast to previous work, such as the MT3v2 model, this thesis explores the application of machine learning to the problem of large-scale multi-object tracking (MOT) using radar data. The field of view (FOV) in this context spans tens of kilometers, with around 100,000 detections (including false detections) originating from around 10,000 objects. The complexity of this problem is further compounded by the requirement for real-time processing, necessitating a careful balance between accuracy and computational efficiency. The aim of this thesis is to address these challenges by adapting transformer-based models for large-scale MOT.

1.4 Thesis Contributions

This thesis began by investigating whether the MT3v2 model could be adapted for large-scale tracking environments. The initial challenge involved addressing the memory requirements associated with processing 100,000 detections, a scale far beyond the original design of MT3v2. Although memory-related challenges were successfully mitigated, additional issues emerged, highlighting the limitations of simply scaling up the MT3v2 model for such complex scenarios.

The focus shifted towards simplifying the problem complexity instead of attempting

large-scale multi-target tracking with MT3v2. A multi-target tracker pipeline was designed, comprising several key components to efficiently manage and solve the large-scale tracking problem. The contributions of this pipeline include:

- **Local Context Generation Module:** An algorithm was developed to group spatially close detections, resulting in local contexts, thus reducing overall complexity by breaking down the large dataset into smaller, manageable subsets.
- **False Detection Filter:** A transformer-encoder-based filtering mechanism was introduced to distinguish between real and false detections, minimizing the computational burden by identifying data points that are likely to be irrelevant early in the process.
- **Partitioning Module:** The detections are then partitioned into independent sub-problems. Four different algorithms are implemented to generate a graph that can be effectively clustered by the Leiden algorithm. Three of them are more traditional statistical approaches while the fourth approach utilizes the attention-scores of the false detection filter. This step makes the following tracking task more manageable.
- **MT3LS Model:** A modified version of the MT3v2 model, referred to as Multi-Target Tracking Transformer Large Scale (MT3LS), was designed to process these localized sub-problems in parallel. This approach allows for effective tracking across the large field of view while maintaining computational efficiency.

This modular approach helped overcome the complexities of large-scale multi-object tracking, focusing on scalability, efficiency, and practicality for real-world applications.

Four different implementations of the pipeline, where each version corresponds to a different partitioning module algorithm, are evaluated and contrasted across a series of simulated scenarios with varying complexities. Metrics such as missed target rate, false prediction rate, and state estimation errors are used to assess their effectiveness. Additionally, the MT3LS model's performance is evaluated separately, with a comparison to the original MT3v2 model using the GOSPA metric.

1.5 Use of AI Tools

In the interest of transparency and adherence to ethical standards, I disclose that AI tools were employed during the writing of this thesis. These tools were exclusively used to assist with sentence rephrasing and improving the clarity of the text. All outputs generated by these tools were carefully reviewed and edited to ensure they met the academic and scientific standards of this work. All scientific and technical contributions were independently developed by the author.

1.6 Structure of the Thesis

Chapter 2: Target Tracking

This chapter introduces the fundamentals of radar-based target tracking, covering the principles of radar operation, measurement noise modeling, and the radar model used in this thesis. It discusses the transformation of radar measurements into Cartesian coordinates, introduces Bayesian filtering for state estimation, and explores multi-object tracking (MOT) techniques, including Random Finite Set (RFS) methods and the Multi-Hypothesis Tracker (MHT). The Generalized Optimal Subpattern Assignment (GOSPA) metric for evaluating tracking performance is also explained.

Chapter 3: Machine Learning

Here we introduce foundational machine learning concepts, including neural networks, normalization, and overfitting prevention. Key ideas behind the transformer architecture, such as attention mechanisms and embeddings, are also discussed, along with binary classification metrics and clustering algorithms like the Leiden algorithm. These concepts provide the theoretical basis for understanding the MT3v2 model in Chapter 4 and the machine learning components of the proposed pipeline in Chapter 5.

Chapter 4: Multi-Target Tracking Transformer v2

This chapter introduces the Multi-Target Tracking Transformer v2 (MT3v2), a transformer-based model developed for Multi-Object Tracking (MOT) using radar detections. The chapter provides an overview of the model's architecture and its core modules, including the encoder, decoder, and loss formulation.

Chapter 5: Large-Scale Target Tracking Pipeline

Here we explore the challenges and complexities of large-scale multi-object tracking (MOT) and present the proposed modular tracking pipeline. This chapter introduces the local context generation module, the false detection filter, the partitioning module with four different implementations, and the modified MT3LS model, highlighting their roles in addressing scalability and efficiency.

Chapter 6: Simulation Setup and Parameter Optimization

Chapter 6 details the simulation environment and outlines the parameter selection process that serves as the basis for the evaluation of the proposed pipeline. This chapter also evaluates the false detection filter and the four different implementations of the partitioning module.

Chapter 7: Evaluation and Discussion

This chapter evaluates the performance of the proposed tracking pipeline on metrics such as missed target rate, false prediction rate, and Euclidean distance for state estimation errors. Additionally, the MT3LS model's performance is evaluated separately, including a comparison to the original MT3v2 model using the GOSPA metric. It concludes with discussions on various topics relevant to the thesis, such as the choice of clustering algorithm in the partitioning module and applications and potential use cases.

Chapter 8: Conclusions and Future Work

The thesis concludes by summarizing the key findings, discussing their implications, and highlighting potential directions for future research.

2
Target Tracking

In this chapter, the fundamentals of radar-based target tracking are discussed, starting with the principles of radar operation, missed and false detections, and the simplified radar model used in this thesis in Section 2.1. The challenge of measurement noise and its mathematical modeling are then introduced in Section 2.1.1. The conversion of radar measurements from polar to Cartesian coordinates and the associated uncertainties are covered in Section 2.1.2. The section ends with a short discussion on the notion of radar scans and brings the concepts in the section together with Figure 2.1.

The concept of Bayesian filtering, which is essential for estimating object states from noisy measurements, is introduced in Section 2.2, while multi-object tracking (MOT) methodologies are discussed in Section 2.3. This section introduces Random Finite Set (RFS) methods in Section 2.3.1 and the Multi-Hypothesis Tracker (MHT) approach in Section 2.3.2, both of which are prominent approaches to addressing the complexities of MOT. Finally, the Generalized Optimal Subpattern Assignment (GOSPA) metric, used to evaluate the performance of multi-target tracking models, is detailed in Section 2.3.3.

This chapter lays the groundwork for understanding the techniques and challenges involved in radar-based multi-target tracking.

2.1 Radar Data in a Target Tracking Context

Radar-based target tracking involves using electromagnetic waves to detect and monitor objects within a specific area, known as the radar field of view (FOV). Typically, this FOV is represented as a circle-sector, defining the spatial boundaries where radar systems can detect targets.

Radar systems operate by emitting electromagnetic waves that travel through the environment and reflect off objects, creating echoes. These reflected signals are processed to generate detections with associated measurements. Each detection represents the radar's estimation of an object's location and, potentially, velocity, and other valuable information based on the returned signal. This information is essential to track the movement and position of objects in various applications, such as surveillance, navigation, and autonomous systems [8].

Since electromagnetic waves can reflect off anything, not only targets of interest, such as ground and raindrops and other *clutter*, the signal processing step needs to discriminate and filter out echoes in order to produce detections of value for the particular application. As this is not a perfect process, detections that are not of interest will sometimes pass through, which we shall henceforth refer to as *false detections*. In addition, detections from targets of interest might be filtered out if the echoes are not strong enough or resemble that of clutter or if the target happens to be occluded from the radar point of view. We will refer to this phenomenon as *missed detections*.

There are different radars for different applications, and because of this, the way they operate and the information they produce in the form of detections can vary just as much. Some radars are airborne and constantly moving with the goal of observing only ground-targets, others are specifically made to track objects such as missiles. The measurements of these radar systems can therefore be completely different.

In the context of this thesis, we will consider a simplified version of a radar. This radar operates in a world where objects are represented by a 4-dimensional vector $\mathbf{x}$ consisting of Cartesian position and velocity, known as the object *state vector*:

$$\mathbf{x} = \begin{bmatrix} x \\ y \\ \dot{x} \\ \dot{y} \end{bmatrix}. \tag{2.1}$$

In addition, targets (objects) are modelled as point targets. That is, they have no extent; they are simply moving dots with a singular position. Because of this, the targets will not occlude one another, but can still be modelled to be missed by the radar with a certain probability.

In Section 2.1.1 the detection measurements and corresponding uncertainties of this simplified radar are described. The following Section 2.1.2 describes the process of converting the radar measurements and uncertainties to Cartesian coordinates and the modeling assumptions that accompany them. The concluding Section 2.1.3 brings together the information of the previous sections and describes how this simplified radar operates together with the introduction of some key terminology such as *scan time*. Some of the concepts are also visualized and can be seen in Figure 2.1

2.1.1 Radar Measurements

A simplified model of a radar is now described, with detections represented by measurements $\mathbf{z}$ in polar coordinates, which capture the range (distance) from the radar r, the angle to the radar φ, and the radial velocity (velocity from or to the radar) known as *range rate* $\dot{r}$ [8]:

$$\mathbf{z} = \begin{bmatrix} r \\ \dot{r} \\ \varphi \end{bmatrix}. \tag{2.2}$$

The measurement vector is thus incomplete compared to the state vector in (2.1), as the tangential velocity component (velocity perpendicular to the radar) is not measured. The measurements relation to the components of the Cartesian state vector (2.1) is as follows:

$$r = \sqrt{x^2 + y^2}, \tag{2.3}$$

$$\dot{r} = \frac{\mathrm{d}r}{\mathrm{d}t} = \frac{x\dot{x} + y\dot{y}}{\sqrt{x^2 + y^2}}, \tag{2.4}$$

$$\varphi = \operatorname{arctan2}(y, x) = \begin{cases} \arctan(\frac{y}{x}) & \text{if } x > 0 \\ \arctan(\frac{y}{x}) + \pi & \text{if } x < 0 \text{ and } y \geq 0 \\ \arctan(\frac{y}{x}) - \pi & \text{if } x < 0 \text{ and } y < 0 \\ +\frac{\pi}{2} & \text{if } x = 0 \text{ and } y > 0 \\ -\frac{\pi}{2} & \text{if } x = 0 \text{ and } y < 0 \\ \text{undefined} & \text{if } x = 0 \text{ and } y = 0 \end{cases}. \tag{2.5}$$

In addition to the measurements being incomplete representations of a target's underlying state vector (2.1), the measurements are also noisy (that is, the measured position and radial velocity are different from the true state). The measurements are thus modeled as random with measurement errors $\mathbf{v}$ added to the true, but incomplete, polar state $\mathbf{s}$:

$$\mathbf{z} = \mathbf{s} + \mathbf{v} = \begin{bmatrix} r_\mathrm{s} \\ \dot{r}_\mathrm{s} \\ \varphi_\mathrm{s} \end{bmatrix} + \begin{bmatrix} r_\mathrm{e} \\ \dot{r}_\mathrm{e} \\ \varphi_\mathrm{e} \end{bmatrix}. \tag{2.6}$$

In this thesis, we assume that the measurement errors $\mathbf{z}_\mathrm{e}$ are constant and are normally distributed with a mean of zero and a diagonal covariance matrix, represented as:

$$\begin{aligned} \mu_\mathrm{e} &= \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}, \\ \mathbf{\Sigma}_\mathrm{e} &= \begin{bmatrix} \sigma_r & 0 & 0 \\ 0 & \sigma_{\dot{r}} & 0 \\ 0 & 0 & \sigma_\varphi \end{bmatrix}, \\ \mathbf{v} &\sim \mathcal{N}\left(\mu_\mathrm{e},\, \mathbf{\Sigma}_\mathrm{e}\right), \end{aligned} \tag{2.7}$$

here, σ_r^2, $\sigma_{\dot{r}}^2$, and σ_φ^2 represent the variances of the range, range rate, and angle measurements, respectively. Moreover, it is assumed that these variances are known a priori.

2.1.2 Polar to Cartesian Measurement and Uncertainty Conversion

As targets are modeled with a Cartesian state vector (2.1), converting measurements and uncertainties to Cartesian coordinates can be beneficial for certain applications.

A straightforward approach to convert from polar to Cartesian coordinates involves multiplying the range r by the cosine and sine of the angle φ. However, this approach introduces bias, as $\mathbb{E}[r\sin\varphi] = r_\mathrm{s}\sin\varphi_\mathrm{s}e^{-\sigma_\varphi^2/2}$ where the subscript s denotes the true values [29]. To correct for this bias, the following unbiased conversion can be applied:

$$\begin{aligned} \lambda_\varphi &= e^{-\sigma_\varphi^2/2}, \\ \mathbf{z}_\mathrm{uc} &= \begin{bmatrix} \lambda_\varphi^{-1} r\cos(\varphi) \\ \lambda_\varphi^{-1} r\sin(\varphi) \\ \dot{r} \end{bmatrix} = \begin{bmatrix} x_\mathrm{uc} \\ y_\mathrm{uc} \\ \dot{r} \end{bmatrix}, \end{aligned} \tag{2.8}$$

here, the subscript uc stands for unbiased conversion. The corresponding components of the covariance matrix for these converted measurements can then be calculated as follows [29]:

$$\begin{aligned} \gamma_\varphi &= e^{-2\sigma_\varphi^2}, \\ \lambda_\varphi &= e^{-\sigma_\varphi^2/2}, \\ \sigma_{x_\mathrm{uc}}^2 &= \left(\lambda_\varphi^{-2} - 2\right) r^2\cos^2(\varphi) + \frac{1}{2}\left(r^2 + \sigma_r^2\right)\left(1 + \gamma_\varphi\cos(2\varphi)\right), \\ \rho_{x_\mathrm{uc}y_\mathrm{uc}}\sigma_{x_\mathrm{uc}}\sigma_{y_\mathrm{uc}} &= \left(\lambda_\varphi^{-2} - 2\right) r^2\cos(\varphi)\sin(\varphi) + \frac{1}{2}\left(r^2 + \sigma_r^2\right)\gamma_\varphi\sin(2\varphi), \\ \sigma_{y_\mathrm{uc}}^2 &= \left(\lambda_\varphi^{-2} - 2\right) r^2\sin^2(\varphi) + \frac{1}{2}\left(r^2 + \sigma_r^2\right)\left(1 - \gamma_\varphi\cos(2\varphi)\right), \\ \sigma_{\dot{r}}^2 &= \sigma_{\dot{r}}^2. \end{aligned} \tag{2.9}$$

Although the polar measurement uncertainties are normally distributed, the unbiased converted measurements are not, making the assumption of normality an approximation for the errors in the converted Cartesian measurements.

2.1.3 Radar Scans

A real radar operates by scanning its field of view (FOV) with electromagnetic waves. It can do this either by standing still and electronically shifting the direction of the waves or by rotating, or both. Since the transmitted waves can reflect off objects (targets) at different distances, these reflections return at different points in time. As a result, detections are also generated at no particular point in time.

In this thesis, as mentioned in 2.1, we are not considering a real radar, but a simplified one. This simplified radar sends out electromagnetic waves over its entire FOV instantaneously and instantaneously receives and processes all echos. These echos arrive at the same time, but can be measured to be at any distance from the radar (i.e. the laws of physics are violated). Between each scan, the radar does nothing for a set amount of time. As such, incoming detections can be grouped in the scans to which they belong, and each scan index can be mapped to the time when the scan took place. We refer to this as the *scan time* of a detection, which

is the measurement time. Let $\mathbf{D}_\mathrm{T}$ denote the sequence of n_s detections originating from the scan T:

$$\mathbf{D}_\mathrm{T} = \left(\begin{bmatrix} r_1 \\ \varphi_1 \\ \dot{r}_1 \\ \sigma_{r_1} \\ \sigma_{\varphi_1} \\ \sigma_{\dot{r}_1} \end{bmatrix}, \begin{bmatrix} r_2 \\ \varphi_2 \\ \dot{r}_2 \\ \sigma_{r_2} \\ \sigma_{\varphi_2} \\ \sigma_{\dot{r}_2} \end{bmatrix}, \ldots, \begin{bmatrix} r_{n_\mathrm{s}} \\ \varphi_{n_\mathrm{s}} \\ \dot{r}_{n_\mathrm{s}} \\ \sigma_{r_{n_\mathrm{s}}} \\ \sigma_{\varphi_{n_\mathrm{s}}} \\ \sigma_{\dot{r}_{n_\mathrm{s}}} \end{bmatrix} \right), \tag{2.10}$$

here, each detection is represented by its measurements $(r, \varphi, \dot{r})$ and the corresponding measurement uncertainties $(\sigma_r, \sigma_\varphi, \sigma_{\dot{r}})$.

In order to gain some intuition of the radar measurements of this simplified radar, such as what is being measured, and how the measurements differ from the ground-truth state vectors (2.1), both in terms of noise (uncertainties), incomplete measurements (tangential velocity missing), together with the prospect of false and missed detections, Figure 2.1 has been created. In this figure, O_T is a sequence that contains the state vectors (2.1) of all targets present in the field of view at the *scan time* corresponding to scan T.

2.2 Single-Object Filtering

Filtering is a fundamental process in state estimation, where the objective is to estimate the state of an object or system (modeled as random) at a specific time T based on a sequence of noisy measurements obtained at times $s \leq \mathrm{T}$.

For example, let us denote $\mathbf{X}_\mathrm{T} = \begin{bmatrix} x_\mathrm{T} & y_\mathrm{T} & \dot{x}_\mathrm{T} & \dot{y}_\mathrm{T} \end{bmatrix}^\mathrm{T}$ as the state of a single object at some scan time T and $\mathrm{Y}_\mathrm{T} = \{\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_\mathrm{T}\}$ as the set of radar detections with measurements corresponding to this single object up to and including scan time T. For this example, we do not consider the possibility of false detections being present. The state estimation (filtration) process is formalized by calculating the filtering distribution $p\left(\mathbf{X}_\mathrm{T} | \mathrm{Y}_\mathrm{T}\right)$, which represents the probability density of the state $\mathbf{X}_\mathrm{T}$ given Y_T [39]. Thus, this distribution contains more information on the state of the underlying object at scan time T than any of the detection measurements in Y_T on its own.

Given models for the state transition (in this case, how the object moves), measurement uncertainties, etc., the filtering distribution can be either estimated or calculated analytically. The Kalman filter is an example where the underlying modeling assumptions allow for analytically calculating the filtering distribution.

2.3 Multi-Object Tracking (MOT)

In this thesis, we are not estimating the state of a single object at some scan time $\mathbf{X}_\mathrm{T}$ given detections Y_T of only this object up to and including scan time T. Instead, we are estimating the state of multiple objects simultaneously, taking into account

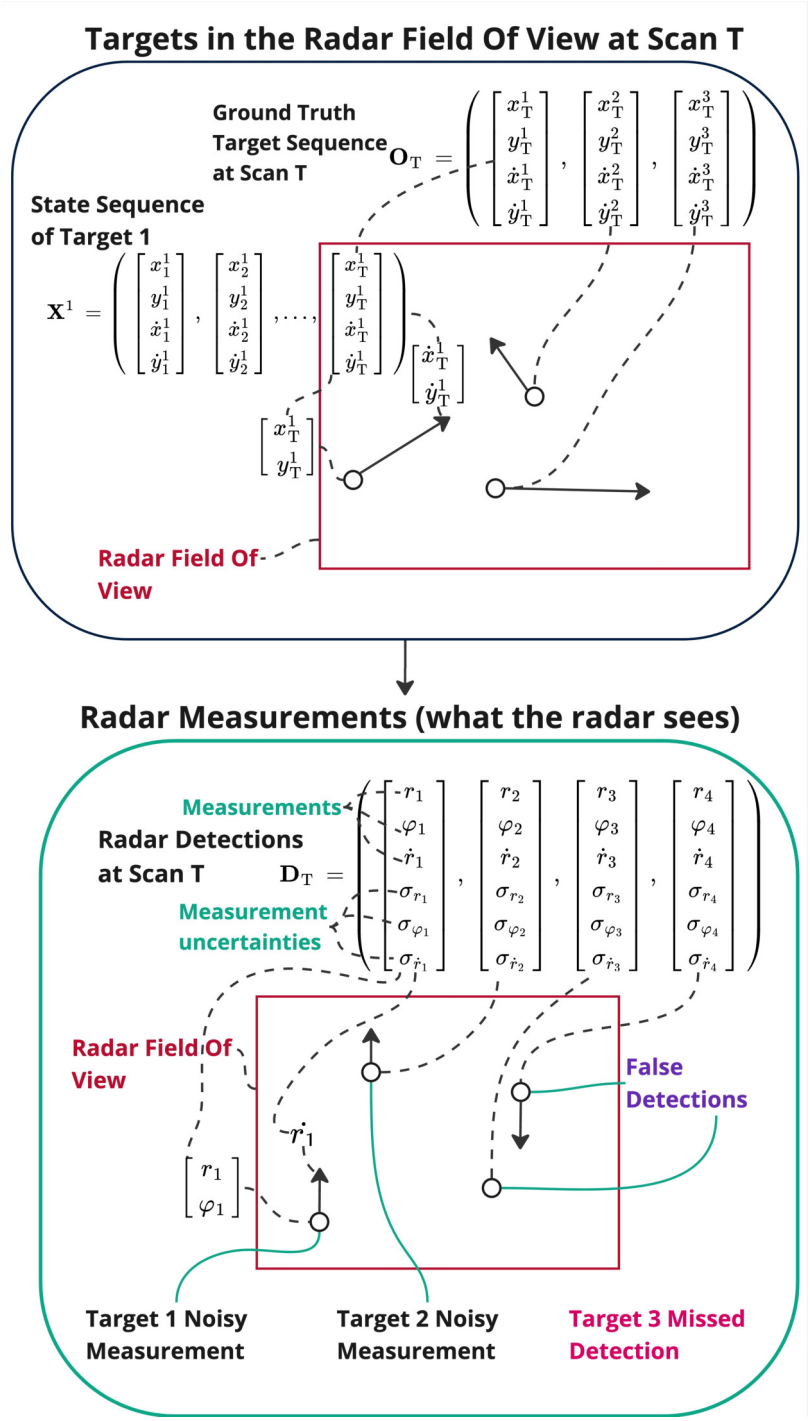


Figure 2.1: Illustration of the radar detection process given some true targets in its FOV.

the possibility of objects entering or leaving the scene or being missed by the sensor (radar), the presence of false detections, and more. This is known as Multi-Object Tracking (MOT), or equivalently Multi-Target Tracking (MTT).

The MOT state estimation process can be viewed as computing the filter distribution $p(\mathrm{O_T}|\mathrm{Y_T})$, where $\mathrm{O_T} = \{\mathbf{X}_\mathrm{T}^1, \mathbf{X}_\mathrm{T}^2, \ldots, \mathbf{X}_\mathrm{T}^N\}$ represents the collective state of N objects at time T, and $\mathrm{Y_T} = \{\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_{n_\mathrm{m}}\}$ denotes the set of all n_m measurements up to time T, possibly including false detections.

The number of objects N is unknown and can change between each time T to $\mathrm{T}+1$. To accurately model the true states of multiple, and possibly changing, number of objects, a model thus needs to estimate the number of underlying objects and their states given measurements. These measurements could in turn originate from any object or represent a false detection. This is a difficult task, and a multitude of approaches have been pursued, and in the following subsections two separate methodologies are described.

2.3.1 Random Finite Set Methods

Bayesian methods for Multi-Target Tracking (MTT) often leverage the Random Finite Set (RFS) framework, which provides a rigorous approach to modeling the uncertainty in the number, and states, of multiple objects. It allows for the joint modeling of all aspects of MOT. The RFS framework models the MOT problem by representing the state of the system as a random finite set. [40]

One common approach to model the multi-target state $\mathrm{O_T}$ at some time T is the multi-Bernoulli (MB) RFS filter [40]. The MB RFS models the fact that targets can be described as either existing or not, and if they exist, their state can be described by some distribution. We can thus model the targets $\mathbf{X}_\mathrm{T}^i$ existence by a Bernoulli distribution and the state of the targets by some other distribution, for example, a normal distribution.

The filter that operates with the MB RFS model assumes that each new measurement can belong to any previous target, represent the birth of a new target, or be a false detection (clutter). Each birth is represented by a Bernoulli RFS, and the births are thus represented by an MB RFS. False detections or clutter are modeled by a Poisson distribution. The filter density can at some time T be represented by N Bernoulli components, which include all potential objects born up to time T. Below is an example where the state of the objects is parametrized by a normal distribution:

$$\mathrm{MB_T} = \left\{\left(\mathbf{p}_\mathrm{T}^i, \mathbf{M}_\mathrm{T}^i, \mathbf{\Sigma}_\mathrm{T}^i\right)\right\}_{i=1}^{N},$$

here, $\mathbf{p}_\mathrm{T}^i$ represents the probability of existence of the object i, $\mathbf{M}_\mathrm{T}^i$ is the mean vector of the normal distribution that describes the state, and $\mathbf{\Sigma}_\mathrm{T}^i$ is the corresponding covariance matrix.

There are many more sophisticated ways to model the MTT problem within the RFS framework, an example is Poisson multi-Bernoulli mixture (PMBM) filter [28]. The PMBM models, among other things, the potential of objects not being detected yet as a Poisson distribution, and has for each potential object an MB RFS representation. Each multi-Bernoulli component for each potential object can be viewed to represent a so-called hypothesis density of this target. This is somewhat implicitly related to the more explicit hypothesis management of a Multi-Hypothesis tracker, discussed in the following section.

2.3.2 Multi-Hypothesis Tracker (MHT)

The Multi-Hypothesis Tracker (MHT) is another key approach in MOT. It addresses the data association and the state estimation problem separately. MHT maintains multiple hypotheses about possible measurement-to-object associations, updating and pruning these hypotheses as new measurements arrive. This method enables effective tracking in cluttered and ambiguous environments, since correct associations may not be immediately apparent, but become clearer in hindsight. In such scenarios, single-hypothesis trackers have a high likelihood of failing [4].

Mathematically, MHT can be represented as maintaining a set of hypotheses $\mathcal{H}_t = \{H_t^1, H_t^2, \ldots, H_t^M\}$ at each time step t, where each hypothesis H_t^m represents a possible sequence of associations from time 0 to t. Often, there are groups of hypotheses that have a common starting point, which usually constitute a tree of hypotheses for a certain believed object. The likelihood of each hypothesis is computed based on the measurement model and the prior probabilities of the object states, with the most likely hypotheses being retained for future updates [4]. At each step, the MHT model presents the most likely so-called global hypothesis, which often is the most likely hypothesis from each group of hypotheses, such that there is no or minimal overlap of detections between the selected hypotheses. For each of these hypotheses, the single-target filtration results can be presented as the states of the believed objects. An illustration of the hypothesis approach is shown in Figure 2.2.

2.3.3 GOSPA metric

When it comes to evaluating the performance of an MOT model (tracker), there is a wide variety of methods. Compared to evaluating a simple single-target filtering model such as the Kalman filter, in MOT there is much more that needs to be taken into account. A metric to evaluate an MOT model is the GOSPA metric [26]. GOSPA takes into account not only the predictions, but also missed objects and false predictions. In the following, we will introduce a special case of the GOSPA metric, denoted $d_1^{c,2}(X, Y)$, which uses the Euclidean distance as a base for the state cost.

The predictions and corresponding ground truth objects are represented as sequences of vectors $X = (\mathbf{x}^1, \ldots, \mathbf{x}^{\mathrm{n_X}})$ and $Y = (\mathbf{y}^1, \ldots, \mathbf{y}^{\mathrm{n_Y}})$ where $\mathrm{n_X} \leq \mathrm{n_Y}$ and $\mathbf{x}^i, \mathbf{y}^i \in \mathbb{R}^{\mathrm{d_s}}$. As such, each vector represents either a predicted state vector or a ground truth state

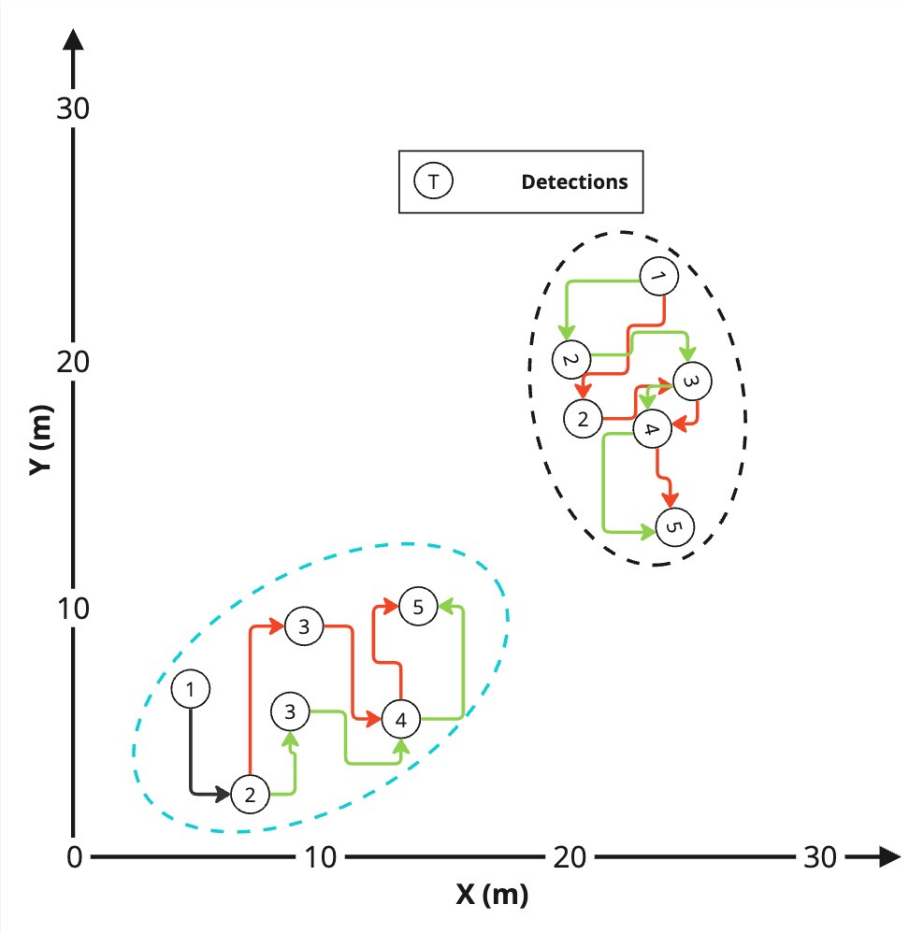


Figure 2.2: Illustration of potential hypothesis handled by an MHT algorithm. Two different groups of hypothesis are shown within ellipses with dashed lines of different colors. The global hypotheses in this scenario are all combinations of the local hypothesis in each group (green-green, red-green, green-red, red-red), since there is no conflict with detections between them.

vector. Either X or Y can represent the predictions, as the metric is symmetric.

The metric consists of a minimization procedure that finds the optimal mapping function π from the contents of the sequence X to Y, such that the value of the metric is minimized. The mapping function can be viewed as a sequence itself, in the space of all possible permutations of the set $\{1, \ldots, \mathrm{n_Y}\}$.

The distance measure $d\left(\mathbf{x}^i, \mathbf{y}^{\pi(i)}\right)$, if using Euclidean distance, is defined as:

$$d\left(\mathbf{x}^i, \mathbf{y}^{\pi(i)}\right) = \sqrt{\sum_{j=1}^{\mathrm{d_s}} \left(\mathbf{x}_j^i - \mathbf{y}_j^{\pi(i)}\right)^2}.$$

The state cost, which penalizes the predictions that are within some matching distance c (also known as cut-off distance) can now be defined as follows:

$$s_{\text{cost}} = \sum_{i=1}^{|X|} d\left(\mathbf{x}^i, \mathbf{y}^{\pi(i)}\right) \mathbb{1}_{d(\mathbf{x}^i, \mathbf{y}^{\pi(i)}) \leq c},$$

here, c is a parameter that can be decided on based on the problem at hand. The number of matched predictions and ground truth states n_{m} is calculated similarly:

$$n_{\text{m}} = \sum_{i=1}^{|X|} \mathbb{1}_{d(\mathbf{x}^i, \mathbf{y}^{\pi(i)}) \leq c}.$$

We can now define the GOSPA metric $d_1^{c,2}(X, Y)$ as follows:

$$d_1^{c,2}(X, Y) \triangleq \min_{\pi \in \prod_{|Y|}} \left(s_{\text{cost}} + \frac{c}{2} \left(|Y| + |X| - 2n_{\text{m}} \right) \right), \tag{2.11}$$

here, $\frac{c}{2}\left(|Y| + |X| - 2n_{\text{m}}\right)$ represents the combined false prediction and missed target cost.

We can decompose the GOSPA metric $d_1^{c,2}(X, Y)$ into components that are easier to interpret. First, we define the normalized localization (state) cost $s_{\text{norm,cost}}$ as:

$$s_{\text{norm,cost}} = \frac{s_{\text{cost}}}{n_{\text{m}}}.$$

This cost gives insight into how far off the matched predictions of the multi-target tracker are on average. Given that the set W corresponds to the sequence of ground truth target states, the missed target cost m_{cost} can be defined as follows:

$$m_{\text{cost}} = \frac{c}{2}\left(|W| - n_{\text{m}}\right).$$

And similarly, the false prediction cost f_{cost}, given that Z corresponds to the sequence of predicted states, is defined as:

$$f_{\text{cost}} = \frac{c}{2}\left(|Z| - n_{\text{m}}\right).$$

3
Machine Learning

We begin in Section 3.1 by discussing elementary concepts in neural networks, such as loss functions, linear layers, activation functions, and feedforward neural networks. Following this, we discuss some normalization procedures and overfitting prevention techniques in Section 3.2.

In Section 3.3, concepts related to the transformer architecture are introduced, such as attention, embeddings, and the encoder-decoder structure.

Essential concepts such as binary classification metrics are discussed in Section 3.4, followed by an overview of clustering algorithms, specifically the Leiden algorithm, in Section 3.5. The chapter concludes with auxiliary theory presented in Section 3.6.

This chapter provides the theoretical foundation for integrating machine learning techniques into multi-object tracking (MOT) systems. The ideas presented here are essential for understanding the MT3v2 model described in Chapter 4 and the machine learning components of the proposed pipeline discussed in Chapter 5.

3.1 Neural Network Preliminaries

Neural networks are functions that take as input some form of data, often a matrix $\mathbf{X} \in \mathbb{R}^{n \times d}$, where d corresponds to the number of so-called input *features* and n to the number of inputs. Each input is represented as a row in $\mathbf{X}$. The neural network can be defined as:

$$f(\mathbf{X}; \theta) = \mathbf{Y},$$

here, $\mathbf{Y}$ is a matrix that can vary in the number of dimensions depending on the task at hand, and $\theta \in \Theta$ represents a specific set of *parameters* that define the network, where Θ denotes the entire space of possible parameter values.

The parameters of a network can take many different forms, but the most common is the so-called *linear layer*. A linear layer consists of a linear transformation matrix $\mathbf{W} \in \mathbb{R}^{d_{\text{in}} \times d_{\text{out}}}$ and often a bias vector $\mathbf{b} \in \mathbb{R}^{1 \times d_{\text{out}}}$. Here, d_{in} is the number of feature dimensions of the input $\mathbf{X}$ to the layer. The elements of the transformation matrix and the bias vector are parameters of the neural network, that is, $\mathbf{W}, \mathbf{b} \in \theta$.

We define a linear layer (affine layer, but we will refer to it as a linear layer), which contains a bias vector $\mathbf{b}$, that transforms some input matrix $\mathbf{X}$ to d_{out} feature dimensions as:

$$\mathrm{L}\left(\mathbf{X}\right)_{\mathbf{b}}^{d_{\text{out}}} = \mathbf{X}\mathbf{W} + \mathbf{b}. \tag{3.1}$$

And if it does not contain a bias vector, we define it as follows:

$$\mathrm{L}\left(\mathbf{X}\right)^{d_{\text{out}}} = \mathbf{X}\mathbf{W}. \tag{3.2}$$

In the context of neural networks, the *neuron* is often mentioned. The neuron can be viewed as a singular processing unit, and in relation to a linear layer, a neuron is defined by a singular column of $\mathbf{W}$ and the corresponding column element of $\mathbf{b}$. This means that a linear layer that transforms the input to the dimensions d_{out} has the same number of neurons, one for each output dimension.

Since linear layers are linear, neural networks need other transformations to be able to learn nonlinear tasks (characterized by the loss function). These transformations are known as *activation functions*. A popular choice of activation function is the ReLU [21] (rectified linear unit), which is defined as:

$$\mathrm{ReLU}\left(\mathbf{X}\right) = \max\left(0,\ \mathbf{X}\right), \tag{3.3}$$

here, the max operation is performed element-wise on $\mathbf{X}$. Another popular activation function is the Softplus activation function, which is a smooth approximation of the ReLU function [11]. Softplus can sometimes be advantageous when training neural networks as it is differentiable everywhere. It is defined as:

$$\mathrm{Softplus}\left(\mathbf{X}\right) = \log\left(1 + \exp\left(\mathbf{X}\right)\right), \tag{3.4}$$

here, just as for ReLU, both log and exp are performed element-wise on $\mathbf{X}$.

By combining linear layers and activation functions, we can create what is known as feedforward neural networks (FFNs). A feedforward neural network can consist of any number of linear layers in sequence, often with some form of activation function in-between the linear layers. For ease of notation, we assume that each layer has a bias vector $\mathbf{b}$ and uses the same activation function f. The last layer here is defined as not having an activation function. The dimensions of the linear transformation for each layer are defined by $d_1 \to \cdots \to d_{\text{n}}$ where n is the number of linear layers and d_{i} are the output dimensions of each layer. We thus define an FFN as follows:

$$\mathrm{FFN}_{\text{f}}^{d_1 \to \cdots \to d_{\text{n}}}\left(\mathbf{X}\right) = \mathrm{L}_{\mathbf{b}}^{d_{\text{n}}}\left(\text{f}\left(\mathrm{L}_{\mathbf{b}}^{d_{\text{n}-1}}\left(\text{f}\left(\ldots\right)\right)\right)\right), \tag{3.5}$$

here, ... refers to the recursiveness of the operations, as the same structure follows until the first linear layer.

The goal of training a neural network is to find the optimal parameters given a task that the neural network should solve. The task is defined by a *loss function*, which allows the learning of the parameters through an optimization procedure. In

supervised learning labeled data $\hat{\mathbf{Y}}$ is provided, which the neural network is supposed to learn to approximate.

To tie the previous concepts together, we now give an example. Consider an input $\mathbf{X} \in \mathbf{R}^{n \times 256}$ where each row represents a flattened representation of a black-and-white 16×16 pixel image (thus $d = 256$), which could represent either a blurry cat or a blurry dog. We have labeled data $\hat{\mathbf{Y}} \in \{0, 1\}^n$, where a 0 represents if the row corresponds to an image of a cat and 1 for an image of a dog. The task of distinguishing between two classes, typically labeled 0 and 1, is known as *binary classification*. We define our neural network as follows:

$$f(\mathbf{X}; \theta) = \text{Sigmoid}\left(\text{FFN}_{\text{ReLU}}^{528 \to 1}(\mathbf{X})\right),$$

here, sigmoid refers to the sigmoid function:

$$\text{Sigmoid}(\mathbf{x})_i = \frac{1}{1 + e^{-\mathbf{x}_i}} \in (0, 1).$$

Thus, the neural network output is $\mathbf{Y} \in (0, 1)^n$, which could be interpreted as the probability that the input is a picture of a dog. With this, we can construct a loss function:

$$\mathcal{L}_{\text{BCE}}\left(\mathbf{Y}, \hat{\mathbf{Y}}\right) = \frac{1}{n} \sum_{i=1}^{n} \hat{\mathbf{Y}}_i \log(\mathbf{Y}_i) - \left(1 - \hat{\mathbf{Y}}_i\right) \log(\mathbf{Y}_i), \tag{3.6}$$

here, predictions $\mathbf{Y}_i$ that are close to 1 when $\hat{\mathbf{Y}}_i$ is 0 (cat) and predictions $\mathbf{Y}_i$ close to 0 when the label is $\hat{\mathbf{Y}}_i$ 1 (dog) are penalized. This loss function is known as *binary cross-entropy* [5]. The lowest possible value of the loss function is 0, and the goal is to find the parameters that make the neural network approximate the labels and thus reduce the loss.

The most common way to optimize the parameters of the neural network, known as *training* the neural network, is through some form of gradient descent that minimizes the value of the loss function (3.6). With gradient descent, the parameters are updated by moving in the direction of the gradient where the loss function decreases the most.

3.2 Normalization Procedures and Overfitting Prevention

Since neural networks can contain many layers and transformations, the function $f(\mathbf{X}; \theta)$ they define is often very complex and consists of a large number of parameters. Because of this, training a neural network through gradient descent-based methods can often be difficult.

Normalization procedures have been observed to help in these scenarios. One reason for this is that they constrain the output of layers in the network, reducing the

risk of gradients becoming very large, which leads to unstable training. One of the most popular normalization procedures is known as *layer norm* and was introduced in [18]; it has been shown to produce more stable training dynamics and reduce training time. It is essentially a standardization procedure, with the addition of learnable parameters $\boldsymbol{\gamma}$, known as gain, and a bias vector $\mathbf{b}$. It is defined as follows:

$$\text{LayerNorm}\left(\mathbf{X}\right)_{i,j} = \frac{\mathbf{X}_{i,j} - \mu_i}{\sigma_i}\boldsymbol{\gamma}_j + \mathbf{b}_j, \tag{3.7}$$

here, μ_i is defined as follows, where d denotes the number of features in $\mathbf{X}$:

$$\mu_i = \frac{\sum_{j=1}^{d} \mathbf{X}_{i,j}}{n}.$$

And σ_i is defined as:

$$\sigma_i = \sqrt{\frac{1}{d}\sum_{j=1}^{d}\left(\mathbf{X}_{i,j} - \mu_i\right)^2}.$$

In addition to being difficult to train, neural networks, especially those with a large number of parameters, can exhibit what is known as *overfitting*. Overfitting refers to the phenomenon in which neural networks perform well, in terms of a loss function, on the data on which they have been trained, but perform significantly worse on other data. That is, the network does not generalize. Some reasons for this can be that the network has too many parameters, and instead of learning patterns in the data that can be extrapolated upon, it instead remembers the training data by encoding it in the parameters.

A technique known as dropout, introduced in the paper [16], aims to reduce overfitting in networks. It does this by randomly setting some of the outputs of certain layers to zero during training. If $\mathbf{X}^{\text{out}}$ is the output of some layer, dropout with probability p is defined as:

$$\text{Dropout}\left(\mathbf{X}^{\text{out}}\right)_{i,j} = \mathbf{X}^{\text{out}}_{i,j} o_{i,j}, \tag{3.8}$$

here, $o_{i,j}$ is a Bernoulli distributed random variable with probability p of being zero, and thus $1 - p$ of being one:

$$o_{i,j} \sim \text{Bernoulli}\left(1 - p\right).$$

By doing this, the network can no longer rely on specific parameters, and encoding the training data in the parameters thus becomes more difficult. In addition to managing problems with overfitting, dropout also forces the network to essentially solve the problem (minimize the loss function) in many different ways. The random element of shutting off and on neurons results in a new network during each training session (gradient update), and thus the network can be viewed in its entirety as a combination of many different subnetworks [16].

3.3 Attention and Transformers

In traditional feedforward neural networks, the different inputs, i.e., the rows of the input matrix $\mathbf{X}$, are processed independently and simultaneously without interaction between them. For an illustrative example, refer to the image classification task discussed in Section 3.1, where each row of $\mathbf{X}$ represents a flattened image of a cat or a dog. In this case, whether a specific row represents a dog or a cat is not dependent on any other row in the dataset.

In contrast, recurrent neural networks (RNNs) are designed to process inputs sequentially, where each input is influenced by the previous. Imagine now that $\mathbf{X}$ contains rows representing successive frames of a video featuring a dog. We now define the task as predicting the next frame of the dog, given all previous frames. In an RNN, each row (frame) is processed in relation to the previous frames, allowing the model to learn from the evolving sequence. This enables the network to understand the video as a continuous sequence rather than as a collection of independent frames. As a result, RNNs are well-suited for tasks that rely on temporal relationships, like forecasting in time-series data or filtering and state estimation from noisy observations (as introduced in Section 2.2), where understanding the dynamics of change over time is important. In connection with the previous Chapter 2, these networks are mostly suitable for single-object filtering, where the inputs represent measurements of a single evolving state, such as successive video frames of a single dog, rather than interlaced frames of multiple entities, like both a dog and a cat.

Although RNNs effectively capture sequential dependencies, they have inherent limitations when dealing with more complex scenarios involving multiple entities or long-range dependencies. This is where the concept of attention becomes highly useful. In 2017, the paper *Attention is All You Need* [27] introduced a new type of architecture known as the transformer, which popularized the use of attention layers. Unlike RNNs, attention layers allow interactions between all inputs simultaneously, without the constraints of sequential processing. This type of processing has become popular for tasks such as translation and text generation (Large Language Models, such as Chat GPT), where the dependencies within the data are often non-sequential. For example, understanding a sentence may require considering words that are far apart or referring to information introduced much earlier (such as character information), which RNNs would struggle to do efficiently.

Similarly, in Multi-Object Tracking, as introduced in Section 2.3, determining the number of objects and their respective positions becomes challenging when attempting to process detections sequentially. Transformers, by using attention, enable the network to consider all detections together, allowing it to establish relationships between multiple potential objects simultaneously. This makes attention a promising alternative for problems where complex interactions between inputs are required, and where processing inputs independently or strictly sequentially would, likely, be insufficient.

Since attention does not rely on sequential ordering, the notion of time or spatial position needs to be explicitly represented within the inputs. This is done through positional embeddings, which encode positional information in a way that allows the network to understand the relationships between inputs, irrespective of their order. The intuition behind positional embeddings, along with the general concept of embeddings, is explored further in Section 3.3.3. However, before delving into positional embeddings, we first introduce the computations that define the attention layer in Sections 3.3.1 and 3.3.2.

3.3.1 Attention Layer

Compared to the sequential nature of recurrent layers, attention allows each input to interact directly with every other input based on their similarities, providing a flexible flow of information. The mechanism achieves this by computing relationships through a query matrix $\mathbf{Q}$, a key matrix $\mathbf{K}$, and a value matrix $\mathbf{V}$. Imagine a scenario where we are trying to understand a story: the query might represent a word we are currently focused on (such as a character name), the keys are representations of all the words in the story (contextual information), and the values contain information tied to each word that we want to extract. By comparing the query with each key, attention determines how much information should flow from each value to help understand the current word (information relevant to the character). In this way, the attention mechanism can help the model focus on the most relevant parts of the input, even if those parts are far apart in the sequence.

Attention layers come in many different forms, but they all share some common principles. An attention layer takes as input a query matrix $\mathbf{Q} \in \mathbb{R}^{n_\mathrm{q} \times d}$, a key matrix $\mathbf{K} \in \mathbb{R}^{n_\mathrm{v} \times d}$, and a value matrix $\mathbf{V} \in \mathbb{R}^{n_\mathrm{v} \times d}$. Thus, the key and value matrix are of the same dimensions and the query matrix might differ. Given these matrices, it outputs an embedding matrix $\mathbf{E}^{\mathrm{out}} \in \mathbb{R}^{n_\mathrm{q} \times d}$, where each row is a convex combination of the rows of a linear transformed $\mathbf{V}$. We define the attention operation as:

$$\mathrm{Attention}\left(\mathbf{Q}, \mathbf{K}, \mathbf{V}\right) = \mathbf{E}^{\mathrm{out}}.$$

The most common form of attention is *Self-attention*. It is defined as when the query, key, and value matrices all originate from the same source ($n_\mathrm{q} = n_\mathrm{v}$). Another common application is *cross-attention*, which occurs when the key and value matrices originate from the same input, while the query matrix does not. These matrixes could, for example, represent the output of one or of separate feedforward neural networks.

To understand the steps of an attention layer:

1. The query, key, and value matrices are first transformed by a linear layer, where $d_1 = d$:

$$\tilde{\mathbf{Q}} = \mathrm{L}^{d_1}\left(\mathbf{Q}\right), \quad \tilde{\mathbf{K}} = \mathrm{L}^{d_1}\left(\mathbf{K}\right), \quad \tilde{\mathbf{V}} = \mathrm{L}^{d_1}\left(\mathbf{V}\right).$$

   This linear transformation of each matrix allows the layer to extract the most important features for the following similarity procedure and, in addition, the

most important features for the convex combination, given the task at hand.

2. Next, the attention score matrix $\mathbf{S} \in \mathbb{R}^{n_\mathrm{q} \times n_\mathrm{v}}$ is calculated by taking the dot product between each row of $\tilde{\mathbf{Q}}$ (queries) and each row of $\tilde{\mathbf{K}}$ (keys), scaled by the square root of the feature dimension d_1:

$$\mathbf{S}_{i,j} = \frac{\tilde{\mathbf{Q}}_i \cdot \tilde{\mathbf{K}}_j^\mathrm{T}}{\sqrt{d_1}}, \tag{3.9}$$

here, the score $\mathbf{S}_{i,j}$ indicates the similarity between the key $\tilde{\mathbf{K}}_j$ and the querie $\tilde{\mathbf{Q}}_i$.

3. The score matrix $\mathbf{S}$ is then normalized by the softmax function so that each row sums up to one, resulting in the attention weight matrix $\mathbf{A}$:

$$\mathbf{A}_{i,j} = \frac{e^{\mathbf{S}_{i,j}}}{\sum_{k=1}^{n} e^{\mathbf{S}_{i,k}}}. \tag{3.10}$$

4. The attention weights are used to compute each output row $\mathbf{E}_i^\mathrm{out}$ as a convex combination of the rows of the value matrix $\tilde{\mathbf{V}}$:

$$\mathbf{E}_i^\mathrm{out} = \sum_{k=1}^{n_\mathrm{v}} \mathbf{A}_{i,k} \tilde{\mathbf{V}}_k.$$

The output matrix $\mathbf{E}^\mathrm{out}$ is often referred to as an embedding, linking it to the concept of *embeddings*, which is discussed further in Section 3.3.3.

3.3.2 Multi-Headed Attention Layer

An extension of the attention layer introduced in Section 3.3.1 is known as multi-headed attention [27]. It consists of the same operations, with the inclusion of a final concatenation and linear transformation step.

Multi-headed attention can be interpreted as extracting and processing different contexts (particular information) present in the query, key, and value matrices. In terms of the story or text example, one head could extract information related to characters, while another could extract information about locations, objects, or events. The different contexts are then combined, and their relative importance can be learned.

In multiheaded attention, we instead produce h different $\tilde{\mathbf{Q}}$, $\tilde{\mathbf{K}}$, and $\tilde{\mathbf{V}}$ matrices from the incoming query $\mathbf{Q}$, key $\mathbf{K}$, and value $\mathbf{V}$ matrices. We define the layer as follows:

$$\mathrm{MHA}^\mathrm{h}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \mathbf{E}^\mathrm{out}. \tag{3.11}$$

Each of the h transformed query, key, and value matrices are now of dimensionality:

$$d_1 = \frac{d}{h}.$$

Instead of $d_1 = d$ as in the case of a single attention-head. This requires that the input dimension d be a multiple of the number of heads h.

Multiheaded attention then calculates the score $\mathbf{S}$, weight $\mathbf{A}$, and the output embedding matrix $\mathbf{E}^{\text{out}}$, for each head h. The output embeddings for each head h are then combined as follows:

$$\mathbf{E}^{\text{out}} = \sum_{i=1}^{h} \mathrm{L}^{d}\left(\mathbf{E}^{\text{out,i}}\right),$$

here, the linear layer is separate for each attention head's output.

3.3.3 Embeddings

In the context of transformers, embeddings refer to vector representations that capture various types of information, such as position, time, or specific attributes such as character information in a story. An embedding can be viewed as the result of transforming an input, often using a linear layer, to create a numerical representation that the model can effectively work with. Although embeddings are used in many types of neural networks, they have gained particular significance in the context of transformers and the attention mechanism.

A specific type of embedding that is important in transformers is the *positional embedding*. As mentioned in Section 3.3, since the attention mechanism does not inherently encode the order of the input elements (unlike RNNs, which process data sequentially), positional embeddings are used to incorporate this missing temporal or spatial information.

The way in which positional embeddings are generated has a direct impact on the attention mechanism's ability to effectively measure similarities between inputs. In the attention mechanism, similarity is typically measured using a dot product between query and key vectors. However, using raw position values (e.g. time or spatial coordinates) might not effectively reflect relative distances through dot products. For this reason, positional embeddings often transform raw positions into a representation that better captures the relationships between inputs, particularly when using dot product similarity.

To illustrate, consider the example of representing times in the continuous range $[0, 10]$ in such a way that the dot product similarity reflects the relative distances between times. We construct *sinusoidal embeddings* with the following frequencies:

$$\boldsymbol{\omega} = \left[0, 0.2475, 0.495, 0.7425, 0.99\right].$$

The resulting sinusoidal embeddings $\boldsymbol{\epsilon}^{i}$ for each input time $\mathbf{t}_i$ are given by:

$$\begin{aligned}
\boldsymbol{\epsilon}^{i}_{j} &= \sin\left(\frac{2\pi\boldsymbol{\omega}_j\mathbf{t}_i}{10}\right), \\
\boldsymbol{\epsilon}^{i}_{j+5} &= \cos\left(\frac{2\pi\boldsymbol{\omega}_j\mathbf{t}_i}{10}\right).
\end{aligned}$$

These sinusoidal embeddings provide a representation that encodes the relative positions of inputs, allowing the dot product similarity in the attention mechanism to reflect the temporal relationships effectively. To compare, the Euclidean distance between two times $\mathbf{t}_i$ and $\mathbf{t}_j$ is given by:

$$\text{Euclidean Distance}_{i,j} = \sqrt{(\mathbf{t}_i - \mathbf{t}_j)^2}.$$

The Euclidean distance can be transformed into a similarity metric as:

$$\text{Euclidean Similarity}_{i,j} = \frac{1}{1 + \text{Euclidean Distance}_{i,j}}.$$

In Figure 3.1, we illustrate the differences in similarity scores between Euclidean similarity of raw times, dot product similarity of raw times, and dot product similarity with the sinusoidal embeddings. The plot shows that the sinusoidal embeddings maintain high similarity for inputs that are close in time, which is essential for effective sequence modeling.

There are multiple ways to create positional embeddings, and in the original transformer architecture [27], sinusoidal embeddings were used. In addition to sinusoidal embeddings, it is common to learn positional embeddings through a linear layer, allowing the model to determine the optimal encoding. Another method is *Fourier feature encoding*, where the frequencies used in sinusoidal embeddings are made learnable, as described in Section 3.3.3.1. For discrete values, sometimes a learnable lookup table is employed, where a vector representation is learned for each discrete position.

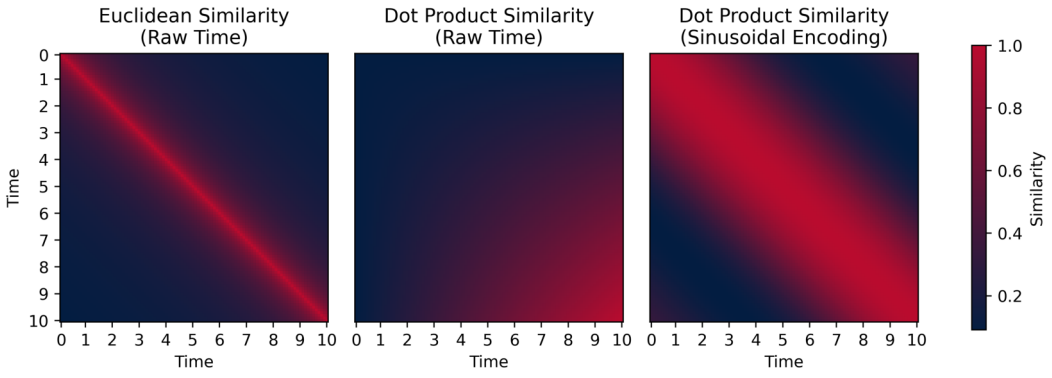


Figure 3.1: The similarities in each subplot have been scaled by their maximum value to achieve a maximum value of 1. The Euclidean similarity of raw time illustrates that points in time close to each other have high similarity. For the dot product similarity of raw time values, we no longer see high similarities for close times; instead, the magnitude of the time controls the similarity. In the rightmost plot, the effect of sinusoidal embeddings can be seen, as the highest similarities now occur when points are close in time.

3.3.3.1 Learnable Fourier Feature Embeddings

Learnable Fourier feature embeddings (encodings) extend the concept of sinusoidal embeddings (encodings) by including the frequencies $\boldsymbol{\omega}$ as learnable parameters. In [34] they showed that the learnable Fourier embeddings outperformed traditional methods on several benchmarks.

Learnable Fourier feature embeddings can be created for raw positions (or any type of features) $\mathbf{X} \in \mathbb{R}^{n \times d}$ in any dimension d, such as time in $d = 1$, or 3-d Cartesian positions (x, y, z) with $d = 3$. The number of features d_{out} in the resulting embeddings $\boldsymbol{\epsilon} \in \mathbb{R}^{n \times d_{\text{out}}}$ is a multiple of two, since the embeddings are constructed from the same number of cos and sin elements.

The frequencies are generated as follows:

$$\boldsymbol{\omega} = 2\pi \mathrm{L}^{d_{\text{out}}/2}(\mathbf{X}).$$

Cosine and sine elements are then constructed as:

$$\begin{aligned} \boldsymbol{\epsilon}_{i,j}^{\cos} &= \cos(\boldsymbol{\omega}_{i,j}), \\ \boldsymbol{\epsilon}_{i,j}^{\sin} &= \sin(\boldsymbol{\omega}_{i,j}). \end{aligned}$$

The cos and sin elements are then combined through another linear transformation, to the desired output dimension d_{out}:

$$\boldsymbol{\epsilon} = \mathrm{L}^{d_{\text{out}}}(\boldsymbol{\epsilon}^{\cos}) + \mathrm{L}^{d_{\text{out}}}\left(\boldsymbol{\epsilon}^{\sin}\right).$$

3.3.4 Transformer Encoder and Decoder

A transformer can be thought of as a two-part system: the *encoder* and the *decoder*. The *encoder* is responsible for understanding the context of the input: understanding how different parts of the input relate to one another, such as learning which parts of a story refer to the same character. The *decoder* then takes this contextual understanding and applies it to solve a specific task, using queries to retrieve relevant information from the encoder's output and generate an appropriate response.

The *encoder* captures relationships within the input data, which are represented in its output embeddings. It is composed of multiple layers, each containing two main parts: a multi-headed self-attention layer, which allows different parts of the input to interact with each other, and a feedforward neural network layer that further processes these interactions. To stabilize training and improve performance, layer normalization and residual connections are applied throughout [27] [22]. Positional embeddings are also added to the input to give a sense of order or position, which is important for capturing the structure of sequences.

The *decoder* takes the embeddings produced by the encoder along with some task-specific input. It also has multiple layers, each consisting of three main parts: a

self-attention layer to process the task-specific input, a cross-attention layer to incorporate information from the encoder, and a feedforward neural network. The cross-attention layer uses the task-specific input to "query" the encoder's output, combining what it has learned about the input data with the specific requirements of the task. The decoder also outputs embeddings, which can be transformed and utilized to produce outputs for the task at hand.

3.4 Binary Classification Metrics

For a binary classification task (mentioned in Section 3.1), it is often of interest not only to judge the neural network by the value of the loss function. This is often done through the construction of metrics that highlight whether the network is good or not, given what one wants the network to achieve. This could be necessary if the input data is highly skewed towards one of the classes- the loss might seem low, but that is because the model has learned to predict only the majority class.

Given predicted labels $\mathbf{Y}_{\text{pl}} \in \{0,1\}^{n\times 1}$ for n inputs (for example, predicted probabilities $\mathbf{Y}$ above 0.5 are considered class 1 predictions), there are several ways to summarize performance using different metrics [6] [9]. The most common one being accuracy:

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}}$$

where:

- **TP:** True Positives is the number of correctly labeled samples with true class 1.
- **TN:** True Negatives is the number of correctly labeled samples with true class 0.
- **FP:** False Positives is the number of falsely labeled samples with true class 0.
- **FN:** False Negatives is the number of falsely labeled samples with true class 1.

Sensitivity is another metric that quantifies how good the model is at identifying class 1 labels:

$$\text{Sensitivity} = \frac{\text{TP}}{\text{TP} + \text{FN}}.$$

The corresponding metric for class 0 is known as specificity:

$$\text{Specificity} = \frac{\text{TN}}{\text{TN} + \text{FP}}.$$

Precision is similar to sensitivity, but the denominator is instead the total number of predicted class 1 samples and not the true number, resulting in a measure of how precise the models' class 1 predictions are:

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}.$$

Balanced accuracy is then defined as the mean of sensitivity and specificity:

$$\text{Balanced accuracy} = \frac{\text{Sensitivity} + \text{Specificity}}{2},$$

which sometimes is preferred to accuracy because it weighs the separate class accuracy equally, which accuracy does not. Another combined metric is known as the F1 score, which is defined as the harmonic mean of precision and sensitivity:

$$\text{F1} = 2\frac{\text{Precision} \cdot \text{Sensitivity}}{\text{Precision} + \text{Sensitivity}},$$

which is useful as it combines the precision of the model's guesses of class 1 with the sensitivity, as either one on their own can be misleading.

3.5 Clustering and The Leiden Algorithm

Clustering algorithms partition data without the use of labels, with the goal of uncovering information from potential structures in the data [12, pp. 875–890]. In this realm, there is a subgroup of algorithms from the field of network science that are commonly termed community detection algorithms [30]. These algorithms are designed for networks, i.e. graphs with nodes and edges that represent connections between nodes. One of the most popular community detection algorithms is the Leiden algorithm [30].

The Leiden algorithm outputs a partition of the nodes, ranging from one subset (all nodes grouped together) to as many subsets as there are nodes (each node in its own subset). The algorithm iteratively partitions the nodes by moving them into different subsets, known as communities, based on the strength (edge weight) and the number of connections between the nodes, starting with each node in its own subset. The objective is to identify partitions where nodes within the same community are highly interconnected, resulting in strong and dense communities [30]. This objective can be defined with a number of different functions, one of the most popular is modularity [30]. Modularity [7] is a measure that is used to assess the strength of a partition of a network into communities or clusters, which is why it is common as an objective function in community detection algorithms [30].

The modularity of a partition ranges from -1 to 1, with higher values indicating a stronger community structure. A positive modularity indicates that there are more edges within communities than would be expected by chance, while a modularity close to zero suggests that the detected community structure is no better than a random partition. The Leiden algorithm can thus be viewed as a heuristic algorithm to maximize modularity for a partition of a graph.

However, even though modularity has been widely used in community detection, there is much debate about whether it is actually a good measure, both for judging a partition and when using it as an objective function, as discussed in [19].

3.5.1 Clustering Metrics

In addition to modularity, there are a number of ways to judge whether a predicted partition $\{\mathcal{P}_1, \mathcal{P}_2, \ldots, \mathcal{P}_k\}$ of a clustering algorithm is good or not. If a ground truth partition $\{\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_n\}$ is known, a comparison can be made with this true partition instead of leaning on measures such as modularity.

The adjusted Rand Index (ARI) [12, pp. 877–878] is one such score, which takes the value of the Rand Index and adjusts it for chance. The Rand Index is computed as:

$$\mathrm{RI} = \frac{\mathrm{TP} + \mathrm{TN}}{\binom{\sum_{i=1}^{k} |\mathcal{P}_i|}{2}},$$

where TP is the number of pairs of points that are in the same subset in both partitions and TN is the number of pairs that are in different subsets in both partitions. The binomial coefficient in the denominator represents the total number of pairs N. For ARI, one then calculates the *expected* RI by assuming the same number of subsets in each partition and assigning memberships at random to create a partition. This is modeled through a generalized hypergeometric distribution [12, pp. 877–878]. The adjusted rand index is between -1 and 1, where 1 represents equal partitions.
Another commonly used score is normalized mutual information (NMI) which is based on information theory and uses entropy for normalization. Just as the rand index, NMI is an adjusted-for-chance version denoted Adjusted Mutual Information (AMI) [10]. The adjusted score is calculated using assumptions similar to those for ARI. The adjusted mutual information score is between 0 and 1, where 1 represents equal partitions.
Furthermore, a simpler metric is defined below:

$$\mathrm{Q}_{\mathrm{pg}} = \frac{\sum_{i=1}^{n} \sum_{j=1}^{k} \mathbb{1}_{|\mathcal{C}_i \cap \mathcal{P}_j| = |\mathcal{C}_i| = |\mathcal{P}_j|}}{n},$$

here, Q_{pg} is the number of perfect subsets (subsets found in both partitions) divided by the total number of true subsets. If the fraction is high, then one can assume that the clustering is good; however, the inverse is not necessarily true. An example would be a predicted partition that for each subset misses one member when compared to the true partition subsets, with these missed members in the same subset. As such, it is not a very informative metric, but in the realm of data association in MTT, it can give valuable insight, since faulty or missed associations can substantially affect filtering performance in some cases.

Partitions can come in any shapes and sizes, that is, many small subsets, some large subsets, or a combination of the two. Some metrics are good for equal-sized subsets, others not. It is often necessary to employ a few different metrics to judge whether a predicted partition is "good" or not.

3.6 Auxilliary Theory

In this section, concepts used in the thesis with no clear connection to earlier topics are introduced.

3.6.1 Mahalanobis Distance

The Mahalanobis distance [33] is a generalized distance metric that is particularly useful for multivariate normal distributions. It extends the concept of standard deviation to higher dimensions.
The Mahalanobis distance D_M between a vector $\mathbf{x}$ and a mean vector $\boldsymbol{\mu}$, given a covariance matrix $\boldsymbol{\Sigma}$, is defined as:

$$D_M\left(\mathbf{x}|\boldsymbol{\mu}, \boldsymbol{\Sigma}\right) = \sqrt{\left(\mathbf{x}-\boldsymbol{\mu}\right)^T \boldsymbol{\Sigma}^{-1} \left(\mathbf{x}-\boldsymbol{\mu}\right)}, \tag{3.12}$$

here D_M^2 follows a chi-square distribution with k degrees of freedom:

$$D_M^2 \sim \chi_k^2. \tag{3.13}$$

This property allows the Mahalanobis distance to be translated into probabilities, enabling the creation of statistical tests based on this metric.

4

Multi-Target Tracking Transformer v2

The Multi-Target Tracking Transformer v2 (MT3v2) is a transformer-based model (see Section 3.3 for an introduction to transformers and the attention mechanism) designed for Multi-Object Tracking (MOT) with radar detections as inputs, developed by researchers at Chalmers University of Technology [37]. Operating within the Random Finite Set (RFS) framework, see Section 2.3.1 for a basic introduction to RFS, MT3v2 is trained to predict a Multi Bernoulli RFS.

In Section 4.1 we give an overview of the model and the different modules it consists of. Following this, we introduce the different modules, beginning with Section 4.1.1, the encoder pre-processing step. The encoder is then described in Section 4.1.2 which is followed by a decoder preprocessing module in Section 4.1.3. In Section 4.1.4 the decoder module is introduced, and finally, the matching and loss procedure is described in Section 4.1.5.

4.1 MT3v2 Overview

As described in the introduction to this chapter, the MT3v2 is a transformer-based model designed to process radar detections and produce outputs in the form of a Multi-Bernoulli RFS. For information on detections, scans, and other radar-related concepts, see Section 2.1. For concepts related to the transformer, such as attention, embeddings, encoder-decoder structure, refer to Section 3.3. For a basic introduction to random finite sets and the Multi-Bernoulli RFS, refer to Section 2.3.1. In the following, the different processing steps and the general notation of MT3v2 are described and summarized in Figures 4.2 and 4.6.

We refer to $\mathbf{D} = \left[\mathbf{r},\, \boldsymbol{\varphi},\, \dot{\mathbf{r}},\, \mathbf{t} \right] \in \mathbb{R}^{n_\mathrm{d} \times 4}$ as the matrix containing the measurements and scan times $\mathbf{t}$ of n_d detections from T scans. An illustration of the input detections $\mathbf{D}$ and what is being measured is given in Figure 4.1. The true underlying target state at scan time T (that is, what the radar is meant to measure) is denoted $\mathrm{O_T} = \{\mathbf{X}_\mathrm{T}^1, \ldots, \mathbf{X}_\mathrm{T}^{n_\mathrm{ts}}\}$. Here, n_ts denotes the number of underlying targets alive at scan time T. Each underlying target state $\mathbf{X}_\mathrm{T}^i$ is characterized by the 2-dimensional

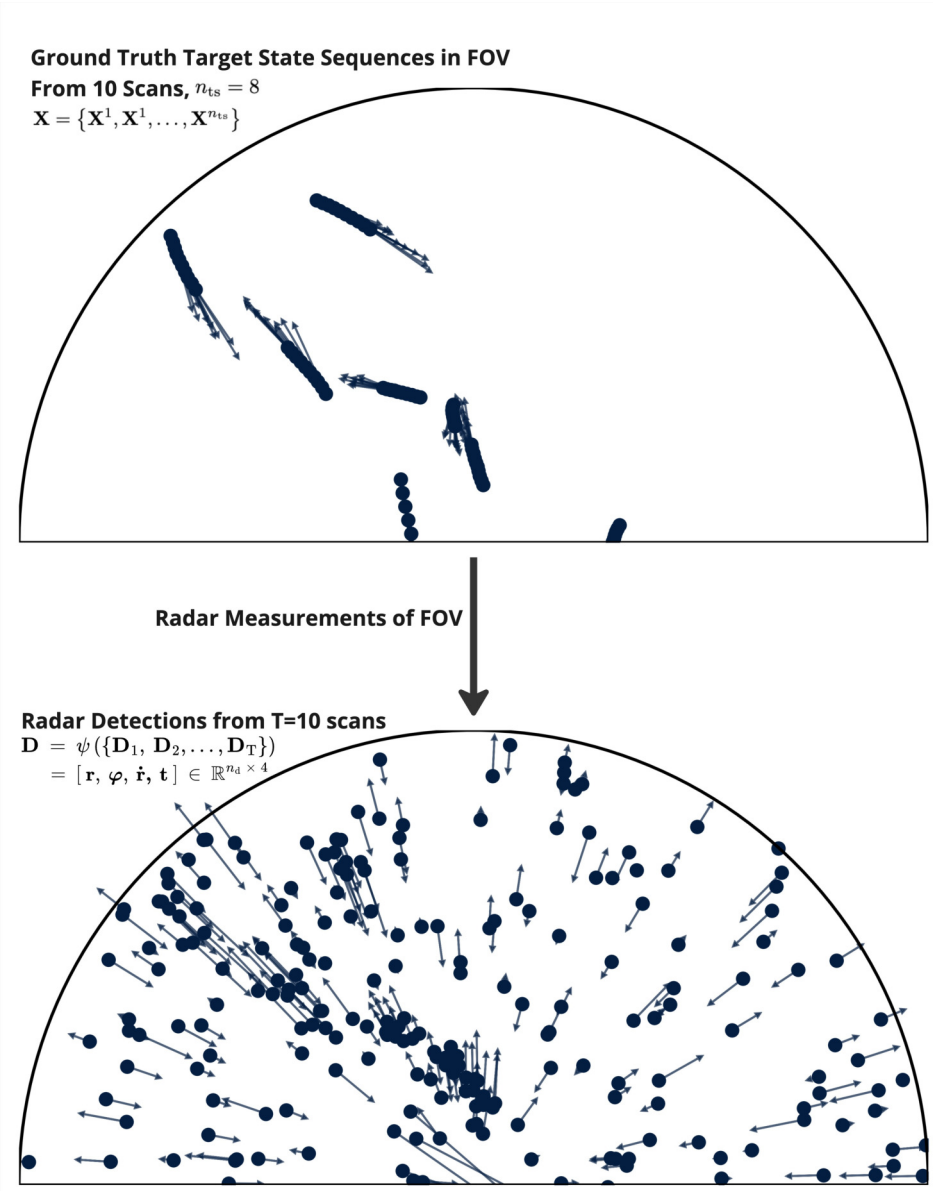


Figure 4.1: The upper part of the figure illustrates what the radar is trying to measure. The lower part of the figure illustrates what the radar actually sees in the form of detections and corresponding measurements. This is a completely synthetic scenario meant to illustrate what the input *could* look like.

Cartesian position and velocity, as mentioned in Section 2.1:

$$\mathbf{X}_{\mathrm{T}}^{i} = \begin{bmatrix} x_{\mathrm{T}}^{i} \\ y_{\mathrm{T}}^{i} \\ \dot{x}_{\mathrm{T}}^{i} \\ \dot{y}_{\mathrm{T}}^{i} \end{bmatrix}.$$

Given the detection matrix $\mathbf{D}$ the MT3v2 model produces predictions $\mathbf{P}_{\mathrm{out}}$ that are meant to model the underlying multi-target state O_{T} at the latest scan time T. The predictions $\mathbf{P}_{\mathrm{out}}$ characterize the density of a Multi-Bernoulli RFS with n_{p} components:

$$\mathrm{MT3v2}\left(\mathbf{D}\right) = \mathbf{P}_{\mathrm{out}} = \left\{\left(\mathbf{p}^{i},\, \mathbf{M}^{i},\, \mathbf{\Sigma}^{i}\right)\right\}_{i=1}^{n_{\mathrm{p}}},$$

here, the $\mathbf{p}^{i}$ is the existence probability of target/object i and the state density is a normal distribution characterized by the mean vector $\mathbf{M}^{i}$:

$$\mathbf{M}^{i} = \begin{bmatrix} \mu_{x^{i}} \\ \mu_{y^{i}} \\ \mu_{\dot{x}^{i}} \\ \mu_{\dot{y}^{i}} \end{bmatrix}.$$

And the diagonal covariance matrix $\mathbf{\Sigma}^{i}$:

$$\mathbf{\Sigma}^{i} = \mathrm{diag}(\sigma_{x^{i}}^{2}, \sigma_{y^{i}}^{2}, \sigma_{\dot{x}^{i}}^{2}, \sigma_{\dot{y}^{i}}^{2}).$$

To gain some intuition for what this MB RFS represents, refer to Figure 4.5.

Before the MT3v2 model can produce outputs as shown in Figure 4.5, the input detections $\mathbf{D}$ are processed by several modules, visualized in Figure 4.2. The different modules are briefly described below, beginning with the encoder preprocessing module.

Given the input detections $\mathbf{D}$, MT3v2 first performs a preprocessing step, called encoder preprocessing (EP), which transforms the radar detections into initial embeddings $\mathbf{E}_{\mathrm{enc},0}$ and time embeddings $\boldsymbol{\epsilon}_{\mathrm{enc}}$:

$$\mathrm{EP}\left(\mathbf{D}\right) = \left(\mathbf{E}_{\mathrm{enc},0}, \boldsymbol{\epsilon}_{\mathrm{enc}}\right).$$

These embeddings represent the radar detections in a form suited for processing by neural networks, and specifically attention layers, enabling the model to capture key features effectively. The encoder receives these embeddings and uses self-attention to capture relationships between different detections, generating processed output embeddings $\mathbf{E}_{\mathrm{enc,out}}$:

$$\mathrm{Encoder}\left(\mathbf{E}_{\mathrm{enc},0}, \boldsymbol{\epsilon}_{\mathrm{enc}}\right) = \mathbf{E}_{\mathrm{enc,out}}.$$

The self-attention mechanism allows the encoder to relate different detections to each other, learning spatial and temporal correlations. The contextual information

learned for each detection can then be used to deduce whether the detection is likely to be a false detection, or, if not, the group of detections it most likely belongs to (through similarity of embeddings), representing the same target.

The encoder output embeddings, $\mathbf{E}_{\text{enc,out}}$, are then passed along with the original detection matrix $\mathbf{D}$ to the decoder preprocessing module (DP). This module generates the encoder predictions $\mathbf{P}_{\text{enc}}$, initial decoder predictions $\mathbf{P}_{\text{dec,0}}$, initial decoder embeddings $\mathbf{E}_{\text{dec,0}}$, and positional embeddings $\boldsymbol{\epsilon}_{\text{dec}}$:

$$\text{DP}\left(\mathbf{E}_{\text{enc,out}}, \mathbf{D}\right) = \left(\mathbf{P}_{\text{enc}}, \mathbf{P}_{\text{dec,0}}, \mathbf{E}_{\text{dec,0}}, \boldsymbol{\epsilon}_{\text{dec}}\right).$$

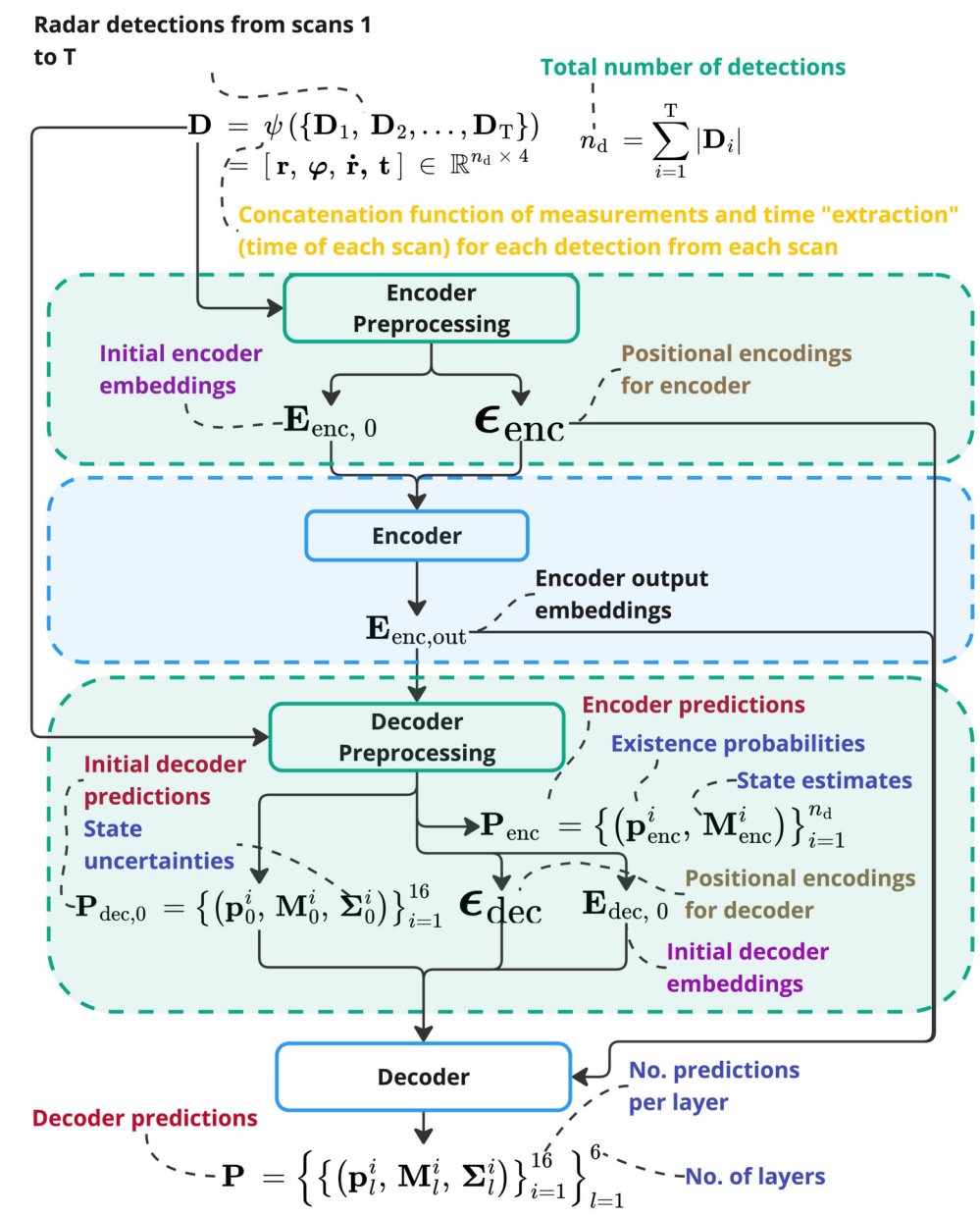


Figure 4.2: An overview of the MT3v2 model.

The initial decoder embeddings $\mathbf{E}_{\text{dec},0}$ represent potential targets and serve as a starting point for further refinement of the initial predictions by the decoder.

The final step is the decoder, which takes as input the initial decoder predictions, embeddings, and positional embeddings, along with the encoder output embeddings and time embeddings. The decoder produces the final multi-target predictions $\mathbf{P}$:

$$\text{Decoder}\left(\mathbf{P}_{\text{dec},0}, \mathbf{E}_{\text{dec},0}, \boldsymbol{\epsilon}_{\text{dec}}, \mathbf{E}_{\text{enc,out}}, \boldsymbol{\epsilon}_{\text{enc}}\right) = \mathbf{P}.$$

The decoder uses self-attention to refine its understanding of the potential targets and their relations, and cross-attention to query the encoder for contextual information that is relevant for each potential target. Each prediction $\mathbf{P}$ represents a Multi-Bernoulli (MB) RFS, which models the target state at a given scan time:

$$\mathbf{P}_l = \left\{ \left(\mathbf{p}_l^i,\, \mathbf{M}_l^i,\, \boldsymbol{\Sigma}_l^i \right) \right\}_{i=1}^{n_{\text{p}}},$$

here, $\mathbf{p}_l^i$ are the existence probabilities, $\mathbf{M}_l^i$ represents the mean state vector, and $\boldsymbol{\Sigma}_l^i$ represents the uncertainty. In the original article, $n_{\text{p}} = 16$, indicating the maximum number of targets MT3v2 can model.

In summary, MT3v2 leverages a transformer encoder-decoder architecture to model multi-target relationships in radar data. The encoder builds a context-rich representation of the detections, and the decoder uses that context to refine predictions about potential targets. The general flow is visualized in Figures 4.2. The loss functions used to train the model are described in Section 4.1.5, and an overview of the loss procedure can be seen in Figure 4.6.

4.1.1 Encoder Preprocessing

Given $\mathbf{D}$, the measurements, excluding scan times, are extracted:

$$\mathbf{Z} = \begin{bmatrix} \mathbf{r} & \dot{\mathbf{r}} & \boldsymbol{\varphi} \end{bmatrix} \in \mathbb{R}^{n_{\text{d}} \times 3}.$$

The range $\mathbf{r}$ is then normalized by the range of the radars field of view FOV, so that they fall within the range $[0, 1]$. A similar procedure also forces the elements of $\boldsymbol{\varphi}$ to the range $[0, 1]$. The range rate $\dot{\mathbf{r}}$ is normalized by a maximum value $\dot{r}_{\max}$ that defines the upper limit of the range rate values, so that its elements fall in the range $[0, 1]$. The normalization procedure is summarized as:

$$\phi\left(\mathbf{Z}\right) = \tilde{\mathbf{Z}}.$$

Next, the normalized measurements $\tilde{\mathbf{Z}}$ are projected to an embedding dimension of 256:

$$\mathbf{E}_{\text{enc},0} = \text{L}_{\mathbf{b}}^{256}\left(\tilde{\mathbf{Z}}\right).$$

The positional embeddings $\boldsymbol{\epsilon}_{\text{enc}}$ for the scan time $\mathbf{t}$ are generated through a learnable lookup table, as the scan times are discrete. For more information, see Section 3.3.3.

Finally, the input embeddings $\mathbf{E}_{\text{enc},0}$ and the time embeddings $\boldsymbol{\epsilon}_{\text{enc}}$ are passed to the encoder of MT3v2. [37]

4.1.2 The Encoder

The encoder part of MT3v2 takes as input the input embeddings $\mathbf{E}_{\text{enc},0}$ and time embeddings $\boldsymbol{\epsilon}_{\text{enc}}$ from the encoder preprocessing step. It is made up of $l_{\text{enc}} = 6$ layers. For information on the notation used in this section and the following, refer to Chapter 3. Below, the different sublayers of each layer are described; these sublayers are visually summarized in Figure 4.3.

Each layer i begins by processing the output embeddings from the previous layer with a multi-head attention layer, with $h = 8$ attention heads:

$$\tilde{\mathbf{E}}_1^i = \text{MHA}^8\left(\mathbf{E}_{\text{enc,i}-1} + \boldsymbol{\epsilon}_{\text{enc}}, \mathbf{E}_{\text{enc,i}-1} + \boldsymbol{\epsilon}_{\text{enc}}, \mathbf{E}_{\text{enc,i}-1}\right).$$

In this layer, encoder embeddings (detection representations) can learn their local contexts by sharing information with similar embeddings (detections). It can be viewed as a slightly modified multi-head self-attention layer, where the query and key matrices include positional embeddings, but the value matrix does not. This allows the positional embeddings to steer the attention weights, influencing how different detections are related, without being part of the final embeddings.

Following the attention layer is a layer-norm layer with a residual connection to $\mathbf{E}_{\text{enc,i}-1}$:

$$\tilde{\mathbf{E}}_2^i = \text{LayerNorm}\left(\tilde{\mathbf{E}}_1^i + \mathbf{E}_{\text{enc,i}-1}\right).$$

A feedforward network then processes $\tilde{\mathbf{E}}_2^i$ by first projecting the embeddings to 2048 dimensions, and then back to 256:

$$\tilde{\mathbf{E}}_3^i = \text{FFN}_{\text{ReLU}}^{2048\to256}\left(\tilde{\mathbf{E}}_2^i\right).$$

The last layer is once again layer-norm with a residual connection to $\tilde{\mathbf{E}}_2^i$, resulting in the output embeddings $\mathbf{E}_i$ for encoder layer i:

$$\mathbf{E}_{\text{enc,i}} = \text{LayerNorm}\left(\tilde{\mathbf{E}}_2^i + \tilde{\mathbf{E}}_3^i\right).$$

The last layer of the encoder, $i = 6$, produces the output embeddings of the encoder module, denoted $\mathbf{E}_{\text{enc,out}}$. These embeddings, along with the input detection matrix $\mathbf{D}$ are passed to the decoder preprocessing module.

The encoder output embeddings can be viewed as containing contextual information for each detection, as they have been compared and updated with information from one another based on similarities in the attention layers. This contextual information might be regarding how many other similar and nearby detections there are, or conversely, if there are not many similar detections close by. In the following section, these embeddings are used to create the initial decoder embeddings, and the initial state predictions for the decoder.

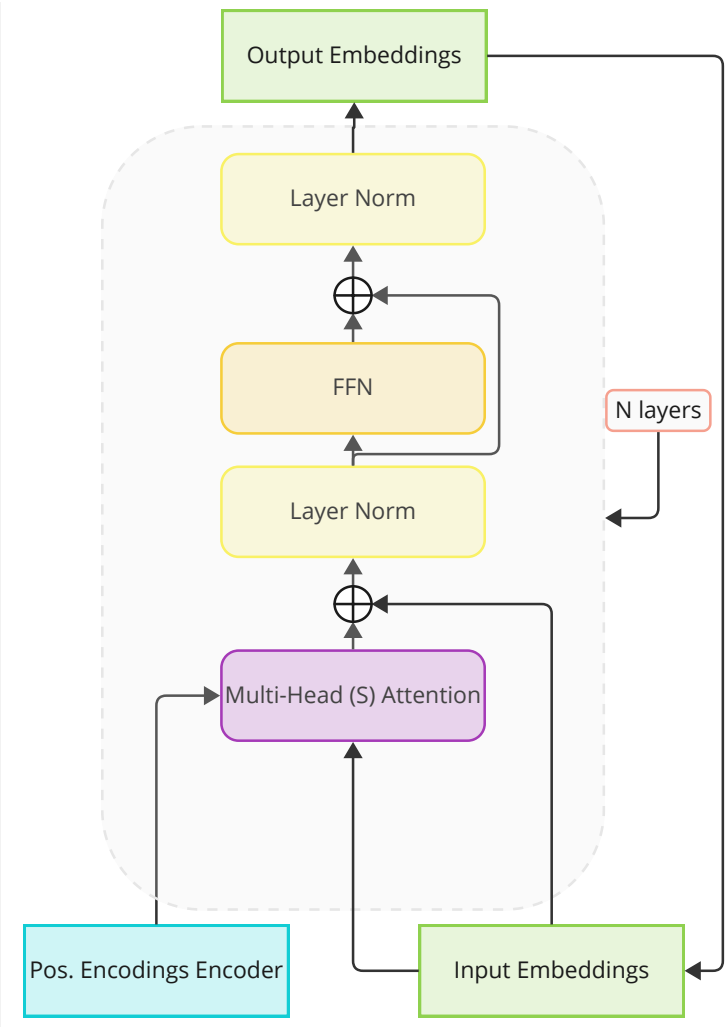


Figure 4.3: An overview of the encoder structure in the MT3v2 model. (S) stands for Self and (C) for cross. Addition operations are denoted by circles with a + symbol, combining inputs directed by the arrows. Positional encodings are the same as positional embeddings.

4.1.3 Decoder Preprocessing

After receiving encoder output embeddings $\mathbf{E}_{\text{enc,out}}$ and input detections $\mathbf{D}$, the decoder preprocessing module begins by transforming the embeddings $\mathbf{E}_{\text{enc,out}}$ as follows:

$$\tilde{\mathbf{E}}_{\text{enc,out}} = \text{LayerNorm}\left(\text{L}_{\mathbf{b}}^{256}\left(\mathbf{E}_{\text{enc,out}}\right)\right).$$

These transformed embeddings are then transformed once again to produce logits $\boldsymbol{\gamma}$, representing pre-sigmoid transformed probabilities (sigmoid here refers to the sigmoid function):

$$\boldsymbol{\gamma} = \text{L}_{\mathbf{b}}^{1}\left(\tilde{\mathbf{E}}_{\text{enc,out}}\right).$$

The top $n_{\text{p}} = 16$, according to the highest values, indices of the logits $\boldsymbol{\gamma}$ are then extracted, represented by the index set $\mathcal{I}$. The rows corresponding to the indices $\mathcal{I}$ of the embeddings $\tilde{\mathbf{E}}_{\text{enc,out}}^{\mathcal{I}}$ are then transformed to produce the initial decoder embeddings:

$$\mathbf{E}_{\text{dec},0} = \text{LayerNorm}\left(\text{L}_{\mathbf{b}}^{256}\left(\tilde{\mathbf{E}}_{\text{enc,out}}^{\mathcal{I}}\right)\right).$$

The decoder embeddings $\mathbf{E}_{\text{dec},0}$ can be viewed as representing the most likely potential targets in the radar field of view (FOV).

The positional encodings (embeddings) of the decoder $\boldsymbol{\epsilon}_{\text{dec}}$ are generated by the same transformation:

$$\boldsymbol{\epsilon}_{\text{dec}} = \text{LayerNorm}\left(\text{L}_{\mathbf{b}}^{256}\left(\tilde{\mathbf{E}}_{\text{enc,out}}^{\mathcal{I}}\right)\right).$$

Predictions

The encoder predictions $\mathbf{P}_{\text{enc}}$ are then generated. For this, the input detections $\mathbf{D}$ are first used to create the matrix $\tilde{\mathbf{M}}$, containing the measurements converted to Cartesian positions, with the columns representing Cartesian velocity zeroed:

$$\tilde{\mathbf{M}} = \begin{bmatrix}\mathbf{r}\cos(\boldsymbol{\varphi}) & \mathbf{r}\sin(\boldsymbol{\varphi}) & \mathbf{0} & \mathbf{0}\end{bmatrix} \in \mathbb{R}^{n_d \times 4}.$$

Following this, the matrix $\tilde{\mathbf{M}}$ is transformed by dividing all elements by $4 \cdot r_{\text{FOV}}$ and then adding 0.5, here r_{FOV} is the range (radius) of the FOV. This transformation forces all elements to be in the range $[0.25, 0.75]$, as the minimum possible value in $\tilde{\mathbf{M}}$ is $-r_{\text{FOV}}$ and the maximum is r_{FOV}. We denote this transformation ν:

$$\nu\left(\mathbf{X}\right) = \frac{\mathbf{X}}{4 \cdot r_{\text{FOV}}} + 0.5. \tag{4.1}$$

The inverse operation is similarly defined as:

$$\nu^{-1}\left(\mathbf{X}\right) = 4 \cdot r_{\text{FOV}}\left(\mathbf{X} - 0.5\right). \tag{4.2}$$

Consequently, the matrix elements are brought into a presigmoid space by the logit function, resulting in $\hat{\mathbf{M}}$:

$$\hat{\mathbf{M}} = \ln\left(\frac{\nu\left(\tilde{\mathbf{M}}\right)}{1-\nu\left(\tilde{\mathbf{M}}\right)}\right).$$

Using $\hat{\mathbf{M}}$, the encoder state predictions are produced by first calculating an adjustment matrix $\boldsymbol{\Delta}$:

$$\boldsymbol{\Delta} = \mathrm{FFN}_{\mathrm{ReLU}}^{128\to128\to4}\left(\tilde{\mathbf{E}}_{\mathrm{enc,out}}\right).$$

The encoder state predictions are then calculated by combining $\hat{\mathbf{M}}$ and $\boldsymbol{\Delta}$, transforming back to a sigmoid space, and then applying ν^{-1}:

$$\mathbf{M}_{\mathrm{enc}} = \nu^{-1}\left(\frac{1}{1+\exp\left(-\boldsymbol{\Delta}-\hat{\mathbf{M}}\right)}\right).$$

Following, the corresponding existence probabilities $\mathbf{p}_{\mathrm{enc}}$ are produced by transforming the logits $\boldsymbol{\gamma}$ to probabilities through the sigmoid function:

$$\mathbf{p}_{\mathrm{enc}} = \frac{1}{1+\exp\left(\boldsymbol{\gamma}\right)}.$$

Resulting in encoder predictions $\mathbf{P}_{\mathrm{enc}}$:

$$\mathbf{P}_{\mathrm{enc}} = \left\{\left(\mathbf{p}_{\mathrm{enc}}^{i},\, \mathbf{M}_{\mathrm{enc}}^{i}\right)\right\}_{i=1}^{n_{\mathrm{d}}}.$$

The initial predictions for the decoder $\mathbf{P}_{\mathrm{dec},0}$ consist of the top $n_{\mathrm{p}} = 16$ indices $\mathcal{I}$ of the encoder state predictions $\mathbf{M}_{\mathrm{enc}}$, as the existence probabilities and covariance matrices are generated without connection to the prior layer guesses in the decoder:

$$\mathbf{P}_{\mathrm{dec},0} = \left\{\left(\mathbf{0},\, \mathbf{M}_{\mathrm{enc}}^{i},\, \mathbf{0}\right)\right\}_{i\in\mathcal{I}}.$$

4.1.4 The Decoder

The decoder takes as input the decoder embeddings $\mathbf{E}_{\mathrm{dec},0}$ and positional embeddings $\boldsymbol{\epsilon}_{\mathrm{dec}}$, along with the initial decoder predictions $\mathbf{P}_{\mathrm{dec},0}$, the encoder output embeddings $\mathbf{E}_{\mathrm{enc,out}}$ and encoder time embeddings $\boldsymbol{\epsilon}_{\mathrm{enc}}$. Given this input, it produces predictions $\mathbf{P}$.

The MT3v2 decoder has $l_{\mathrm{dec}} = 6$ layers, the same as the encoder. It differentiates itself from the encoder by having an extra sub-layer consisting of cross-attention with the encoder output embeddings $\mathbf{E}_{\mathrm{enc,out}}$. In the following, the different sublayers of each layer are described. In Figure 4.4 these sublayers are visualized.

Each layer i begins by processing the embeddings from the previous layer with a multi-head attention layer, with $h = 8$ attention heads:

$$\tilde{\mathbf{E}}_{\mathrm{dec},1}^{i} = \mathrm{MHA}^{8}\left(\mathbf{E}_{\mathrm{dec,i-1}} + \boldsymbol{\epsilon}_{\mathrm{dec}}, \mathbf{E}_{\mathrm{dec,i-1}} + \boldsymbol{\epsilon}_{\mathrm{dec}}, \mathbf{E}_{\mathrm{dec,i-1}}\right).$$

In this layer, the decoder embeddings interact through self-attention, effectively sharing information between potential targets within the radar field of view. This

allows each embedding to refine its understanding by considering the characteristics of other possible targets, resulting in a more accurate representation of the multi-target environment.

Following this attention layer, a layer-norm layer with a residual connection to $\mathbf{E}_{\text{dec,i}-1}$ is employed:

$$\tilde{\mathbf{E}}^i_{\text{dec},2} = \text{LayerNorm}\left(\tilde{\mathbf{E}}^i_{\text{dec},1} + \mathbf{E}_{\text{dec,i}-1}\right).$$

Now, the embeddings $\tilde{\mathbf{E}}^i_{\text{dec},2}$ serve as the query matrix in multi-headed cross-attention, $h = 8$, with the encoder embedding outputs $\mathbf{E}_{\text{enc,out}}$ as keys and value matrix:

$$\tilde{\mathbf{E}}^i_{\text{dec},3} = \text{MHA}^8\left(\tilde{\mathbf{E}}^i_{\text{dec},2} + \boldsymbol{\epsilon}_{\text{dec}}, \mathbf{E}_{\text{enc,out}} + \boldsymbol{\epsilon}_{\text{enc}}, \mathbf{E}_{\text{enc,out}}\right).$$

This cross-attention step can be intuitively understood as follows. The decoder embeddings represent the most probable targets in the radar field of view (FOV). By acting as queries with the encoder output embeddings, which represent contextual information learned about the detections, the decoder embeddings can learn to query information that aids in representing the state and related uncertainties for the supposed target.

Following the cross-attention layer is another layer-norm layer with a residual connection to $\tilde{\mathbf{E}}^i_{\text{dec},2}$:

$$\tilde{\mathbf{E}}^i_{\text{dec},4} = \text{LayerNorm}\left(\tilde{\mathbf{E}}^i_{\text{dec},2} + \tilde{\mathbf{E}}^i_{\text{dec},3}\right).$$

A feedforward network then processes $\tilde{\mathbf{E}}^i_{\text{dec},4}$ by first projecting the embeddings to 2048 dimensions, and then back to 256:

$$\tilde{\mathbf{E}}^i_{\text{dec},5} = \text{FFN}^{2048\to256}_{\text{ReLU}}\left(\tilde{\mathbf{E}}^i_{\text{dec},4}\right).$$

Just as for the encoder layers, the last sub-layer of the decoder layer is layer-norm with a residual connection to $\tilde{\mathbf{E}}^i_{\text{dec},4}$, resulting in the output embeddings $\mathbf{E}_{\text{dec},i}$ for the decoder layer i:

$$\mathbf{E}_{\text{dec},i} = \text{LayerNorm}\left(\tilde{\mathbf{E}}^i_{\text{dec},5} + \tilde{\mathbf{E}}^i_{\text{dec},4}\right).$$

Decoder Predictions

At the end of every decoder layer, predictions $\mathbf{P}_i$ for the current layer i are produced:

$$\mathbf{P}_i = \left\{\left(\mathbf{p}^j_i,\, \mathbf{M}^j_i,\, \boldsymbol{\Sigma}^j_i\right)\right\}_{j=1}^{n_{\text{p}}}.$$

The existence probabilities $\mathbf{p}_i$ are generated by first producing logits $\boldsymbol{\gamma}_i$ by passing the decoder embeddings $\mathbf{E}_{\text{dec},i}$ through a linear layer:

$$\boldsymbol{\gamma}^i = \text{L}^1_{\mathbf{b}}\left(\mathbf{E}_{\text{dec},i}\right).$$

And applying the sigmoid function:

$$\mathbf{p}_i = \frac{1}{1 + \exp\left(\boldsymbol{\gamma}^i\right)}.$$

The state mean vectors $\mathbf{M}_i^j$ are updated using a strategy called iterative refinement, where the previous predictions $\mathbf{M}_{i-1}^j$ are updated (refined) by adding an adjustment vector $\boldsymbol{\Delta}_i^j$. The matrix $\boldsymbol{\Delta}_i$ consisting of the adjustment vectors is calculated as follows:

$$\boldsymbol{\Delta}_i = \mathrm{FFN}_{\mathrm{ReLU}}^{128 \to 128 \to 4}\left(\mathbf{E}_{\mathrm{dec},i}\right).$$

The adjustments are made in a presigmoid space, and because of this, the previous layer predictions $\mathbf{M}_{i-1}^j$ first need to be transformed by the ν function (4.1) and then logit transformed:

$$\hat{\mathbf{M}}_{i-1} = \ln\left(\frac{\nu\left(\mathbf{M}_{i-1}\right)}{1 - \nu\left(\mathbf{M}_{i-1}\right)}\right).$$

The decoder state predictions $\mathbf{M}_i^j = \left[\mu_{x_i^j}, \mu_{y_i^j}, \mu_{\dot{x}_i^j}, \mu_{\dot{y}_i^j}\right]^{\mathrm{T}}$ are then calculated by combining $\hat{\mathbf{M}}_{i-1}$ and $\boldsymbol{\Delta}$, transforming back to a sigmoid space, and then applying the inverse function ν^{-1} (4.2):

$$\mathbf{M}_i = \nu^{-1}\left(\frac{1}{1 + \exp\left(-\boldsymbol{\Delta}_i - \hat{\mathbf{M}}_{i-1}\right)}\right).$$

What remains to be predicted are now the state uncertainties, which are represented by a diagonal covariance matrix $\boldsymbol{\Sigma}_i^j = \mathrm{diag}(\sigma_{x_i^j}^2, \sigma_{y_i^j}^2, \sigma_{\dot{x}_i^j}^2, \sigma_{\dot{y}_i^j}^2)$. The predicted elements of $\boldsymbol{\Sigma}_i^j$ represent the standard deviations, not the variances. These are calculated in a similar fashion to the other components, with a feedforward neural network; however, the output is passed through a softplus activation function (3.4) to ensure that all elements are non-negative (since standard deviations can not be negative):

$$\boldsymbol{\Sigma}_i = \mathrm{Softplus}\left(\mathrm{FFN}_{\mathrm{ReLU}}^{128 \to 128 \to 4}\left(\mathbf{E}_{\mathrm{dec},i}\right)\right).$$

The predictions for the decoder layer i can now be summarized as a Multi-Bernoulli RFS (for more information see Section 2.3.1) modeling the existence and potential non-existence together with the states and corresponding uncertainties of up to $n_{\mathrm{p}} = 16$ targets:

$$\mathbf{P}_i = \left\{\left(\mathbf{p}_i^j,\, \mathbf{M}_i^j,\, \boldsymbol{\Sigma}_i^j\right)\right\}_{j=1}^{n_{\mathrm{p}}}.$$

The predictions of the final layer are the only predictions that are used after the model has been trained. The predictions for the other layers, including the decoder predictions, are only used for training purposes. The loss functions and ground-truth matching procedure used to train the model are outlined in Section 4.1.5.

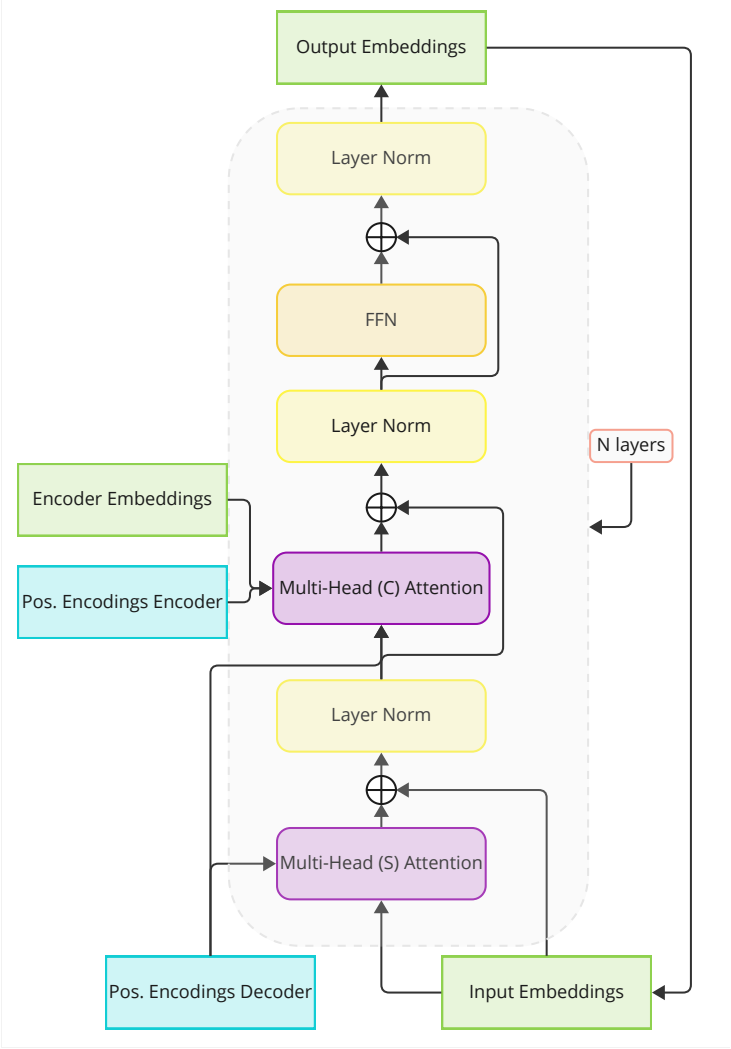


Figure 4.4: An overview of the decoder structure in the MT3v2 model. (S) stands for Self and (C) for cross. Addition operations are denoted by circles with a + symbol, combining inputs directed by the arrows. Positional encodings are the same as positional embeddings.

Visualizing the Predictions

The predictions $\mathbf{P}_i$ of the decoder can be, somewhat, intuitively understood through visualizations. Given the example input seen in Figure 4.1, the existence probabilities and the positional part of the state distribution (the first two mean elements and standard deviations) can be visualized by overlaying each component over the radar's Field of View. This is done by evaluating points on the field of view on the positional part of the state distribution of each component and scaling the density by the existence probability. The densities of each component are then summed together. This is visualized in the upper part of Figure 4.5 with the example input, where $n_\mathrm{p} = 10$.

Furthermore, we can also visualize the complete state distribution of each component by overlaying each component distribution over the field of view. In the middle part of Figure 4.5 this can be seen, where the blue gradient ellipses represent the positional uncertainties and the gold ellipses the velocity uncertainties. Components with an existence probability above some threshold τ_p are colored red, while those below are colored blue.

The lower figure in Figure 4.5 illustrates the ground-truth target states. All figures referred to in this section, including 4.1, are purely demonstrative and do not represent actual input or predictions of the model. These figures are only meant to help the reader gain some intuitive understanding of the model and the problem it is designed to solve.

4.1.5 Matching Process and Loss

The MT3v2 model predicts Multi-Bernoulli RFS $\mathbf{P}_l$ for each decoder layer l, and existence and state estimates for the encoder $\mathbf{P}_\mathrm{enc}$. The underlying ground-truth state sequence at scan time T is characterized by $\mathrm{O_T} = \{\mathbf{X}_\mathrm{T}^1, \ldots, \mathbf{X}_\mathrm{T}^{n_\mathrm{ts}}\}$. To produce a loss for MT3v2, a mapping between these two sets, the prediction components, and the ground truth target states $\mathbf{X}_\mathrm{T}^i$ must be created. This mapping essentially labels the predictions with ground-truth states.

To construct this mapping, the authors of the article [37] construct a function which is minimized to find the optimal mapping. The function emphasizes that predictions and ground-truth states that are close are likely connected. However, if the predicted existence probability is low, the model does not "think" that there is a target there - thus some other, not as spatially close prediction might be the more correct mapping.

We define the mapping function σ as being part of the set of all permutations of the sequence $(1, \ldots, n_\mathrm{p})$, such that $\sigma(i)$ corresponds to the i'th element of the sequence. The component-wise loss function is defined as

$$\mathcal{L}_\mathrm{m}\left(\mathrm{O_T}, \sigma(i),\, \mathbf{p}_l^i,\, \mathbf{M}_l^i\right) = \begin{cases} 0 & \text{if } \sigma(i) > n_\mathrm{ts} \\ \left\| \mathbf{M}_l^i - \mathbf{X}_\mathrm{T}^{\sigma(i)} \right\|_1 - \log\left(\mathbf{p}_l^i\right) & \text{otherwise} \end{cases}.$$

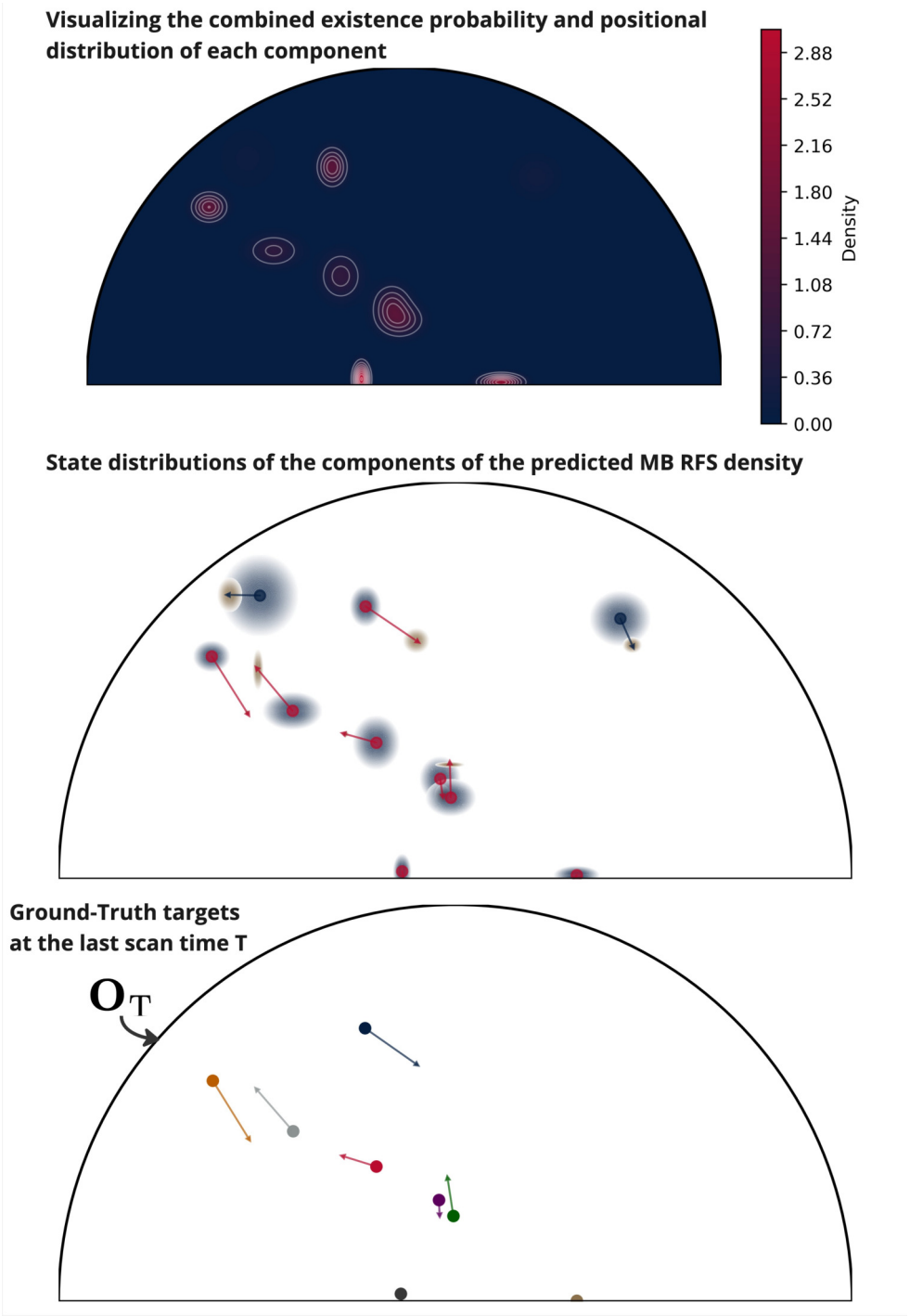


Figure 4.5: The upper part of the figure visualizes the combined existence probability and positional state densities of all components summed over the radar field of view. The middle figure visualizes the entire state distribution of each component, where the components with a existence probability below some threshold τ_p are colored blue and those above red. Blue ellipses are positional uncertainties and gold are for the velocity. The lower figure shows the ground truth target states O_T at scan time T. This is a completely synthetic scenario meant to illustrate what the output *could* look like, given the synthetic input from Figure 4.1.

Here $\left\|\mathbf{M}_l^i - \mathbf{X}_{\mathrm{T}}^{\sigma(i)}\right\|_1$ is the Manhattan distance:

$$\left\|\mathbf{M}_l^i - \mathbf{X}_{\mathrm{T}}^{\sigma(i)}\right\|_1 = \sum_{j=1}^{4} |\mathbf{M}_l^{i,j} - \mathbf{X}_{\mathrm{T}}^{\sigma(i),j}|,$$

here j refers to the j'th element of the vectors, and l is either enc for the encoder, or any integer between one and six for the decoder layers.

The optimization procedure to find the optimal mappings $\tilde{\sigma}^l$ (where the l also refers to the encoder predictions) can now be defined as follows:

$$\tilde{\sigma}^l = \arg\min_{\sigma} \sum_{i=1}^{k} \mathcal{L}_{\mathrm{m}} \left(\mathrm{O}_{\mathrm{T}}, \sigma(i), \mathbf{p}_l^i, \mathbf{M}_l^i\right).$$

Given these mappings, or matchings, $\tilde{\sigma}^l$, we can now construct the loss functions of MT3v2.

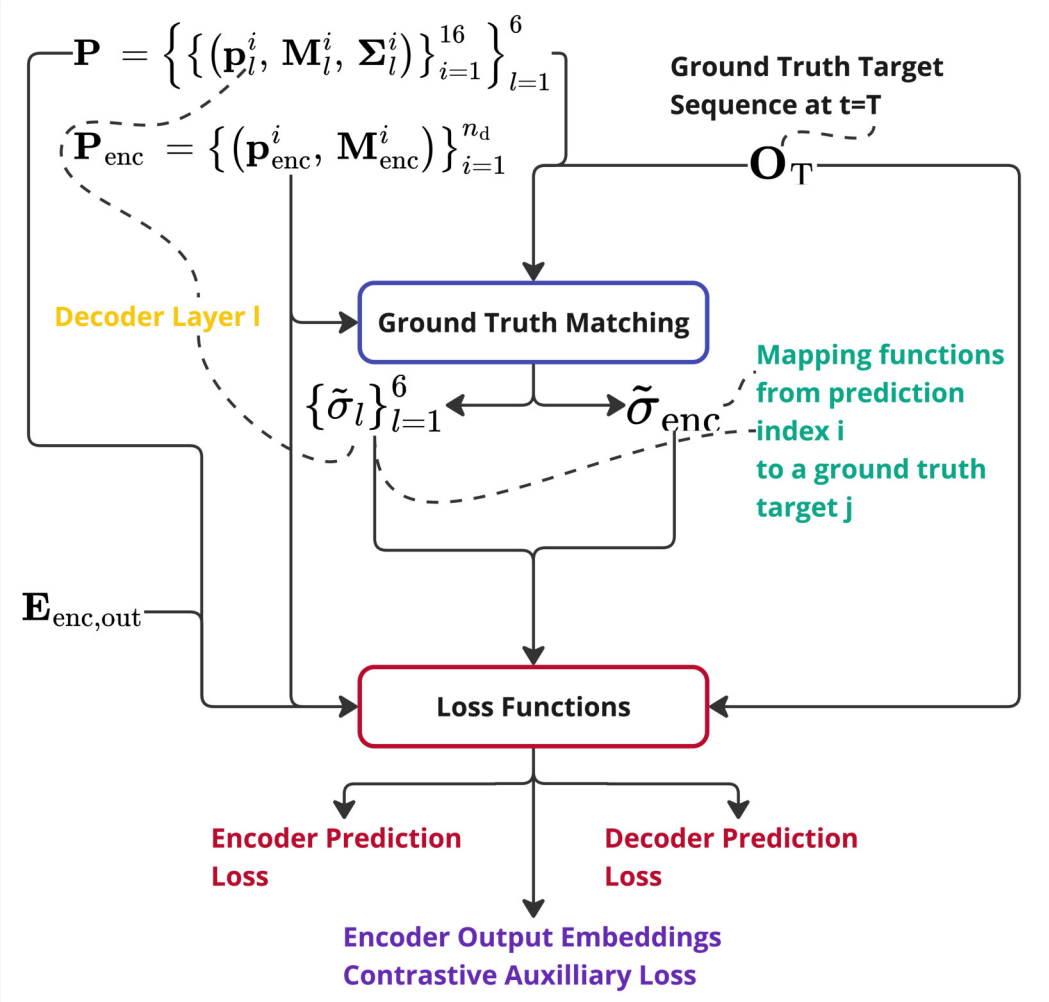


Figure 4.6: An overview of the MT3v2 models ground-truth matching procedure and loss functions.

4.1.5.1 Loss Functions and Training

The training of MT3v2 is based on a combination of loss functions to optimize the performance of the model across its various layers, in combination with dropout (3.8), which is described in Section 3.2.

The loss function for the decoder predictions is a simplified version of the Multi-Bernoulli Negative Log-Likelihood (NLL) loss. It is approximated by calculating the loss from only the optimal matching function $\tilde{\sigma}^l$ instead of all possible permutations. Given the optimal assignments $\tilde{\sigma}^l$ for each layer l, the decoder prediction loss $\mathcal{L}_{\text{dec}}$ is computed as follows:

$$\mathcal{L}_{\text{dec}}\left(\mathrm{O}_{\mathrm{T}}, \tilde{\sigma}, \mathbf{P}\right) = -\frac{1}{l_{\text{dec}} \cdot n_{\mathrm{p}}} \sum_{l=1}^{l_{\text{dec}}} \sum_{i=1}^{n_{\mathrm{p}}} f_{\text{dec}}\left(\mathrm{O}_{\mathrm{T}}, \tilde{\sigma}^l(i), \mathbf{P}_l\right),$$

here, the function f_{dec} is defined as:

$$f_{\text{dec}}\left(\mathrm{O}_{\mathrm{T}}, \tilde{\sigma}^l(i), \mathbf{P}_l\right) = \begin{cases} \alpha \log\left(1 - \mathbf{p}_l^i\right) & \text{if } \tilde{\sigma}^l(i) > n_{\text{ts}} \\ \alpha \log\left(\mathbf{p}_l^i\right) + \beta \log\left(\mathcal{N}\left(\mathbf{X}_{\mathrm{T}}^{\tilde{\sigma}^l(i)} \mid \mathbf{M}_l^i, \mathbf{\Sigma}_l^i\right)\right) & \text{otherwise .} \end{cases}$$

With $\mathcal{N}\left(\mathbf{X}_{\mathrm{T}}^{\sigma(i)} \mid \mathbf{M}_l^i, \mathbf{\Sigma}_l^i\right)$ representing the multivariate normal pdf defined by $\mathbf{M}_l^i$ and $\mathbf{\Sigma}_l^i$ evaluated at $\mathbf{X}_{\mathrm{T}}^{\sigma(i)}$. The loss penalizes false predictions, that is, no match with a ground truth target, with $\alpha \log\left(1 - \mathbf{p}_l^i\right)$, where α is a parameter set to 1 in the case of MT3v2. The higher the predicted probability, the larger the resulting loss. When a prediction has been matched with a ground-truth target, the loss instead incentivizes the predicted probability to be large. Furthermore, the state distribution component $\beta \log\left(\mathcal{N}\left(\mathbf{X}_{\mathrm{T}}^{\sigma(i)} \mid \mathbf{M}_l^i, \mathbf{\Sigma}_l^i\right)\right)$ forces the model to take uncertainty regarding the underlying state into account. The parameter β is also set to 1 for MT3v2.

For the encoder predictions $\mathbf{P}_{\text{enc}}$, a similar loss is employed that utilizes a different state loss component:

$$\mathcal{L}_{\text{enc}}\left(\mathrm{O}_{\mathrm{T}}, \tilde{\sigma}^{\text{enc}}, \mathbf{P}_{\text{enc}}\right) = -\frac{1}{n_{\mathrm{d}}} \sum_{i=1}^{n_{\mathrm{d}}} f_{\text{enc}}\left(\mathrm{O}_{\mathrm{T}}, \tilde{\sigma}^{\text{enc}}(i), \mathbf{P}_{\text{enc}}^i\right),$$

here, the function f_{enc} is defined as follows:

$$f_{\text{enc}}\left(\mathrm{O}_{\mathrm{T}}, \tilde{\sigma}^{\text{enc}}(i), \mathbf{P}_{\text{enc}}^i\right) = \begin{cases} \alpha \log\left(1 - \mathbf{p}_{\text{enc}}^i\right) & \text{if } \tilde{\sigma}^l(i) > n_{\text{ts}} \\ \alpha \log\left(\mathbf{p}_{\text{enc}}^i\right) + \beta \frac{1}{4} \sum_{j=1}^{4} |\mathbf{M}_{\text{enc}}^{i,j} - \mathbf{X}_{\mathrm{T}}^{\tilde{\sigma}^l(i),j}| & \text{otherwise} \end{cases},$$

here, we can see that the state loss is defined as the Manhattan distance for the encoder predictions. This loss incentivizes similar behavior as the decoder loss; however, uncertainties are no longer taken into account. The same parameter values $\alpha = 1$ and $\beta = 1$ are used for the encoder loss.

Finally, a contrastive learning loss is used to accelerate training and improve model

performance, as shown in various ablation studies [37]. Contrastive here refers to the fact that embeddings of detections that originate from the same target are incentivized to be similar; otherwise, they are incentivized to be different. The embeddings refer to the encoder output embeddings $\mathbf{E}_{\text{enc,out}}$. These are first processed through a linear layer followed by normalization to unit vectors to generate $\mathbf{C}$:

$$\tilde{\mathbf{C}} = \mathrm{L}_{\mathbf{b}}^{256}\left(\mathbf{E}_{\text{enc,out}}\right),$$
$$\mathbf{C}_{i,j} = \frac{\tilde{\mathbf{C}}_{i,j}}{\sqrt{\sum_{k=1}^{256} \tilde{\mathbf{C}}_{i,k}^2}}.$$

We also define the sets $\mathcal{K}_i$ as containing all detection indices that originate from the same target as the detection represented by the index i, excluding i itself. All false detections are regarded to originate from the same target. We denote $\mathcal{W}_i$ as the set of all detection indices, except for index i. Using $\mathbf{C}$, $\mathcal{W}_i$, and $\mathcal{K}_i$ we can now construct the contrastive loss as follows:

$$\mathcal{L}_{\text{c}}\left(\mathbf{C}, \mathcal{W}, \mathcal{K}\right) = \beta_{\text{c}} \sum_{i=1}^{n_{\text{d}}} \frac{-1}{|\mathcal{K}_i|} \sum_{k \in \mathcal{K}_i} \log \frac{e^{\mathbf{C}_i^\top \mathbf{C}_k}}{\sum_{j \in \mathcal{W}_i} e^{\mathbf{C}_i^\top \mathbf{C}_j}},$$

here, β_{c} is a parameter set to 4 in the original paper [37].

The final loss is constructed as the sum of $\mathcal{L}_{\text{enc}}$, $\mathcal{L}_{\text{dec}}$, and $\mathcal{L}_{\text{c}}$. For more details on the loss functions, see the original paper [37].

5

Large-Scale Target Tracking Pipeline

This chapter delves into the specific challenges and methodologies used in this thesis to handle large-scale target tracking scenarios.

Section 5.1 outlines the problem, highlighting the need for methods that address memory constraints, scalability, and the general complexity of the problem. Section 5.2 explores a variety of strategies to overcome these challenges, such as sparse attention mechanisms and a divide-and-conquer approach. The main focus of this chapter is the divide-and-conquer approach, which is outlined in Section 5.3.

5.1 Problem Description

As introduced earlier, this thesis focuses on target tracking scenarios involving a radar field of view (FOV) spanning tens of kilometers, where each scan or timestep generates approximately 10^4 detections. Of these detections, roughly half are expected to be false. The total number of tracked objects is also on the order of 10^4. Over the course of ten scans, this results in approximately 10^5 detections.

Given the scale of this scenario and the architecture of the MT3v2 model, several challenges and potential complexities arise that need to be addressed:

- **Memory Constraints:** Processing 10^5 detections with 8 attention heads and using 32-bit floating-point precision would require approximately 320 GB of memory, which far exceeds the capacity of current GPUs.
- **Large FOV:** The detections are distributed across a FOV that is more than 2000 times larger than what MT3v2 was designed for. This could be a problem for MT3v2's data association ability and thus filtering performance, as much larger embedding dimensions might be needed to accurately represent the enlarged FOV.
- **Complex Tracking Requirements:** The model must be able to track up to 10^4 objects throughout the FOV among around 10^5 detections, significantly increasing complexity. Similarily to above, much larger embedding dimensions might be needed to handle this complexity.

5.2 Exploration of Methods and Algorithms

Throughout the course of this thesis, a variety of approaches were explored to address the challenges posed by large-scale target tracking.

The initial focus was on developing methods that utilized sparse attention to improve memory efficiency and reduce time complexity. The idea was to capitalize on the spatial distribution of detections; given the vast FOV, detections separated by large distances are unlikely to be correlated. Sparse attention was used to exploit this by limiting the attention mechanism to keys within a reasonable proximity to each query. To support this, fast partitioning and contexting algorithms were developed, allowing queries to represent specific partitions while contexts of keys often overlapped.

Although the sparse attention methods showed promise in terms of reducing computational complexity and memory usage, the overall performance of the model was not satisfactory. The reasons for the underperformance were not fully investigated. However, one reason may have been that the number of embedding dimensions was not enough. Instead of scaling up the number of embedding dimensions and potentially facing other complexity issues, an approach focused on complexity reduction was adopted.

As a result, the strategy was amended to simplify the problem complexity before attempting multi-target filtering. As a first step, an algorithm was developed to group spatially close detections into subsets (contexts). This would ensure the management of large volume data in the subsequent steps. These subsets passed through a false detection filter to reduce the number of irrelevant (false) detections. Following this, a partitioning algorithm divided the remaining detections into independent sub-problems. These sub-problems would then be processed in parallel by a modified version of the MT3v2 model (MT3LS). This approach is detailed in the Section 5.3, starting with an overview of the proposed pipeline in Section 5.3.1.

5.3 Multi-Target Tracking Pipeline

In this section, the proposed large-scale multi-target tracker pipeline is first given an overview in Section 5.3.1. In the following sections, the different components of the pipeline are described in detail. The generation of features used throughout the pipeline is detailed in 5.3.2. The Local Context Generation module is described in Section 5.3.3 and in the following Section 5.3.4 the False Detection Filter is detailed. The partitioning module along with the four different implementations is then described in Section 5.3.5. Finally, the last step of the pipeline, a model named Multi Target Tracking Transformer Large Scale (MT3LS), based on the MT3v2 model (MT3v2 is described in detail in Chapter 4), is outlined in Section 5.3.6.

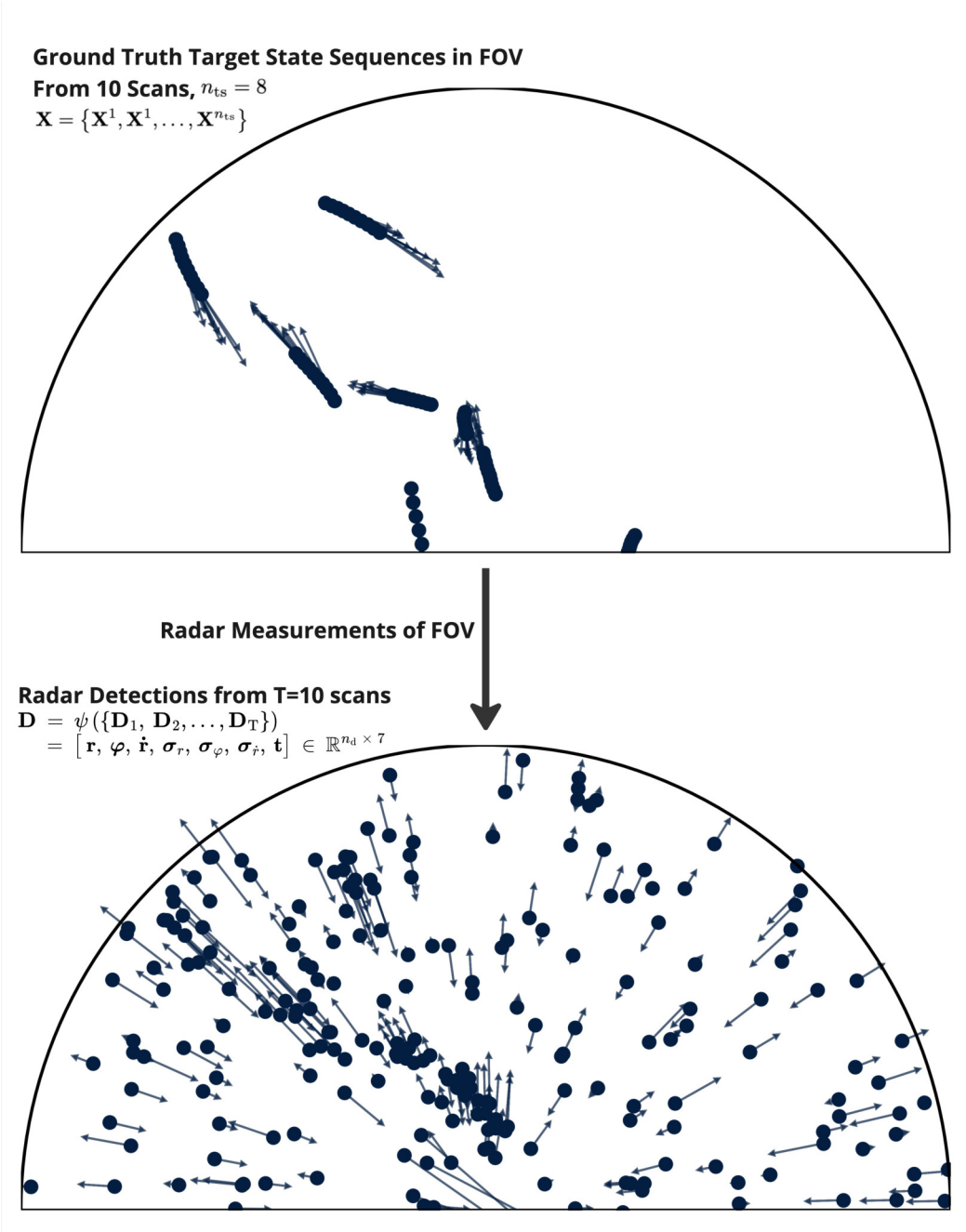


Figure 5.1: Underlying ground truth in the upper part, and corresponding input detections in the lower part, as part of an example scenario to illustrate the pipeline. Compare to the input to MT3v2 in Figure 4.1. This is a completely synthetic scenario meant to illustrate what the input *could* look like.

5.3.1 Pipeline Overview

In Chapter 4 an example scenario is shown to illustrate the potential input and output of the model with Figures 4.1 and 4.5. The same scenario will be used in this chapter, with slightly different information in the input detection matrix $\mathbf{D}$, as seen in Figure 5.1. The same scenario will also be used to illustrate some of the different pipeline modules, starting with the Local Context Generation Module in Figure 5.3.

In Figure 5.2 the information flow between the different modules of the Multiple Target Tracking pipeline outlined below can be seen.

The pipeline receives as input a matrix $\mathbf{D} \in \mathbb{R}^{n_\mathrm{d} \times m}$ of n_d radar detections with $m = 7$ features, generated from T radar scans (see Figure 2.1 for context of radar scans). These features include range r, range rate $\dot{r}$, angle φ, their corresponding uncertainties $\sigma_r, \sigma_{\dot{r}}, \sigma_\varphi$, and finally scan times. This input is illustrated with an example scenario in Figure 5.1. For more information on radar theory and the meaning of scan times, false detections, and missed detections in the context of this thesis, refer to Section 2.1.

The Feature Generation Module takes as input $\mathbf{D}$ and generates a new feature matrix $\mathbf{F} \in \mathbb{R}^{n_\mathrm{d} \times n_\mathrm{f}}$, with $n_\mathrm{f} = 16$ features:

$$\mathbf{F} = \mathrm{FeatureGeneration}(\mathbf{D}).$$

Consequently, the Local Context Generation (LCG) takes as input the feature matrix $\mathbf{F}$ and generates n_g (n_g is data-dependent and therefore cannot be determined a priori) different subsets $\mathcal{G} = \{\mathcal{G}_1, \mathcal{G}_2, \ldots, \mathcal{G}_{n_\mathrm{g}}\}$ (contexts) of $\mathcal{D} = \{1, 2, \ldots, n_\mathrm{d}\}$:

$$\mathcal{G} = \mathrm{LCG}(\mathbf{F}),$$

where:

$$\begin{aligned} &\mathcal{G}_i \subseteq \mathcal{D}, \\ &\bigcup_{i=1}^{n_g} \mathcal{G}_i = \mathcal{D} \quad \text{(the union covers all detections)}, \\ &|\mathcal{G}_i \cap \mathcal{G}_j| \geq 0 \quad \text{for } i \neq j \quad \text{(subsets can overlap)}. \end{aligned}$$

The False Detection Filter (FDF) takes as input the contexts $\mathcal{G}$ and $\mathbf{F}$ and generates the vectors $\mathbf{p}_\mathrm{td} \in \mathbb{R}^{n_\mathrm{d}}$ (true detection probabilities) and $\boldsymbol{\zeta}_\mathrm{td} \in \{0, 1\}^{n_\mathrm{d}}$ (true detection decisions, determined by thresholding $\mathbf{p}_\mathrm{td}$):

$$(\boldsymbol{\zeta}_\mathrm{td}, \mathbf{p}_\mathrm{td}) = \mathrm{FDF}\left(\mathcal{G}, \mathbf{F}\right).$$

The Partitioning Module takes as input the vector $\boldsymbol{\zeta}_\mathrm{td}$ from the FDF, $\mathcal{G}$ from the LCG, and $\mathbf{F}$ from the Feature Generation Module. It outputs a partition $\mathcal{P} = \{\mathcal{P}_1, \mathcal{P}_2, \ldots, \mathcal{P}_k\}$ (the number of subsets k is data-dependent and therefore cannot be determined a priori) of a subset $\tilde{\mathcal{D}} \subseteq \mathcal{D}$ of the original detections:

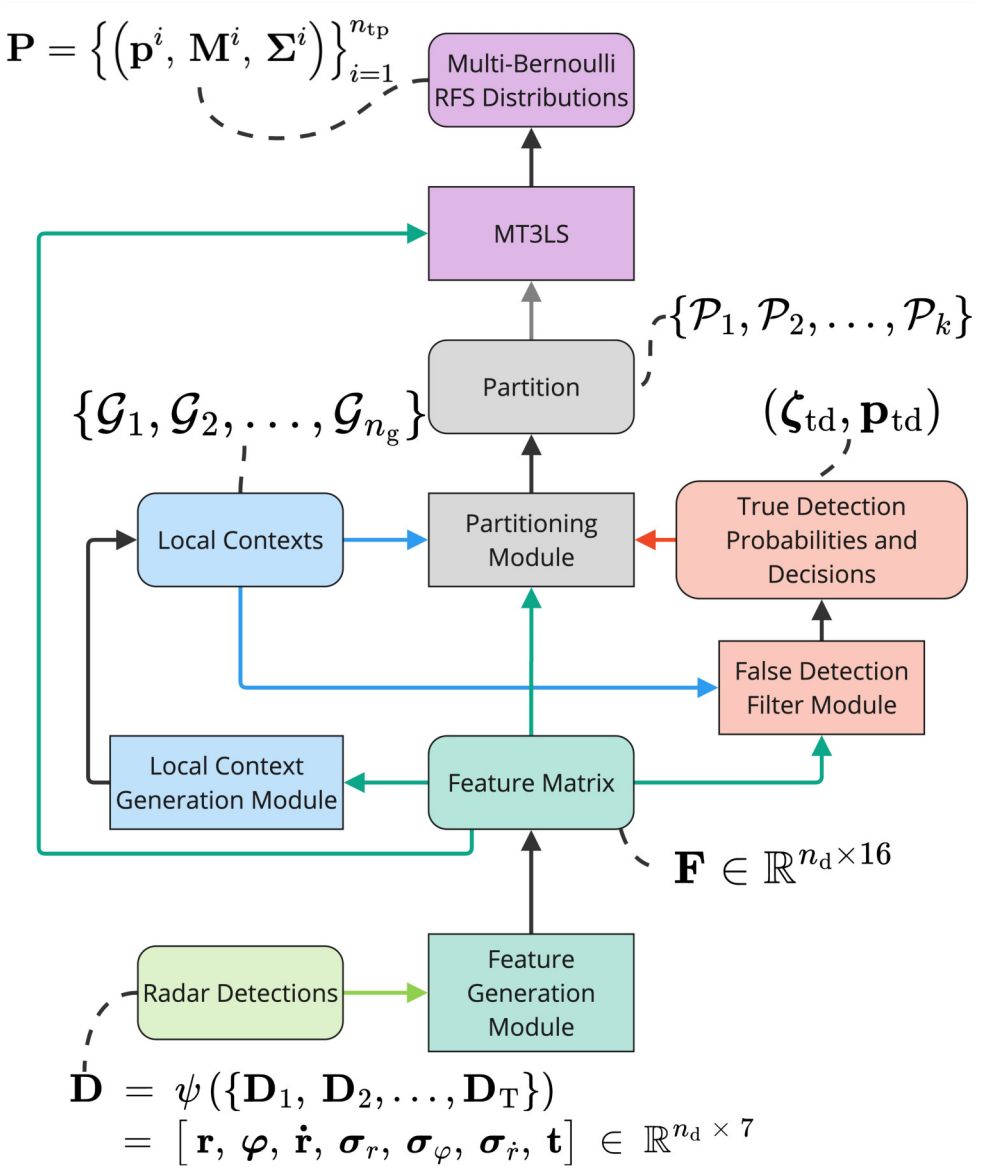


Figure 5.2: This figure illustrates how the different modules of the proposed multi-target tracker interact with each other on a high level. The different modules are rectangular with straight angles and color-coded. The output of the different modules are rectangular with smooth edges. The arrows from the inputs (smooth edges) to the different modules are colored the same color as the module that the input originates from.

$$\mathcal{P} = \text{Partitioning}\left(\boldsymbol{\zeta}_{\text{td}}, \mathcal{G}, \mathbf{F}\right),$$

where:

$$\begin{aligned} \mathcal{P}_i \neq \emptyset, \quad & i \in \{1, 2, \ldots, k\} \quad \text{(no subset is empty)}, \\ \mathcal{P}_i \cap \mathcal{P}_j = \emptyset, \quad & i \neq j, \quad i, j \in \{1, 2, \ldots, k\} \quad \text{(disjoint subsets)}, \\ \bigcup_{i=1}^{k} \mathcal{P}_i = \tilde{\mathcal{D}}. & \end{aligned}$$

The MT3LS Module, which is a modified version of MT3v2 (see Chapter 4 for information on MT3v2), takes as input the partition $\mathcal{P}$ from the partitioning module and $\mathbf{F}$ from the Feature Generation Module. It outputs a Multi-Bernoulli RFS filtration distribution $\mathbf{P}$:

$$\mathbf{P} = \text{MT3LS}\left(\mathcal{P}, \mathbf{F}\right) = \left\{\left(\mathbf{p}^i,\, \mathbf{M}^i,\, \boldsymbol{\Sigma}^i\right)\right\}_{i=1}^{n_{\text{tp}}},$$

here, n_{tp} is data-dependent and therefore cannot be determined a priori.

5.3.2 Feature Generation Module

The Feature Generation Module takes as input n_{d} radar measurements (detections) together with the radar uncertainties and scan times for each detection (see Sections 2.1 for more information). The input feature matrix is thus of the form:

$$\begin{aligned} \mathbf{D} &= [\mathbf{r}, \dot{\mathbf{r}}, \boldsymbol{\varphi}, \boldsymbol{\sigma}_r, \boldsymbol{\sigma}_{\dot{r}}, \boldsymbol{\sigma}_{\varphi}, \mathbf{t}] \\ &\in \mathbb{R}^{n_{\text{d}} \times 7}. \end{aligned}$$

In Figure 5.1 an example scenario is illustrated with input $\mathbf{D}$ together with the ground truth target sequences $\mathbf{X}$ that are being measured.

Using the input features, the corresponding unbiased converted Cartesian measurements, their standard deviations, and correlation are appended (see Section 2.1.2). In addition to this, the features $x_{\text{uc,p}}$ and $y_{\text{uc,p}}$ are appended. They represent unbiased converted Cartesian measurements after predicting the range r to the same point in time t_c for all detections. This is done by calculating the predicted range as follows:

$$r_p = r + \dot{r}\left(t_c - t\right),$$

keeping φ and $\dot{r}$ constant, as we have no knowledge of the potentially underlying target yet. Cartesian velocities $\dot{x}$, $\dot{y}$ are included by approximating them with the magnitude of range rate $\dot{r}$ in their corresponding directions:

$$\dot{x} = \dot{r}\cos(\varphi)\,,$$
$$\dot{y} = \dot{r}\sin(\varphi)\,.$$

The resulting feature matrix is thus characterized by:

$$\begin{aligned}\mathbf{F} = [&\mathbf{r}, \dot{\mathbf{r}}, \boldsymbol{\varphi}, \boldsymbol{\sigma}_r, \boldsymbol{\sigma}_{\dot{r}}, \boldsymbol{\sigma}_{\varphi}, \mathbf{t},\\ &\mathbf{x}_{\mathrm{uc}}, \mathbf{y}_{\mathrm{uc}}, \boldsymbol{\sigma}_{x_{\mathrm{uc}}}, \boldsymbol{\sigma}_{y_{\mathrm{uc}}}, \boldsymbol{\rho}_{x_{\mathrm{uc}}y_{\mathrm{uc}}},\\ &\dot{\mathbf{x}}, \dot{\mathbf{y}}, \mathbf{x}_{\mathrm{uc,p}}, \mathbf{y}_{\mathrm{uc,p}}]\\ &\in \mathbb{R}^{n_{\mathrm{d}}\times 16}.\end{aligned}$$

5.3.3 Local Context Generation

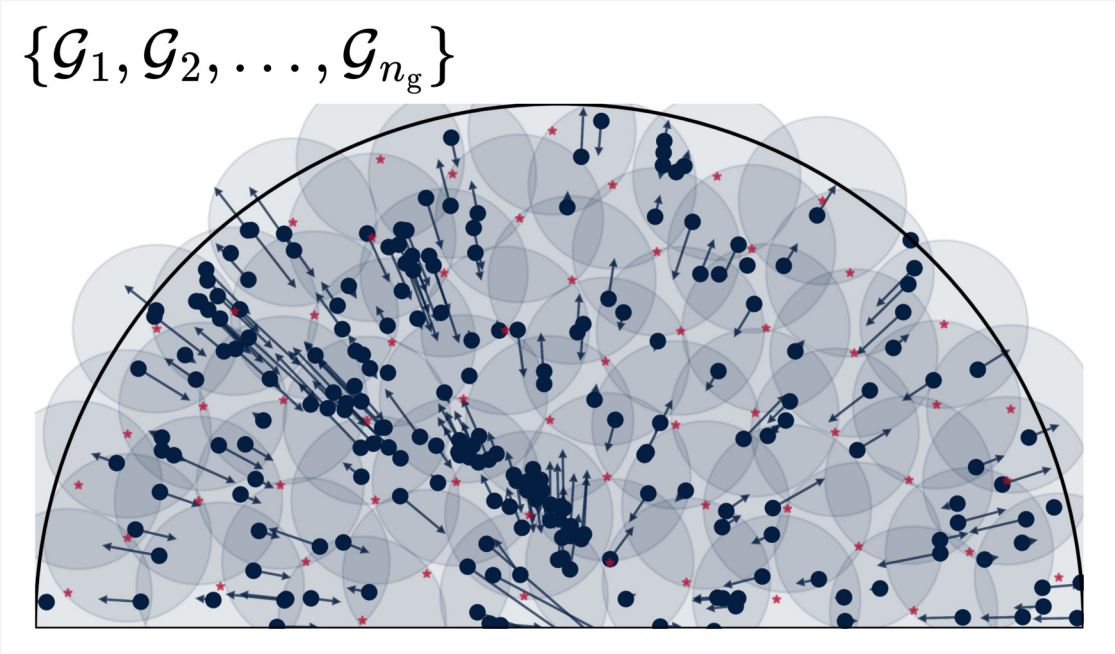


Figure 5.3: Local Context Generation Module is here illustrated in the example scenario by overlaying the local contexts that are generated over the input detections, which can be seen in Figure 5.1. Each red star represents the center of a local context, and the see-through blue disc represent the local context. Here the overlapping property of the local contexts is visualized, as multiple local contexts can contain the same detections. This is a completely synthetic scenario meant to illustrate what the local contexts *could* look like, given the synthetic input from Figure 5.1.

The Local Context Generation Module is an algorithm designed to alleviate the memory complexity issues related to subsequent attention and transformer blocks in the proposed MOT pipeline. To achieve this, the algorithm leverages the inherent correlation of detections that are spatially close to create subsets (local contexts) of the original n_{d} detections.

These subsets are generated by placing anchor locations throughout the radar FOV, inspired by the compact distribution of sunflower seeds (see Appendix 1 A.2). How many of these anchor locations that are created is described in Section 5.3.3.1.

The anchor locations represent centers of discs $\left(x_{\text{al}}^k, y_{\text{al}}^k\right)$ with a radius r_{anchors} across the radar field of view, an example of these discs can be seen in Figure 5.3. The radius r_{anchors} is part of the parameter selection algorithm, described in Section 5.3.3.1.

The generated discs, represented by $\left(x_{\text{al}}^k, y_{\text{al}}^k, r_{\text{anchors}}\right)$, are then used to create local contexts $\mathcal{G}_k$.

A detection, characterized by the index i, is added to the local context $\mathcal{G}_k$, defined by the disc $\left(x_{\text{al}}^k, y_{\text{al}}^k, r_{\text{anchors}}\right)$, if the detection features $(\mathbf{x}_{\text{uc,p}}^i, \mathbf{y}_{\text{uc,p}}^i)$ are inside the disc and if the number of elements in the corresponding context is less than $d_{\max}$:

$$\text{If } (\mathbf{x}_{\text{uc,p}}^i - x_{\text{al}}^k)^2 + (\mathbf{y}_{\text{uc,p}}^i - y_{\text{al}}^k)^2 \leq r_{\text{anchors}}^2 \text{ and } |\mathcal{G}_k| < d_{\max}, \text{ then } \mathcal{G}_k \leftarrow \mathcal{G}_k \cup \{i\},$$

here, $d_{\max}$ is a parameter that is described in Section 5.3.3.1.

The algorithm includes a second step to handle detections that might be missed due to an excessive number of detections within some discs. New discs and the corresponding contexts $\mathcal{G}_m$ are created using $(\mathbf{x}_{\text{uc,p}}^m, \mathbf{y}_{\text{uc,p}}^m)$ as disc centers, where m is a detection that does not belong to any context $\mathcal{G}_k$. Each of these new discs has a radius halved from r_{anchors}. This process is repeated iteratively until all detections are included in at least one context:

$$\begin{aligned}
&\text{For each detection } m \text{ such that } m \notin \bigcup_k \mathcal{G}_k, \\
&\text{a new context } \mathcal{G}_c \text{ is created with disc } (\mathbf{x}_{\text{uc,p}}^m, \mathbf{y}_{\text{uc,p}}^m, \frac{r_{\text{anchors}}}{2}). \\
&\text{This process is repeated until } \bigcup_k \mathcal{G}_k = \{1, 2, \ldots, n_{\text{d}}\}.
\end{aligned}$$

The Local Context Generation then outputs the resulting n_{g} contexts $\mathcal{G} = \{\mathcal{G}_1, \mathcal{G}_2, \ldots, \mathcal{G}_{n_{\text{g}}}\}$:

$$\mathcal{G} = \text{LCG}(\mathbf{F}).$$

Since discs can overlap, detections can be part of multiple local contexts $\mathcal{G}_k$. This overlap is advantageous, as it enables detections to be processed by the False Detection Filter and Partitioning Module in different contexts. For the False Detection Filter, described in 5.3.4, this can result in an ensemble-like effect, as the false detection probability is averaged across different contexts [12, pp. 580].

5.3.3.1 Parameter Settings

The algorithm is built on the choice of parameters s_{f} and $d_{\max}$, where s_{f} determines the number of anchor locations relative to the total number of detections n_{d} and

$d_{\max}$ sets the maximum number of detections for each context. These parameters should be determined in unison so that as few detections as possible are missed and the computational complexity of the resulting contexting is minimized. From these parameters, the rest of the parameter values are generated. This process is outlined below.

First, the area of the FOV, in square kilometers, is calculated:

$$\mathrm{FOV}_{\mathrm{area}} = \frac{(r_{\max}^2 - r_{\min}^2)\,(\varphi_{\max} - \varphi_{\min})}{10^6} \cdot 0.5,$$

here, $r_{\max}$ and $r_{\min}$ are the maximum and minimum radii of the FOV in meters, and $\varphi_{\max}$ and $\varphi_{\min}$ are the maximum and minimum angles in radians.

Given this area, the parameter N for generating the sunflower seed sequence in A.2 is determined by:

$$N = \left\lfloor s_{\mathrm{f}}\, n_{\mathrm{d}}\, \frac{r_{\max}^2\, \pi}{\mathrm{FOV}_{\mathrm{area}}} \right\rceil,$$

here, s_{f} is a fraction that controls the number of anchor locations as a fraction of the total number of detections n_{d}.

The approximate number of anchors within the FOV is then:

$$n_{\text{anchors in FOV}} \approx \lfloor s_{\mathrm{f}}\, n_{\mathrm{d}} \rceil.$$

The density of detections in the FOV, d_{d}, is given by:

$$d_{\mathrm{d}} = \frac{n_{\mathrm{d}}}{\mathrm{FOV}_{\mathrm{area}}}.$$

Using this density, the radius of each anchor, in meters, is calculated as:

$$r_{\mathrm{anchors}} = \sqrt{\frac{d_{\max}/2}{d_{\mathrm{d}}\, \pi}} \cdot 10^3,$$

here, $d_{\max}$ is the maximum number of detections in each context. This ensures that the average number of detections within the area of the anchor locations, assuming a uniform distribution of detections within the FOV, is $d_{\max}/2$.

5.3.4 False Detection Filter Module

The False Detection Filter (FDF) module aims to identify and eliminate false detections from the radar data, reducing the number of irrelevant detections before subsequent processing. Taking as input the feature matrix $\mathbf{F}$ and local contexts $\mathcal{G}$ generated by the Local Context Generation Module, the FDF computes true detection probabilities for each local context. By aggregating these probabilities, the module derives a global estimation of true detections, distinguishing genuine detections from false ones. This filtering step helps streamline the tracking pipeline, ensuring that only detections beleived to be meaningful are considered in the next

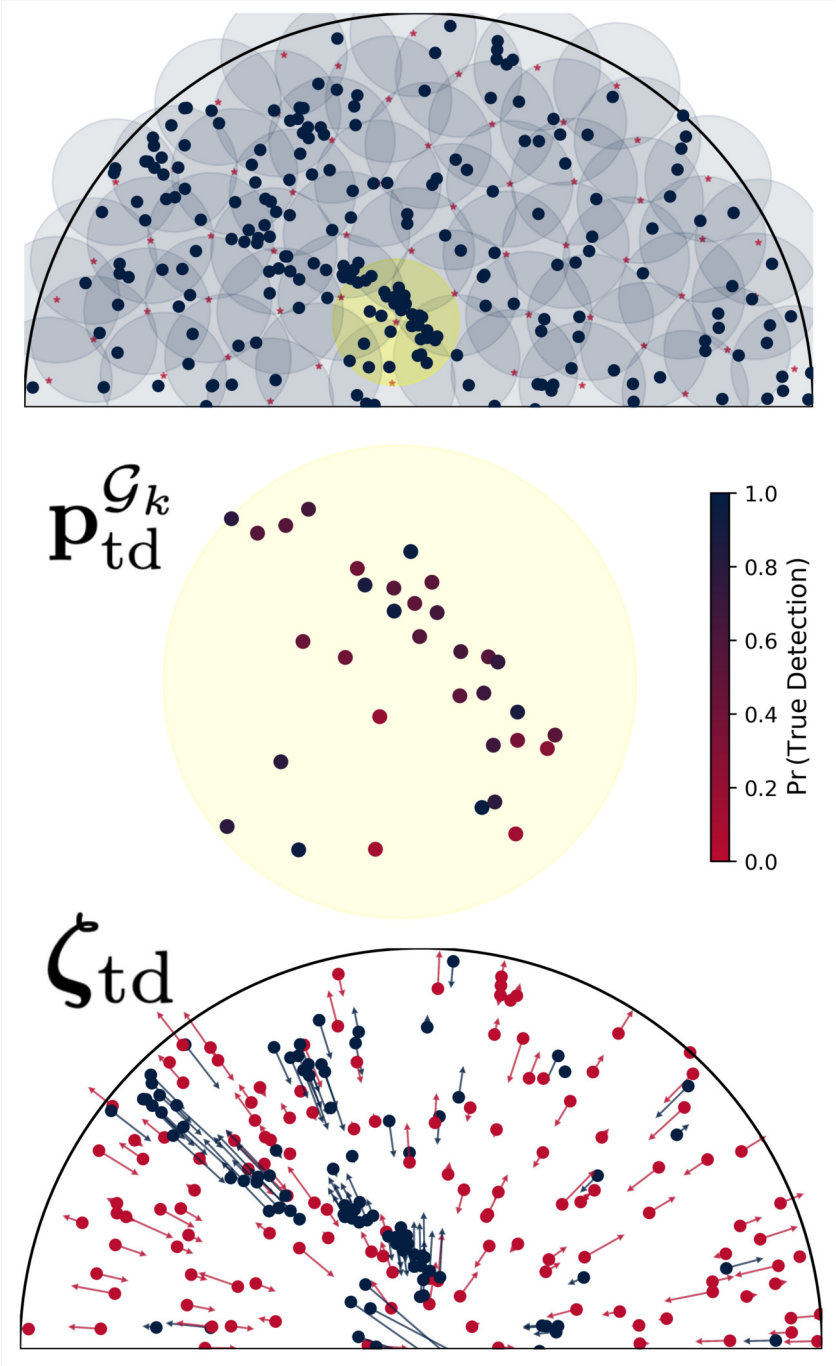


Figure 5.4: The upper part of the figure illustrates the input to the False Detection Filter (see Figure 5.3), with velocity components removed for a less cluttered figure. The yellow highlighted local context in the upper part of the figure is then chosen to illustrate the localized computations, resulting in true detection probabilities $\mathbf{p}_{\text{td}}^{\mathcal{G}_k}$ for each local context k. These local probabilities are then aggregated to form the global true detection probabilities $\mathbf{p}_{\text{td}}$ and the true detection decisions $\boldsymbol{\zeta}_{\text{td}}$. True detections are represented by a blue color, and false by a red color. This is a completely synthetic scenario meant to illustrate what the output *could* look like, given the synthetic input from Figure 5.1.

stages of the algorithm.

The corresponding feature matrix for each context $\mathbf{F}^{\mathcal{G}_k} \in \mathbb{R}^{|\mathcal{G}_k| \times 16}$ is first transformed to:

$$\mathbf{FDF}^{\mathcal{G}_k} = \begin{bmatrix} \boldsymbol{\psi}(\mathbf{x}_{\mathrm{uc}}) \\ \boldsymbol{\psi}(\mathbf{y}_{\mathrm{uc}}) \\ \boldsymbol{\eta}(\dot{\mathbf{x}}) \\ \boldsymbol{\eta}(\dot{\mathbf{y}}) \\ \boldsymbol{\psi}(\mathbf{x}_{\mathrm{uc,p}}) \\ \boldsymbol{\psi}(\mathbf{y}_{\mathrm{uc,p}}) \\ \boldsymbol{\eta}(\boldsymbol{\sigma}_{x_{\mathrm{uc}}}) \\ \boldsymbol{\eta}(\boldsymbol{\sigma}_{y_{\mathrm{uc}}}) \\ \boldsymbol{\rho}_{\mathrm{x_{uc}y_{uc}}} \\ \mathbf{t} \end{bmatrix}^{\mathrm{T}} \in \mathbb{R}^{|\mathcal{G}_k| \times 10},$$

where:

$$\boldsymbol{\psi}(\mathbf{x}) = 2\frac{\mathbf{x} - \bar{\mathbf{x}}}{r_{\mathrm{anchors}}},$$
$$\boldsymbol{\eta}(\mathbf{x}) = 2\frac{\mathbf{x}}{r_{\mathrm{anchors}}},$$

here, $\bar{\mathbf{x}}$ represents the mean of the input vector $\mathbf{x}$.

The location-based features, $\mathbf{x}_{\mathrm{uc}}, \mathbf{y}_{\mathrm{uc}}, \mathbf{x}_{\mathrm{uc,p}}, \mathbf{y}_{\mathrm{uc,p}}$, are transformed by subtracting the mean of the context and dividing with half the radius of the anchors. As such, the positional features have a range between -2 and 2. The Cartesian velocities are divided by half the radius as well, so that the position and velocity are scaled in unison. This scaling ensures that the velocities are accurately represented in this new transformed space. The Cartesian standard deviations are also scaled by half of the seed radius, as their values can exceed 150, to minimize the range of values. In addition to this, scaling also ensures that the corresponding variances are accurately transformed to represent the new space. Since correlation is bounded between -1 and 1 it is not transformed.

The transformed context-wise feature matrices $\mathbf{FDF}^{\mathcal{G}_k}$ are then fed to a model based on the encoder part of a transformer, trained to perform binary classification. The model outputs a score between 0 and 1 for each detection, where scores close to 1 indicate that the model perceives that the detection is a true detection. After processing all contexts $\mathcal{G}$, the scores are aggregated by taking the mean score of each detection based on all contexts in $\mathcal{G}$ it is part of. The result is the true detection probabilities $\mathbf{p}_{\mathrm{td}}$ and the true detection decisions $\boldsymbol{\zeta}_{\mathrm{td}}$. The process is described in the following.

Model Architecture

The inputs $\mathbf{FDF}^{\mathcal{G}_k}$ are first preprocessed the same way as detailed in Section 4.1.1, however, the positional embeddings $\boldsymbol{\epsilon}_{\text{enc}}$ are generated slightly differently.

The positional embeddings are generated by adding the scan-time embeddings (obtained from a learnable lookup table as mentioned in Section 4.1.1) and a separate positional encoding derived from the features, excluding scan-times $\mathbf{t}$. These separate positional embeddings are generated by a learnable Fourier feature transformation (see Section 3.3.3.1). This additional positional encoding is generated since the detections can be widely dispersed, making a learnable spatial positional encoding potentially beneficial.

The input embeddings $\mathbf{E}^{\mathcal{G}_k}_{\text{enc},0}$ and the positional embeddings are subsequently fed into the encoder part of the model, with the same structure as described in Section 4.1.2. The only difference in the sub-layers, is that the feedforward neural networks now transform to a dimension of 528 instead of 2048 before transforming back to 256. The same number of encoder layers and attention heads is used.

The resulting embeddings $\mathbf{E}^{\mathcal{G}_k}_{\text{enc,out}}$ are then passed through a linear layer that reduces their dimension to 256, followed by a layer normalization and another linear layer that transforms them to a dimension of 1, resulting in pre-sigmoid probabilities $\boldsymbol{\gamma}^{\mathcal{G}_k}$:

$$\boldsymbol{\gamma}^{\mathcal{G}_k} = \mathrm{L}^1\left(\mathrm{LayerNorm}\left(\mathrm{L}^{256}\left(\mathbf{E}^{\mathcal{G}_k}_{\text{enc,out}}\right)\right)\right).$$

The pre-sigmoid probabilities $\boldsymbol{\gamma}^{\mathcal{G}_k}$ are then passed through a sigmoid function resulting in true detection probabilities:

$$\mathbf{p}^{\mathcal{G}_k}_{\text{td}} = \frac{1}{1 + e^{-\boldsymbol{\gamma}^{\mathcal{G}_k}}}. \tag{5.1}$$

After processing all contexts $\mathcal{G}$, the result is aggregated to form global true detection probabilities $\mathbf{p}_{\text{td}}$ and true detection decisions $\boldsymbol{\zeta}_{\text{td}}$. The aggregation process is detailed below.

Aggregating Localized Outputs

Since detections may appear in multiple contexts, the final score $\mathbf{p}_{\text{td},j}$ for detection j is obtained by aggregating the scores in all contexts where detection j appears. Specifically, the aggregation is performed by taking the mean of the scores from each context:

$$\mathbf{p}_{\text{td},j} = \frac{1}{|\mathcal{G}(j)|} \sum_{\mathcal{G}_i \ni j} \mathbf{p}^{\mathcal{G}_i}_{\text{td},\mathcal{J}(j,\mathcal{G}_i)},$$

here $\mathcal{G}(j)$ denotes the set of all contexts that contain detection j, and $|\mathcal{G}(j)|$ is the number of such contexts. $\mathcal{J}(j, \mathcal{G}_i)$ returns the index of detection j in $\mathbf{p}^{\mathcal{G}_i}_{\text{td}}$. This process integrates the context-wise context, producing a unified score for each detection.

A binary output denoted $\boldsymbol{\zeta}_{\text{td}}$, which represents whether a detection is considered false or not, is constructed by applying a threshold τ_{fdf}:

$$\boldsymbol{\zeta}_{\text{td},j} = \begin{cases} 1, & \text{if} \quad \mathbf{p}_{\text{td},j} \geq \tau_{\text{fdf}}, \\ 0, & \text{otherwise.} \end{cases} \tag{5.2}$$

To gain an intuitive understanding of the False Detection Filter, Figure 5.4 has been created based on the example scenario. In this figure, the local context processing is highlighted along with the input of detections and local contexts. The end result is illustrated through a visualization of true detection decisions $\boldsymbol{\zeta}_{\text{td}}$.

Loss Functions

For training of the False Detection Filter, two different implementations of binary cross-entropy are used. For more information, refer to Section 3.4.

The first loss is constructed by calculating the binary cross-entropy loss (see (3.6) in Section 3.1) of the predicted probabilities $\mathbf{p}_{\text{td}}^{\mathcal{G}_k}$ with the underlying labels $\hat{\mathbf{Y}}_{\text{td}}^{\mathcal{G}_k}$ corresponding to 1 for true detections and 0 for false detections:

$$\mathcal{L}_0 = \mathcal{L}_{\text{BCE}}\left(\mathbf{p}_{\text{td}}^{\mathcal{G}_k},\ \hat{\mathbf{Y}}_{\text{td}}^{\mathcal{G}_k}\right).$$

The second part of the loss function is also a binary cross-entropy loss, but related to the ground-truth associations between detections. The loss can be regarded as related to contrastive learning, described in Section 4.1.5.1. With ground-truth associations, we refer to whether a detection i originated from the same underlying target as detection j. We denote the ground-truth association matrix for context k as $\mathbf{H}^{\mathcal{G}_k} \in \mathbb{R}^{|\mathcal{G}_k| \times |\mathcal{G}_k|}$, where:

$$\mathbf{H}_{ij}^{\mathcal{G}_k} = \begin{cases} 1 & \text{if detections } i \text{ and } j \text{ are associated with the same target,} \\ 0 & \text{otherwise,} \end{cases}$$

here, false detections are treated as coming from different objects, resulting in 0s everywhere except on the diagonal.

We construct predictions by extracting the multi-head attention scores $\mathbf{S}^h$ from the last layer of the encoder for each head h. For information on the attention mechanism, see Section 3.3.1. We then construct the so-called association probability predictions $\tilde{\mathbf{A}}$ as follows:

$$\tilde{\mathbf{A}}_{ij} = \frac{1}{8} \sum_{h=1}^{8} \frac{1}{1 + e^{-\mathbf{S}_{ij}^h}}. \tag{5.3}$$

The loss is then calculated and summed for every row i

$$\mathcal{L}_1 = \frac{1}{|\mathcal{G}_k|} \sum_{i=1}^{|\mathcal{G}_k|} \mathcal{L}_{\text{BCE}}\left(\tilde{\mathbf{A}}_i,\ \mathbf{H}_i\right).$$

Finally, the total loss is a weighted sum of these two components:

$$\mathcal{L} = \mathcal{L}_0 + 8\mathcal{L}_1.$$

The secondary loss $\bar{\mathcal{L}_1}$ is multiplied by 8 to increase its impact.

5.3.5 Partitioning Module

The partitioning module receives as input the local contexts $\mathcal{G}$, true detection decisions $\boldsymbol{\zeta}_{\text{td}}$, and true detection probabilities $\mathbf{p}_{\text{td}}$ as input. Given these inputs, it outputs a partition $\mathcal{P} = \{\mathcal{P}_1, \mathcal{P}_2, \ldots, \mathcal{P}_k\}$. In Figure 5.5 an overview of the first parts of the algorithm is illustrated with the example scenario. The resulting output of the module can be seen in Figure 5.6.

The detections are partitioned with the aim of extracting independent multi-target tracking sub-problems that can be solved independently. There are a wide variety of ways to partition detections and many approaches were explored in the context of this thesis. The approach of this module relies on interpreting the detections as nodes in a graph $\mathcal{A}$, the edges between them representing the strength or weight of the connection (association), represented by an edge matrix $\mathbf{G}$. What must be determined are the edge values $\mathbf{G}$.

The algorithm presented in Sections 5.3.5.1, 5.3.5.2, 5.3.5.3 and 5.3.5.4 relies on first calculating localized association graphs $\mathcal{A}^{\mathcal{G}_k}$ for each local context, described in detail in Section 5.3.5.1. The weights in each local context are decided by an association strength function $\omega(i, j)$ between the detection pairs (i, j). The localized association graphs for each context are then aggregated to form a global graph $\mathcal{A}$ with edge values $\mathbf{G}$, described in Section 5.3.5.2. Given the aggregated edge values $\mathbf{G}$, the partitioning approach with the Leiden algorithm is described in Section 5.3.5.3 (for more information on the Leiden algorithm, see Section 3.5). The partition is then post-processed, which is described in detail in Section 5.3.5.4

In Sections 5.3.5.5, 5.3.5.6, 5.3.5.7 and 5.3.5.8 four different implementations of $\omega(i, j)$ are outlined.

5.3.5.1 Localized Association Graphs

The first step is the creation of localized association graphs $\mathcal{A}^{\mathcal{G}_k}$, where each graph corresponds to a context $\mathcal{G}_k$ generated by the Local Context Generation. The nodes of the graphs are represented by the detections in each context $\mathcal{G}_k$, and the edges $\mathbf{G}_{ij}^{\mathcal{G}_k}$ correspond to the connection strength between the detections (nodes).

The connection strength between detections is determined by a connection strength function $\omega(i, j)$ with values in $[0, 1]$:

$$\mathbf{G}_{ij}^{\mathcal{G}_k} = \omega(i, j).$$

In Sections 5.3.5.5, 5.3.5.6, 5.3.5.7 and 5.3.5.8 four different implementations of this function are outlined.

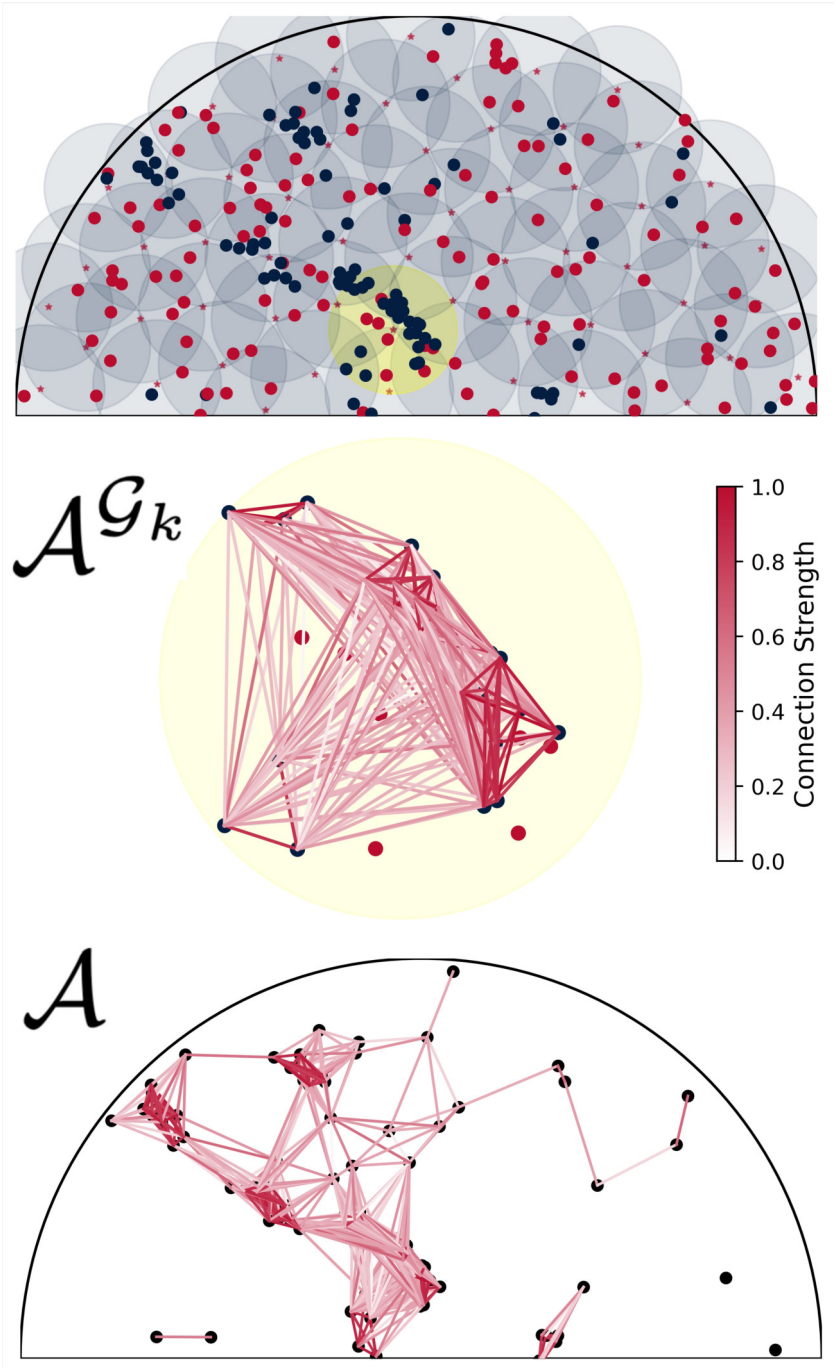


Figure 5.5: The upper part of the figure illustrates the input to the partitioning module, see Figure 5.4 for the False Detection Filter output and Figure 5.3 for the Local Context Generation output. The same local context as in Figure 5.4 is then highlighted in order to illustrate the localized graphs $\mathcal{A}^{\mathcal{G}_k}$ that are created, and then aggreagated to the global graph $\mathcal{A}$. Red dots in the upper and middle figure represent detections i deemd as false by the false detection filter $\boldsymbol{\zeta}_{\text{td},i} = 0$. This is a completely synthetic scenario meant to illustrate what the different outputs *could* look like, given the synthetic input from Figure 5.1.

In Figure 5.5 an example of a local association graph is illustrated in the middle portion of the figure, based on the example scenario.

5.3.5.2 Graph Aggregation

The complete association graph $\mathcal{A}$ is created by aggregating all localized edges $\mathbf{G}_{ij}^{\mathcal{G}_k}$ where $\boldsymbol{\zeta}_{\text{td},i} = \boldsymbol{\zeta}_{\text{td},j} = 1$ (see (5.2)).

If the edges are binary, that is, $\mathbf{G}_{ij}^{\mathcal{G}_k} \in \{0, 1\}$, a maximum aggregation of the connection strength is applied:

$$\mathbf{G}_{ij} = \begin{cases} \max_{\mathcal{G}_k \ni i,j} \mathbf{G}^{\mathcal{G}_k}_{\mathcal{J}(i,\mathcal{G}_k)\mathcal{J}(j,\mathcal{G}_k)}, & \text{if } \boldsymbol{\zeta}_{\text{td},i} = 1 \text{ and } \boldsymbol{\zeta}_{\text{td},j} = 1, \\ 0, & \text{otherwise,} \end{cases}$$

here, $\mathcal{J}(i, \mathcal{G}_k)$ returns the index of detection i in the edge matrix $\mathbf{G}^{\mathcal{G}_k}$ corresponding to context $\mathcal{G}_k$. The aggregation is over all local contexts $\mathcal{G}_k$ and results in $\mathbf{G}_{ij}$ receiving the value 1 if any localized edge between detection i and j has the value 1.

If the edges are not binary, a mean connection strength aggregation is performed:

$$\mathbf{G}_{ij} = \begin{cases} \frac{1}{|\mathcal{G}(i,j)|} \sum_{\mathcal{G}_k \ni i,j} \mathbf{G}^{\mathcal{G}_k}_{\mathcal{J}(i,\mathcal{G}_k)\mathcal{J}(j,\mathcal{G}_k)}, & \text{if } \boldsymbol{\zeta}_{\text{td},i} = 1 \text{ and } \boldsymbol{\zeta}_{\text{td},j} = 1, \\ 0, & \text{otherwise,} \end{cases}$$

here, $|\mathcal{G}(i,j)|$ is the number of contexts that contain both detection i and j.

These aggregations convert the potentially directed localized graphs, that is, $\mathbf{G}_{ij}^{\mathcal{G}_k}$ does not have to equal $\mathbf{G}_{ji}^{\mathcal{G}_k}$, into an undirected graph $\mathbf{G}_{ij} = \mathbf{G}_{ji}$. Furthermore, all edges where either detection is considered false by the False Detection Filter are set to zero.

The resulting global graph $\mathcal{A}$ for the example scenario is illustrated in the lower part of Figure 5.5.

5.3.5.3 Partitioning

Now, the undirected graph $\mathcal{A}$ with the edge matrix $\mathbf{G}$ can be partitioned. This is done by applying the Leiden algorithm with modularity as an objective function (see Section 3.5).

The resulting partition subsets $\{\tilde{\mathcal{P}}_1, \tilde{\mathcal{P}}_2, \ldots, \tilde{\mathcal{P}}_p\}$ are then post-processed before passing the resulting subsets to the MT3LS module. The post-processing is detailed in the following section.

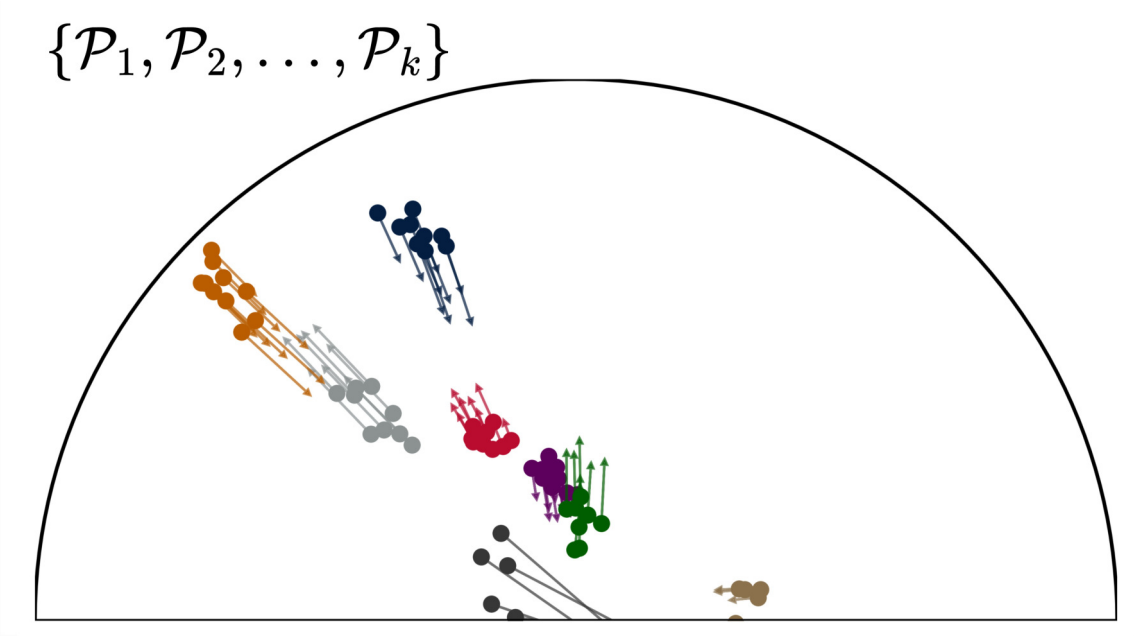


Figure 5.6: This figure is an illustration of the output subsets $\{\mathcal{P}_1, \mathcal{P}_2, \ldots, \mathcal{P}_k\}$ after partioning $\mathcal{A}$ with the Leiden algorithm and performing post-processing. Refer to Figure 5.5 for an illustration of the earlier steps of the partioning module. This is a completely synthetic scenario meant to illustrate what the output *could* look like, given the synthetic input from Figure 5.1.

5.3.5.4 Post-Processing

Each subset $\tilde{\mathcal{P}}_i$ undergoes a screening step to ensure reliability. Subsets are excluded if they contain fewer than four detections to minimize the risk of subsets consisting only of false detections that by accidence appear as a target. Additionally, subsets missing detections from the last two time steps are excluded. This step removes subsets corresponding to potential targets that are likely no longer detectable and thus not of interest.

Formally, a subset $\tilde{\mathcal{P}}_i$ is discarded if:

$$\tilde{\mathcal{P}}_i \text{ is discarded if} \quad |\tilde{\mathcal{P}}_i| < 4 \quad \text{or} \quad \{t_{\max}, t_{\max-1}\} \not\subseteq \mathcal{T}\left(\tilde{\mathcal{P}}_i\right),$$

where:

- $|\tilde{\mathcal{P}}_i|$: The number of detections in subset $\tilde{\mathcal{P}}_i$,
- $t_{\max}$: The most recent scan time,
- $t_{\max-1}$: The second most recent scan time,
- $\mathcal{T}\left(\tilde{\mathcal{P}}_i\right)$: The set of scan times covered by the detections in $\tilde{\mathcal{P}}_i$.

Let $\mathcal{D}_t \subseteq \mathcal{D}$ denote the subset of detections associated with the scan time t. Using this notation, we proceed to define further conditions on the subsets $\tilde{\mathcal{P}}_i$. For subsets $\tilde{\mathcal{P}}_i$ that satisfy the following conditions:

1. A majority of detections in $\tilde{\mathcal{P}}_i$ belong to unique time steps:

$$|\mathcal{T}\left(\tilde{\mathcal{P}}_i\right)| > \frac{|\tilde{\mathcal{P}}_i|}{2},$$

2. $\tilde{\mathcal{P}}_i$ does not have more than one detection from each of the last two time steps:

$$|\tilde{\mathcal{P}}_i \cap \mathcal{D}_{t_{\max}}| \leq 1 \quad \text{and} \quad |\tilde{\mathcal{P}}_i \cap \mathcal{D}_{t_{\max}-1}| \leq 1,$$

3. $\tilde{\mathcal{P}}_i$ contains at least two detections from the same scan:

$$\exists t \in \mathcal{T}(\tilde{\mathcal{P}}_i) \text{ such that } |\tilde{\mathcal{P}}_i \cap \mathcal{D}_t| \geq 2,$$

only the detection with the highest probability of being a true detection, denoted $\mathbf{p}_{\text{td}}$, is retained for each scan time t. Formally:

$$\tilde{\mathcal{P}}_i \leftarrow \bigcup_{t \in \mathcal{T}(\tilde{\mathcal{P}}_i)} \arg \max_{j \in (\tilde{\mathcal{P}}_i \cap \mathcal{D}_t)} \mathbf{p}_{\text{td},j},$$

here $\mathbf{p}_{\text{td},j}$ is the probability of detection j being true. This processing is based on the assumption that a single target can at a maximum result in one detection per radar scan. By processing subsets that fulfill these criteria, we could remove false detections or detections from different targets that might have been grouped together, and thus aid the MT3LS module.

The resulting subsets are denoted $\{\mathcal{P}_1, \mathcal{P}_2, \ldots, \mathcal{P}_k\}$ and represent a partition $\mathcal{P}$ of a subset $\tilde{\mathcal{D}} \subseteq \mathcal{D}$ of the set of detections $\mathcal{D}$. The partition $\mathcal{P}$ should contain subsets of highly connected detections with little connection to other subsets of detections. They can thus be interpreted as separate problems in the sphere of multi-target tracking. In Figure 5.6 the resulting partition is illustrated for the example scenario.

5.3.5.5 Connection Strength by Positional Uncertainty

Detections that are relatively close in time and originate from the same object should be spatially close when predicted to the same point in time. Furthermore, their corresponding measurement uncertainty should define a likely spread in space. This can be utilized as a way to create connection strengths between detections.
One way to determine $\omega(i,j)$ is to interpret the features $\mathbf{x}_{\text{uc,p},i}, \mathbf{y}_{\text{uc,p},i}, \boldsymbol{\sigma}_{x_{\text{uc},i}}, \boldsymbol{\sigma}_{y_{\text{uc},i}}, \boldsymbol{\rho}_{x_{\text{uc},i}y_{\text{uc},i}}$ (see Section 5.3.2) of detection i as the mean vector $\boldsymbol{\mu}^i$ and the covariance matrix $\boldsymbol{\Sigma}^i$ of a multivariate normal distribution:

$$\boldsymbol{\mu}^i = \begin{bmatrix} \mathbf{x}_{\text{uc,p},i} \\ \mathbf{y}_{\text{uc,p},i} \end{bmatrix},$$
$$\boldsymbol{\Sigma}^i = \begin{bmatrix} \boldsymbol{\sigma}_{x_{\text{uc},i}} & \boldsymbol{\sigma}_{x_{\text{uc},i}}\boldsymbol{\sigma}_{y_{\text{uc},i}}\boldsymbol{\rho}_{x_{\text{uc},i}y_{\text{uc},i}} \\ \boldsymbol{\sigma}_{x_{\text{uc},i}}\boldsymbol{\sigma}_{y_{\text{uc},i}}\boldsymbol{\rho}_{x_{\text{uc},i}y_{\text{uc},i}} & \boldsymbol{\sigma}_{y_{\text{uc},i}} \end{bmatrix}.$$

The Mahalanobis distance (see Section 3.6.1) can then be calculated to the sample $\mathbf{x}^j = [\mathbf{x}_{\text{uc,p},j}, \mathbf{y}_{\text{uc,p},j}]^{\text{T}}$ represented by detection j. If the two detections are close, determined by a threshold $\tau_{\text{md,PU}}$, the connection strength can be set to 1, otherwise, 0:

$$\omega(i,j) = \begin{cases} 1, & \text{if} \quad D_M\left(\mathbf{x}^j | \boldsymbol{\mu}^i, \boldsymbol{\Sigma}^i\right) \leq \tau_{\text{md,PU}}, \\ 0, & \text{otherwise.} \end{cases}$$

This approach ensures that detections that are spatially close, considering their positional uncertainty, are connected within the association graph.

5.3.5.6 Connection Strength by Normal Approximations I

Another way to determine whether two detections should have a connection is to construct statistical tests that incorporate more information than the method described in Section 5.3.5.5. Here, indices i and j refer to the two detections that are being compared.

Range and Range Rate Statistic

The first test statistic, denoted z_1, is based on the assumption that if the detections originate from the same object, the difference in their range divided by their time difference should approximately match the range rate of detection i (or j). Thus, subtracting the range rate of detection i, should result in a statistic close to zero:

$$\tilde{z}_1 = \frac{r_j - r_i}{t_j - t_i} - \dot{r}_i \sim \mathcal{N}\left(0,\ \sigma^2_{\tilde{z}_1}\right).$$

Assuming that this statistic is normally distributed, a statistic following the standard normal distribution can be reached by dividing by its standard deviation. The standard deviation of $\tilde{z}_1$ can be approximated by:

$$\sigma_{\tilde{z}_1} = \sqrt{\frac{\sigma^2_{r_j} + \sigma^2_{r_i}}{(t_j - t_i)^2} + \sigma^2_{\dot{r}_i} + \sigma^2_{a_i}|t_j - t_i|}, \tag{5.4}$$

here $\sigma^2_{r_i}$ and $\sigma^2_{r_j}$ are the variances of the range measurements for detections i and j, respectively, and $\sigma^2_{\dot{r}_i}$ is the variance of the range rate of detection i. The parameter $\sigma^2_{a_i}$ can be regarded as the process noise variance, related to the movement model of the object from which the detection i originates. This parameter can be set on the basis of assumptions as to how the objects move. The test statistic z_1 can thus be calculated as follows:

$$z_1 = \frac{\tilde{z}_1}{\sigma_{\tilde{z}_1}} \sim \mathcal{N}\left(0,\ 1\right).$$

Angular Difference Statistic

Similarly, we construct another statistic z_2, related to the angular difference between the detections. By assuming a normal distribution, we first arrive at the following.

$$\tilde{z}_2 = \varphi_j - \varphi_i \sim \mathcal{N}\left(0,\ \sigma^2_{\tilde{z}_2}\right),$$

whose standard deviation can be approximated by:

$$\sigma_{\tilde{z}_2} = \sqrt{\sigma_{\varphi_j}^2 + \sigma_{\varphi_i}^2 + \frac{\sigma_{\omega_i}^2}{r_i^2}(t_j - t_i)^2}, \tag{5.5}$$

here $\sigma_{\varphi_i}^2$ and $\sigma_{\varphi_j}^2$ are the variances of the angle measurements. The parameter $\sigma_{\omega_i}^2$ is the variance in angular velocity for detection i, which is a parameter setting as it is unknown. It can be set on the basis of knowledge of the objects' average velocity. We can now construct z_2:

$$z_2 = \frac{\tilde{z}_2}{\sigma_{\tilde{z}_2}} \sim \mathcal{N}(0,\ 1).$$

Combining the Statistics

The connection strength function $\omega(i, j)$ returns 1 if the absolute values of both test statistics $|z_1|$ and $|z_2|$ are below a predefined threshold $\tau_{\text{md,NAI}}$, indicating that the two detections are likely from the same object. Otherwise, it returns 0:

$$\omega(i,j) = \begin{cases} 1 & \text{if } |z_1| \leq \tau_{\text{md,NAI}} \text{ and } |z_2| \leq \tau_{\text{md,NAI}}, \\ 0 & \text{otherwise.} \end{cases}$$

5.3.5.7 Connection Strength by Normal Approximations II

Instead of reducing the test statistics from Section 5.3.5.6 to a binary value, one could instead take advantage of the probability density functions of the test statistics given the assumed normal distributions.

For each test statistic, the corresponding probability density can be calculated using the standard normal distribution. The probability density associated with the test statistic z_1 is given by:

$$p_1 = \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{z_1^2}{2}\right).$$

The probability density associated with the test statistic z_2 is:

$$p_2 = \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{z_2^2}{2}\right).$$

Assuming independence between the test statistics, the joint probability density of p_1 and p_2 is their product:

$$p_{12} = p_1 \cdot p_2.$$

To convert this joint probability density into the range $[0, 1]$, a min-max normalization is applied for the context currently processed $\mathcal{G}_k$:

$$\text{score}_{ij} = \frac{p_{12} - \min(\mathbf{p}_{12}^{\mathcal{G}_k})}{\max(\mathbf{p}_{12}^{\mathcal{G}_k}) - \min(\mathbf{p}_{12}^{\mathcal{G}_k})},$$

here $\mathbf{p}_{12}^{\mathcal{G}_k} \in \mathbb{R}^{|\mathcal{G}_k| \times |\mathcal{G}_k|}$ contains all the joint probabilities in the subset $\mathcal{G}_k$, which represents the localized graph $\mathcal{A}^{\mathcal{G}_k}$ that is currently being created.
The connection strength function $\omega(i, j)$ for this approach utilizes a threshold $\tau_{\text{da,NAII}}$ that sets a lower bound for the connection strengths between detections:

$$\omega(i,j) = \begin{cases} \text{score}_{ij} & \text{if score}_{ij} > \tau_{\text{da,NAII}}, \\ 0 & \text{otherwise.} \end{cases}$$

5.3.5.8 Connection Strength as Attention-Score

Another function that essentially calculates the connection strengths between detections is the last attention layer of the False Detection Filter as we train $\tilde{\mathbf{A}}$ to accurately represent the ground-truth association matrices $\mathbf{H}^{\mathcal{G}_k}$. We could thus interpret $\tilde{\mathbf{A}}$ from (5.3) as the edge matrix $\mathbf{G}_{ij}^{\mathcal{G}_k}$. The connection strength function can thus be constructed as follows:

$$\omega(i,j) = \begin{cases} \tilde{\mathbf{A}}_{ij} & \text{if } \tilde{\mathbf{A}}_{ij} > \tau_{\text{da,AS}}, \\ 0 & \text{otherwise,} \end{cases}$$

here, $\tau_{\text{da,AS}}$ is a threshold that sets a lower bound for the connection strength between detections, just as in the previous section.

This approach differs from the others, since each connection strength is based on the entire local context, not just the two detections represented by the weight.

5.3.6 MT3LS

After the Partitioning Module has partitioned the detections into subsets $\{\mathcal{P}_1, \mathcal{P}_2, \ldots, \mathcal{P}_k\}$, they are processed in parallel as separate batches by a modified version of the MT3v2 model, which we will call MT3 Large Scale or MT3LS (see Chapter 4). By processing smaller, reliable subsets of data, the MT3LS Module can perform filtering and tracking with greater accuracy and computational efficiency, compared to attempting to process all detections across the entire FOV at once. This approach ensures that the model can scale to the demands of large FOVs and high detection counts, maintaining performance even in complex and challenging tracking scenarios.

Feature Transformations

In the original MT3v2 model, all radar detections were processed jointly, which allowed normalization of the detections by the radars constant field of view (FOV) characteristics. However, given that detections are now processed in separate, disjoint batches without a fixed area (FOV), the original approach is no longer feasible. To adapt to these changes, the input feature vector (see Section 5.3.2) for each subset $\mathcal{P}_j$ is transformed:

$$\mathbf{LF}^{\mathcal{P}_j} = \begin{bmatrix} \boldsymbol{\psi}\left(\mathbf{x}_{\mathrm{uc}}, \boldsymbol{\sigma}_{\mathrm{x_{uc}}}\right) \\ \boldsymbol{\psi}\left(\mathbf{y}_{\mathrm{uc}}, \boldsymbol{\sigma}_{\mathrm{y_{uc}}}\right) \\ \boldsymbol{\lambda}\left(\dot{\mathbf{x}}, \boldsymbol{\sigma}_{\mathrm{x_{uc}}}\right) \\ \boldsymbol{\lambda}\left(\dot{\mathbf{y}}, \boldsymbol{\sigma}_{\mathrm{y_{uc}}}\right) \\ \boldsymbol{\psi}\left(\mathbf{x}_{\mathrm{uc,p}}, \boldsymbol{\sigma}_{\mathrm{x_{uc}}}\right) \\ \boldsymbol{\psi}\left(\mathbf{y}_{\mathrm{uc,p}}, \boldsymbol{\sigma}_{\mathrm{y_{uc}}}\right) \\ \boldsymbol{\eta}\left(\boldsymbol{\sigma}_{\mathrm{x_{uc}}}\right) \\ \boldsymbol{\eta}\left(\boldsymbol{\sigma}_{\mathrm{y_{uc}}}\right) \\ \boldsymbol{\rho}_{\mathrm{x_{uc}y_{uc}}} \\ \mathbf{t} \end{bmatrix}^{\mathrm{T}} \in \mathbb{R}^{|\mathcal{P}_j| \times 10},$$

here the normalization functions are defined as:

$$\boldsymbol{\psi}\left(\mathbf{x}, \mathbf{y}\right) = \frac{\mathbf{x} - \bar{\mathbf{x}}}{\bar{\mathbf{y}}}, \quad \boldsymbol{\eta}\left(\mathbf{x}\right) = \frac{\mathbf{x}}{150}, \quad \boldsymbol{\lambda}\left(\mathbf{x}, \mathbf{y}\right) = \frac{\mathbf{x}}{\bar{\mathbf{y}}},$$

here, $\bar{\mathbf{x}}$ and $\bar{\mathbf{y}}$ represent the mean of the input vectors $\mathbf{x}$ and $\mathbf{y}$.

These transformations ensure that the features are appropriately scaled and standardized, which aids the learning process as the input is constrained to a consistent range. Just as for the False Detection Filter, the velocity-related features are scaled in unison with the positional features. Including features such as standard deviations and correlations provides the model with valuable information about the underlying uncertainties, which could aid in the accuracy of filtering predictions $\mathbf{P}$.

Encoder Preprocessing

The encoder preprocessing of the features in the MT3LS Module remains consistent with the original MT3v2 model. However, it does not perform the scaling operations. See Section 4.1.1 for more information on this topic.

Encoder

The encoder architecture also mirrors the original MT3v2, retaining the same number of attention heads and the dimensions of the feedforward network (FFN) layers.

Decoder Preprocessing

Similarly, the decoder preprocessing step follows the same strategy. However, instead of selecting the top-16 inputs as decoder queries, the model now selects the top-4, using the features $x_{\mathrm{uc}}, y_{\mathrm{uc}}$ as initial position estimates. Furthermore, the predictions for the mean vectors $\mathbf{M}_i$ are no longer transformed by a function ν and its inverse ν^{-1}. Additionally, adjustments are no longer performed in a pre-sigmoid space. For more details on how MT3v2 produces predictions, refer to Sections 4.1.3 and 4.1.4.

Instead, the encoder predictions, and thus the initial decoder predictions, are generated as follows:

$$\hat{\mathbf{M}} = \left[\mathbf{x}_{\text{uc}}\ \mathbf{y}_{\text{uc}}\ \mathbf{0}\ \mathbf{0}\right] \in \mathbb{R}^{|\mathcal{P}_j| \times 4}.$$

These initial guesses are then combined with the adjustments $\boldsymbol{\Delta}$ to create the encoder state mean estimates $\mathbf{M}_{\text{enc}}$, and as a result the mean state estimates for the initial decoder predictions $\mathbf{P}_{\text{dec},0}$:

$$\mathbf{M}_{\text{enc}}^{i} = \hat{\mathbf{M}}^{i} + \boldsymbol{\Delta}^{i} \odot \begin{bmatrix} \bar{\boldsymbol{\sigma}}_{\text{x}_{\text{uc}}} \\ \bar{\boldsymbol{\sigma}}_{\text{y}_{\text{uc}}} \\ \bar{\boldsymbol{\sigma}}_{\text{x}_{\text{uc}}} \\ \bar{\boldsymbol{\sigma}}_{\text{y}_{\text{uc}}} \end{bmatrix},$$

here, $\bar{\boldsymbol{\sigma}}$ refers to the mean of the vector $\boldsymbol{\sigma} \in \mathbb{R}^{|\mathcal{P}_j|}$ and $\odot$ refers to the element-wise product of the two vectors. In addition, i refers to the row corresponding to the state mean predictions for the element $i \in \{1, \ldots, |\mathcal{P}_j|\}$.

This process essentially re-scales the adjustments from the standardized space and adds the adjustments to the initial guesses $\hat{\mathbf{M}}$. The same scaling process is also applied to the adjustment predictions in each decoder layer, as described in the following section.

The encoder predictions for the subset $\mathcal{P}_j$ can be summarized as follows:

$$\mathbf{P}_{\text{enc}}^{\mathcal{P}_j} = \left\{\left(\mathbf{p}_{\text{enc}}^{i},\ \mathbf{M}_{\text{enc}}^{i},\right)\right\}_{i=1}^{|\mathcal{P}_j|}.$$

Decoder

The decoder is largely the same, however, the adjustment process has been changed. Refer to the previous section on decoder preprocessing for what has changed, as the same applies for the decoder predictions. Furthermore, the model now predicts the correlation coefficients $\rho_{\text{xy}}, \rho_{\dot{\text{x}}\dot{\text{y}}}$ together with the standard deviations.

As described in the previous section, the mean state estimates for each prediction i are generated as follows for each decoder layer l:

$$\mathbf{M}_{l}^{i} = \mathbf{M}_{l-1}^{i} + \boldsymbol{\Delta}_{l}^{i} \odot \begin{bmatrix} \bar{\boldsymbol{\sigma}}_{\text{x}_{\text{uc}}} \\ \bar{\boldsymbol{\sigma}}_{\text{y}_{\text{uc}}} \\ \bar{\boldsymbol{\sigma}}_{\text{x}_{\text{uc}}} \\ \bar{\boldsymbol{\sigma}}_{\text{y}_{\text{uc}}} \end{bmatrix}.$$

The standard deviation and correlation predictions are generated similar to MT3v2, however, as correlations are also predicted, the following is done:

$$\tilde{\boldsymbol{\Sigma}}_{l} = \text{FFN}_{\text{ReLU}}^{128 \times 128 \times 6}\left(\mathbf{E}_{\text{dec},l}\right),$$

here, $\tilde{\boldsymbol{\Sigma}}_{l}$ contain the unprocessed standard deviations for prediction i in the first 4 columns $\tilde{\boldsymbol{\Sigma}}_{l}^{i,1:4}$ and the unprocessed correlations for detection i in the last 2 columns $\tilde{\boldsymbol{\Sigma}}_{l}^{i,5:6}$. The standard deviation predictions are constructed as follows:

$$\boldsymbol{\Sigma}_l^{i,1:4} = \text{Softplus}\left(\tilde{\boldsymbol{\Sigma}}_l^{i,1:4}\right).$$

Ensuring that the correlations remain within the valid range of $[0, 1]$. And the predictions of the correlation coefficients are devised as follows:

$$\boldsymbol{\Sigma}_l^{i,5:6} = \frac{\tilde{\boldsymbol{\Sigma}}_l^{i,5:6}}{1 + |\tilde{\boldsymbol{\Sigma}}_l^{i,5:6}|}.$$

Ensuring that the correlations remain within the valid range of $[-1, 1]$.

The entries of $\boldsymbol{\Sigma}_l^i$ characterize a covariance matrix for position ($\boldsymbol{\Sigma}_{\text{pos},l}^i$) and velocity ($\boldsymbol{\Sigma}_{\text{vel},l}^i$) for prediction i at decoder layer l. The covariance matrix for position can be constructed as:

$$\boldsymbol{\Sigma}_{\text{pos},l}^i = \begin{bmatrix} \left(\boldsymbol{\Sigma}_l^{i,1}\right)^2 & \boldsymbol{\Sigma}_l^{i,5}\boldsymbol{\Sigma}_l^{i,1}\boldsymbol{\Sigma}_l^{i,2} \\ \boldsymbol{\Sigma}_l^{i,5}\boldsymbol{\Sigma}_l^{i,1}\boldsymbol{\Sigma}_l^{i,2} & \left(\boldsymbol{\Sigma}_l^{i,2}\right)^2 \end{bmatrix},$$

and the covariance matrix for velocity can be constructed similarly:

$$\boldsymbol{\Sigma}_{\text{vel},l}^i = \begin{bmatrix} \left(\boldsymbol{\Sigma}_l^{i,3}\right)^2 & \boldsymbol{\Sigma}_l^{i,6}\boldsymbol{\Sigma}_l^{i,3}\boldsymbol{\Sigma}_l^{i,4} \\ \boldsymbol{\Sigma}_l^{i,6}\boldsymbol{\Sigma}_l^{i,3}\boldsymbol{\Sigma}_l^{i,4} & \left(\boldsymbol{\Sigma}_l^{i,4}\right)^2 \end{bmatrix},$$

where:

- $\left(\boldsymbol{\Sigma}_l^{i,1}\right)^2$: Variance of x.
- $\left(\boldsymbol{\Sigma}_l^{i,2}\right)^2$: Variance of y.
- $\left(\boldsymbol{\Sigma}_l^{i,3}\right)^2$: Variance of $\dot{x}$.
- $\left(\boldsymbol{\Sigma}_l^{i,4}\right)^2$: Variance of $\dot{y}$.
- $\boldsymbol{\Sigma}_l^{i,5}$: Predicted correlation coefficient ρ_{xy} for x and y.
- $\boldsymbol{\Sigma}_l^{i,6}$: Predicted correlation coefficient $\rho_{\dot{\text{x}}\dot{\text{y}}}$ for $\dot{x}$ and $\dot{y}$.

Finally, the existence probabilities for layer l, $\mathbf{p}_l$ are generated in the same way as for the MT3v2 model.

Predictions

The predictions for the l'th decoder layer for the subset $\mathcal{P}_j$ can be summarized as follows:

$$\mathbf{P}_l^{\mathcal{P}_j} = \left\{\left(\mathbf{p}_l^i,\, \mathbf{M}_l^i,\, \boldsymbol{\Sigma}_l^i\right)\right\}_{i=1}^{n_{\text{p}}=4}.$$

After predictions for each subset $\mathcal{P}_j$ have been generated, the last layer predictions ($l = 6$) are aggregated to form a Multi-Bernoulli RFS that covers the radar's entire field of view:

$$\mathbf{P} = \bigcup_{j=1}^{k} \mathbf{P}_6^{\mathcal{P}_j}.$$

The total number of predictions n_{tp} in $\mathbf{P}$ is given by:

$$n_{\mathrm{tp}} = k \cdot n_{\mathrm{p}}.$$

Thus, the final set of predictions can be characterized as:

$$\mathbf{P} = \left\{ \left(\mathbf{p}^i, \mathbf{M}^i, \mathbf{\Sigma}^i \right) \right\}_{i=1}^{n_{\mathrm{tp}}}.$$

Given these predictions, a threshold τ_{p} could be applied on the existence probabilities to extract the most likely targets.

In Figure 5.7 the predictions of MT3LS are illustrated in the example scenario, where k could be interpreted as $k = 2$ and $n_{\mathrm{p}} = 5$. In the upper part of the figure, the existence probabilities and positional distributions of each component of the MB RFS are visualized. The middle part of the figure shows predicted targets with a red color that are above some threshold τ_{p} and those below with a blue color. In contrast to the corresponding figure for MT3v2 predictions, see Figure 4.5, the state covariance matrix for both position and velocity is full for MT3LS. This allows for the prediction of rotated covariance ellipses, and this fact is shown in the middle part of Figure 5.7. The lower part of the figure represents the actual target states at the last included scan time T in the example scenario.

Loss Functions

The loss functions of MT3LS are applied for each processed subset independently, that is, it does not act on the aggregated predictions but on the predictions for each subset $\mathcal{P}_j$.

The target states $\mathrm{O_T}$ are defined as the set of all ground-truth target states for the alive targets at the latest scan time T. The ground-truth target states associated with a subset $\mathcal{P}_j$ consist of the states of targets that have detections in $\mathcal{P}_j$ and are a part of $\mathrm{O_T}$. The set of associated target states for subset $\mathcal{P}_j$ can be expressed as:

$$\mathrm{O_T}^{\mathcal{P}_j} = \left\{ \mathbf{X}_T^i \in \mathrm{O_T} \mid \mathcal{D}_{\mathrm{target},i} \cap \mathcal{P}_j \neq \emptyset \right\},$$

where:

- $\mathbf{X}_T^i$: State of target i at time T, including positional and velocity components,
- $\mathcal{D}_{\mathrm{target},i} \subset \{1, \ldots, n_{\mathrm{d}}\}$: Set of all detections that correspond to target i,

The number of targets in the resulting set $\mathrm{O_T}^{\mathcal{P}_j}$ is denoted by $n_{\mathrm{ts}} = |\mathrm{O_T}^{\mathcal{P}_j}|$.

The loss function for the linear assignment problem differs from the original MT3v2 model. It now consists of two different loss functions depending on whether the number of alive underlying objects is equal to one or more than one. The assignment loss for *one* alive object is, that is $n_{\mathrm{ts}} = 1$, (for information on the matching/mapping process and parameters, refer to Section 4.1.5):

$$\mathcal{L}_{\mathrm{m}} \left(\sigma(i), \mathbf{p}_l^i \right) = \begin{cases} 0 & \text{if } \sigma(i) \neq 1 \\ -\log\left(\mathbf{p}_l^i\right) & \text{otherwise} \end{cases},$$

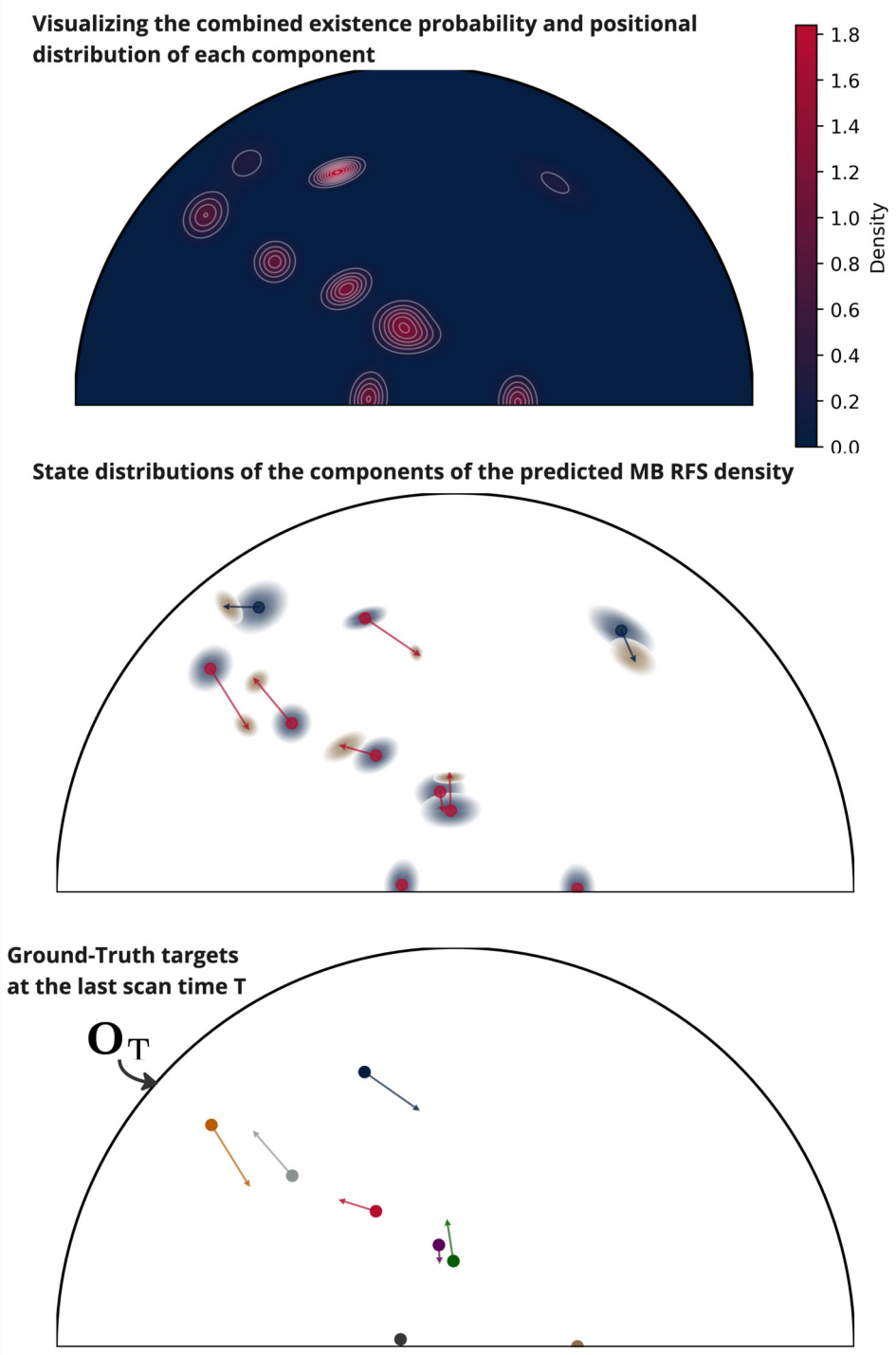


Figure 5.7: The upper part of the figure visualizes the combined existence probability and positional state densities of all components summed over the radar field of view. The middle figure visualizes the entire state distribution of each component, where the components with a existence probability below some threshold τ_p are colored blue and those above red. Blue ellipses are positional uncertainties and gold are for the velocity. The lower figure shows the ground truth target states O_T at scan time T. This is a completely synthetic scenario meant to illustrate what the output *could* look like, given the synthetic input from Figure 5.1.

here, $n_{\text{ts}} = |\mathrm{O_T}^{\mathcal{P}_j}|$ refers to the number of alive ground truth targets at the latest scan time T, in this case $n_{\text{ts}} = 1$.

This loss was preferred to the original one, as the objects are much further apart in the scenario considered here. Because of this, the positional part of the loss would dominate. Furthermore, since only one underlying object is alive, it is reasonable that the prediction with the largest existence probability should be matched.

The assignment loss for multiple alive objects separates the position- and velocity-related loss into different components. First, the different loss components are calculated for each possible target and prediction mapping. The existence probability loss component $\mathbf{L}_{\mathbf{p}} \in \mathbb{R}^{n_{\text{ts}} \times |\mathcal{P}_j|}$ is calculated as:

$$\mathbf{L}_{\mathbf{p}}^{ij} = -\log\left(\mathbf{p}_l^j\right).$$

The positional state loss component $\mathbf{L}_{\mathbf{M}_\text{p}} \in \mathbb{R}^{n_{\text{ts}} \times |\mathcal{P}_j|}$ is calculated similarly:

$$\mathbf{L}_{\mathbf{M}_\text{p}}^{ij} = -\left\|\mathbf{M}_l^{j,1:2} - \mathbf{X}_{\mathrm{T}}^{i,1:2}\right\|_1,$$

here, the superscript $1:2$ refers to the positional part of the mean state estimates and the ground truth target state. The velocity state loss component is then calculated as:

$$\mathbf{L}_{\mathbf{M}_\text{v}}^{ij} = -\left\|\mathbf{M}_l^{j,3:4} - \mathbf{X}_{\mathrm{T}}^{i,3:4}\right\|_1,$$

here, the superscript $3:4$ refers to the velocity part of the mean state estimates and the ground truth target state.

Each row i of the different loss components are then transformed by the min-max function κ:

$$\boldsymbol{\kappa}\left(\mathbf{x}\right) = \frac{\mathbf{x} - \min\left(\mathbf{x}\right)}{\max(\mathbf{x}) - \min(\mathbf{x})},$$

which results in:

$$\tilde{\mathbf{L}}_{\mathbf{p}}^{i} = \boldsymbol{\kappa}\left(\mathbf{L}_{\mathbf{p}}^{i}\right),\ \tilde{\mathbf{L}}_{\mathbf{M}_\text{p}}^{i} = \boldsymbol{\kappa}\left(\mathbf{L}_{\mathbf{M}_\text{p}}^{i}\right),\ \tilde{\mathbf{L}}_{\mathbf{M}_\text{v}}^{i} = \boldsymbol{\kappa}\left(\mathbf{L}_{\mathbf{M}_\text{v}}^{i}\right).$$

These normalized component losses are then combined to create the resulting matching loss for multiple alive ground truth targets $n_{\text{ts}} > 1$:

$$\mathcal{L}_{\text{m}}\left(\sigma(i), \tilde{\mathbf{L}}_{\mathbf{p}}^{,i}, \tilde{\mathbf{L}}_{\mathbf{M}_\text{p}}^{,i}, \tilde{\mathbf{L}}_{\mathbf{M}_\text{v}}^{i}\right) = \begin{cases} 0 & \text{if } \sigma(i) > n_{\text{ts}} \\ \tilde{\mathbf{L}}_{\mathbf{p}}^{\sigma(i),i} + \tilde{\mathbf{L}}_{\mathbf{M}_\text{p}}^{\sigma(i),i} + \tilde{\mathbf{L}}_{\mathbf{M}_\text{p}}^{\sigma(i),i} & \text{otherwise} \end{cases}.$$

Because of this, the position, velocity, and existence probability components contribute equally to the resulting loss, rather than the positional loss component dominating.

When it comes to prediction loss functions, there are some differences from the implementation of MT3v2 described in Section 4.1.5. In the case of the MT3v2 model, both of its parameters α and β were set to $\alpha = \beta = 1$ (see Section 4.1.5.1),

here we set $\beta = 1$ and $\alpha = 100$. The motivation here is the same as for the assignment loss, to increase the importance of the existence probability based loss as the positional loss is much larger. This is especially true for the encoder loss, but also for the decoder loss, as the covariance matrix has a much larger spread as a result of the increased size of the radar field of view. The larger field of view affects the measurement uncertainties, as while the polar coordinate uncertainties are constant, the corresponding Cartesian uncertainties are not. Refer to Section 2.1 for more information on measurement uncertainties.

Another change is that β_c, the parameter that controls the impact of the contrastive loss, is set to 75 instead of 4. Once again, to increase the impact of this loss.

6

Simulation Setup and Parameter Optimization

This chapter introduces the simulation environment and the rationale behind the key parameter choices for the proposed multi-target tracking pipeline.

We begin by detailing the system hardware in Section 6.1. The chapter then provides a detailed description of a radar-based surveillance scenario in Section 6.2, including the field of view (FOV), object birth, motion, and death models, as well as false detection models used to simulate a realistic environment.

Next, we examine jointly the parameter tuning of the Local Context Generation Module and the False Detection Filter in Section 6.3, which are designed to reduce computational and time complexity while maintaining high accuracy. Following this, the chapter delves into the settings and optimization of the Partitioning Algorithms in Section 6.4. Optimal parameter values are determined on the basis of clustering metrics and computational efficiency.

6.1 System Information

The hardware used consists of an NVIDIA A100 GPU and 40 Intel(R) Xeon(R) Silver 4210 CPUs, each clocked at 2.20 GHz.

6.2 A Surveillance Based Scenario

Radar Field Of View

The radar FOV was set to have the lower and upper limits $[2000, 40000]$ in meters in terms of range, and $[-1.57, 1.57]$ as lower and upper limits in radians for angle. This results in an FOV similar to a half-circle, as can be seen in Figure 6.1.

Radar Measurement Uncertanties

The uncertainties of the radar measurements were set to (see Section 2.1.1):

$$\mathbf{\Sigma}_{\mathrm{e}} = \mathrm{diag}\left(\sigma_r,\, \sigma_{\dot{r}},\, \sigma_{\varphi}\right) = \mathrm{diag}\left(15,\, 0.9778,\, 0.004\right),$$

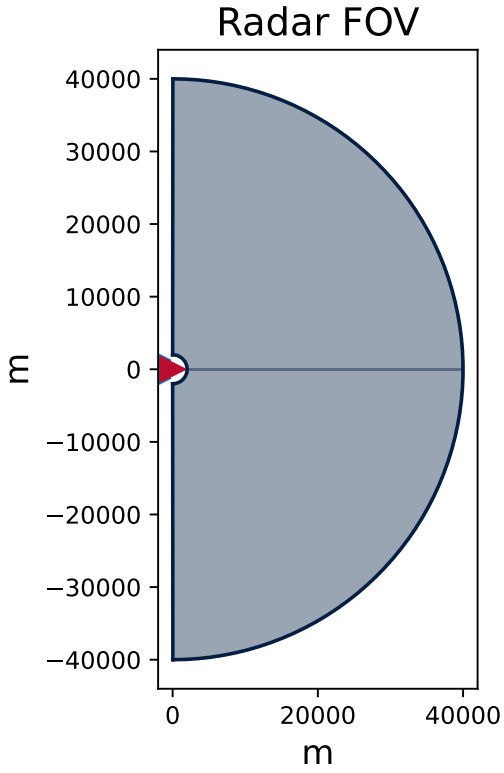


Figure 6.1: The light blue shaded area corresponds to the radar FOV. The location of the radar is indicated by the red triangle at (0,0).

here, σ_r, $\sigma_{\dot{r}}$, and σ_φ represent the uncertainties in range (meters), range rate (meters per second), and azimuth (radians), respectively.

These values provide a simplified representation of the measurement uncertainties in modern surveillance radars and were determined in consultation with the supervisors of this thesis. Given the use of a simplified radar model in this study, along with the fact that detailed information on the performance and measurement uncertainties of surveillance radars is often classified or unavailable in the public domain, these values were considered both practical and appropriate for the scope of this work.

Radar Scans and Simulation Length

The simulation produces 10 scans over the entire radar FOV. Each scan corresponds to an instantaneous sweep of the radar FOV. This means that all detections from each scan have the same scan time t. Between each scan, the radar has a downtime of 0.1 seconds. A total of 10 scans are performed, resulting in a simulation time of 1 second.

Object Birth and Death Modeling

The radar FOV is simulated to have $N_{\text{so}} \sim \text{Pois}(\lambda_o)$ starting objects, $\lambda_o = 8000$. The position and velocity of each object are modeled as follows:

$$\begin{aligned} x &\sim \mathcal{U}_{[0,40000]}, \\ y &\sim \mathcal{U}_{[-40000,40000]}, \\ \dot{x},\, \dot{y} &\sim \mathcal{N}(0, 1500), \end{aligned}$$

here, $\mathcal{U}$ is the uniform distribution with subscripts that denote the range of the distribution and normal distributions are defined by their mean μ and variances σ^2. Since the FOV is not a rectangle, the position of each object is sampled using a rejection sampling approach, where positions are drawn iteratively until they fall within the FOV. The model thus assumes that the objects are uniformly spread across the FOV. In addition, the maximum range rate $\dot{r}$ is set to 150 m/s. We use rejection sampling until each object has a velocity with a corresponding range rate $\dot{r}$ less than 150 m/s. The distribution of the speed of the objects is thus a truncated (zero density after 150 m/s) Rayleigh with $\sigma = \sqrt{1500}$ and is illustrated, approximately, in Figure 6.2.

This range of speeds captures all types of birds and the speeds of most, if not all, commercially available drones, as well as the speed of some slower airplanes. Because of this, it captures a range of speeds that is becoming of more interest during modern warfare as drones are incorporated on a large scale, as witnessed from the Russia-Ukraine war, discussed in the following Reuters article [42].

At each scan, each object in the FOV has the probability $p_{\text{d}} = 0.05$ of dying or

being removed, and objects that have moved outside of the FOV are also removed. Furthermore, each object has a probability $p_{\text{meas}} = 0.95$ of being measured (detected) at each scan. The measurement model is described in Section 2.1.1.

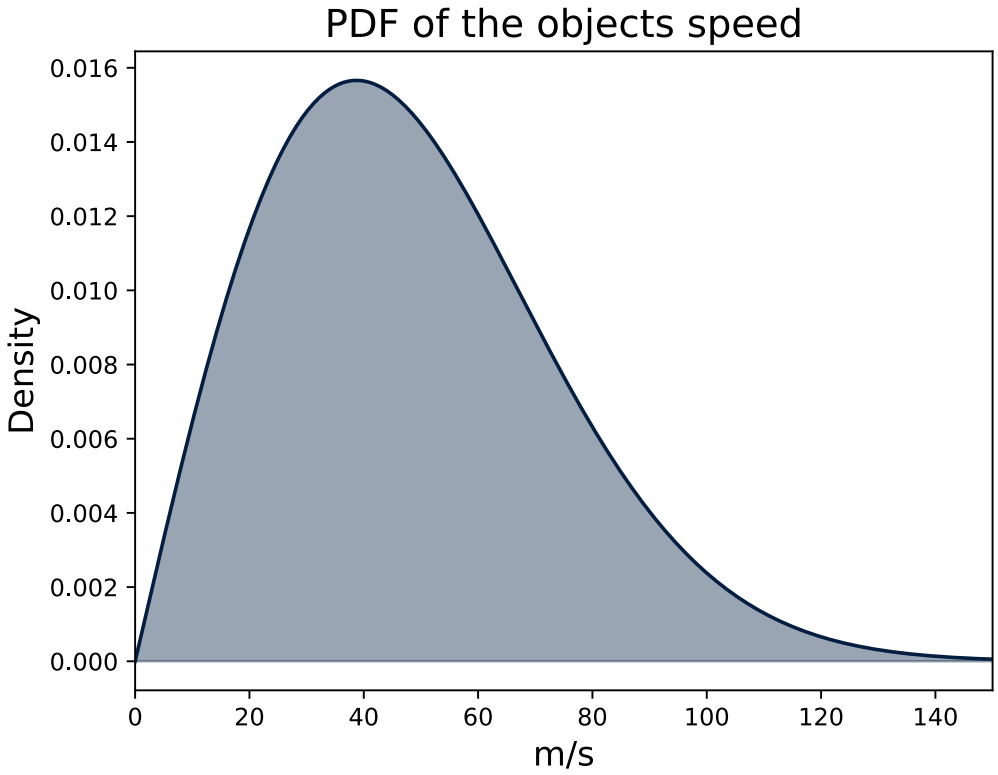


Figure 6.2: PDF of a Rayleigh distribution with $\sigma = \sqrt{1500}$ representing the objects speed. Most objects thus exhibit speeds between 0 and 120 m/s.

Object Movement Model

The movements of the objects are modeled as follows:

$$\begin{bmatrix} x_{i+1} \\ y_{i+1} \\ \dot{x}_{i+1} \\ \dot{y}_{i+1} \end{bmatrix} = \begin{bmatrix} 1 & 0 & \Delta\mathrm{T} & 0 \\ 0 & 1 & 0 & \Delta\mathrm{T} \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_i \\ y_i \\ \dot{x}_i \\ \dot{y}_i \end{bmatrix} + \mathbf{q}, \tag{6.1}$$

here, $\Delta\mathrm{T}$ is the time between scans (0.1 seconds), the subscript $_i$ denotes the state of the object at scan i, and $\mathbf{q}$ represents the process noise, which is modeled as:

$$\mathbf{q} \sim \mathcal{N}\left(\begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}, \sigma_a^2 \begin{bmatrix} \mathbf{I}\Delta\mathrm{T}^3/3 & \mathbf{I}\Delta\mathrm{T}^2/2 \\ \mathbf{I}\Delta\mathrm{T}^2/2 & \mathbf{I}\Delta\mathrm{T} \end{bmatrix} \right),$$

here $\mathbf{I}$ is an identity matrix with two rows and columns and $\sigma_a^2 = 0.2$ is the process noise variance. This motion model makes the velocity of the objects almost constant, with some acceleration.

False Detection Modeling

When it comes to false detection measurements, they are modeled in a similar way to the real objects. At each scan, the number of false detections drawn is modeled as $N_\mathrm{c} \sim \mathrm{Pois}(\lambda_c)$, with $\lambda_c = 5500$. False detection measurements are modeled as follows:

$$
\begin{aligned}
x &\sim \mathcal{U}_{[0,40000]}, \\
y &\sim \mathcal{U}_{[-40000,40000]}, \\
r &= \sqrt{x^2 + y^2}, \\
\dot{r} &\sim \mathcal{U}_{[-150,150]}, \\
\varphi &= \mathrm{atan2}(y, x).
\end{aligned}
$$

As with object births, rejection sampling is used to generate measurements within the FOV.

Simulation Summary

During the entire simulation, an approximate average of the total number of detections belonging to real objects is:

$$
n_\mathrm{om} \approx \sum_{i=0}^{10} (1 - p_\mathrm{d})^i \lambda_o p_\mathrm{meas} \approx 60992,
$$

while the mean number of false detections during the entire simulation is $10\lambda_c = 55000$, resulting in a total of around 116,000 detections for the entire simulation. The number of alive objects at the last time step of the simulation is roughly 5042.

6.3 Local Context Generation Module and False Detection Filter Parameters

This section evaluates the parameter settings for the Grouping Algorithm and the False Detection Filter (FDF) to optimize performance and computational efficiency.

The parameter d_max, representing the maximum number of detections per local context, of the Local Context Generation Module (LCG) was set to 100. The setting $d_\mathrm{max} = 100$ was motivated in part by being a reasonable batch size for processing by the False Detection Filter and partly by limiting the parameter space to optimize over. The remaining parameter for LCG is therefore s_f, determining the number of local contexts (subsets $\mathcal{G}_i$) to generate as a fraction of the total number of detections (see Section 5.3.3). For the FDF, the threshold τ_fdf (see (5.2)) needs to be

determined. Detections below this threshold are considered false detections.

The parameters are optimized with the following objectives:

Maximize: Binary classification metrics of the FDF,
Minimize: Computational time of the FDF.

Here, binary classification metrics refer to the F1 score, accuracy, and balanced accuracy, which are detailed in Section 3.4. We use these metrics together, as either one on its own can be misleading. All these metrics have maximum values at 1, and the larger the value of the metrics, the better.

Figure 6.3 shows that a threshold value of $\tau_{\text{fdf}} \approx 0.5$ consistently yields optimal results across different fractions s_{f}. Performance drops significantly only at $s_{\text{f}} = 0.025$, indicating that too few local contexts affect the accuracy of classification.

In Figure 6.4 the performance of the False Detection Filter at $\tau_{\text{fdf}} = 0.5$ is summarized. The execution times in the lower panel indicate a significant reduction when using $s_{\text{f}} = 0.05$ compared to $s_{\text{f}} = 0.1$, with only a minor performance loss in terms of classification metrics in the upper panel. The benefits in execution time at $s_{\text{f}} = 0.025$ compared to $s_{\text{f}} = 0.05$ are not as great, while the difference in classification performance has increased.

The pseudo-optimal parameter values of the False Detection Filter and the Local Context Generation Module, identified through the above analysis, are summarized below:

Local Context Generation Module

$$\begin{aligned} d_{\max} &= 100, \\ s_{\text{f}} &= 0.05. \end{aligned} \tag{6.2}$$

False Detection Filter

$$\tau_{\text{fdf}} = 0.5. \tag{6.3}$$

Processing local contexts in parallel in the FDF substantially reduces memory usage compared to processing all detections jointly. At $s_{\text{f}} = 0.05$, with 8 attention heads, and 32-bit floating point numbers, the memory requirement for attention calculations drops to 1.6 GB from 320 GB if considering processing around 10^5 detections simultaneously. Although processing local contexts as separate inputs decreases the efficiency of GPU operations, the overall reduction in memory complexity justifies the trade-off, particularly in large-scale tracking scenarios.
In Appendix B.2 the training procedure and inference optimizations for the False Detection Filter are described.

6.4 Partitioning Algorithms Settings

This section explores the parameter settings for the different implementations of the connection strength function $\omega(i, j)$ outlined in Sections 5.3.5.5, 5.3.5.6, 5.3.5.7,

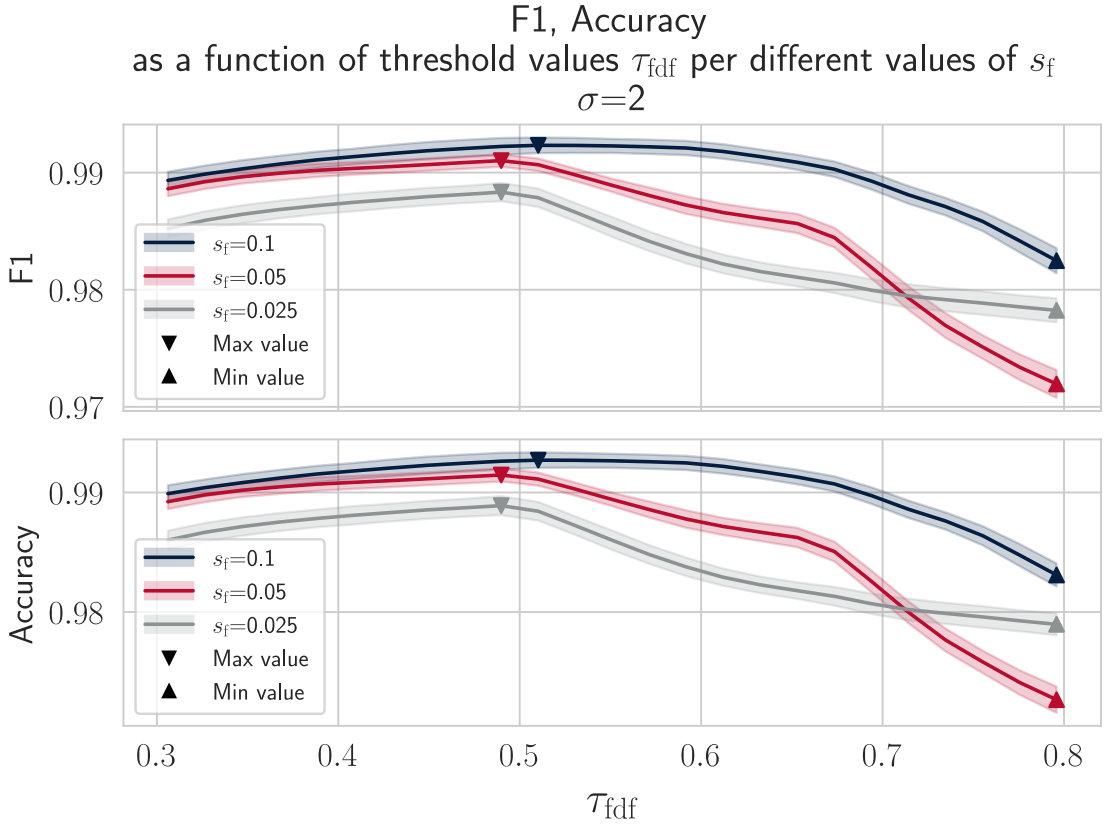


Figure 6.3: Effect of threshold value τ_{fdf} and the fraction parameter s_{f} on the binary classification metrics F1 and Accuracy for the FDF, see Section 3.4. The slightly see-through colors encompass $\sigma = 2$ standard deviations based on $n_{\mathrm{runs}} = 20$ separate simulations while the lines represent the mean.

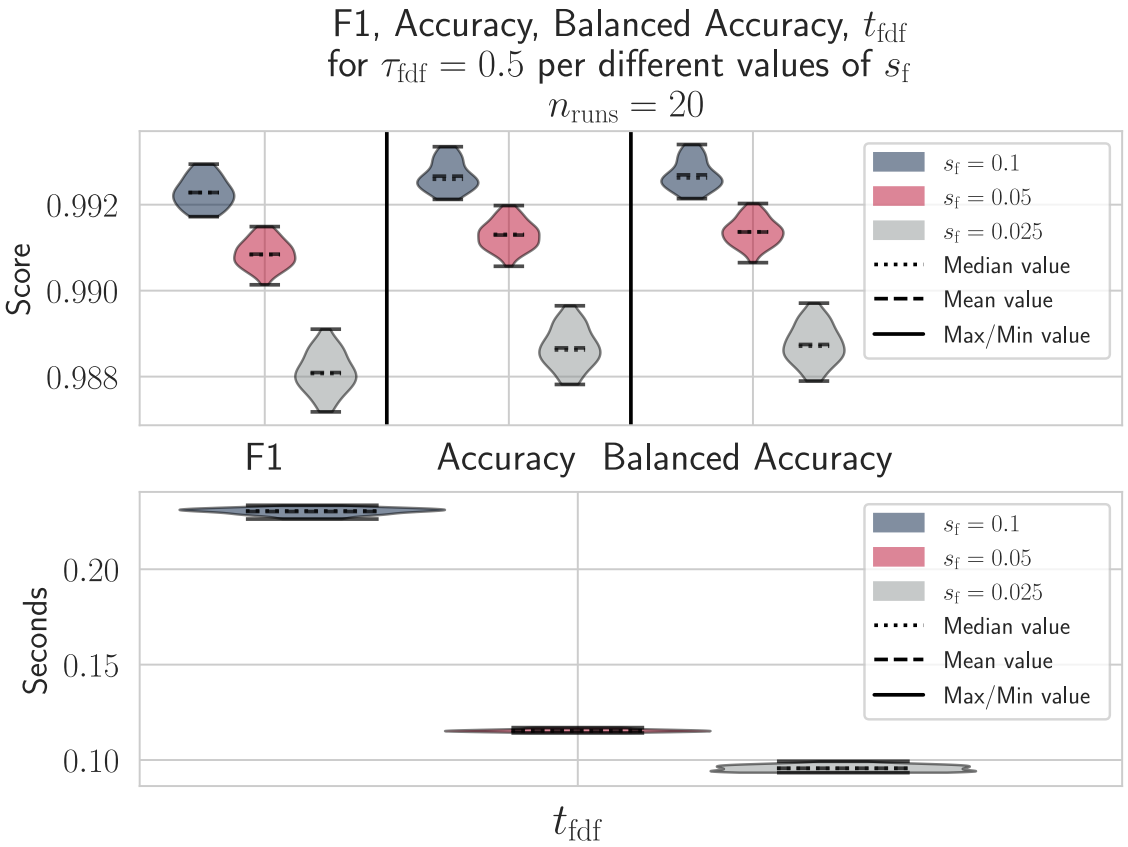


Figure 6.4: Performance summary at $\tau_{\text{fdf}} = 0.5$ with Balanced Accuracy added for robustness and a lower panel with execution time across settings. The violin plots display the distribution of scores over $n_{\text{runs}} = 20$ simulations, where the width of the violin reflects the density of results at each value.

and 5.3.5.8. The variations in the connection strength function lead to four distinct partitioning algorithms.

The parameters of the Local Context Generation Module and the False Detection Filter were fixed to their pseudo-optimal values (see (6.2) and (6.3), respectively), as the partitioning algorithm implementations depend on them.

For easier reference, the different implementations are denoted as follows, together with the parameters upon which they depend:

- **Positional Uncertainty**: **E** (described in Section 5.3.5.5)
  - $\tau_{\text{md,PU}}$
- **Normal Approximations I**: **ST** (described in Section 5.3.5.6)
  - $\sigma_{a_i}^2$
  - $\sigma_{\omega_i}^2$
  - $\tau_{\text{md,NAI}}$
- **Normal Approximations II**: **JP** (described in Section 5.3.5.7)
  - $\sigma_{a_i}^2$
  - $\sigma_{\omega_i}^2$
  - $\tau_{\text{da,NAII}}$
- **Attention-Scores**: $\tilde{\mathbf{A}}$ (described in Section 5.3.5.8)
  - $\tau_{\text{da,AS}}$

For **ST** and **JP**, the values of the parameters $\sigma_{a_i}^2$ (see (5.4)) and $\sigma_{\omega_i}^2$ (see (5.5)) were set before to reduce the scope of parameter tuning. Initial empirical testing led to $\sigma_{\omega_i}^2 = 15^2$. The value of $\sigma_{a_i}^2$ was aligned with the setting in Section 6.2, i.e., it mirrors the true underlying value of the process noise. These parameters should primarily be set to reasonable values rather than undergoing extensive optimization.

The parameter values that remain to be decided upon are thus the thresholds that govern the detection association weights. These are divided into two types: τ_{md} (**E** and **ST**) and τ_{da} (**JP** and $\tilde{\mathbf{A}}$).

Threshold Settings

The threshold parameter values are optimized with the following objectives:

Maximize:	Clustering metrics of the partitioning algorithms
Minimize:	Computational time of the partitioning algorithms

Here, clustering metrics refer to the Adjusted Rand Index (ARI), Adjusted Mutual Information (AMI), refer to Section 3.5.1 for more information. Both metrics are used, as either one on their own could be misleading, depending on the structure of the evaluated partition. These metrics have maximum values at 1, and the higher the value of the metrics, the better.

In Figure 6.5 it can be observed that both **ST** and **E** reach their peak performance at similar threshold values. However, the method **E** exhibits slightly higher ARI and AMI values. Furthermore, **E** is consistently faster than **ST** in all threshold choices.

The increase in execution time at higher thresholds is due to the association graph edge matrix $\mathbf{G}$ (see Section 5.3.5 for more information on $\mathbf{G}$) becoming denser, as more non-zero entries are introduced.

Based on the results, the pseudo-optimal threshold values are $\tau_{\text{md}} \approx 1.8$ for **ST** and $\tau_{\text{md}} \approx 2.8$ for **E**. These thresholds balance performance (in terms of ARI and AMI) and computational efficiency. Although lowering the threshold could improve execution time, this could result in reduced clustering performance.

Figure 6.6 illustrates the effect of τ_{da} on the **JP** and $\tilde{\mathbf{A}}$ methods. Unlike the previous figure, the effect on execution time t_{c} is less pronounced, except at very low thresholds (near $\tau_{\text{da}} \approx 0$). This is due to a substantial increase in the density of the association graph edge matrix $\mathbf{G}$ (see Section 5.3.5 for more information on $\mathbf{G}$), as lower thresholds allow more non-zero entries in the matrix.

In terms of clustering performance, both **JP** and $\tilde{\mathbf{A}}$ peak at similar values of τ_{da}, with $\tilde{\mathbf{A}}$ tending to perform better at lower thresholds. The optimal threshold for both methods, balancing ARI and AMI, is approximately $\tau_{\text{da}} \approx 0.05$.

6.4.1 Summary and Algorithm Comparison

In this section, the parameter choices for the entire pipeline are summarized for the different partitioning algorithms. The partitioning algorithms are then compared in terms of the clustering metrics Adjusted Rand Index (ARI) Adjusted Mutual Information (AMI) and Q_{pg} (for more information on these metrics see Section 3.5.1) and execution time t_{c} at these parameter values. All these metrics have a maximum value of 1 and the larger the value of the metrics, the better.

Parameter Settings							
Method	$d_{\max}$	s_{f}	τ_{fdf}	τ_{md}	τ_{da}	$\sigma^2_{\text{a}_\text{i}}$	$\sigma^2_{\omega_i}$
$\tilde{\mathbf{A}}$	100	0.05	0.5	n/a	0.05	n/a	n/a
E	100	0.05	0.5	2.8	n/a	n/a	n/a
JP	100	0.05	0.5	n/a	0.05	0.2	15^2
ST	100	0.05	0.5	1.8	n/a	0.2	15^2

Table 6.1: Summary of the pseudo-optimal parameter settings of the pipeline for different partitioning algorithms.

Table 6.1 summarizes the pseudo-optimal parameter values for the different partitioning algorithms and the preceding pipeline (Local Context Generation Module and False Detection Filter).

In Figures 6.7 and 6.8 the performance of the different implementations of the partitioning algorithm is displayed in terms of clustering metrics and execution time through violin plots. The higher the clustering metric score, the better for all included metrics (see Section 3.5.1), and the lower the execution time t_{c}, the better. In

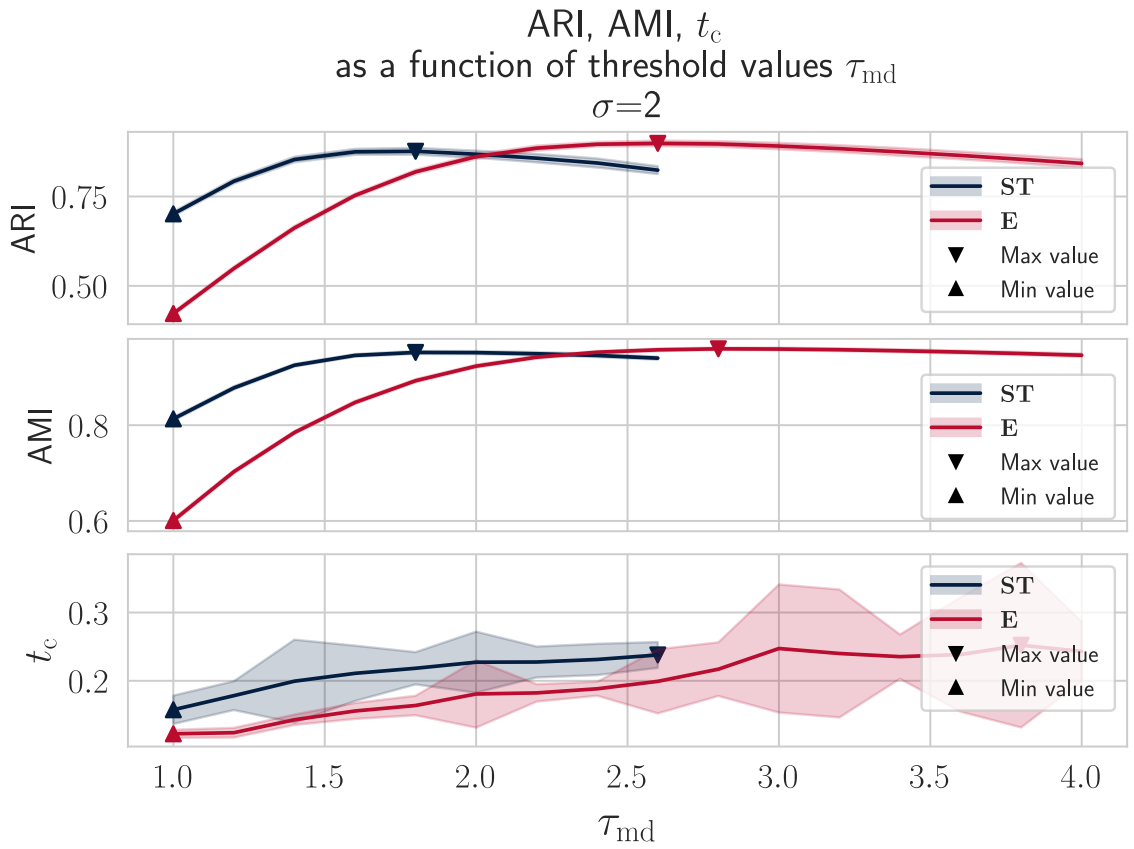


Figure 6.5: This figure illustrates the cluster metrics Adjusted Rand Index (ARI) and Adjusted Mutual Information (AMI), along with the execution time t_c, as a function of the threshold value τ_{md} for the association methods **ST** and **E**. The transparent bands represent two standard deviations $\sigma = 2$ at each value of τ_{md} based on 20 simulations, while the lines show the mean.

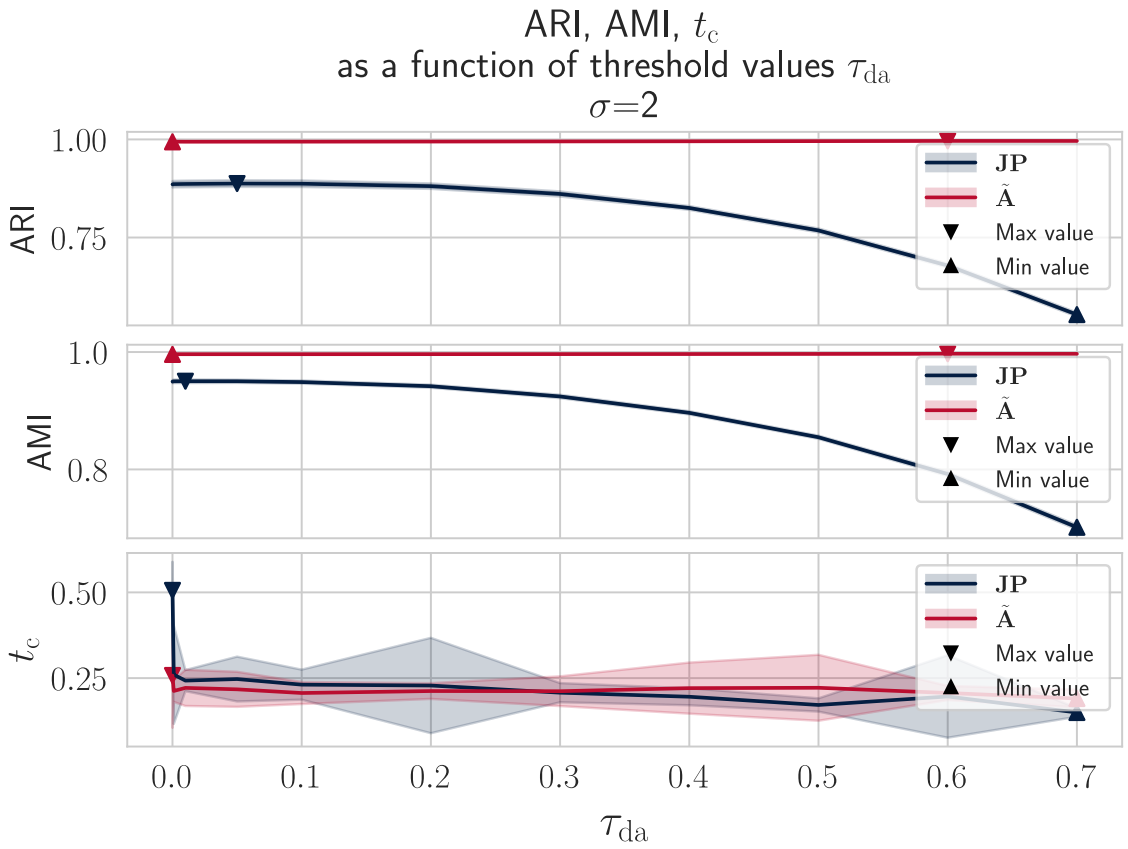


Figure 6.6: This figure shows the cluster metrics Adjusted Rand Index (ARI) and Adjusted Mutual Information (AMI), along with the execution time t_c, as a function of the threshold value τ_{da} for the association methods **JP** and $\tilde{\mathbf{A}}$. The transparent bands represent two standard deviations $\sigma = 2$ at each value of τ_{da} based on 20 simulations, with the lines indicating the mean.

both figures, the partition algorithm $\tilde{\mathbf{A}}$ has its own y-axis located to the right. For ease of comparison, a symbol Δ has been placed above all other partitioning algorithms in Figures 6.7 and 6.8. The value of Δ represents the difference in the median values between the method under it and the method $\tilde{\mathbf{A}}$. It should be interpreted as follows:

- **ARI, AMI,** Q_{pg}: The *higher* the value of Δ, the *better* $\tilde{\mathbf{A}}$ is compared to the method beneath.
- t_c: The *lower* the value of Δ, the *better* $\tilde{\mathbf{A}}$ is compared to the method beneath.

In Figure 6.7, the performance of the partitioning algorithms in terms of the ARI and AMI metrics is illustrated. The attention-based method, $\tilde{\mathbf{A}}$, significantly outperforms the other methods in both ARI and AMI metrics, with the gap being most pronounced for ARI. The positional uncertainty method, **E**, comes in second, showing a noticeable advantage over the remaining methods in terms of AMI.

In Figure 6.8 the difference between the different methods and $\tilde{\mathbf{A}}$, in terms of the Q_{pg} metric, is even more pronounced. $\tilde{\mathbf{A}}$ produces 18% more perfect subsets (Q_{pg} represents fractions of perfect subsets in the predicted partition; see Section 3.5.1 for more information) than **E**, the next best-performing method. Additionally, $\tilde{\mathbf{A}}$ is faster, probably due to not needing to explicitly create localized association graphs, instead using $\tilde{\mathbf{A}}$ directly from the False Detection Filter (see Sections 5.3.5 and 5.3.5.8 for more information on the partitioning module).

Despite these advantages, the differences in execution time among the methods are not very significant with overlapping distributions. The long tails in the execution time distributions are partly due to varying access to CPU resources during testing, which introduces variability in multi-processing capabilities.

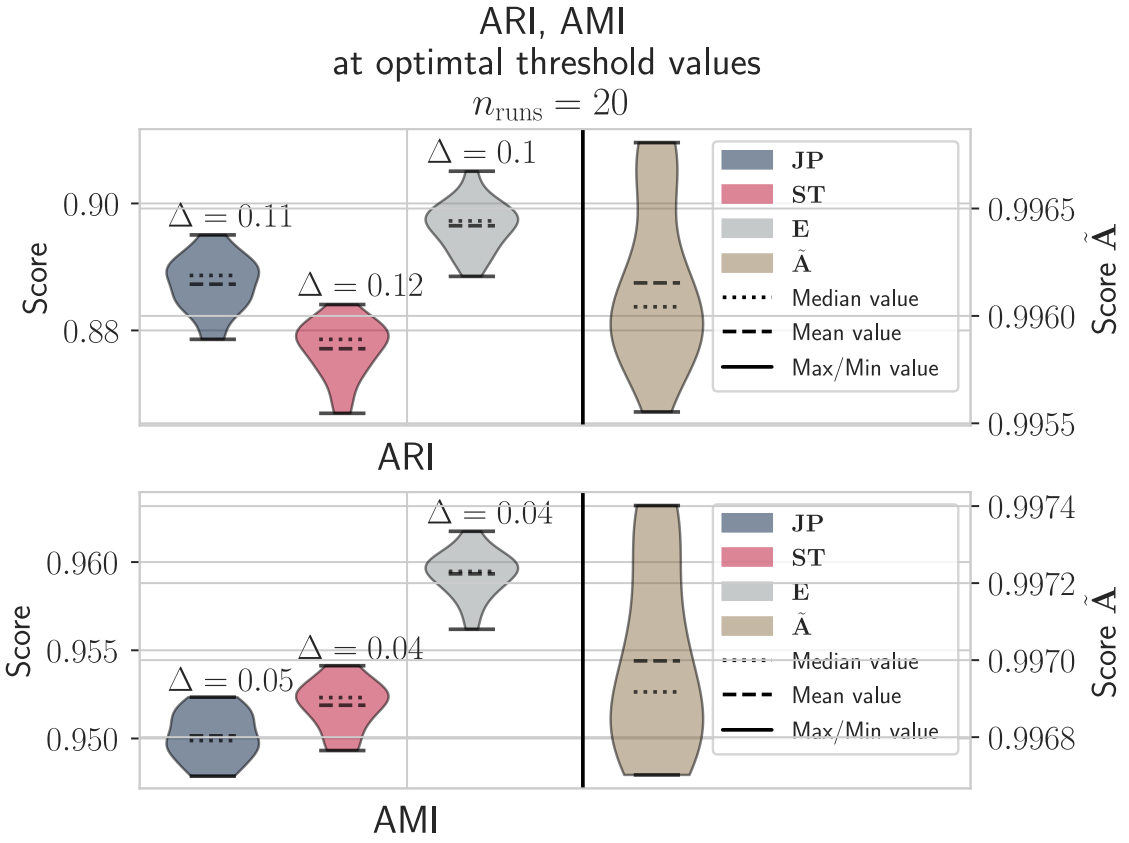


Figure 6.7: This figure using the cluster metrics ARI and AMI. The higher the value of ARI and AMI the better. Due to its significantly different performance, $\tilde{\mathbf{A}}$ has its own axis for the score (furthest to the right in both panels). n_{runs} is the number of simulations that were run to produce the results.

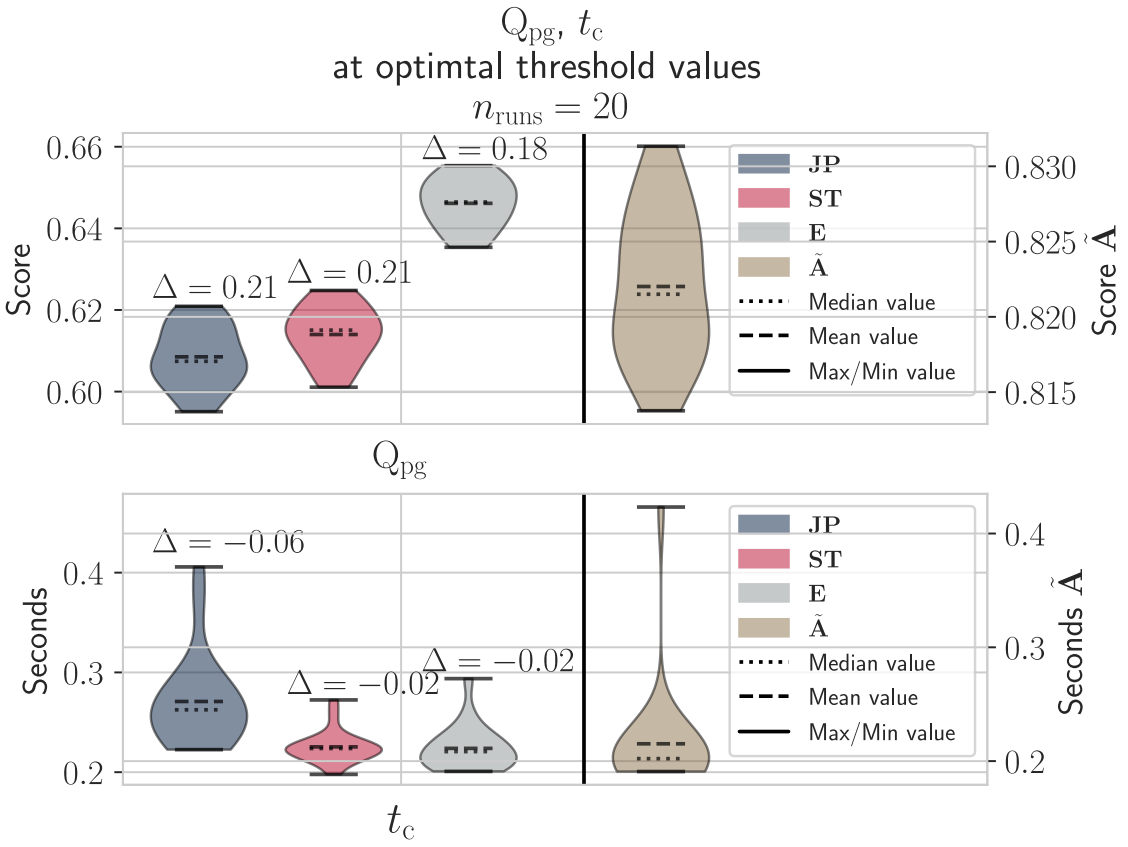


Figure 6.8: This figure compares all the data association methods based on the cluster metric Q_{pg} and execution time t_c. The higher the value of Q_{pg} the better, and the lower the value of t_c the better. Similar to the previous figure, $\tilde{\mathbf{A}}$ has its own axis to the right, for both score and time (in seconds). n_{runs} is the number of simulations that were run to produce the results.

7
Evaluation and Discussion

In this chapter, a detailed evaluation of the proposed multi-target tracking pipeline is presented in Section 7.1. Following this, the performance of MT3LS is evaluated in Section 7.2. The chapter concludes with a discussion of various topics raised in the course of this thesis in Section 7.3.

The ground truth objects that are used to produce the results for all sections, except Section 7.2.1, are the states of alive objects at the last time step, referred to as O_T in previous chapters (see Section 6.2 for information on the modeling of targets).

7.1 Pipeline Evaluation

This section begins by outlining the different pipeline implementations and naming conventions, along with the metrics used to evaluate the multi-target tracking (MTT) performance in Section 7.1.1.

Section 7.1.2 evaluates different multi-target tracking pipeline implementations on the task outlined in Section 6.2. The impact of the False Detection Filter on performance is evaluated and discussed in Section 7.1.3.

In Section 7.1.4 a more complex task is introduced, for which the False Detection Filter is retrained, and the performance of the different multi-target tracking pipeline implementations is discussed in Section 7.1.4.1. This section also evaluates the adaptability of the different pipelines to the different scenarios in Section 7.1.4.2.

See Appendix B for implementation and training details for MT3LS and the False Detection Filter.

7.1.1 Pipeline Implementations and Metrics

The multi-target tracking (MTT) pipeline outlined in Section 7.1 consists of several modular components. The Partitioning Module, described in detail in Section 5.3.5.3, contains four different implementations of the association strength function, $\omega(i, j)$. In the following, we will define four different implementations of the multi-target tracking pipeline, which differ only by the choice of association strength function $\omega(i, j)$:

- **Elliptical Proximity Pipeline (EPP)**: Utilizes positional uncertainty and

Mahalanobis distance to connect detections. The association strength function $\omega(i, j)$ is described in Section 5.3.5.5. For parameters used in the resulting pipeline, refer to the method **E** in Table 6.1.

- **Statistical Range-Rate Pipeline (SRRP)**: Combines statistical tests based on range-rate and angular differences to determine connection strength. The association strength function $\omega(i, j)$ is described in Section 5.3.5.6. For parameters used in the resulting pipeline, refer to the method **ST** in Table 6.1.
- **Probabilistic Normalization Pipeline (PNP)**: Extends SRRP by using the normalized probability densities of range-rate and angular differences to define a smooth connection strength between detections. The association strength function $\omega(i, j)$ is described in Section 5.3.5.7. For parameters used in the resulting pipeline, refer to the method **JP** in Table 6.1.
- **Attention Score Pipeline (ASP)**: Leverages attention scores from the False Detection Filter module to compute connection strengths. The association strength function $\omega(i, j)$ is described in Section 5.3.5.8. For parameters used in the resulting pipeline, refer to the method $\tilde{\mathbf{A}}$ in Table 6.1.

Prediction-Target Matching

In order to produce evaluation metrics, the predictions $\mathbf{P}$ of the MT3LS module need to be matched with the alive target states at the latest scan time $\mathrm{O_T}$.

The procedure begins by filtering out all predictions i with an existence probability $\mathbf{p}^i$ less than 0.5. The remaining predictions are then assigned to targets in $\mathrm{O_T}$ that have detections in the corresponding prediction i's subset $\mathcal{P}_j$, refer to Section 5.3.6 for more information on the MT3LS module. For each target that has more than one prediction mapped to it, the matched prediction is decided by weighing the Euclidean distance in position and velocity equally. This is done by ranking the distances from closest to furthest, for both velocity and position. The prediction with the lowest combined rank that has not been matched with any other target is matched with the target.

We denote $\mathcal{V}$ as the set of mappings (i, j) between predictions i and targets j. Furthermore, we define the set of targets contained in $\mathrm{O_T}$ as $\mathcal{O}$ and the set of predictions with existence probability $\mathbf{p}^i \geq 0.5$ as $\mathcal{F}$. Given these definitions, we can define the metrics used to evaluate the performance of the different pipeline implementations.

Evaluation Metrics

In the following definitions, $\mu_{x^i}, \mu_{y^i}, \mu_{\dot{x}^i}, \mu_{\dot{y}^i}$ refers to the elements of the mean state vector $\mathbf{M}^i$ contained in the prediction $\mathbf{P}_i$:

$$\mathbf{M}^i = \begin{bmatrix} \mu_{x^i} \\ \mu_{y^i} \\ \mu_{\dot{x}^i} \\ \mu_{\dot{y}^i} \end{bmatrix},$$

and $x_{\mathrm{T}}^j, y_{\mathrm{T}}^j, \dot{x}_{\mathrm{T}}^j, \dot{y}_{\mathrm{T}}^j$ refers to the elements of the target j's state vector $\mathbf{X}_{\mathrm{T}}^j$:

$$\mathbf{X}_{\mathrm{T}}^{j} = \begin{bmatrix} x_{\mathrm{T}}^{j} \\ y_{\mathrm{T}}^{j} \\ \dot{x}_{\mathrm{T}}^{j} \\ \dot{y}_{\mathrm{T}}^{j} \end{bmatrix}.$$

- **Mean Positional Error:**
  The positional accuracy of a prediction $\mathbf{P}_i$ matched to a target $\mathbf{X}_{\mathrm{T}}^{j}$ is given by the Euclidean distance between their positions:

  $$d_{\mathrm{pos}}(i,j) = \sqrt{\left(\mu_{x^i} - x_{\mathrm{T}}^{j}\right)^2 + \left(\mu_{y^i} - y_{\mathrm{T}}^{j}\right)^2}.$$

  The mean positional error over all matched predictions is:

  $$\text{Mean Positional Error} = \frac{1}{|\mathcal{V}|} \sum_{(i,j)\in\mathcal{V}} d_{\mathrm{pos}}(i,j).$$

- **Mean Velocity Error:**
  The velocity accuracy is computed as the Euclidean distance between the velocity components of $\mathbf{P}_i$ and $\mathbf{X}_{\mathrm{T}}^{j}$:

  $$d_{\mathrm{vel}}(i,j) = \sqrt{\left(\mu_{\dot{x}^i} - \dot{x}_{\mathrm{T}}^{j}\right)^2 + \left(\mu_{\dot{y}^i} - \dot{y}_{\mathrm{T}}^{j}\right)^2}.$$

  The mean velocity error over all matched predictions is:

  $$\text{Mean Velocity Error} = \frac{1}{|\mathcal{V}|} \sum_{(i,j)\in\mathcal{V}} d_{\mathrm{vel}}(i,j).$$

- **Missed Target Rate (MTR):**
  The missed target rate is the proportion of alive targets $\mathbf{X}_{\mathrm{T}}^{j} \in \mathrm{O}_{\mathrm{T}}$ that are not matched to any prediction:

  $$\mathrm{MTR} = \frac{|\mathcal{O} \setminus \{j \mid (i,j) \in \mathcal{V}\}|}{|\mathcal{O}|}.$$

  Here, $\mathcal{O} \setminus \{j \mid (i,j) \in \mathcal{V}\}$ represents the set of targets in O_{T} that were not matched to any prediction.
- **False Prediction Rate (FPR):**
  The false prediction rate is the proportion of predictions $\mathbf{P}_i$ with a probability of existence above 0.5 that are not matched to any target:

  $$\mathrm{FPR} = \frac{|\mathcal{F} \setminus \{i \mid (i,j) \in \mathcal{V}\}|}{|\mathcal{F}|},$$

  here, $\mathcal{F} \setminus \{i \mid (i,j) \in \mathcal{V}\}$ represents the set of predictions with a probability of existence above 0.5 that were not matched to any target.

7.1.2 Original Scenario

Figure 7.1 provides a comparative analysis of the four multi-target tracking pipelines - ASP, EPP, PNP, and SRRP - under the original task scenario. Evaluation metrics include missed target rate, false prediction rate, mean position error, and mean velocity error. The approximate number of missed targets and false predictions for each pipeline is shown in Table 7.1.

The Attention Score Pipeline (ASP) consistently outperforms the other pipelines across all metrics. It achieves the lowest missed target rate (about 10 times lower) and false prediction rate (between 2.5-10 times lower), highlighting its ability to reliably associate detections originating from the same targets while minimizing false predictions. Additionally, ASP reduces positional errors on average by more than 4 meters and velocity errors by more than 1 m/s, reflecting a performance gain of 8% and 4%, respectively. As all pipelines employ the same MT3LS model, the difference is due to more accurate partitioning.

In contrast, the Statistical Range-Rate Pipeline (SRRP) exhibits the highest missed target rate, suggesting that it struggles to cluster target detections with each other.

The Probabilistic Normalization Pipeline (PNP) has the highest false prediction rate. This could be due to fragmentations of detections originating from the same target into different clusters, or an effect of more false detections being clustered together, or a combination of both.

The Elliptical Proximity Pipeline (EPP) performs moderately well, achieving the second lowest false prediction rate and the third lowest missed detection rate.

In terms of execution time, it was observed that the total pipeline procedure took approximately 0.5 seconds. The False Detection Filter took approximately 0.1 seconds, as can be seen in Section 6.3 and the partitioning algorithm took around 0.22 seconds, refer to Section 6.4. The prediction time for MT3LS was around 0.15 seconds.

7.1.3 Excluding the False Detection Filter

In this subsection, the impact of deactivating the False Detection Filter (FDF) (by setting τ_{fdf} to a negative value) on the performance of multi-target tracking pipelines is analyzed. For the ASP pipeline, the FDF attention scores are still used. This setup evaluates the robustness of each pipeline under increased noise and dependence on the FDF. The results are summarized in Figure 7.2 together with the results of the previous section as a contrast.

When the FDF is removed, the performance of ASP remains largely unchanged, likely because the attention scores generated by the FDF provide information closely aligned with the true target decisions (see Section 5.3.4). However, the performance of the other pipelines significantly deteriorates, as shown by a 10-30 time increase in

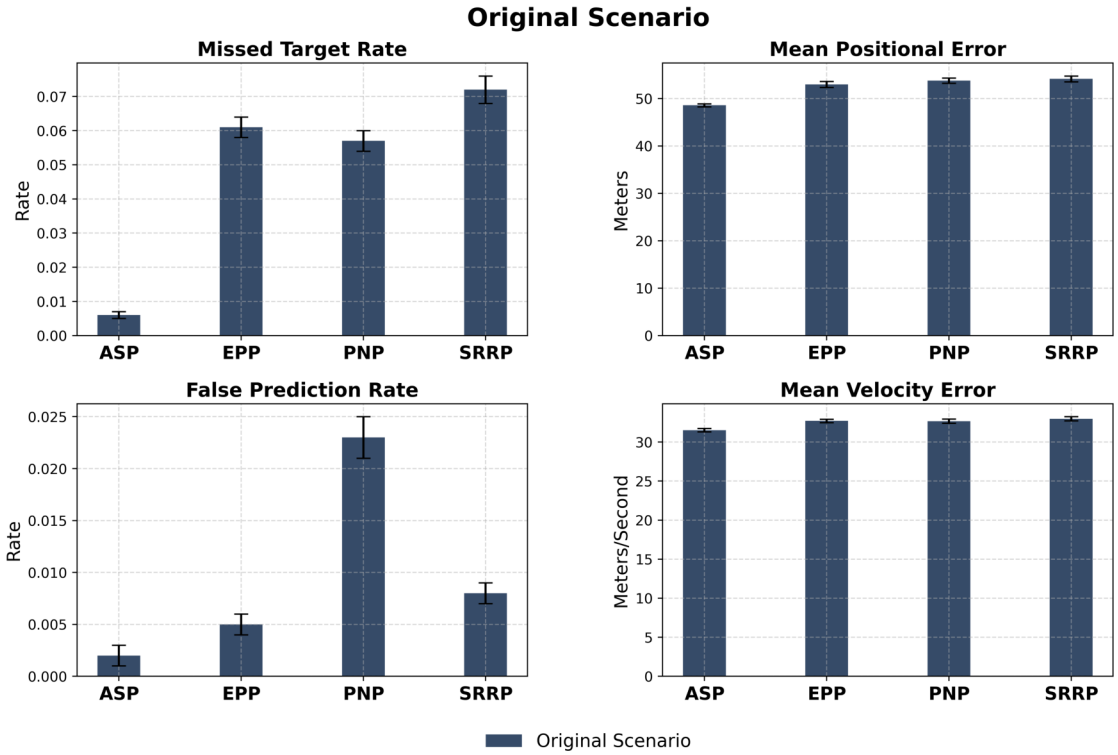


Figure 7.1: Performance of the multi-target tracking pipelines on the original scenario, for information on the scenario see Section 6.2, evaluated using Missed Target Rate, False Prediction Rate, Mean Positional Error, and Mean Velocity Error. The bars represents the mean across 20 simulations while the whiskers represent one standard deviation. For metric definitions, refer to Section 7.1.1.

Average Number of Missed Targets and False Predictions			
		$\approx \boldsymbol{\mu}_d$	$\approx \boldsymbol{\sigma}_d$
Missed Targets	**ASP**	30	5
	EPP	308	15
	PNP	287	15
	SRRP	363	15
False Predictions	**ASP**	10	5
	EPP	25	5
	PNP	116	5
	SRRP	40	5

Table 7.1: Approximated average number of missed targets and false predictions per simulation for different multi-target pipelines. Ratios (both mean and standard deviation) were multiplied by the average number of alive objects at the end of the simulation (see Section 6.2). This scaling slightly exaggerates the number of false predictions.

false prediction rates compared to when the FDF is active. This suggests that without the FDF, non-object (false detection) clusters are more frequently identified, and a higher likelihood of detections from the same target being fragmented into different clusters. Although missed target rates for these pipelines decrease slightly, this improvement is outweighed by the substantial increase in false predictions.

In conclusion, the absence of the FDF significantly affects all pipelines except ASP, highlighting the importance of the FDF in ensuring robust tracking performance. While retraining the MT3LS model on scenarios with higher noise might improve results, it is unlikely to close the performance gap entirely.

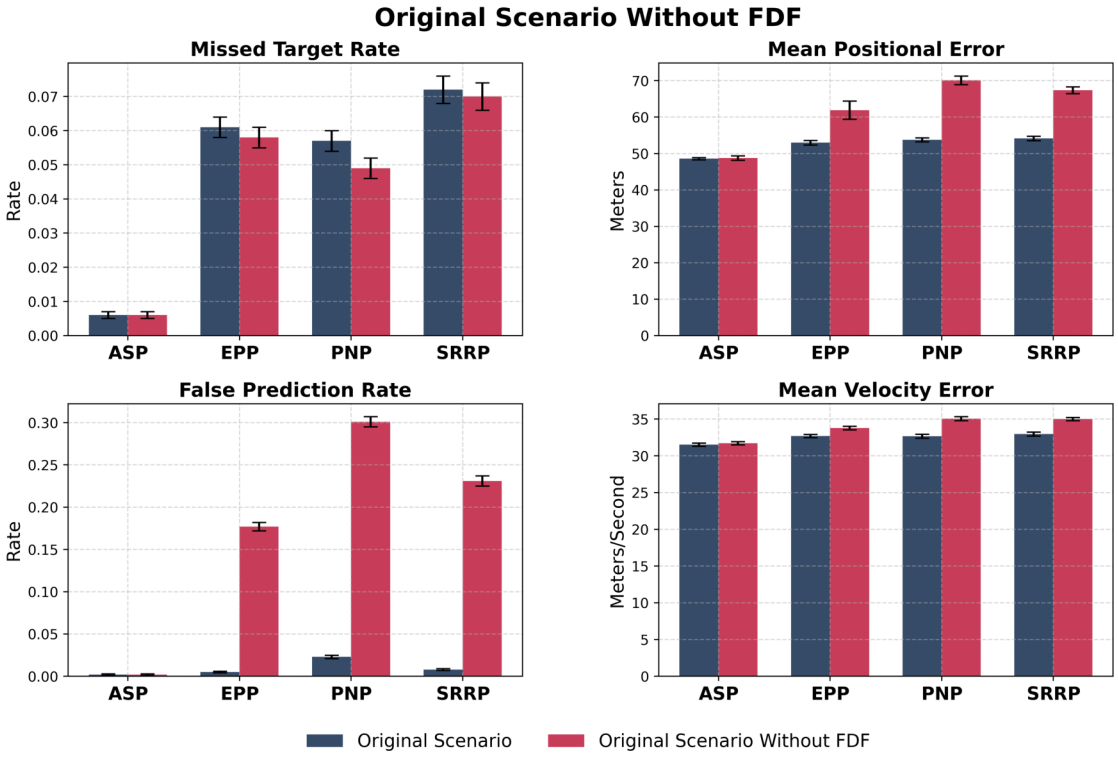


Figure 7.2: Performance comparison of multi-target tracking pipelines in the original scenario without using the False Detection Filter (FDF) decisions, equivalent to setting $\tau_{\text{fdf}} < 0$ (see Section 5.3.4). However, the ASP pipeline still uses the attention scores from the FDF, see Section 5.3.5.8. The results from Section 7.1.2, where the FDF was active, are included as a contrast. The bars represents the mean across 20 simulations while the whiskers represent one standard deviation. Metrics include Missed Target Rate, False Prediction Rate, Mean Positional Error, and Mean Velocity Error. Refer to Section 7.1.1 for metric definitions.

7.1.4 Increasing Scenario Complexity

This section examines the robustness and adaptability of the multi-target tracking pipelines under a more complex scenario, highlighting how performance is affected

in a more challenging environment. The following changes were made to the original scenario described in Section 6.2:

- **False Detection Velocity Distribution**: The velocity of false detections, previously drawn only modeled by range rate $\dot{r}$ by drawing directly from a uniform distribution
$$\dot{r} \sim \mathcal{U}_{[-150,150]},$$
is now drawn from the same distribution as true object velocities before being converted to range rate:
$$\dot{x}, \dot{y} \sim \mathcal{N}(0, 1500),$$
where 1500 refers to the variance of the distribution, making false detections more similar to real objects and thus increasing the difficulty of separating the two. Refer to Section 6.2 for more information on exact modeling aspects and Section 2.1 for information on range rate and Cartesian velocity.
- **Increased False Detection Density**: The mean number of false detections per scan has almost doubled from $\lambda_c = 5500$ to $\lambda_c = 10000$.
- **Increased Object Density**: The mean number of objects generated at the first scan has been tripled, from $\lambda_o = 8000$ to $\lambda_o = 24000$.

Figure 7.3 illustrates the increased complexity of the new scenario compared to the original. To address this added complexity, a new False Detection Filter (FDF), trained specifically for the Complex Scenario, is introduced.

This section evaluates the performance of the multitarget tracking pipelines, ASP, EPP, PNP, and SRRP, across two new scenarios while contrasting these results with their performance on the Original Scenario. The evaluations are defined as follows:

- **Original Scenario** (Section 7.1.2): Evaluation of the different pipelines in the original scenario when using the FDF trained on data from the original scenario. This serves as a reference point for evaluating pipeline performance.
- **Complex FDF on Complex Scenario** (Section 7.1.4.1): Evaluation of the different pipelines in the complex scenario when using the new FDF trained on data from the complex scenario.
- **Complex FDF on Original Scenario** (Section 7.1.4.2): Evaluation of the different pipelines in the original scenario when using the new FDF trained on data from the complex scenario. This provides insight into the adaptability of the different pipelines using the new FDF.

By contrasting the results from these evaluations, we gain a deeper understanding of the robustness and adaptability of the multi-target tracking pipelines and the Complex (new) FDF under varying levels of scenario complexity.

7.1.4.1 Complex FDF on Complex Scenario

This section evaluates the performance of the multi-target tracking pipelines, ASP, EPP, PNP, and SRRP, under the complex scenario using the False Detection Filter (FDF) trained for these conditions. The results, shown in Figure 7.4, are compared with the Original Scenario (Section 7.1.2) to assess the impact of the increased complexity.

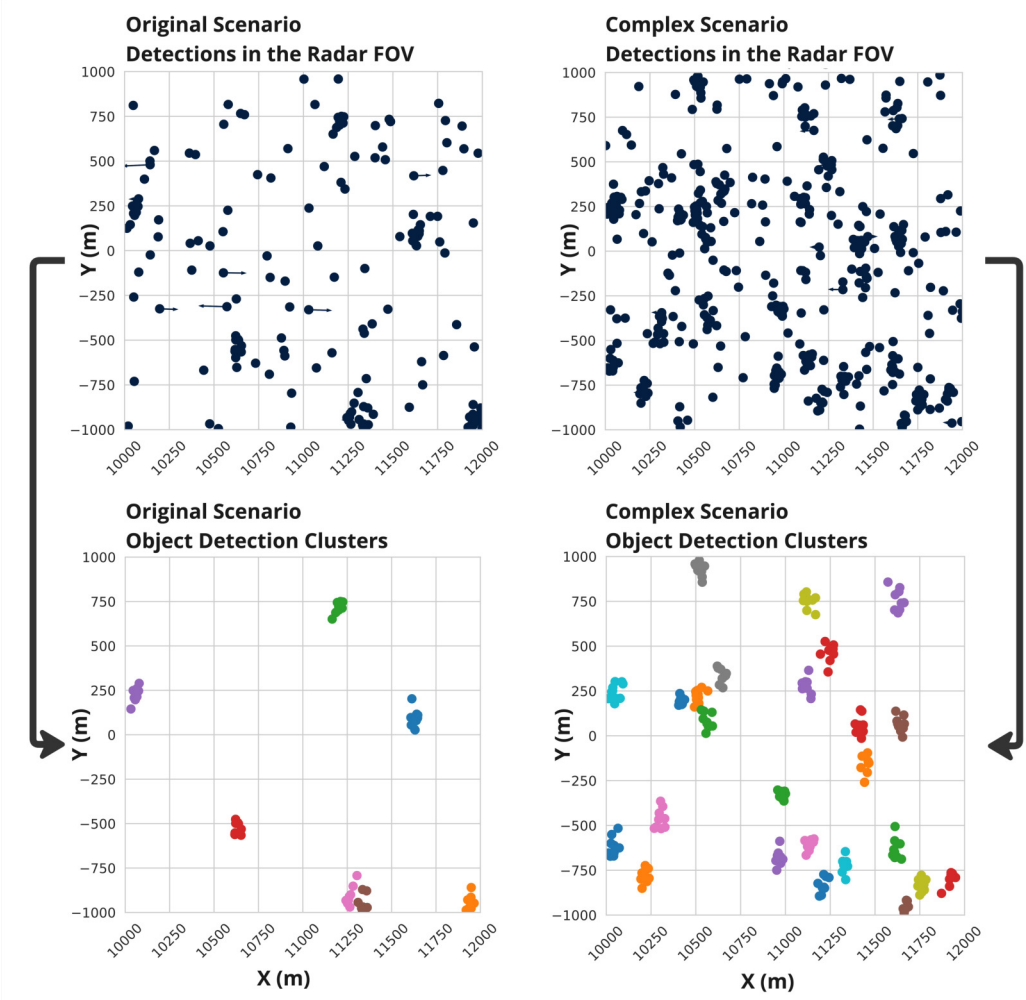


Figure 7.3: Illustration of the difference in complexity between the two scenarios in a zoomed-in view of the radars field of view with detections aggregated from all 10 scans. Different colored clusters represent detections from separate objects. Same-colored clusters with some separation correspond to different objects.

ASP shows the greatest robustness, with a false prediction rate roughly three times higher than in the original scenario, but still significantly lower, about half, than that of the next best performing method. Its missed target rate increases only by 33%, remaining 17-23 times lower than other pipelines, emphasizing its adaptability to more challenging conditions.

The other pipelines, EPP, PNP and SRRP, struggle with increased complexity, with false prediction rates rising 10-30 times and missed target rates doubling or tripling. These results indicate their limited ability to manage higher object densities and false detection rates.

Overall, ASP demonstrates robust performance, maintaining significantly lower false prediction and missed target rates compared to the other pipelines, even under increased scenario complexity. In contrast, the other pipelines exhibit substantial declines in all metrics evaluated, which emphasizes the difficulty of tracking in this more challenging environment. These results underscore ASP's scalability and set the stage for assessing the adaptability of the complex FDF in simpler conditions in Section 7.1.4.2.

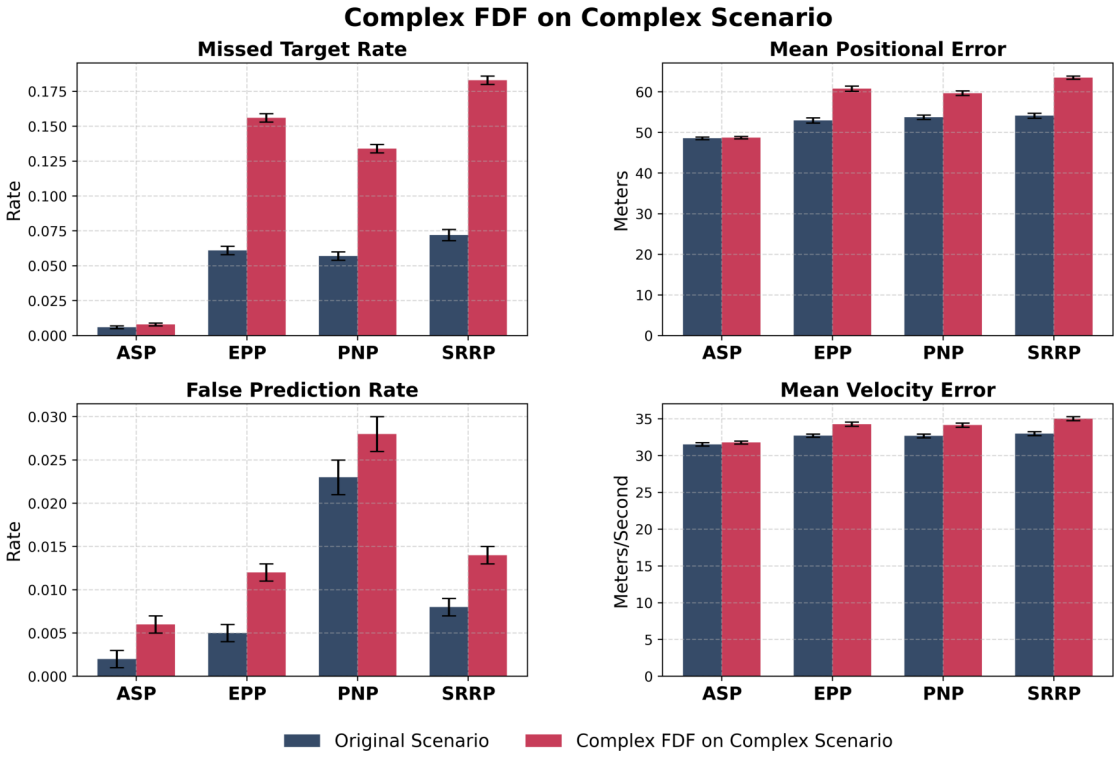


Figure 7.4: Performance of the multi-target tracking pipelines in the Complex Scenario using the FDF trained for this scenario. Evaluation metrics include Missed Target Rate, False Prediction Rate, Mean Positional Error, and Mean Velocity Error. The bars represents the mean across 20 simulations while the whiskers represent one standard deviation. For metric definitions, see Section 7.1.1, and for Original Scenario performance comparisons, refer to Section 7.1.2.

7.1.4.2 Complex FDF on Original Scenario

This section evaluates the performance of the multi-target tracking pipelines, ASP, EPP, PNP, and SRRP, on the original scenario when using the False Detection Filter (FDF) trained on the complex scenario. This evaluation provides insight into the adaptability of the different pipelines at varying levels of complexity. The results can be seen in Figure 7.5.

The results indicate that the pipelines exhibit performance that is nearly identical to their performance in the original scenario using the FDF trained specifically for it. Minor variations are observed, with false prediction rates increasing by approximately 0.002 across all pipelines and slight decreases in missed target rates, ranging from 0.001 to 0.002, for some methods. These changes are minimal and do not signify any substantial degradation in tracking performance.

This consistency across all pipelines suggests that the FDF trained on the complex scenario generalizes effectively to simpler conditions. This adaptability enables robust performance across different scenarios, allowing training efforts to focus on more challenging environments without risking degraded performance in less complex tasks - an essential capability for handling the diverse and unpredictable nature of real-world tracking scenarios.

7.2 Evaluation of MT3LS

In Section 7.2.1, we compare MT3LS with MT3v2 on a task close to what MT3v2 was created for. In Section 7.2.2 the single-target tracking performance of MT3LS is evaluated, as the partitioning module produces, mostly, non-ambiguous clusters, that is, clusters containing only detections originating from the same target.

7.2.1 Comparison of MT3LS and MT3v2

This subsection compares MT3LS with the original MT3v2 model on a benchmark task, similar to Task 1 in the MT3v2 paper [37], with a slightly increased false detection density.

Figure 7.6 shows the GOSPA score and its components for both models. The GOSPA metric was calculated using the same settings as in Section 2.3.3, with a cutoff distance parameter $c = 10$.

The results indicate minimal differences between MT3LS and MT3v2 for all GOSPA components. Both models demonstrate similar behavior in missed targets and false prediction costs, and their normalized localization costs are closely aligned, indicating comparable state estimation capabilities.

In general, MT3LS achieves almost identical performance as MT3v2, despite the changes introduced. This suggests that the modifications did not negatively affect

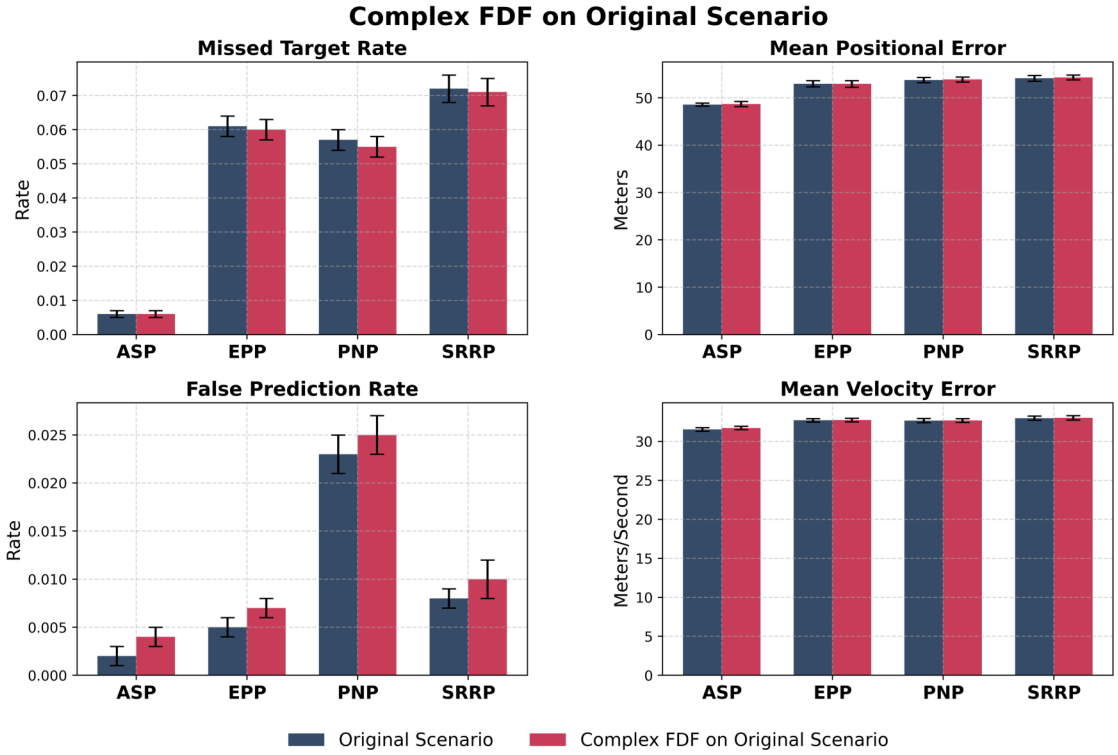


Figure 7.5: Performance of the multi-target tracking pipelines in the Original Scenario when using the Complex FDF. Metrics include Missed Target Rate, False Prediction Rate, Mean Positional Error, and Mean Velocity Error. The bars represents the mean across 20 simulations while the whiskers represent one standard deviation. Refer to Section 7.1.1 for metric definitions.

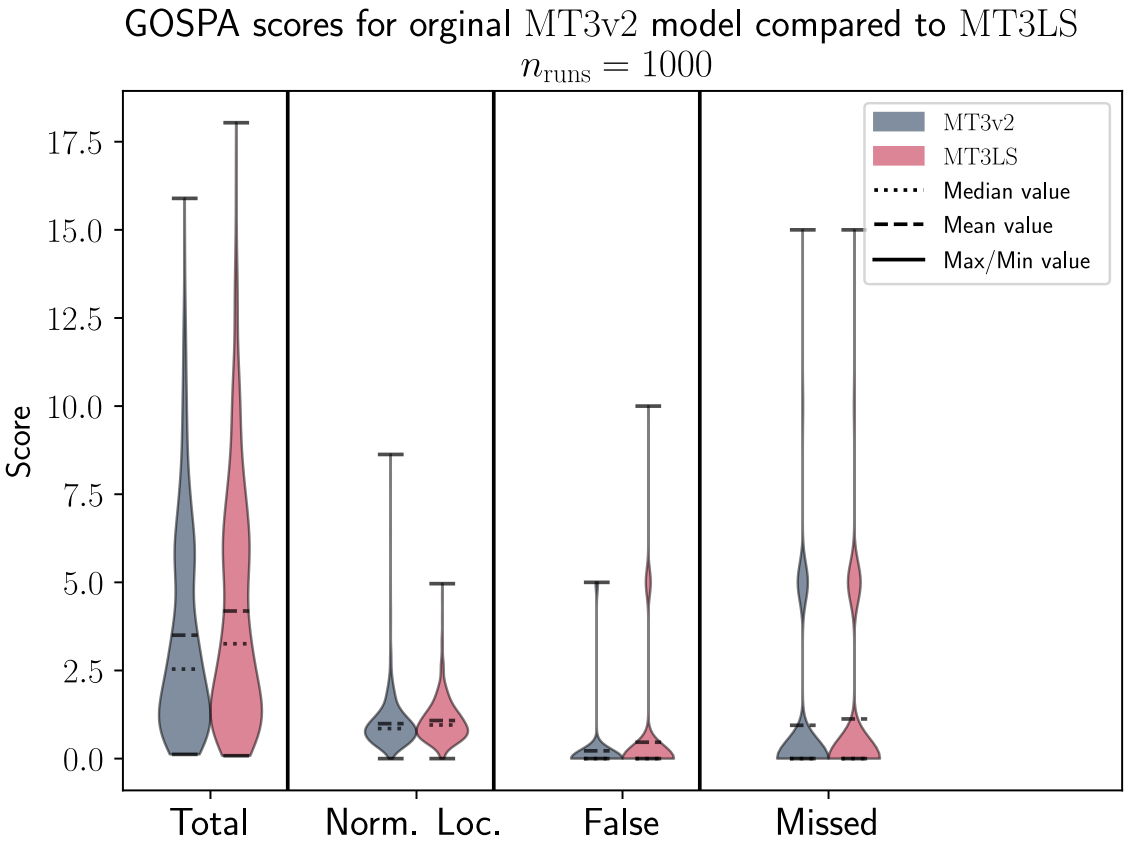


Figure 7.6: GOSPA score and its components (see Section 2.3.3) for the MT3LS model compared to the original MT3v2 model, summarized over 1000 simulations. Here, Norm. Loc. refers to the normalized localization cost, False refers to the false prediction cost, and missed refers to the missed target cost defined in Section 2.3.3.

the model's core multi-target tracking abilities. For details on the training process and the adjustments made to MT3LS, refer to Section B.4.

7.2.2 Single-Target Filtering Performance

In this section, we evaluate the *single-target* filtering performance of the proposed MT3LS model against a modified Kalman filter, using a similar matching procedure as described earlier. Details of the Kalman filter implementation are provided in Appendix A.1. The comparison utilizes the attention-based partitioning algorithm, with parameter values as specified in Table 6.1 by the method $\tilde{\mathbf{A}}$. In this comparison, the Kalman filter should be regarded as an optimal baseline filter. The Kalman filter predictions can be characterized in the same way as the MT3LS predictions, but without the existence probability:

$$\mathbf{P}_i = \left(\mathbf{M}^i,\, \mathbf{\Sigma}^i\right),$$

here, the first two components of the $\mathbf{M}^i$ vector refer to the mean state estimate of the Cartesian position, and the last and the following two components are the mean state estimates for the Cartesian velocity.

As this is an evaluation of single-target filtering performance, only the subsets $\mathcal{P}^j$ that do not contain multiple detections of the same scan time are selected. Because of this, the Kalman filter can process the detections in each subset in a sequential manner under the assumption that all detections originate from the same target. This assumption is valid for a large majority of subsets.

Prediction-Target Matching for Filter Evaluation

For evaluating single-target filtering performance, we adopt the same prediction-target matching procedure used for the pipeline evaluation.

The set of valid prediction-target mappings, denoted by $\mathcal{V}$, is used to evaluate the performance of MT3LS and the Kalman filter in different sectors of the field of view (FOV). These mappings can also refer to mappings between the predictions of the Kalman filter and the MT3LS, which just constitutes the predictions for each method from the same subset $\mathcal{P}^j$.

Evaluation Metrics

We employ two key metrics in this evaluation: the median Euclidean distance between $\mathbf{X}_\mathrm{z}$ and $\mathbf{Y}_\mathrm{z}$ given some mapping $\mathcal{V}^k$, denoted by $\mathcal{L}_2(\mathbf{X}_\mathrm{z}, \mathbf{Y}_\mathrm{z})$, and the median negative log-likelihood (NLL) value between $\mathbf{X}_\mathrm{z}$ and a predicted distribution $\mathbf{P}_\mathrm{z}$, denoted by $\mathrm{NLL}(\mathbf{X}_\mathrm{z} \mid \mathbf{P}_\mathrm{z})$. Here, k refers to a specific sector in the radar field of view (FOV). These metrics offer insights into both the accuracy and reliability of the model's predictions, as described below:

- **Median Euclidean Distance ($\mathcal{L}_2(\mathbf{X}_\mathrm{z}, \mathbf{Y}_\mathrm{z})$)**:

Given a set of matched pairs $(i, j) \in \mathcal{V}^k$ in a sector k, the median Euclidean distance is defined as follows:

$$\mathcal{L}_2(\mathbf{X}_\mathrm{z}, \mathbf{Y}_\mathrm{z}) = \mathrm{Median}\left(\left\{\sqrt{\left(\mathbf{X}_\mathrm{z}^{i,1} - \mathbf{Y}_\mathrm{z}^{j,1}\right)^2 + \left(\mathbf{X}_\mathrm{z}^{i,2} - \mathbf{Y}_\mathrm{z}^{j,2}\right)^2} \,\middle|\, (i,j) \in \mathcal{V}^k\right\}\right),$$

here, $\mathbf{X}_\mathrm{z}^{i,1}, \mathbf{X}_\mathrm{z}^{i,2}, \mathbf{Y}_\mathrm{z}^{j,1}, \mathbf{Y}_\mathrm{z}^{j,2}$ represent components of the state vectors for position or velocity, depending on whether z is p or v.

- **Median Negative Log-Likelihood ($\mathbf{NLL}(\mathbf{X}_\mathrm{z} \mid \mathbf{P}_\mathrm{z})$):**
  For each matched prediction-target pair $(i, j) \in \mathcal{V}^k$ in a particular FOV sector k, the median Negative Log-Likelihood is computed as:

$$\mathrm{NLL}(\mathbf{X}_\mathrm{z} \mid \mathbf{P}_\mathrm{z}) = \mathrm{Median}\left(\left\{-\log\left(\mathcal{N}\left(\mathbf{X}_\mathrm{z}^j \mid \mathbf{M}_\mathrm{z}^i, \mathbf{\Sigma}_\mathrm{z}^i\right)\right) \,\middle|\, (i,j) \in \mathcal{V}^k\right\}\right),$$

  here, the subscript z refers to either position (p) or velocity (v). The term $\mathcal{N}\left(\mathbf{X}_\mathrm{z}^j \mid \mathbf{M}_\mathrm{z}^i, \mathbf{\Sigma}_\mathrm{z}^i\right)$ represents the Gaussian likelihood of the state $\mathbf{X}_\mathrm{z}^j$, given the predicted Gaussian distribution $\mathbf{P}_i$, characterized by mean vector $\mathbf{M}_\mathrm{z}^i$ and covariance matrix $\mathbf{\Sigma}_\mathrm{z}^i$.
  This metric provides information on the reliability of the model's uncertainty estimates and state estimates.

In Figures 7.8, 7.10, 7.9, 7.11, the following naming convention appears:

- **$\mathbf{KF}_\mathrm{p}$:** Refers to the positional part of the filtering distributions contained in the Kalman predictions $\mathbf{P}_i$. Only the mean state vector is considered if the metric is the Euclidean distance or if it is on the left-hand side of $\mathrm{NLL}(\mathbf{X}_\mathrm{z} \mid \mathbf{P}_\mathrm{z})$.
- **$\mathbf{KF}_\mathrm{v}$:** Refers to the velocity part of the filtering distribution contained in the Kalman predictions $\mathbf{P}_i$. Only the mean state vector is considered if the metric is the Euclidean distance or if it is on the left-hand side of $\mathrm{NLL}(\mathbf{X}_\mathrm{z} \mid \mathbf{P}_\mathrm{z})$.
- **$\mathbf{M}_\mathrm{p}$:** Refers to the positional part of the filtering distributions contained in the MT3LS predictions $\mathbf{P}_i$. Only the mean state vector is considered if the metric is the Euclidean distance or if it is on the left-hand side of $\mathrm{NLL}(\mathbf{X}_\mathrm{z} \mid \mathbf{P}_\mathrm{z})$.
- **$\mathbf{M}_\mathrm{v}$:** Refers to the Cartesian velocity part of the filtering distribution contained in the MT3LS predictions $\mathbf{P}_i$. Only the mean state vector is considered if the metric is the Euclidean distance or if it is on the left-hand side of $\mathrm{NLL}(\mathbf{X}_\mathrm{z} \mid \mathbf{P}_\mathrm{z})$.
- **$\mathbf{GT}_\mathrm{p}$:** Refers to the ground-truth Cartesian position in the state vectors $\mathbf{X}_\mathrm{T}^j$
- **$\mathbf{GT}_\mathrm{v}$:** Refers to the ground-truth Cartesian velocity in the state vectors $\mathbf{X}_\mathrm{T}^j$

7.2.2.1 Position Estimation Performance

In this section, we evaluate the positional filtering performance of the proposed MT3LS model against the Kalman filter defined in Section 7.2.2 across various regions of the radar's field of view (FOV). Performance is measured using the median Euclidean distance $\mathcal{L}_2$ between the predicted and ground truth positions and the median negative log-likelihood (NLL) of ground truth given the predicted distribution. To get a sense of the difficulty of the filtering task, Figure 7.7 illustrates the combined Cartesian uncertainties in the radar field of view. For more information on the simulated scenario, see Section 6.2.

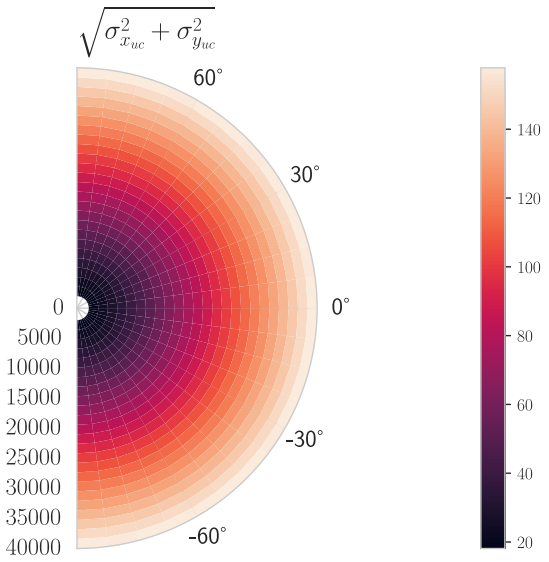


Figure 7.7: Measurement uncertainties in unbiased Cartesian coordinates across the FOV. Refer to Section 2.1 for more information. The plot visualizes the combined positional pseudo-standard deviation, calculated as $\sqrt{\sigma^2_{x_{\text{uc}}} + \sigma^2_{y_{\text{uc}}}}$ (see Section 2.1.2). Lighter colors indicate larger measurement uncertainties.

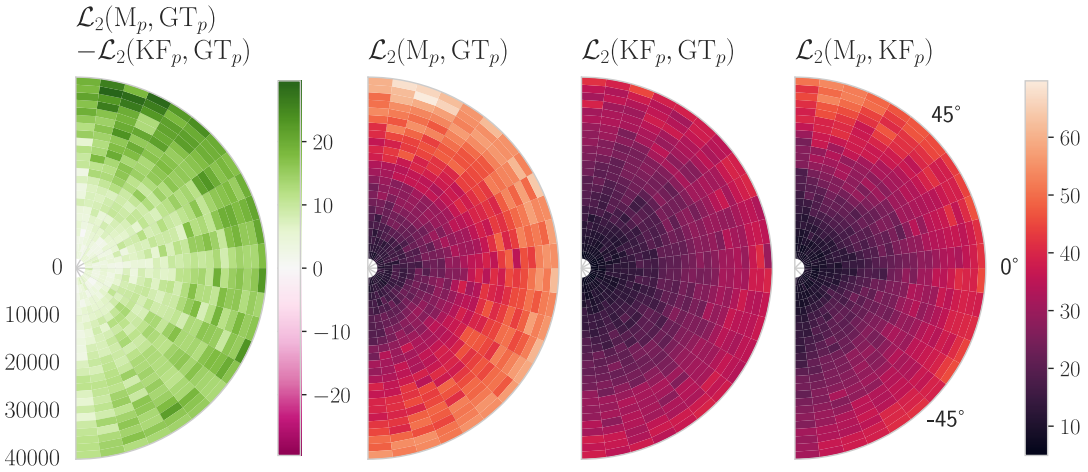


Figure 7.8: Median Euclidean distance $\mathcal{L}_2$ between the predicted and ground truth positions, comparing Kalman filter predictions and MT3LS predictions across different sectors of the radar's FOV. "M" refers to the MT3LS model, "KF" to the Kalman Filter, and "GT" to the Ground Truth Objects.

Figure 7.8 shows that the Kalman filter consistently outperforms MT3LS in terms of positional accuracy across all sectors of the FOV. In the upper FOV (between 45°and 90°), MT3LS exhibits significantly larger errors, possibly due to incomplete training. Generally, the MT3LS predictions are closer to Kalman estimates than to the ground truth, indicating that the model might have learned to process the data similarly to the Kalman filter. Both models show increasing errors with greater distance from the radar, consistent with higher uncertainties.

Figure 7.9 reinforces these findings: the Kalman filter consistently achieves lower NLL values, reflecting more reliable uncertainty estimates. In contrast, MT3LS performs better in regions where x or y are close to zero, but struggles when both x and y are not close to zero, suggesting difficulties in estimating correlations and handling measurement uncertainties effectively.

In summary, the Kalman filter demonstrates superior positional accuracy and reliability, particularly in regions with high measurement uncertainty, as expected. Although MT3LS captures similar trends, it underperforms in challenging areas of the FOV, highlighting potential areas for improvement in the handling of measurement uncertainties.

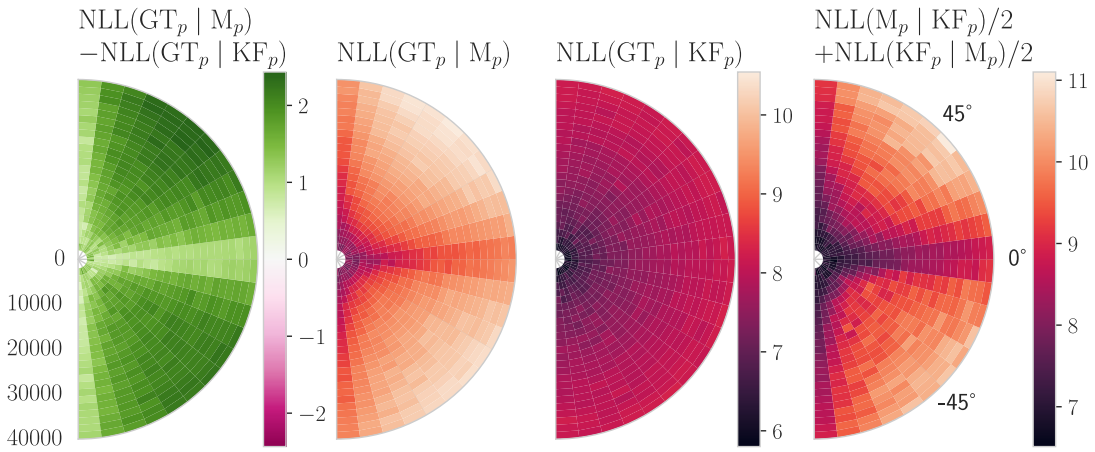


Figure 7.9: Median NLL of positional predictions, comparing ground truth, Kalman filter predictions, and MT3LS predictions across different sectors of the radar's FOV. "M" refers to the MT3LS model, "KF" to the Kalman Filter, and "GT" to the Ground Truth Objects.

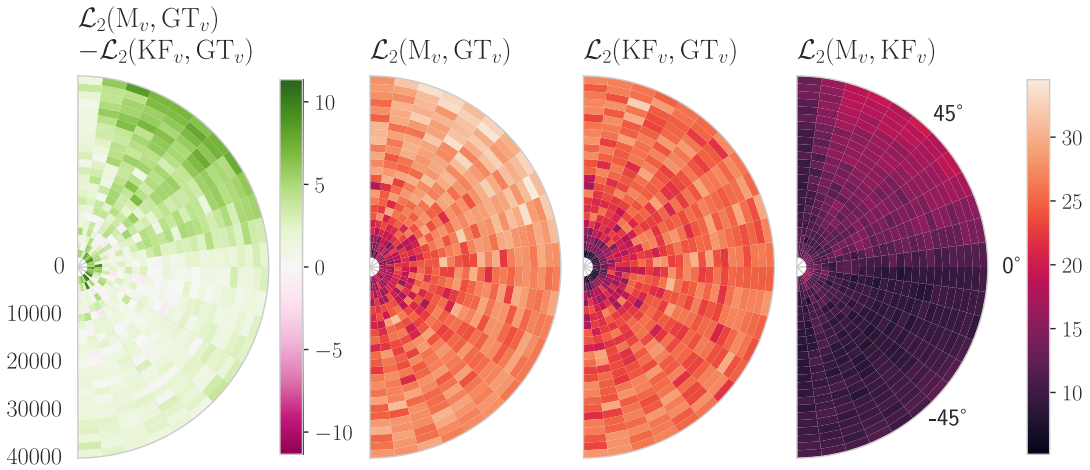


Figure 7.10: Median Euclidean distance $\mathcal{L}_2$ between predicted and ground-truth velocities, comparing Kalman filter predictions and MT3LS predictions across different sectors of the radar's FOV. "M" refers to the MT3LS model, "KF" to the Kalman Filter, and "GT" to the Ground Truth Objects.

7.2.2.2 Velocity Estimation Performance

In this section, we assess the velocity filtering performance of the proposed MT3LS model against the Kalman filter, as discussed in Section 7.2.2. Similarly to the positional evaluation in the previous section, performance is measured using the median Euclidean distance $\mathcal{L}_2$ between predicted and ground-truth velocities, as well as the median negative log-likelihood (NLL) of the ground truth given the predicted velocity distributions.

Figure 7.10 shows that the Kalman filter consistently outperforms MT3LS in terms of velocity accuracy in most sectors of the FOV. In the lower half of the FOV, the performance gap between the two models is reduced, with MT3LS showing a closer alignment to the Kalman filter. However, in the upper region (between 45°and 90°), MT3LS exhibits significantly larger errors (evident by the subplot farther to the left in Figure 7.10), which might indicate incomplete training.

As in the positional evaluation in the previous section, the velocity estimates of MT3LS tend to align more closely with the Kalman filter than with the ground truth, suggesting that the model may be learning similar estimation behaviors to the Kalman filter.

Figure 7.11 further emphasizes the differences in the uncertainty estimates of the models. The Kalman filter consistently achieves lower NLL values across the FOV, indicating more representative confidence in its velocity estimates, reflecting the

Kalman filter's ability to accurately assess underlying uncertainties. Although the Euclidean velocity distances for MT3LS are somewhat comparable to those of the Kalman filter, the higher median NLL values suggest that MT3LS struggles to accurately estimate the associated uncertainties, especially when x and y are not close to zero.

In summary, the Kalman filter maintains superior performance in velocity estimation, particularly in terms of uncertainty estimation as indicated by the NLL values. MT3LS, while showing comparable accuracy in some regions, struggles with covariance estimation, especially in areas with increased distance from the radar and when x and y are not close to zero, the same as for position.

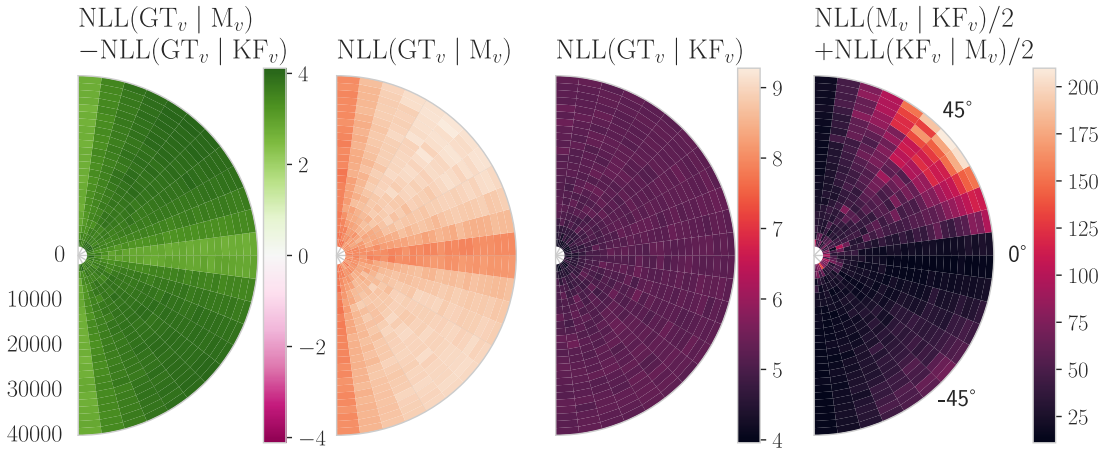


Figure 7.11: Median NLL of velocity predictions, comparing ground truth, Kalman filter predictions, and MT3LS predictions across different sectors of the radar's FOV. "M" refers to the MT3LS model, "KF" to the Kalman Filter, and "GT" to the Ground Truth Objects.

7.3 Further Discussion

In this section, we briefly explore several areas that warrant further consideration on the basis of the findings of this thesis. These include potential improvements in the choice of clustering algorithm, enhancements to the simulation setup, and the broader applications of the proposed methods. Each of these topics presents opportunities for future refinement and exploration.

7.3.1 Transformer-Aided Partitioning

The use of transformers for learning associations between detections, is not entirely novel; it has been explored in works such as [38], authored by the creators of the MT3v2 model [37]. In [38], the data association (partitioning) and smoothing objectives (see [39] for information on smoothing) are decoupled into separate stages, similar to the approach adopted in this thesis. The partitioning method in [38] also employs an encoder architecture, though, unlike this thesis, it formulates the problem as a multiclass classification task. This approach has both advantages and limitations. On the positive side, the model predicts class probabilities directly, thereby eliminating the need for a separate clustering step. However, a fixed number of classes constrains the model's adaptability and scalability.

The challenges of the multiclass approach include increased complexity in learning accurate class probabilities, which might be better handled through clustering algorithms relying on embedding similarities. Training also becomes more complicated due to the need for optimal assignment of class predictions to tracks, which is computationally expensive and may not scale well to scenarios involving a large number of targets (e.g., 10,000 or more). Furthermore, a fixed class space poses issues when dealing with a highly dynamic number of targets, as is typical in multi-target tracking scenarios. Additionally, it would not be possible to use a local context approach as in this thesis to handle complexities. Since class predictions are not necessarily consistent across different groupings, merging results into a cohesive global model is highly impractical. In contrast, the proposed transformer-aided approach in this thesis relies on similarities and local graphs with shared nodes, with natural aggregations to global graph representations.

7.3.2 Choice of Clustering Algorithm

The development of data association strategies in this thesis involved extensive evaluation of various clustering approaches before arriving at the final method. Initially, density-based algorithms like DBSCAN [2] seemed promising for identifying objects in the radar field of view (FOV) as clusters of detections in high-density regions. However, the large FOV and significant variations in measurement uncertainty rendered this approach ineffective. The concept of "density" became ambiguous in this context, leading to the poor performance of DBSCAN.

HDBSCAN [25], an extension of DBSCAN designed to handle varying densities, was subsequently evaluated. Although it outperformed DBSCAN, it still struggled with the complexities inherent in the simulated radar data, indicating the need for a more nuanced solution. HDBSCAN was revisited after the graphical approach via the false detection filter's association scores was found. As it can handle pre-computed metrics, the aggregated graph was transformed to instead represent a distance matrix. In this context, HDBSCAN showed significant improvements, as the pre-computed distances addressed many of the initial challenges. However, it was ultimately not chosen due to its slow inference times and the lack of efficient open-source implementations capable of handling pre-calculated distance matrices

effectively.

The Leiden algorithm demonstrated effective performance in empirical tests, as is evident from the results in this chapter and the previous chapter. However, as discussed in Section 3.5, its reliance on maximization of modularity lacks theoretical rigor for certain applications.

An alternative worth considering is clustering based on Stochastic Block Model (SBM) inference. SBM-based techniques use a Bayesian approach, providing a more theoretically grounded framework for modeling data association graphs. Despite their stronger theoretical foundation, SBM-based methods tend to be computationally slower compared to modularity-based algorithms. However, recent advances, such as those presented in [15], have led to more efficient SBM inference algorithms, which show promise in narrowing the gap between theoretical soundness and computational feasibility.

7.3.3 Simulation Enhancements

The current simulation framework, while effective, could be enhanced to more accurately reflect real-world conditions. One potential improvement involves introducing multiple object types, each sampled with a specific probability, to create a more realistic, multi-modal velocity distribution. Incorporating groups of targets, such as bird flocks or drone swarms, would also be an intriguing addition.

Incorporating a wider variety of motion models would further enhance the simulation. Given the short time frame of the simulations used in this thesis, this might not add much; however, expanding the simulation's scope to include more complex motion models would be beneficial for longer-term or more dynamic tracking scenarios.

7.3.4 Applications and Potential Use Cases

The results of this thesis provide various avenues for practical applications, with the attention-based partitioning method emerging as one of the most promising contributions. This approach offers a relatively simple yet effective training method, which, when combined with fast clustering algorithms, can achieve high-quality partitions in scenarios that would otherwise require more time-consuming manual techniques.

However, this method is not without limitations. One of its primary drawbacks is the occasional inability to produce entirely unambiguous object groupings, especially in cluttered environments. To address this, the attention-based method could be employed as a complexity reduction step, filtering out easy-to-process groups before applying more computationally expensive algorithms.

8

Conclusions and Future Work

This thesis presents a comprehensive pipeline for large-scale multi-target tracking, comprising a Local Context Generation Module, a False Detection Filter, a Partitioning Module, and a final transformer-based MTT model denoted MT3LS.

Key Achievements

The Local Context Generation Module demonstrated effective complexity reduction by dividing the detections in the radar field of view into smaller, manageable subsets. This method significantly decreased the memory requirements of subsequent processing, reducing memory consumption from 320 GB to 1.6 GB, enabling the efficient handling of over 100,000 detections.

The ASP pipeline, leveraging the attention scores generated by the False Detection Filter, consistently outperformed other implementations (EPP, PNP, and SRRP) across all metrics. These gains were particularly evident in the complex scenario, where the attention-based method exhibited strong scalability and robustness to increased complexity.

The False Detection Filter proved to be highly effective in discriminating between true and false detections, achieving balanced accuracies and F1 scores near 99%. In particular, for the EPP, PNP, and SRRP pipelines, i.e. based on more traditional statistical methods, performance was significantly degraded if the filter was removed.

The adaptability of the False Detection Filter was demonstrated by its performance when trained on a complex scenario and tested on the original scenario. Despite the shift in conditions, performance degradation was minimal across all evaluated metrics, demonstrating a capacity to generalize effectively.

In conclusion, the False Detection Filter, and its use of attention scores for partitioning, proved to be a critical component of the pipeline. It enabled efficient handling of large-scale scenarios and demonstrated robust scalability to complex environments, all while maintaining an average processing time of approximately 0.5 seconds for over 100,000 detections.

Future Work

Future work can build on the findings of this thesis by addressing specific challenges, such as environments with dense targets and high clutter (false detections). Developing more robust context generation modules that can handle dense regions without relying on assumptions of uniformly distributed targets and false detections would significantly enhance the scalability of the pipeline and allow for more robust handling of these dense scenarios.

Based on the results of the MT3LS model, exploring alternative implementations to mitigate covariance estimation and training issues in order to improve tracking accuracy presents a promising direction. Additionally, integrating traditional MOT methods, such as the PMBM filter, as a replacement for MT3LS, could provide a complementary approach to the final multi-target tracking part.

Another avenue for future research is the implementation of a recursive process that leverages the results of prior predictions and partitions. For instance, tracking could be initialized using the principles outlined in the proposed pipeline, with subsequent computations minimized by utilizing already identified targets and their predicted trajectories. This iterative refinement could further reduce computational costs and potentially improve tracking performance.

Bibliography

[1] Helmut Vogel. "A better way to construct the sunflower head". In: *Mathematical Biosciences* 44.3 (1979), pp. 179–189. ISSN: 0025-5564. DOI: https://doi.org/10.1016/0025-5564(79)90080-4.

[2] Martin Ester et al. "A density-based algorithm for discovering clusters in large spatial databases with noise". In: *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*. KDD'96. Portland, Oregon: AAAI Press, 1996, pp. 226–231.

[3] D.F. Winkler and United States. Air Force. Air Combat Command. *Searching the Skies: The Legacy of the United States Cold War Defense Radar Program*. Headquarters Air Combat Command, 1997, pp. 29–30. ISBN: 9781907521911. URL: https://books.google.se/books?id=iXrfAAAAMAAJ.

[4] S.S. Blackman. "Multiple hypothesis tracking for multiple target tracking." In: *IEEE Aerospace and Electronic Systems Magazine, Aerospace and Electronic Systems Magazine, IEEE, IEEE Aerosp. Electron. Syst. Mag* 19.1 (2004), pp. 5–18. ISSN: 0885-8985. URL: https://search.ebscohost.com/login.aspx?direct=true&db=edseee&AN=edseee.1263228&site=eds-live&scope=site&authtype=guest&custid=s3911979&groupid=main&profile=eds.

[5] Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Berlin, Heidelberg: Springer-Verlag, 2006, pp. 234–235. ISBN: 0387310738.

[6] Tom Fawcett. "An introduction to ROC analysis". In: *Pattern Recognition Letters* 27.8 (2006). ROC Analysis in Pattern Recognition, pp. 861–874. ISSN: 0167-8655. DOI: https://doi.org/10.1016/j.patrec.2005.10.010. URL: https://www.sciencedirect.com/science/article/pii/S016786550500303X.

[7] M. E. J. Newman. "Modularity and community structure in networks". In: *Proceedings of the National Academy of Sciences* 103.23 (2006), pp. 8577–8582. DOI: https://doi.org/10.1073/pnas.0601602103. eprint: https://www.pnas.org/doi/pdf/10.1073/pnas.0601602103.

[8] M.I. Skolnik. *Radar Handbook, Third Edition*. McGraw Hill LLC, 2008. ISBN: 9780071589420. URL: https://books.google.se/books?id=tA-KAwAAQBAJ.

[9] Kay Henning Brodersen et al. "The Balanced Accuracy and Its Posterior Distribution". In: *2010 20th International Conference on Pattern Recognition*. 2010, pp. 3121–3124. DOI: https://doi.org/10.1109/ICPR.2010.764.

[10] Nguyen Xuan Vinh, Julien Epps, and James Bailey. "Information Theoretic Measures for Clusterings Comparison: Variants, Properties, Normalization and

Correction for Chance". In: *J. Mach. Learn. Res.* 11 (Dec. 2010), pp. 2837–2854. ISSN: 1532-4435.

[11] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. "Deep Sparse Rectifier Neural Networks". In: *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*. Ed. by Geoffrey Gordon, David Dunson, and Miroslav Dudík. Vol. 15. Proceedings of Machine Learning Research. Fort Lauderdale, FL, USA: PMLR, Nov. 2011, pp. 315–323. URL: https://proceedings.mlr.press/v15/glorot11a.html.

[12] Kevin P. Murphy. *Machine Learning: A Probabilistic Perspective*. The MIT Press, 2012. ISBN: 0262018020.

[13] Richards Mark A. *Fundamentals of Radar Signal Processing, Second Edition*. McGraw-Hill Education, 2013. ISBN: 9780071798327. URL: https://search.ebscohost.com/login.aspx?direct=true&db=edsebk&AN=2696174&site=eds-live&scope=site&authtype=guest&custid=s3911979&groupid=main&profile=eds.

[14] William L. Melvin and James A. Scheer. *Principles of Modern Radar, Volume 3 - Radar Applications*. Institution of Engineering and Technology (The IET), 2014, pp. 1–16. ISBN: 978-1-89112-154-8. URL: https://app.knovel.com/hotlink/toc/id:kpPMRVRA06/principles-modern-radar/principles-modern-radar.

[15] Tiago P. Peixoto. "Efficient Monte Carlo and greedy heuristic for the inference of stochastic block models". In: *Phys. Rev. E* 89 (1 Jan. 2014), p. 012804. DOI: https://doi.org/10.1103/PhysRevE.89.012804.

[16] Nitish Srivastava et al. "Dropout: A Simple Way to Prevent Neural Networks from Overfitting". In: *Journal of Machine Learning Research* 15.56 (2014), pp. 1929–1958. URL: http://jmlr.org/papers/v15/srivastava14a.html.

[17] B.-N. Vo, B.-T. Vo, and D. Phung. "Labeled Random Finite Sets and the Bayes Multi-Target Tracking Filter." In: *IEEE Transactions on Signal Processing, Signal Processing, IEEE Transactions on, IEEE Trans. Signal Process* 62.24 (2014), pp. 6554–6567. ISSN: 1053-587X. URL: https://search.ebscohost.com/login.aspx?direct=true&db=edseee&AN=edseee.6928494&site=eds-live&scope=site&authtype=guest&custid=s3911979&groupid=main&profile=eds.

[18] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. *Layer Normalization*. 2016. arXiv: 1607.06450 [stat.ML]. URL: https://arxiv.org/abs/1607.06450.

[19] Santo Fortunato and Darko Hric. "Community detection in networks: A user guide". In: *Physics Reports* 659 (2016). Community detection in networks: A user guide, pp. 1–44. ISSN: 0370-1573. DOI: https://doi.org/10.1016/j.physrep.2016.09.002.

[20] Gaspare Galati. *100 Years of Radar. [electronic resource]*. Springer International Publishing, 2016, p. 274. ISBN: 9783319005843. URL: https://search.ebscohost.com/login.aspx?direct=true&db=cat07472a&AN=clec.SPRINGERLINK9783319005843&site=eds-live&scope=site&authtype=guest&custid=s3911979&groupid=main&profile=eds.

[21] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016, p. 171.

[22] Kaiming He et al. "Deep Residual Learning for Image Recognition". In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016, pp. 770–778. DOI: https://doi.org/10.1109/CVPR.2016.90.

[23] Christian L. Staudt, Aleksejs Sazonovs, and Henning Meyerhenke. "NetworKit: A Tool Suite for Large-scale Complex Network Analysis". In: *Network Science* 4.4 (Dec. 2016). DOI: https://doi.org/10.1017/nws.2016.20.

[24] Diederik P. Kingma and Jimmy Ba. *Adam: A Method for Stochastic Optimization*. 2017. arXiv: 1412.6980 [cs.LG]. URL: https://arxiv.org/abs/1412.6980.

[25] Leland McInnes, John Healy, and Steve Astels. "hdbscan: Hierarchical density based clustering". In: *Journal of Open Source Software* 2.11 (2017), p. 205. DOI: https://doi.org/10.21105/joss.00205.

[26] Abu Sajana Rahmathullah, Angel F. Garcia-Fernandez, and Lennart Svensson. "Generalized optimal sub-pattern assignment metric." In: *2017 20th International Conference on Information Fusion (Fusion), Information Fusion (Fusion), 2017 20th International Conference on* (2017), pp. 1–8. ISSN: 978-0-9964-5270-0. URL: https://search.ebscohost.com/login.aspx?direct=true&db=edseee&AN=edseee.8009645&site=eds-live&scope=site&authtype=guest&custid=s3911979&groupid=main&profile=eds.

[27] Ashish Vaswani et al. "Attention is all you need". In: *Proceedings of the 31st International Conference on Neural Information Processing Systems*. NIPS'17. Long Beach, California, USA: Curran Associates Inc., 2017, pp. 6000–6010. ISBN: 9781510860964.

[28] Ángel F. García-Fernández et al. "Poisson Multi-Bernoulli Mixture Filter: Direct Derivation and Implementation". In: *IEEE Transactions on Aerospace and Electronic Systems* 54.4 (2018), pp. 1883–1901. DOI: https://doi.org/10.1109/TAES.2018.2805153.

[29] Hongqiang Liu et al. "Two unbiased converted measurement Kalman filtering algorithms with range rate". In: *IET Radar, Sonar & Navigation* 12.11 (2018), pp. 1217–1224. DOI: https://doi.org/10.1049/iet-rsn.2018.5154. eprint: https://ietresearch.onlinelibrary.wiley.com/doi/pdf/10.1049/iet-rsn.2018.5154.

[30] V. A. Traag, L. Waltman, and N. J. van Eck. "From Louvain to Leiden: guaranteeing well-connected communities". In: *Scientific Reports* 9.1 (Mar. 2019), p. 5233. ISSN: 2045-2322. DOI: https://doi.org/10.1038/s41598-019-41695-z.

[31] Nicolas Carion et al. "End-to-End Object Detection with Transformers". In: *Computer Vision – ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part I*. Glasgow, United Kingdom: Springer-Verlag, 2020, pp. 213–229. ISBN: 978-3-030-58451-1. DOI: https://doi.org/10.1007/978-3-030-58452-8_13.

[32] Charles R. Harris et al. “Array programming with NumPy”. In: *Nature* 585.7825 (Sept. 2020), pp. 357–362. DOI: https://doi.org/10.1038/s41586-020-2649-2.

[33] Elisa Cabana, Rosa E. Lillo, and Henry Laniado. “Multivariate outlier detection based on a robust Mahalanobis distance with shrinkage estimators.” In: *Statistical Papers* 62.4 (2021), pp. 1583–1609. ISSN: 09325026. URL: https://search.ebscohost.com/login.aspx?direct=true&AuthType=sso&db=bsu&AN=151456640&authtype=sso&custid=s3911979&site=ehost-live&scope=site&authtype=sso&custid=s3911979.

[34] Yang Li et al. *Learnable Fourier Features for Multi-Dimensional Spatial Positional Encoding*. 2021. arXiv: 2106.02795 [cs.LG]. URL: https://arxiv.org/abs/2106.02795.

[35] Xizhou Zhu et al. *Deformable DETR: Deformable Transformers for End-to-End Object Detection*. 2021. arXiv: 2010.04159 [cs.CV]. URL: https://arxiv.org/abs/2010.04159.

[36] Tim Meinhardt et al. “TrackFormer: Multi-Object Tracking with Transformers.” In: *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), Computer Vision and Pattern Recognition (CVPR), 2022 IEEE/CVF Conference on, CVPR* (2022), pp. 8834–8844. ISSN: 978-1-6654-6946-3. URL: https://search.ebscohost.com/login.aspx?direct=true&db=edseee&AN=edseee.9879668&site=eds-live&scope=site&authtype=guest&custid=s3911979&groupid=main&profile=eds.

[37] Juliano Pinto et al. “Deep Learning for Model-Based Multiobject Tracking”. In: *IEEE Transactions on Aerospace and Electronic Systems* 59.6 (2023), pp. 7363–7379. DOI: https://doi.org/10.1109/TAES.2023.3289164.

[38] Juliano Pinto et al. *Transformer-Based Multi-Object Smoothing with Decoupled Data Association and Smoothing*. 2023. arXiv: 2312.17261 [cs.CV]. URL: https://arxiv.org/abs/2312.17261.

[39] Simo Särkkä and Lennart Svensson. *Bayesian Filtering and Smoothing*. English. 2nd ed. Institute of Mathematical Statistics Textbooks. Cambridge, England: Cambridge University Press, June 2023.

[40] Weihua Wu et al. “Basics of Random Finite Sets”. In: *Target Tracking with Random Finite Sets*. Singapore: Springer Nature Singapore, 2023, pp. 61–108. ISBN: 978-981-19-9815-7. DOI: https://doi.org/10.1007/978-981-19-9815-7_3.

[41] Jason Ansel et al. “PyTorch 2: Faster Machine Learning Through Dynamic Python Bytecode Transformation and Graph Compilation”. In: *29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2 (ASPLOS '24)*. ACM, Apr. 2024. DOI: https://doi.org/10.1145/3620665.3640366.

[42] Mariano Zafra et al. “How drone combat in Ukraine is changing warfare”. In: *Reuters* (Mar. 26, 2024). URL: https://www.reuters.com/graphics/UKRAINE-CRISIS/DRONES/dwpkeyjwkpm/ (visited on 09/21/2024).

A

Appendix: Algorithms

A.1 A Modified DUCMKF-R Algorithm

The DUCMKF-R algorithm (decorrelated unbiased converted measurement Kalman filter algorithm with range rate), introduced in [29], is a Kalman filter developed to filter polar measurements that have been converted to Cartesian coordinates, as described in [29].
The DUCMKF-R algorithm starts by converting the polar measurements as follows (described previously in Section 2.1.2):

$$\begin{aligned} \lambda_{\varphi} &= e^{-\sigma_{\varphi}^2/2}, \\ \mathbf{z}_{uc} &= \begin{bmatrix} \lambda_{\varphi}^{-1} r\cos(\varphi) \\ \lambda_{\varphi}^{-1} r\sin(\varphi) \\ \dot{r} \end{bmatrix} = \begin{bmatrix} x_{uc} \\ y_{uc} \\ \dot{r} \end{bmatrix}. \end{aligned} \tag{A.1}$$

The subscript $_{uc}$ stands for unbiased Cartesian measurements.
The measurement model is:

$$\mathbf{h}(\mathbf{x}) = \begin{bmatrix} x \\ y \\ \dot{x}\frac{x}{\sqrt{x^2+y^2}} + \dot{y}\frac{y}{\sqrt{x^2+y^2}} \end{bmatrix}. \tag{A.2}$$

The state vector is $\mathbf{x}_k = [x,\, y,\, \dot{x},\, \dot{y}]^T$ and thus the measurement model needs to be linearized. In the DUCMKF-R this is done with the help of the predicted state mean vector $\mu_{k+1|k}$:

$$\mu_{k+1|k} = \begin{bmatrix} x_{k+1|k} \\ y_{k+1|k} \\ \dot{x}_{k+1|k} \\ \dot{y}_{k+1|k} \end{bmatrix}, \tag{A.3}$$

$$\mathrm{H}_{k+1|k} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \frac{x_{k+1|k}}{\sqrt{x_{k+1|k}^2+y_{k+1|k}^2}} & \frac{y_{k+1|k}}{\sqrt{x_{k+1|k}^2+y_{k+1|k}^2}} \end{bmatrix}. \tag{A.4}$$

This differs from the standard Extended Kalman filter setting, where non-linear observation models such as (A.2) are linearized through its Jacobian and then evaluated at predicted state estimates (A.3) [39]. The subscript k denotes the values at

time k while $k+1|k$ denotes the predicted values at time $k+1$ given information at time k and earlier.

Since the measurements have been converted, the original diagonal measurement error covariance matrix (see Section 2.1.1) can no longer be used. The algorithm utilizes predicted state estimates to construct the new measurement error covariance matrix $\mathrm{R}_{k+1|k}$, and the resulting entries can be seen below:

$$
\begin{aligned}
r_{k+1|k} &= \sqrt{x_{k+1|k}^2 + y_{k+1|k}^2}, \\
\varphi_{k+1|k} &= \operatorname{atan2}\left(y_{k+1|k},\, x_{k+1|k}\right), \\
\mathrm{P}_{xy,k+1|k} &= \begin{bmatrix} \mathrm{P}_{k+1|k}^{0,0} & \mathrm{P}_{k+1|k}^{0,1} \\ \mathrm{P}_{k+1|k}^{1,0} & \mathrm{P}_{k+1|k}^{1,1} \end{bmatrix}, \\
\sigma_{r,k+1|k}^2 &= \frac{1}{r_{k+1|k}^2} \begin{bmatrix} x_{k+1|k} & y_{k+1|k} \end{bmatrix} \mathrm{P}_{xy,k+1|k} \begin{bmatrix} x_{k+1|k} & y_{k+1|k} \end{bmatrix}^T, \\
\sigma_{\varphi,k+1|k}^2 &= \frac{1}{r_{k+1|k}^4} \begin{bmatrix} -y_{k+1|k} & x_{k+1|k} \end{bmatrix} \mathrm{P}_{xy,k+1|k} \begin{bmatrix} -y_{k+1|k} & x_{k+1|k} \end{bmatrix}^T, \\
\gamma_\varphi &= e^{-2\sigma_\varphi^2}, \\
\tilde{\gamma}_\varphi &= e^{-2\sigma_{\varphi,k+1|k}^2}, \\
\mathrm{R}_{k+1|k}^{0,0} &= \frac{e^{\sigma_\varphi^2}}{2} \left(r_{k+1|k}^2 + \sigma_r^2 + \sigma_{r,k+1|k}^2\right) \left(1 + \cos\left(2\varphi_{k+1|k}\right) \tilde{\gamma}_\varphi \gamma_\varphi\right), \\
&\quad - \frac{1}{2} \left(r_{k+1|k}^2 + \sigma_{r,k+1|k}^2\right) \left(1 + \cos\left(2\varphi_{k+1|k}\right) \tilde{\gamma}_\varphi\right), \\
\mathrm{R}_{k+1|k}^{0,1} &= \frac{e^{\sigma_\varphi^2}}{2} \left(r_{k+1|k}^2 + \sigma_r^2 + \sigma_{r,k+1|k}^2\right) \left(\sin\left(2\varphi_{k+1|k}\right) \tilde{\gamma}_\varphi \gamma_\varphi\right), \\
&\quad - \frac{1}{2} \left(r_{k+1|k}^2 + \sigma_{r,k+1|k}^2\right) \left(\sin\left(2\varphi_{k+1|k}\right) \tilde{\gamma}_\varphi\right), \\
&= \mathrm{R}_{k+1|k}^{0,1}, \\
\mathrm{R}_{k+1|k}^{1,1} &= \frac{e^{\sigma_\varphi^2}}{2} \left(r_{k+1|k}^2 + \sigma_r^2 + \sigma_{r,k+1|k}^2\right) \left(1 - \cos\left(2\varphi_{k+1|k}\right) \tilde{\gamma}_\varphi \gamma_\varphi\right), \\
&\quad - \frac{1}{2} \left(r_{k+1|k}^2 + \sigma_{r,k+1|k}^2\right) \left(1 - \cos\left(2\varphi_{k+1|k}\right) \tilde{\gamma}_\varphi\right), \\
\mathrm{R}_{k+1|k}^{2,2} &= \sigma_{\dot{r}}^2, \\
\mathrm{R}_{k+1|k}^{0,2} &= \mathrm{R}_{k+1|k}^{2,0} = \mathrm{R}_{k+1|k}^{2,1} = \mathrm{R}_{k+1|k}^{1,2} = 0.
\end{aligned}
\tag{A.5}
$$

Given these assumptions and converted measurements, the standard Kalman filter algorithm and assumptions can be applied, with some exceptions. The state at time k (A.8) is assumed to be a transformation F_{k-1} (the transition matrix) from the state at time k with the addition of the so-called process noise $\mathbf{q}_{k-1}$ (A.6) which is assumed to be normally distributed [39]:

$$\mathbf{q}_k \sim \mathcal{N}\left(0,\, \mathrm{Q}_k\right), \tag{A.6}$$
$$\mathbf{r}_k \sim \mathcal{N}\left(0,\, \mathrm{R}_{k|k-1}\right), \tag{A.7}$$
$$\mathbf{x}_k = \mathrm{F}_{k-1}\mathbf{x}_{k-1} + \mathbf{q}_{k-1}, \tag{A.8}$$
$$\mathbf{z}_k = \mathbf{h}\left(\mathbf{x}_k\right) + \mathbf{r}_k. \tag{A.9}$$

In (A.9) measurements are assumed to be a transformation from the current state $\mathbf{x}_k$ into the (converted) measurement space, with the addition of the measurement noise $\mathbf{r}_k$ (A.7).
In the following, the prediction step of the Kalman filter is outlined along with related distributions:

$$\mu_{k+1|k} = \mathrm{F}_k\mu_{k|k}, \tag{A.10}$$
$$\mathrm{P}_{k+1|k} = \mathrm{F}_k\mathrm{P}_{\mathrm{k|k}}\mathrm{F}_k^T + \mathrm{Q}_k, \tag{A.11}$$
$$\mathbf{z}_{k+1|k} \sim \mathcal{N}\left(\mathrm{H}_{k+1|k}\mu_{k+1|k},\, \mathrm{S}_{k+1|k+1}\right), \tag{A.12}$$
$$\mathbf{x}_{k+1|k} \sim \mathcal{N}\left(\mu_{k+1|k},\, \mathrm{P}_{k+1|k}\right). \tag{A.13}$$

Following the prediction step is the update step:

$$\mathrm{S}_{k+1|k+1} = \mathrm{H}_{k+1|k}\mathrm{P}_{k+1|k}\mathrm{H}_{k+1|k}^T + \mathrm{R}_{k+1|k}, \tag{A.14}$$
$$\mathrm{K}_{k+1|k+1} = \mathrm{P}_{k+1|k}\mathrm{H}_{k+1|k}^T\mathrm{S}_{k+1|k+1}^{-1}, \tag{A.15}$$
$$\mu_{k+1|k+1} = \mu_{k+1|k} + \mathrm{K}_{k|k}\left(\mathbf{z}_k - \mathrm{H}_{k+1|k+1}\mu_{k+1|k}\right), \tag{A.16}$$
$$\mathrm{P}_{k+1|k+1} = \mathrm{P}_{k+1|k} - \mathrm{K}_{k|k}\mathrm{S}_{k|k}\mathrm{K}_{k|k}^T, \tag{A.17}$$
$$\mathbf{x}_{k+1|k+1} \sim \mathcal{N}\left(\mu_{k+1|k+1},\, \mathrm{P}_{k+1|k+1}\right). \tag{A.18}$$

The resulting filtering density for the state is (A.18).

A.1.1 Implementation Details

The DUCMKF-R algorithm was implemented with PyTorch [41] so that the GPU could be used. The filter was initialized with the following mean vector and covariance matrix:

$$\mu_{0|0} = \begin{bmatrix} x_{uc,0} \\ y_{uc,0} \\ \dot{r}\cos(\varphi) \\ \dot{r}\sin(\varphi) \end{bmatrix}, \tag{A.19}$$

$$\mathrm{P}_{0|0} = \begin{bmatrix} 10^4 & 0 & 0 & 0 \\ 0 & 10^4 & 0 & 0 \\ 0 & 0 & 1500 & 0 \\ 0 & 0 & 0 & 1500 \end{bmatrix}, \tag{A.20}$$

here, the diagonal entries for the position are set to a high enough value to account for the uncertainty and the velocity components are set equal to the variance of the velocity components, see Section 6.2.
The process noise distribution was set to equal the underlying model in the task:

$$\mathbf{q} \sim \mathcal{N}\left(\begin{bmatrix}0\\0\\0\\0\end{bmatrix}, \sigma_a^2 \begin{bmatrix}\mathbf{I}\Delta\mathrm{T}^3/3 & \mathbf{I}\Delta\mathrm{T}^2/2\\ \mathbf{I}\Delta\mathrm{T}^2/2 & \mathbf{I}\Delta\mathrm{T}\end{bmatrix}\right)$$

A.2 Sunflower Seeds Distribution

One way to approximately uniformly distribute n points within a circle is by imitating patterns of, for example, sunflower seeds. In [1] the mechanics of the sunflower seed patterns is examined and a formula is given for the angle pattern of consecutive seeds. For the radii of consecutive seeds, if equal spacing of seeds is desired, the area at the seed radii should be proportional to the number of consecutive seeds, i.e. $\pi r_k^2 \propto k$ [1]. One way to achieve this is by setting the radii of the seed k (r_k) as in:

$$\begin{aligned}\delta &= 180(3-\sqrt{5}) \approx 137.5^\circ\\ \varphi_k &= \delta k\\ r_k &= \sqrt{k}\end{aligned} \tag{A.21}$$

The angle formula relies on the golden angle $\delta \approx 137.5^\circ$ that determines the angles of consecutive seeds placed φ_k (A.21), and is derived from the well-known golden ratio commonly found in nature, as in the case with sunflower seed placement.
To achieve a more even distribution of points along the edge of the resulting circle from the pattern, one can set a fraction of the last seeds' radii to the same value. To achieve an approximately proportional scaling with the circumference of the resulting circle, the fraction n_{edge} should be proportional to the radii and thus the square root of the total number of seeds (n) $n_{\text{edge}} \propto \sqrt{n}$. Depending on how many points one wants on the edge, a parameter α can be adjusted. From this, the seed sequences can be derived as follows:

$$\begin{aligned}n &= N,\\ (\varphi_k)_{k=1}^{n} &= (\delta 1, \delta 2, \ldots, \delta n),\\ n_{\text{edge}} &= \alpha\lfloor\sqrt{n}\rceil,\\ n_{\text{int}} &= n - n_{\text{edge}},\\ (r_k)_{k=1}^{n_{\text{int}}} &= (\sqrt{1}, \sqrt{2}, \ldots, \sqrt{n_{\text{int}}}),\\ (r_k)_{k=n_{\text{int}}+1}^{n} &= (\sqrt{n_{\text{int}}}, \sqrt{n_{\text{int}}}, \ldots, \sqrt{n_{\text{int}}}),\end{aligned} \tag{A.22}$$

for an arbitrary number of seeds (points). The radii sequence can be scaled with any number to make the points be contained in a circle of arbitrary radius.

B

Appendix: Implementation Details and Training Procedures

B.1 Implementation Details

The False Detection Filter and MT3LS were implemented in PyTorch [41]. The ADAM optimizer was used for training [24] and during training, dropout, see Section 3.2 for more details, was utilized between sublayers of the encoder and decoder for MT3LS.

The Leiden algorithm used is from the networkit library [23] with standard settings. NumPy [32] was also used throughout the different module implementations.

The MT3LS model and code for simulations are based on the openly available GitHub repository connected to the original article [37] on the MT3v2 model.

B.2 FDF: Training Procedure and Inference Optimization

The False Detection Filter was trained using the Adam optimizer [24]. Each local context served as a separate input. During training, the batch size was set to 1024, meaning one gradient update per processing of 1024 local context. The training process followed the same Adam settings as [37], except for a more aggressive learning rate reduction schedule - triggered after 10,000 gradient steps without improvement, instead of the 50,000 steps used in the original configuration. Dropout was used in a similar way to [37], and training was carried out until convergence was achieved.

For inference, optimizations were implemented to improve computational efficiency. The False Detection Filter was adapted to process 16-bit floating point numbers instead of the standard 32-bit, reducing memory requirements and speeding up computation. Additional PyTorch optimization techniques were applied, further reducing the inference time. Despite these improvements, there remains substantial potential to further enhance inference speed through additional optimizations.

B.3 MT3LS: Training Procedure

The MT3LS model was trained using subsets generated by the attention-based partitioning method $\tilde{\mathbf{A}}$. Consequently, the model was mainly exposed to subsets with low levels of data ambiguity (subsets with detections belonging to a single underlying target), since most of the subsets in the produced partition are perfect (see Sections 6.4.1 and 3.5.1 for details). Given that all partitioning methods with their pseudo-optimal parameter values produced a majority of subsets with minimal ambiguity, evidenced by median values of Q_{pg} exceeding 60%, this was not considered a significant concern.

Training was carried out with batch sizes ranging from 200 to 512, as the number of subsets containing detections of active objects varied (see Section 5.3.6). The model was trained until a *reasonable* level of convergence was observed, which was achieved after about a day with minimal to no performance improvement. Although training with a constant batch size might have improved efficiency and stability, the model was deemed to have converged adequately under the given conditions. The Adam optimizer and dropout were used with the same settings as in [37].

B.4 MT3LS vs MT3v2: Training

The training of the proposed MT3LS model closely follows the procedure outlined in the original MT3v2 framework [37]. The standard deviation features in MT3LS were normalized by a factor of 15 instead of 150, reflecting the task-specific scaling required for this implementation.

Both models, MT3LS and MT3v2, were trained using identical loss functions and hyperparameter settings as detailed in the original MT3v2 paper [37]. The only difference is that MT3LS predicts the full covariance matrices. This consistency in the training protocol, assuming convergence in training, ensures that any observed differences in performance are most likely directly attributable to architectural adjustments in MT3LS.

DEPARTMENT OF MATHEMATICAL SCIENCES
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden
www.chalmers.se

CHALMERS
UNIVERSITY OF TECHNOLOGY