Christopher Strøm

Towards robust and flexible point-object multi-target tracking using transformer neural networks

Master's thesis in Cybernetics and Robotics
Supervisor: Edmund Førland Brekke
Co-supervisor: Erik Wilthil, Geir Hamre, Rudolf Mester, Tom Arne Pedersen
June 2023

Master's thesis

Christopher Strøm

Towards robust and flexible point-object multi-target tracking using transformer neural networks

Master’s thesis in Cybernetics and Robotics
Supervisor: Edmund Førland Brekke
Co-supervisor: Erik Wilthil, Geir Hamre, Rudolf Mester, Tom Arne Pedersen
June 2023

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Engineering Cybernetics

Abstract

A target tracker is a key component of autonomous navigation. Target trackers can be defined using Bayesian probabilistic modelling, or end-to-end with neural networks. The MT3v1 and MT3v2 are two generations of Transformer-based neural trackers which aim to provide an alternative to Bayesian trackers, by trading online computation cost for offline training, without sacrificing performance. Certain limitations with these two have motivated the development of the MT3v3. This next generation Transformer tracker is shown to perform better than its predecessor and the Bayesian IMM-JIPDA tracker in certain scenarios, while being more flexible and adaptable to new autonomy pipelines. A pre-trained MT3v3 network is provided, alongside a user-friendly implementation. However, it is also shown that the MT3v3 performs very poorly in the worst-case, and struggles to track targets that cross over one another or run in parallel. Since the MT3v3 is a black-box system, it is also difficult to interpret why this is the case. As such, more development work is needed for the MT3v3 to improve reliability and interpretability.

Sammendrag

En målfølger er en nøkkelkomponent i autonom navigasjon. Målfølgingsalgoritmer kan defineres ved hjelp av Bayesiansk probabilistisk modellering, eller ende-til-ende med nevrale nettverk. MT3v1 og MT3v2 er to generasjoner av Transformer-baserte nevrale målfølgere hvis mål er å tilby et alternativ til Bayesianske målfølgingsalgoritmer, ved å bytte ut online beregningskostnad for offline trening, uten å ofre ytelse. Visse begrensninger med disse to har motivert utviklingen av MT3v3. Denne neste-generasjons Transformer-målfølger er vist å yte bedre enn sin forgjenger, samt den Bayesianske IMM-JIPDA-målfølgeren i visse tilfeller, samtidig som at den er mer fleksibel og tilpasningsdyktig til nye autonomisystemer. Et forhåndstrent MT3v3-nettverk er gjort tilgjengelig, sammen med en brukervennlig implementasjon. Det vises dog at MT3v3 presterer veldig dårlig i de verste tilfellene, og har problemer med å følge mål som krysser over hverandre eller kjører parallelt. Siden MT3v3 er et sort-boks system, er det i tillegg vanskelig å tolke hvorfor ytelsen er som den er. MT3v3 trenger derfor mer utviklingsarbeid for å forbedre pålitelighet og tolkbarhet.

Preface

This thesis marks the end of a 5 year long journey through a Master's degree in Cybernetics and Robotics. It is a continuation of my project thesis, and parts of the theory and simulation work is partially borrowed from this.

Neural networks are not necessarily a central component of the Cybernetics field as presented through classes I have taken. Because of this, the decision to make deep learning the main focus of my thesis was somewhat of a risk. My goal in doing so was to add neural networks as yet another tool in my toolbox. As a result, this thesis sits in between the two fields of Bayesian estimation and deep learning with Transformers. The way I present my work is heavily influenced by this.

I would firstly like to thank my main supervisor Associate Professor Edmund F. Brekke for great insight into the sometimes overwhelming field of target tracking. I would also like to thank my co-supervisors – Erik Wilthil (Zeabuz) for the idea of the "CKF" and guidance when developing the detector simulator, Geir Hamre (DNV) and Tom Arne Pedersen (DNV) for useful feedback both in meetings and on my writing, as well as being good mentors throughout my previous time at DNV. Thanks to Professor Rudolf Mester I have arguably improved both how I write and how I structure my work. Another thank you goes out to Audun G. Hem for providing an IMM-JIPDA implementation which saved me a great deal of development work. Lastly, a massive thank you to all my friends from the Vortex NTNU office. Our waffle Fridays, social gatherings, procrastination opportunities, coffee breaks and brainstorming sessions have made this difficult process a lot more manageable!

Christopher Strøm
Trondheim, June 2023

Table of Contents

1 Introduction

The title of this thesis implies one of two things:

1. Transformer-based point-object multitarget trackers do not currently exist
2. There exists Transformer-based point-object multitarget trackers that are not robust and/or flexible

As will become apparent, the latter case is true. The overarching goal of this thesis is thus to develop a Transformer-based point-object multitarget tracker that is robust in scenarios of differing complexity and is flexible enough to be used in various tracking pipelines. This tracker will be a continuation of the development started in (Pinto et al. 2021a) and (Pinto et al. 2022).

1.1 Motivation

Multitarget target tracking (MTT) is a critical aspect of modern autonomous systems, with applications ranging from robotics and driverless vehicles to surveillance and defense (Tian et al. 2019), (Wang et al. 2017). The core challenge of MTT, as the name implies, lies in accurately identifying and tracking multiple targets using measurements that are corrupted by noise. Further complicating MTT is the fact that a target at any given time may not produce a measurement at all, as well as the presence of clutter measurements which originate from sources other than the intended targets. Point-object MTT refers to the problem where targets are assumed to be points in 2D or 3D space, and existing solutions to this problem predominantly employ Bayesian methods and statistical models for describing both the targets and measurements. A counterpart to these solutions are the trackers based on neural networks, such as recurrent neural networks (RNN) or long-short term memory (LSTM) networks. Instead of mathematically modelling MTT as a Bayesian estimation problem, neural trackers are deep learners, trained on large amounts of data to automatically learn the relevant statistical and geometric properties embedded in the measurements.

In recent times, a family of neural networks known as Transformers have emerged as a powerful alternative to architectures like the RNNs and LSTMs. Initially introduced by (Vaswani et al. 2017), transformer networks have revolutionized the field of natural language processing, and their applications have extended to other domains such as computer vision and time series analysis. Transformers excel at identifying both simple and complicated dependencies in the input data, irrespective of their spatial or temporal arrangement, and are especially well suited for handling sequential data.

The MTT problem is inherently a sequential problem as the positions and velocities of targets evolve over time, and sensors observing these targets produce time-series. Given these characteristics, it is reasonable to consider employing Transformers for tackling the MTT problem, as they have demonstrated remarkable performance in other domains. However, existing research on using Transformers for tracking predominantly focuses on camera-based bounding-box tracking, leaving somewhat of a gap in the literature with regards to their application in point-object tracking. This presents an opportunity for further investigation and potential advancements by exploring the capabilities of Transformers in addressing the challenges associated with point-object MTT.

1.2 Problem formulation

(Pinto et al. 2022) ask the question

Can Deep Learning be Applied to Model-Based Multi-Object Tracking?

and implement the MultiTarget Tracking Transformer v2 (MT3v2) as an improvement over the MT3v1 (Pinto et al. 2021a). It is demonstrated that the MT3v2 is capable of matching the performance of state-of-the-art Bayesian trackers in simpler tasks, while outperforming them in more complex tasks. Exploring this was the main topic of the project thesis (Strøm 2022), where it was shown that while the trained MT3v2 networks perform well on the same tasks that they were trained on, crossing tasks and pretrained networks yielded poor performance. Furthermore, the way the MT3v2 was benchmarked did not show the capability of the MT3v2 as a tracker, and the trained networks can only be run in a specific configuration, rendering them unusable for scenarios outside of those used in training. Furthermore, the localization errors for the predictions made by the MT3v2 were significantly higher than the Bayesian tracker counterparts, and the data generator used for training and testing do not necessarily produce bona fide multitarget tracking scenarios.

These observations have motivated a more comprehensive examination and continued refinement of the MT3v2. The aim of this thesis is thus to address some of the limitations and shortcomings of the MT3v1 and MT3v2 with a new architecture, offer additional analysis beyond what is presented in (Pinto et al. 2021a) and (Pinto et al. 2022), and benchmark its performance against other Bayesian methods in a broader range of scenarios.

1.3 Main contributions and thesis outline

The following list is to be considered as the main contributions of this thesis:

1. A thorough review of the MT3v1 and MT3v2 architectures
2. The MT3v3, an improved multitarget tracking Transformer architecture
3. Benchmarking and comparing an MT3-based tracker against a new set of baselines and in new scenarios

All relevant code implementations related to the new MT3v3 architecture, as well as pretrained versions of the MT3v3, will be made publicly available at `https://github.com/chrstrom/mt3v3`.

Chapter 2 presents theory for both the Bayesian and Transformer solutions to the MTT problem, as well formally introducing what is meant by the MTT problem in the first place. The sections related to the Transformer network architecture is partially borrowed and condensed from that which is presented in the project thesis (Strøm 2022). Additions were made to specifically tailor this theory section to explain the architecture changes made to the MT3v2 to produce the MT3v3. **Chapter 3** is dedicated to going in-depth with the trackers MT3v1 and MT3v2, in terms of their architecture changes as compared to the original Transformer, their reported performance, and limitations that plague them. This is also work that was started in (Strøm 2022), but the presentation here is much more thorough and describes issues related to the MT3v2 data generation scheme that was previously not known. Some data obtained by the original creators of these architectures will also be presented and used to generate figures in this chapter. The previous work on the MT3v1 and MT3v2 and the limitations of the respective architectures have motivated the development of the MT3v3, and **chapter 4** presents the changes made in this new architecture. This is by far the most important contribution of the three, and will be the focus on the subsequent chapters. **Chapter 5** describes the setup of the simulators used for training and testing the MT3v3, and how the benchmark pipeline is designed. Results obtained with the MT3v3 are presented in **chapter 6**, and contains various tests and comparisons between different trackers and scenarios. These results are discussed in **chapter 7**, while the conclusion and suggestions for future work is presented in **chapter 8**.

2 Theory

This chapter serves to provide a theoretical foundation that is necessary for formulating the multitarget tracking problem for point-objects and a set of corresponding solutions. Firstly, the mathematical notation used throughout the thesis will be presented. Then, the most important aspects of the Transformer neural networks are explained, since these are required to understand the implications of the changes made to the MT3v3. After this, a short introduction to Bayesian statistics is given, before the multi-target tracking problem itself is defined and explained. Lastly, some possible solutions to the target tracking problem are briefly introduced, as well as some methods and metrics for measuring the performance of a tracker.

2.1 Notation

Vectors and matrices receive bold typesetting, while scalars do not. Measurements and state vectors are denoted $\mathbf{z}$ and $\mathbf{x}$ respectively, and can be subscripted to indicate time. For example, all measurements between time 1 and k is written $\mathbf{z}_{1:k}$. "x" is used as a "placeholder parameter" throughout this section, and unless specified, this has no particular interpretation. Hats are added to variables to indicate that they represent some estimate of the underlying true value, i.e. that $\hat{\mathbf{x}}$ is an estimate of the true $\mathbf{x}$. Any variables related to the neural network section are named according to the context they appear in, and is made explicit as new concepts are introduced.

Most probability distributions will be denoted $p(\cdot)$ and conditional distributions as $p(\cdot|\cdot)$. It is important to remark that the subscript for each pdf is left out for the sake of brevity. I.e. if Z is a random variable and z is a realization, the pdf of Z evaluated at z is denoted as the simplified $p(z)$ as opposed to $p_Z(z)$. Subscript is instead used in some places to avoid ambiguity. A handful of distributions are mentioned throughout, and $\mathcal{U}(\text{lower}, \text{upper})$ is used to denote the uniform distribution, while $\mathcal{N}(\mu, \sigma^2)$ and $\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ are used to denote the single and multivariate

Gaussian respectively. The Poisson distribution is denoted $\mathcal{P}(\lambda)$. $||\cdot||_{(\cdot)}$ is used to denote normalization, where the subscript is used to differentiate normalization schemes like layer-norm (LN) and field-of-view (FOV). All neural networks, both individual components and an entire network from input to output is represented by some mapping f_{name}, where "name" is used to specify what the mapping refers to.

2.2 Transformer neural networks

The Transformer was first introduced in (Vaswani et al. 2017) as a novel architecture designed to address sequence mapping tasks, such as natural language processing, without using recurrent or convolutional layers. Instead, *attention*-mechanisms are employed in order to capture long-range dependencies and effects in the input data while parallelizing computations more efficiently. The success of Transformers in natural language processing has inspired applications in other areas, such as computer vision, speech recognition, generative models, and graph-structured data. This section provides an overview of the most important modules that are used in Transformers.

2.2.1 Preliminary theory

Before presenting the main components that make up a Transformer, it is important to be aware of the fundamental building blocks and techniques employed in each of these.

Feed-forward network

The purpose of a feed-forward network is to map inputs to outputs by learning a representation of the underlying mapping function through a composition of simpler functions (Bishop and Nasrabadi 2006). This is achieved by organizing a set of *nodes* into layers, and connecting the outputs of the nodes in a given layer to the inputs of the nodes of the subsequent layer. The input layer will get its inputs from the data itself, while the output of the output layer is either used directly or fed to some other component of a more complex system. In short, a feed-forward network is simply some learned function $f_{nn}(\cdot)$ between the input and the output. The input and output will depend entirely on the problem that the network is designed to solve, and can be scalar, vectors, or matrices. A single node of the network, also referred to as a neuron or unit, has a set of weights $\mathbf{w}$, a bias b and an activation function f_a. The output of a node is generated by applying its activation function to the sum of its bias and the weighted sum of the inputs. Letting h^l denote the output of the l'th layer, the output of node i in the next layer can be expressed as

$$h_i^{l+1} = f_a(a_i^l + b_i^l) \qquad a_i^l = \mathbf{w}_i^{l\top}\mathbf{h}^l \tag{2.1}$$

and each h_i^{l+1} will be concatenated to form the layer output $\mathbf{h}^{l+1}$. This notation style is consistent with (Ba et al. 2016) and will be used throughout the rest of these sections. If a non-linear activation function is chosen, the network is enabled to learn complex and non-linear mapping functions between the input and output of the network as a whole. Some common activation functions include tanh, sigmoid and ReLU, which are presented in short below.

The hyperbolic tangent (tanh) is a function which maps its input to the range $[-1, 1]$ and is defined as

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{2.2}$$

providing a smooth, differentiable transition between values. The tanh function is often used in situations where it is beneficial to have both positive and negative outputs that are limited in magnitude.

The sigmoid function is defined as

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{2.3}$$

and maps the input to a range between 0 and 1, creating a smooth, differentiable S-shaped curve. The sigmoid function is frequently used when modeling probabilities or binary classification tasks due to the output range. The Softmax function is a sigmoid that is generalized to work with $N > 2$ classes.

The Rectified Linear Unit (ReLU) function is defined as

$$\text{ReLU}(x) = \max(0, x) \tag{2.4}$$

which is a simple function which acts as a gate by making all negative values of x evaluate to 0. The ReLU function has become popular due to its computational efficiency and ability to mitigate the vanishing gradient problem in deeper networks, but may encounter the problem of "ReLU-death", where a node using ReLU produces a gradient that evaluates to zero for all inputs. (Lu et al. 2019). It is also not differentiable at the origin, so some care must be taken in implementation for this.

A fully-connected feed-forward network is a fundamental building block of Transformer networks, which to summarize is a composition of nodes stacked into layers, where each layer is fully interconnected with the next. ReLU is almost exclusively utilized in the FFN modules, while the input and output layers are defined from the specifics of the task the network is supposed to solve.

Learning

The weights and biases of a network are initialized either randomly or with some fixed scheme, which in general will not produce a network that correctly solves the input-output mapping problem (Bishop and Nasrabadi 2006). As such, the weights and biases must be fine-tuned in such a way as to minimize the difference between the expected and actual network output, for a given set of inputs. Doing this manually is infeasible, and so automatic methods have been developed to solve the tuning task. Automatically changing the weights and biases of the network based on the output error of the network can collectively be referred to as the networks learning or training phase. Firstly, the network is used to calculate the output given some input, where the correct output is known. This stage is known as the *forward pass*. Then, the generated output and the corresponding known correct output is used in a *loss function* to calculate a score of how well the network performed. An automatic learning scheme then uses the gradient of the loss function in order to find the direction each weight and bias should be adjusted in order to minimize the loss function. That is, denoting the weights and biases as $\mathbf{W}$ and the loss function as $\mathcal{L}$, then

$$\mathbf{W} \leftarrow \mathbf{W} - \lambda \frac{\partial \mathcal{L}}{\partial \mathbf{W}} \tag{2.5}$$

where λ is referred to as the learning rate of the training process and is a tunable hyperparameter. Calculating the gradient of the loss function is usually done by means of *backpropagation*. If the gradient of the loss function is calculated across all available sets of inputs and known outputs, this method is called batch gradient descent. However, if this training set is very large, it becomes computationally infeasible to calculate the full gradient for every step. As such, stochastic gradient descent is more commonly used. The main difference in stochastic gradient descent is that a single training point is used to calculate the gradient $\mathcal{L}_i$, which is then used to update $\mathbf{W}$. This allows for more efficient learning, but may introduce more noise in the process. Mini-batch gradient descent is also commonly used, where more than one but less than the total training samples are used to calculate the gradient of the loss function.

There is no one-rule-fits-all for determining λ, but different schemes have emerged to adapt the learning rate depending on how the loss function evolves. A commonly used algorithm for this is the Adam optimizer of (Kingma and Ba 2014), which adaptively computes learning rates for different parameters, and in practice requires little to no tuning.

Automatic learning means that explicitly modelling dependencies between the inputs and outputs is not required. This is beneficial for tasks where defining models can be problematic or impossible. However, it also means that most neural network solutions are "black boxes", where it is difficult to interpret why a solution does or does not work.

Residual learning

Residual learning is a concept that simplifies the training process of deep neural network by taking an alternate approach of representing the mapping between inputs and outputs. (He et al. 2016) The idea behind residual learning is to use shortcut connections that skip one or more layers in the network, and in the most extreme case, have a skip connection that goes all the way from the input to the output. This makes the network learn the residual (or difference) between the input and the desired output, rather than the entire mapping directly. One example that illustrates the benefits of residual learning is in the case where identity is the correct mapping between the input and output, i.e. $f_{\text{nn}}(\mathbf{y}) = \mathbf{y}$. With a skip connection that attaches the output directly to the input, the network will simply learn that all weights and biases should be zero, thus making the shortcut connection the only path that affects the output. Without any shortcuts, the network would have to learn to approximate the identity mapping by combining a set of non-linear operations, something which is more complex.

Layer normalization

Layer normalization is a common technique utilized in neural networks in order to speed up the training process and was introduced by (Ba et al. 2016) as an alternative to other normalization techniques. It is motivated by the fact that the output of any given layer is correlated with the output of the previous layer. This in turn yields gradients with respect to the weights of a given layer which highly depend on the outputs of the previous layer. Over the course of training, the output distributions of the hidden layers may shift in sync with one another. This in turn will make it harder for the network to learn an approximation of the true mapping from input to output. Layer normalization approaches this problem by fixing the mean and variance of the input sum for each layer, and is done using the normalization statistics

$$\mu^l = \frac{1}{L}\sum_{i=l}^{L} a_i^l \qquad \sigma^l = \sqrt{\frac{1}{L}\sum_{i=l}^{L}\left(a_i^l - \mu^l\right)^2} \tag{2.6}$$

where L is the number of hidden units in layer l. This can be computed on-demand and has low computational and spatial complexity since it does not require to maintain any history across training steps. Using layer normalization, equation 2.1 becomes

$$h_i^{l+1} = f\left(\left(\frac{g}{\sigma^l}\left(a_i^l - \mu^l\right) + b_i^l\right)\right. \tag{2.7}$$

where g is an additional gain factor that can be tuned.

2.2.2 Input embedding

The very first stage of the Transformer architecture is the input embedding stage, and the purpose of this module is to convert the input data to a consistent format that is usable by the network. This is required since the inputs for a specific Transformer architecture could be anything from english words to measurements from a radar. For example, an English sentence can be translated into a vector of indices representing the position of each word in a dictionary of available words, which in the natural language processing domain is referred to as tokenization (Zhou et al. 2021). Using the example of radar measurements, the raw input already contains real numbers, and so tokenization is not required. However, input embedding is still necessary, since it serves another important purpose, namely to capture and utilize semantic and/or syntactic information in the raw input data. Each input element is mapped to a vector of dimension d_{model}, which is a tunable parameter of the network. In (Vaswani et al. 2017), this is set to 512. The final output of the input embedding for a sequence of length T is thus $T \times d_{\text{model}}$. The mapping from a raw input to its embedding is learned through the training process of the network. It is not known a priori what this mapping will prioritize or learn from the input data, and will vary from task to task. It is also hard to interpret directly, and getting insight into what goes on in this stage simply due to the high dimensionality of the embeddings and the complex interaction effects that that may be embedded.

2.2.3 Positional encoding

Transformers are by default invariant to permutations of the input sequence, which means that temporal dependencies are not inherently learnable by the network. Positional encoding is a technique that injects positional information in the input sequence into the embeddings, and is done before the input of the encoder. This added information enables the model to effectively model dependencies between tokens, even when they are distant from one another in the input sequence. Encodings can be both relative and absolute. Relative encodings capture the relationship between tokens based on their positions relative to one another, while absolute encodings embed information about the global position of each token. There are also multiple different ways to implement these two methods.

Sinusoidal position encoding

Sinusoidal encoding is the method used in the original Transformer in (Vaswani et al. 2017). Position encodings are generated by applying sinusoidal functions to each position index. Specifically, for position p and dimension i, the encoding is calculated as

$$PE(p, 2i) = \sin\left(\frac{p}{10000^{\frac{2\pi}{d}}}\right), \quad PE(p, 2i+1) = \cos\left(\frac{p}{10000^{\frac{2\pi}{d}}}\right) \tag{2.8}$$

and was originally chosen as it was hypothesized that the network would learn to attend to relative positions more easily, since $PE(p+k, i)$ can be expressed as a linear function of $PE(p, i)$ for any fixed offset k. Since the position encodings will have the same dimension as the input embeddings, this method allows the network to extrapolate onto input sequences longer than those seen in training.

Learned position encoding

Learned encoding is an alternative approach to sinusoidal encodings and is also touched upon in (Vaswani et al. 2017). Instead of using some fixed deterministic function, this method initializes a position encoding matrix with values that will be learned in training. In doing so, it may allow the network to learn task-specific positional information that may be more relevant to the problem domain. However, this approach does not generalize well, since the maximum supported input sequence length is determined by the size of the position encoding matrix, here denoted τ. When learning absolute position encodings, the learned matrix will

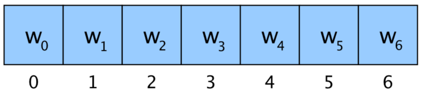
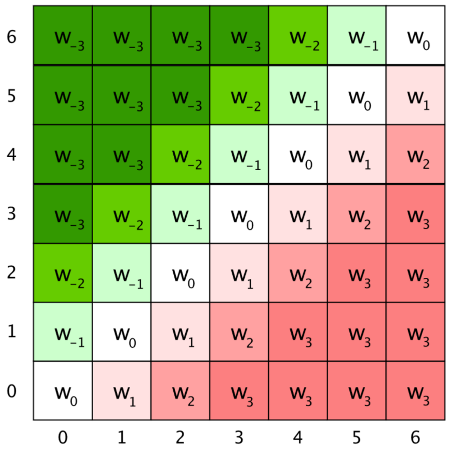


Figure 2.1: An example of the absolute position encoding vector for $\tau = 6$. Original figure is from (Huang et al. 2020).



6: w_{-3} w_{-3} w_{-3} w_{-3} w_{-2} w_{-1} w_0
5: w_{-3} w_{-3} w_{-3} w_{-2} w_{-1} w_0 w_1
4: w_{-3} w_{-3} w_{-2} w_{-1} w_0 w_1 w_2
3: w_{-3} w_{-2} w_{-1} w_0 w_1 w_2 w_3
2: w_{-2} w_{-1} w_0 w_1 w_2 w_3 w_3
1: w_{-1} w_0 w_1 w_2 w_3 w_3 w_3
0: w_0 w_1 w_2 w_3 w_3 w_3 w_3
0 1 2 3 4 5 6

Figure 2.2: An example of the relative position encoding matrix for $\tau = 6$. Note how the same weight is used for elements that are at the same distance from one another. Also note that the same weights are used for multiple distances above a certain threshold. Original figure is from (Huang et al. 2020).

instead be a vector of dimension τ. Relative encodings will be a lookup matrix of size $\tau \times \tau$, where elements along the anti-diagonals will be the same. Figures 2.1 and 2.2 show what this will look like. Note that the lookup matrix displayed here also incorporates an addition effect by defining a threshold that determines if a distance should receive a unique value. I.e. it is possible to define a concept of ”far away”, where all elements that are further apart from one another than this threshold receive the same position encoding.

Time2Vec

Time2Vec is a more recent positional encoding method proposed by (Kazemi et al. 2019) and is, as the name implies, a method specifically adapted to representing the temporal aspect of the input sequence through the position encoding. Time2Vec extends the idea of learned position encodings by introducing both trainable and non-trainable components to the position embedding. In doing so, the idea is to provide richer representations of time or position information in the input data.

Mathematically, Time2Vec is defined as

$$\mathbf{t2v}(\tau)[i] = \begin{cases} \omega_i \tau + \varphi_i, & \text{if } i = 0 \\ \mathcal{F}\left(\omega_i \tau + \varphi_i\right), & \text{if } 1 \leq i \leq k \end{cases} \tag{2.9}$$

where τ is some scalar notion of time, $\mathcal{F}$ is some periodic function, while ω_i and φ_i are learnable parameters. For example, $\mathcal{F}$ could be chosen to be a sinusoid, which makes Time2Vec somewhat resemble the original sinusoidal position encoding. Letting the first dimension be linear in τ also allows the network to capture time-dependent but non-periodic patterns in the input data. Despite having learned parameters, Time2Vec is defined across the dimensions of the input embeddings, meaning that it is not limited to a fixed size of the input sequence. (Kazemi et al. 2019) show promising results when using the position encodings as inputs to the network, instead of adding it to the input embeddings.

2.2.4 Attention

At the very core of the Transformer architecture lies various *attention* modules. Attention allows the Transformer to efficiently process information in the input sequences by only focusing on relevant parts of the input data and selectively attending to different elements based on their relevance. Assessing inputs by relevance is part of what a Transformer learns through the training process. Attention is a powerful mechanism to capture complex dependencies in input data, while remaining computationally feasible for longer input sequences.

Mathematically speaking, the attention mechanism is a nonlinear mapping of a query and a set of key-value pairs. The output can be interpreted as a masking of the input values, where the mask is calculated using the query and key. (Vaswani et al. 2017) introduced the query, key, and value notation, which is conceptually similar to a content retrieval system. In this analogy, the query represents a search term, the keys are a set of items to be searched, and the values correspond to the best matches found between the queries and the keys. Attention mechanisms were explored ahead of the introduction of the Transformer, such as in (Bahdanau et al. 2014) and (Luong et al. 2015), as both additive and multiplicative attention. The original Transformer uses the latter, applied in parallel to a set of queries, keys and values for the sake of efficiency. This mechanism is coined as scaled dot-product attention and takes the form of the function

$$f_{\text{attention}}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \mathbf{M}\mathbf{V}, \quad \mathbf{M} = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d_k}}\right) \tag{2.10}$$

$\mathbf{M}$ in this formulation is the mask that is applied to the values in $\mathbf{V}$, and it becomes apparent why this is a form of multiplicative attention function. The dimension of $\mathbf{K}$ is $d_k \times d_k$ and the scaling factor $\sqrt{d_k}$ is introduced to counteract performance loss that is observed for very high-dimensional matrices. (Vaswani et al. 2017) illustrates this effect by considering a hypothetical case where all components of the queries and keys are i.i.d $\mathcal{N}(0, 1)$. In this case their dot product is $\mathcal{N}(0, d_k)$

which would yield very small gradients for large values of d_k. This in turn would hinder learning when training the network.

The single attention function in equation 2.10 can be used to form a multi-head attention function. It is an extension of the attention mechanism, designed to enable the use of attention to capture different types of relationships in the input data by computing attention independently in multiple subspaces, and concatenating these at the output. The basic idea of multi-head attention is to expand the attention function into several parallel "heads", each with their own inputs. Letting the number of heads be denoted h, the multi-head attention mechanism is thus

$$f_{\text{mha}}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = [\text{head}_1, \dots, \text{head}_h]\, \mathbf{W}^O \tag{2.11}$$

Where $\mathbf{W}^O$ is an additional projection used at the output in order to ensure that dimensions of the final output are correct in terms of how the model is defined. Also note that each head is computed according to

$$\text{head}_i = f_{\text{attention}}(\mathbf{Q}\mathbf{W}_i^Q, \mathbf{K}\mathbf{W}_i^K, \mathbf{V}\mathbf{W}_i^V) \tag{2.12}$$

where the projection matrices $\mathbf{W}_i^Q$, $\mathbf{W}_i^K$ and $\mathbf{W}_i^V$ are all learned and unique for a single head.

By letting $\mathbf{Q} = \mathbf{K} = \mathbf{V}$ the normal attention mechanisms become so-called self-attention mechanisms. In doing so, the mask $\mathbf{M}$ can be interpreted as a score for which elements in the input sequence are important, and how important they are relative to one another. This is a powerful technique to interpret the input embeddings as it allows for capturing dependencies between any two elements in the input sequence, regardless of the distance between them, and to ignore inputs that are deemed to not be important. In the target tracking case, a tracking Transformer could learn to attend to the input by weighing measurements deemed to have come from the same target very heavily with one another, and to ignore clutter.

2.2.5 Architecture overview

In general, a Transformer contains two main modules; the encoder and decoder. The former is responsible for generating an alternative representation of the input sequence, while the latter uses this representation to produce the network output.

Encoder

The purpose of the encoder is to capture information in the form of features and contextual effects that exists in the input sequence. This is done by producing an alternative representation of the input data, referred to as a *continuous representation* or *embeddings*, akin to those produced by the input embedding module. Mapping the input to this continuous representation is done using stack of N identical layers which iteratively process the input data, where N is a configurable hyperparameter. The choice of N affects the capacity of the encoder, potentially allowing the Transformer to capture more complex relationships and patterns in

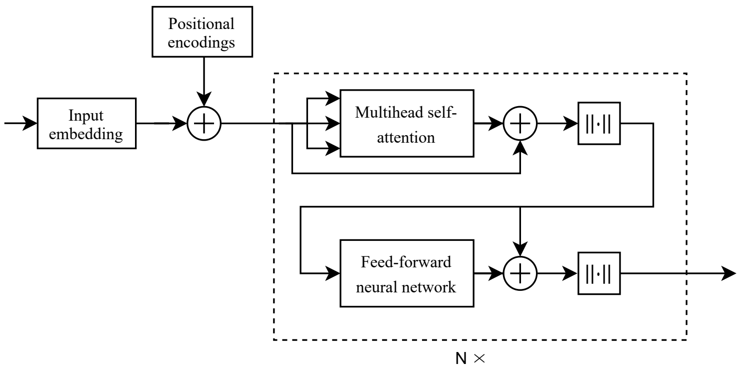


Figure 2.3: A block diagram of the encoder block, based on the figure in (Vaswani et al. 2017). The input data is fed to the input embedding block. $||\cdot||$ represents the layer norm expressed in equation 2.6. Figure is taken from (Strøm 2022).

the input data as N increases. However, a larger N also increases the number of learnable parameters, which in turn comes at the cost of greater computational complexity during inference and an increase in training time.

Each layer in the encoder is composed of two primary sub-layers; multi-head self-attention and a position-wise feed-forward network. The multi-head self-attention block weighs every input token with all the other tokens, and is learned to weigh the relative importance of the different input tokens. By calculating self-attention for multiple heads in parallel, the network can attend to multiple different features of its input. The output of the multi-head self-attention is then passed to a feed-forward network, which is applied independently to each position of the attended input sequence. This is what allows the decoder to learn interactions between the input features. This structure is presented in figure 2.3. Both sub-layers are short-cut, where the input to each sub-layer is added to the output before passing on. This is how residual learning is applied in Transformer networks. Both sub-layer outputs are additionally normalized using the statistics presented in equation 2.6 before being passed along, in order to stabilize training and improve convergence.

Decoder

The decoder is responsible for generating an output sequence based on the encoded input data. Like the encoder, the decoder consists of stacked identical layers. The decoder also uses multi-head self-attention and a feed-forward network in each sub-layer, but also employs an additional encoder-decoder attention block.

The decoder is responsible for generating an output sequence by leveraging the encoded information from the inputs as calculated by the encoder. Similarly to the the encoder, the decoder is composed of a stack of N identical layers, where N is typically the same as the number of encoder layers. Each decoder layer differs from the encoder layer in that it contains three sub-layers instead of two. The first sub-layer is a *masked* multi-head self-attention module, while the second is an encoder-decoder cross-attention mechanism, while the final is a position-wise feed-forward network. Like the encoder, these layers are interconnected by residual connections and followed by layer normalization. A graphical view of this can be seen in figure 2.4.

Masked attention is similar to the multi-head self-attention mechanism used in the encoder but with an added masking mechanism to prevent the model from attending to future positions in the output sequence during training. This is crucial for autoregressive tasks, where the model needs to generate the output sequence one element at a time without access to future information. However, since the ground-truth information is available during training, it is possible to also compute this in parallel. The encoder is connected to the decoder through the cross-attention mechanism. Here, the keys and values are the output of the final encoder layer, while the queries come from the current decoder layer's masked attention module. This allows the decoder to iteratively produce an output sequence while dynamically using information in the encoded inputs. Note that the skip connection in the cross-attention mechanism passes the query matrix along, and not the keys and values from the encoder.

Depending on the specific Transformer task, the final decoder layer is processed in different ways. In the original Transformer architecture, a linear projection is used to map the decoder output back to dictionary indices. This vector is then passed through a softmax function to generate a vector of output probabilities. The word with the highest probability is then typically chosen as the final output.

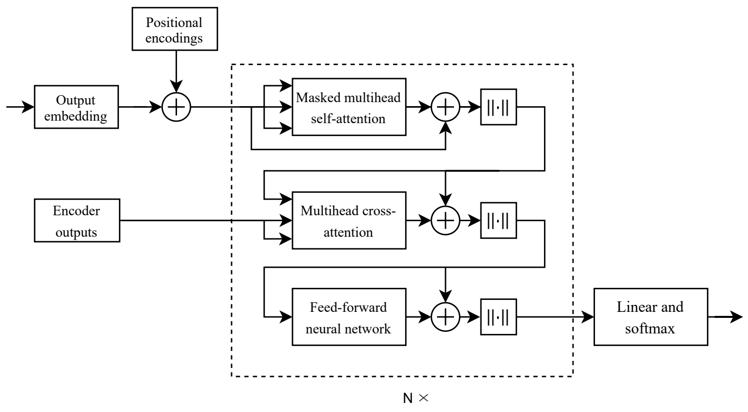


Figure 2.4: A block diagram of the decoder block, based on the figure in (Vaswani et al. 2017). The output embedding block takes right-shifted outputs as its inputs. $||\cdot||$ represents the layer norm expressed in equation 2.6. Figure is taken from (Strøm 2022).

2.3 Bayesian estimation

Bayesian estimation is a powerful probabilistic approach to infer information about stochastic systems by combining prior knowledge with observed data. Systems can be modelled using a wide range of probability distributions, and provides a way of injecting assumptions or priors about how a system behaves from one point in time to another, or the relationships between variables.

2.3.1 The Bayes filter

The problem of estimating some underlying state of a system through noisy measurements can be expressed in general terms as two equations that constitute the Bayes filter (Brekke 2020), under a set of assumptions. Firstly, the process model, i.e. how the system evolves over time, should be on the form

$$p(\mathbf{x}_k|\mathbf{x}_{1:k-1}, \mathbf{z}_{1:k-1}) = p(\mathbf{x}_k|\mathbf{x}_{k-1}) \tag{2.13}$$

which means that when $\mathbf{x}_{k-1}$ is given, then $\mathbf{x}_k$ should be independent of every other prior states and measurements. That is, at time $k-1$, $\mathbf{x}_{k-1}$ holds all the available information in the system about the distribution of $\mathbf{x}_k$. For the measurement model, i.e. how $\mathbf{x}$ maps to $\mathbf{z}$, the assumption is that

$$p(\mathbf{z}_k|\mathbf{x}_{1:k}, \mathbf{z}_{1:k-1}) = p(\mathbf{z}_k|\mathbf{x}_k) \tag{2.14}$$

which can be interpreted in a similar fashion to the process model - that the information about the measurement at time k is entirely contained in the state vector at that time.

Armed with these assumptions, the Bayes filter equation can be stated as a *prediction* step and an *update* step, which will be carried out cyclically. The prediction step is defined as

$$p(\mathbf{x}_k|\mathbf{z}_{1:k-1}) = \int p(\mathbf{x}_k|\mathbf{x}_{k-1})p(\mathbf{x}_{k-1}|\mathbf{z}_{1:k-1})d\mathbf{x}_{k-1} \tag{2.15}$$

while the update step is

$$p(\mathbf{x}_k|\mathbf{z}_{1:k}) = \frac{p(\mathbf{z}_k|\mathbf{x}_k)p(\mathbf{x}_k|\mathbf{z}_{1:k-1})}{p(\mathbf{z}_k|\mathbf{z}_{1:k-1})} \tag{2.16}$$

The result presented in equations 2.15 and 2.16 is very powerful due to its generality and because it provides an optimal way of fusing information between the belief of how a state is propagated and measurements of this state, regardless of what each $p(\cdot)$ actually is. However this design is deceptively simple, for the very same reason it is powerful, as specifying what each distribution should actually be is not trivial (Robert et al. 2007). In particular, there is no guarantee that $p(\mathbf{x}_k|\mathbf{z}_{1:k})$ maintains its shape through multiple prediction and update steps, which is problematic since any implementation of a Bayes filter in all likelihood will execute multiple of these cycles. It is thus common to invoke some assumptions to ensure that the prediction and update steps always end in the same family of probability distributions, which also allows for closed-form solutions to be expressed. The Kalman filter is one such example.

2.3.2 The Kalman filter

The Kalman filter is a specific formulation of the Bayes filter where the state transition and measurement model are assumed to be linear, while the noise terms are assumed to be additive white gaussian. Under these assumptions, equation 2.13 becomes

$$p(\mathbf{x}_k|\mathbf{x}_{k-1}) = \mathcal{N}\left(\mathbf{x}_k; \mathbf{F}\mathbf{x}_{k-1}, \mathbf{Q}\right) \tag{2.17}$$

while equation 2.14 becomes

$$p(\mathbf{z}_k|\mathbf{x}_k) = \mathcal{N}\left(\mathbf{z}_k; \mathbf{H}\mathbf{x}_k, \mathbf{R}\right) \tag{2.18}$$

When these assumptions hold, the prediction and update steps are Bayes optimal, i.e. the estimates $\hat{\mathbf{x}}$ of the true state are as good as they can be. The Kalman filter algorithm works by firstly defining the initial estimate $\hat{\mathbf{x}}_0$ and its corresponding initial covariance $\mathbf{P}_0$ which contains the initial uncertainty of the estimate. The initial covariance matrix is often set to be diagonal.

Prediction

The prediction step for the Kalman filter is perhaps best expressed in terms of the predicted density

$$p(\mathbf{x}_k|\mathbf{z}_{1:k-1}) = \mathcal{N}\left(\mathbf{x}_k; \hat{\mathbf{x}}_{k|k-1}, \mathbf{P}_{k|k-1}\right), \tag{2.19}$$

where

$$\hat{\mathbf{x}}_{k|k-1} = \mathbf{F}\hat{\mathbf{x}}_{k-1}, \qquad \mathbf{P}_{k|k-1} = \mathbf{F}\mathbf{P}_{k-1}\mathbf{F}^\top + \mathbf{Q} \tag{2.20}$$

are the mean and covariance of the prior posterior density $\mathcal{N}\left(\mathbf{x}_{k-1}; \hat{\mathbf{x}}_{k-1}, \mathbf{P}_{k-1}\right)$.

Update

The Kalman update takes the form of the posterior density

$$p\left(\mathbf{x}_k|\mathbf{z}_{1:k}\right) = \mathcal{N}\left(\mathbf{x}_k; \hat{\mathbf{x}}_k, \mathbf{P}_k\right) \tag{2.21}$$

where

$$\hat{\mathbf{x}}_k = \hat{\mathbf{x}}_{k|k-1} + \mathbf{W}_k\boldsymbol{\nu}_k, \qquad \mathbf{P}_k = \left(\mathbf{I} - \mathbf{W}_k\mathbf{H}\right)\mathbf{P}_{k|k-1} \tag{2.22}$$

$\mathbf{W}_k$ is referred to as the Kalman gain, and is obtained from

$$\mathbf{W}_k = \mathbf{P}_{k|k-1}\mathbf{H}^\top\left(\mathbf{H}\mathbf{P}_{k|k-1}\mathbf{H}^\top + \mathbf{R}\right)^{-1} \tag{2.23}$$

while $\boldsymbol{\nu}_k$ is the innovation, i.e. the difference between an actual measurement and the predicted measurement,

$$\nu_k = \mathbf{z}_k - \mathbf{H}\hat{\mathbf{x}}_{k|k-1} \tag{2.24}$$

Although a formal derivation of the Kalman filter equations is not provided here, there is still valuable insight to be gained by examining the prediction and update steps. The assumption of linearity makes the filter computationally efficient and straightforward to implement. However, it may not be suitable for systems with highly nonlinear dynamics or measurement models. In such cases, an extension like the Extended Kalman Filter (EKF) can be used instead. By the assumption of white Gaussian noise, the prediction and update step are simple and will remain simple across multiple iterations, as the product of Gaussians is also Gaussian. Consequently, the state estimate and its covariance can be fully described by the posterior mean and covariance across all steps. Using the Kalman filter in practice reduces down to selecting appropriate values for the elements of $\mathbf{Q}$ and $\mathbf{R}$, which often requires domain knowledge and some understanding of a system's behavior. Measurements will not always come at regular intervals, but due to the two-stage nature of the Kalman filter, multiple prediction stages can run in between updates. Lastly, using a Kalman filter on its own assumes that an incoming measurement is known to have originated from a specified target.

2.4 Multitarget tracking

The core of this thesis revolves around multitarget tracking solutions - but what is the multitarget tracking problem that needs to be solved in the first place? It is essential to define what exactly this question entails, any assumptions that are made, as well as the theoretical and practical concerns that may arise when defining and implementing these solutions.

The multitarget tracking problem as a whole is a fundamental challenge in many fields, such as air- and surface surveillance systems (Blackman 2004), computer vision, and autonomous vehicles (Challa et al. 2011), where the main objective condenses down to estimating the trajectories of multiple maneuvering targets over time. This process involves establishing tracks as they appear, associating measurements to existing tracks, and predicting the motion of tracked targets. This process is complicated by factors such as clutter measurements, measurement noise, false alarms and misdetections, and varying target dynamics.

A possible approach is, as mentioned by both (Brekke 2020) and (Challa et al. 2011) to simply use solutions for the *single* target tracking problem and run them in parallel, which will work perfectly fine when all targets are separated. However, when two or more targets are close to one another, the effectiveness of this approach may degrade substantially. This has motivated the development of true multitarget trackers.

2.4.1 Assumptions and challenges

In order to solidify the somewhat loose terms that have been used so far, a set of assumptions need to be listed. Firstly, a first-order Markov assumption about the evolution of a target state is made. That is, all the information about $\mathbf{x}_k$ is contained in $\mathbf{x}_{k-1}$. Thus, $\mathbf{x}_k$ only depends on $\mathbf{x}_{k-1}$. Furthermore, it is assumed that any measurement $\mathbf{z}_k$ is only dependent on the state $\mathbf{x}_k$, and not any other target state or measurements. These are fairly standard assumptions to make in both the single and multitarget case. The following list of assumptions can be defined:

I. Any target generates at most one measurement, and any measurement comes from at most one target.

II. The state of a live target evolves from one timestep to another according to some model $f_{\mathbf{x}}(\mathbf{x}_k|\mathbf{x}_{k-1})$

III. A target measurement is related to its state according to some model $f_{\mathbf{z}}(\mathbf{x}_k|\mathbf{x}_{k-1})$

IV. A target generates a measurement with probability $P_D(\mathbf{x}_k)$

V. Existing targets survive from timestep *k-1* to k with probability $P_S(\mathbf{x}_{k-1})$

VI. New targets are born according to a PPP[1] with intensity function $\mu(\cdot)$

VII. Clutter measurements appear according to a PPP with intensity function $c(\cdot)$

Many of the challenges associated with multitarget tracking can be expressed in terms of these assumptions. For example, **V** and **VI** express how there will be an unknown number of objects, potentially none at all, at any given time instance. However, because of **IV**, zero measurements does not necessarily mean that there are no targets present. **IV** and **VII** together further implies that the total number of measurements does not perfectly correlate with the number of targets present. Note that while not explicitly stated for the sake of generality, P_D is often times assumed to be constant, and assumption **I** rarely holds on its own. This introduces practical concerns such as measurement clustering, in order to adhere to these assumptions. The alternative would be to lift the assumption entirely, which is what is done for extended object tracking, but this is not considered at all in this thesis.

A challenge unique to multitarget tracking presents itself under these assumptions: Multiple measurements may arise close to multiple targets, and these measurements may both have come from the targets, but also from clutter. As such, there may be many plausible ways of associating measurements and tracks.

Another challenge arises with the models $f_{\mathbf{x}}$ and $f_{\mathbf{z}}$. Aside from the inherent modelling challenge itself, there are some practical concerns that must be taken. Specifically, in order to generate closed-form solution, the structure of these models needs to be maintained through the Bayes' filter update and predictions. These steps contain multiplications between the aforementioned $f_{\mathbf{x}}$ and $f_{\mathbf{z}}$, which limits the choice of models to those that are *conjugate priors*. Conveniently, Gaussians are conjugate priors, and as such it may be beneficial to introduce the further assumptions that the motion and measurements models are linear and Gaussian, which means

$$f_{\mathbf{x}}(\mathbf{x}_k|\mathbf{x}_{k-1}) = \mathcal{N}(\mathbf{x}_k; \mathbf{F}\mathbf{x}_{k-1}, \mathbf{Q}) \tag{2.25}$$

and

$$f_{\mathbf{z}}(\mathbf{z}_k|\mathbf{x}_k) = \mathcal{N}(\mathbf{z}_k; \mathbf{H}\mathbf{x}_k, \mathbf{R}). \tag{2.26}$$

[1]Poisson Point Process. See (Streit and Streit 2010) for details.

which are on a form that correspond to the previously introduced Kalman filter formulation.

2.4.2 Clairvoyant Kalman Filter

The previously mentioned challenges with multitarget tracking arises from the fact that in practice, given a sensor scan $\mathbf{z}_k = \{\mathbf{z}_k^1, \cdots \mathbf{z}_k^n\}$, there are in general no distinguishing features between true and false measurements or any explicit information about which true measurement originated from which target. However, given perfect association information, i.e. the knowledge of which target produced which measurements and which measurements are clutter, the tracking problem collapses to a simple state estimation problem akin to the Kalman filter. This is the motivation behind what has been named the Clairvoyant Kalman filter (CKF). By invoking the Gaussian-linearity assumptions previously presented, and providing ground-truth association information, the CKF is Bayes optimal and will never produce a false detection. The CKF may miss a target, but only in the cases where a spawned target has not yet produced any measurements. This "tracker" will provide the strongest possible baseline which can be used to empirically assess the optimality of other trackers. It is important to note that the CKF is not intended for any practical use, since the required information is rarely ever available - and in the case that this information is available, there is no target tracking problem to solve.

The workflow of the CKF is fairly simple. The event B_k^i denotes that target i produced a measurement for the first time, at timestep k. This does not correspond to the ground truth birth time of the target, but instead the observable birth through the measurements. In a similar manner, the event D_k^i denotes the time a target despawns, but in the ground-truth sense, as opposed to the time at which the last measurement was produced. When B^i occurs, a new Kalman filter KF_i is initialized. The predictor of KF_i will run for every timestep until D^i occurs, while the corrector will run whenever target i produces a measurement. In order to distinguish measurements from one another, $\mathbf{z}_k$ is embedded in the super-measurement $\tilde{\mathbf{z}}_k = [\mathbf{z}_k,\ i]$, where $i \in \{-1\} \cup \mathbb{N}$ denotes a unique identity of the target that the measurement came from, and $i = -1$ is used as an indicator for false measurements.

Again, it is very important to clarify that the CKF is *not* a tracker, in the sense that it does not solve any of the challenges related to the tracking problem. Regardless, the Kalman filter or another formulation of the Bayes' filter is a core component of many multitarget trackers, which in turn makes the CKF the strongest and a reasonable baseline to compare against. Additionally, a tracker that performs better than a CKF may indicate that there is something wrong with how the tracker is being benchmarked, or that the tracker uses additional information outside of what is given to the CKF.

2.4.3 The joint integrated probabilistic data association filter

The Joint Integrated Probabilistic Data Association Filter (JIPDA) was introduced by (Musicki and Evans 2004) and is an extension of the integrated probabilistic data association (IPDA) filter to multiple targets. The IPDA is again an extension of the he probabilistic data association filter (PDAF) to estimate the probability of track existence. This section introduces these concepts and how they build off of one another. However, it does not dive into the details, as this can be found in separate literature. The following subsections are highly based on (Brekke 2020).

The probabilistic data association filter

The PDAF is a target tracker that enforces stricter assumptions than those presented in section 2.4.1. Most notably, it is assumed that there is one and only one target in the sensor view, which is equivalent to a birth intensity of zero and a survival probability of one. Furthermore, it is assumed that P_D is constant. The assumptions of how the target state evolves and how target-oriented measurements are related to the target state remain the same. The problem that the PDAF solves is then to track a single object over time, where an unknown number of clutter measurements and either zero or one target measurement is given for each timestep. "Tracking a single object" means to approximate the density of the target state, $p_k(\mathbf{x}_k)$. While the PDAF is a somewhat dated tracking algorithm, first introduced in (Bar-Shalom and Tse 1975), it serves as a good introduction to the data association problem and how a target tracker differs from algorithms like the Kalman filter, as well as how it is similar. Assuming that a track has already been established, the PDAF can be split into two steps like the Kalman filter, namely the prediction and update steps. In the prediction step, the state of the target at time k is estimated using the state at $k-1$ and a model of the target's motion, again like the Kalman filter. The update step is where the main difference lies. The set of measurements at time k may not contain any measurement belonging to the true target, and it may also contain clutter measurements. To model this, the events $a_k = 1, \dots a_k = m_k$ are defined, where $a_k = 1$ represents measurement "1" originating from the target and so on. Additionally, $a_k = 0$ is used to model that no measurement originated from the target. Then, instead of modelling the measurement-conditional state distribution as $p(\mathbf{x}_k | \mathbf{z}_{1:k})$, it is additionally conditioned on the aforementioned events as $p(\mathbf{x}_k | a_k, \mathbf{z}_{1:k})$. To define the posterior density of the target state, the probability of each event conditioned on the measurements up to time k needs to be defined. These event probabilities are also referred to as the association probabilities. The fundamental difference between an estimator like the Kalman filter and a tracker lies in this measurement-to-track data association problem. These event probabilities incorporate information about the target detection probability, the number of measurements at time k, the predicted target state, as well as the the clutter and measurement models. It is also unreasonable to consider every single measurement at time k when performing the update step. Instead, a *validation gate* is introduced. This is a region of the sensor surveilance region that is usually defined as an ellipse shaped according to the predicted state covariance. This means that only measurements that are in a region where the true target is expected to be are considered for the update step. A detailed explanation of this can be found in chapter 7.3 in (Brekke 2020).

After the data association step, the posterior distribution of the target state is, under the initial Gaussian-linearity assumption, a mixture of Gaussians. This assumption would thus immediately be broken at the next timestep. Mixture reduction is employed to prevent this, and can be considered as the final step of the PDAF update step. In short, mixture reduction in the PDAF involves moment matching the mixture down to a single Gaussian that has the same expectation and covariance as the mixture.

Lastly, the prediction and update steps assume that a track is already established, but exactly how this is done is not part of the PDAF formulation directly. Instead, an ad-hoc track management scheme such as 2/2&M/N can be used. In short, this is a process that starts by establishing a "preliminary track" if two consecutive measurements appear close to one another. This preliminary track is maintained for N steps according to the PDAF steps. If this track received at least M measurements inside its gate through these N steps, it is considered a "confirmed track" and continues to be updated by the PDAF. If not, the preliminary track is killed and the process starts over. Once a track is confirmed it can be maintained using the same test, but inverted. That is, if the track does not receive more than M measurements inside its gate for N timesteps, it is terminated. The M and N for track confirmation and termination are user-defined, and do not necessarily have to be the same. It is worth mentioning that while the PDAF assumes that one single target exists in the surveilance region, it is possible to approximate a solution to the MTT problem by using a set of single-target trackers like the PDAF in parallel. As long as none of the validation gates of the multiple tracks overlap, this can be a very good approximation. However, if they *do* overlap, the performance of this approach may quickly deteriorate. Two immediate improvements that can be made over the PDAF are thus to introduce a more rigid way of dealing with track existence and to properly formulate the data association and update step for multiple targets.

Track existence probability and the IPDA

The integrated PDA differs from the PDAF by including an extra state indicating whether or not a target exists. This existence state evolves over time, and an associated *existence probability* is included in the prediction and update steps. Doing this allows for managing tracks by thresholding. A track starts its life cycle from a measurement with some initial existence probability. If this existence probability reaches above some threshold, it can transition into a confirmed track. Likewise, a confirmed track can be terminated if its existence probability drops below some potentially different threshold. It is worth noting that unlike the 2/2&M/N approach, no decision about track existence *has* to be made. Instead, the existence probabilities simply give a measure of the probability that a track corresponds to a true target – The decision of whether or not to establish a track is not fixed. Details on how to calculate the posterior distribution for the existence probability for a single track can be found in chapter 7.5 of (Brekke 2020). This existence-based approach thus offers more flexibility than the PDAF. However, the IPDA is still formulated under the assumption that at most one target is present.

Joint formulation of the association problem

Moving from the assumption of a single target to multiple targets being present is achieved by modelling the probabilities of all possible assignments of multiple measurements to multiple targets, under the constraint detailed by assumption **I** from section 2.4.1. Like the PDAF, this joint formulation utilizes association hypotheses. However, this is now defined as $a_k = \left[a_k^1, \dots a_k^n\right]$ where $a_k^t = j$ if measurement j is claimed by target t, and $a_k^t = 0$ if no measurement is claimed by target t. When a measurement is "claimed" by a target, it cannot be claimed by another target, as per assumption **I**. As (Brekke 2020) writes, these association hypotheses can be defined as the mapping

$$a_k : \{1, \dots, n\} \to \{0, 1, \dots, m_k\} \ \mid \ a_k^s = a_k^t \neq 0 \ \Rightarrow \ s = t. \tag{2.27}$$

Where the PDAF and IPDA define the pdf $p\left(\mathbf{x}_k | a_k, \mathbf{z}_{1:k}\right)$, the joint formulation used in the JPDA models the joint distribution $p\left(\mathbf{x}_k^1, \dots \mathbf{x}_k^n | a_k, \mathbf{z}_{1:k}\right)$. While the derivation of the final mixture-reduced target state posterior is somewhat more involved than with the single-target trackers, it follows the same steps. Using the event-conditional probabilities, a mixture of Gaussians is constructed for each track. The difference lies in that multiple tracks are defined – The mixture reduction is thus performed per-track, leaving one Gaussian to describe each target. This formulation, in addition to modelling existence probability loosely constitute the inner workings of the JIPDA. The JIPDA will use the same association events as the JPDA, but the numerical values for the association probabilities will be slightly different due to the introduction of the existence probability for each target.

Implementation details

Certain considerations have to be made in order to make these trackers feasible to implement and run in practice, especially for the J(I)PDA (Brekke 2020). Summing across all association hypothesis has exponential complexity, which will quickly become computationally infeasible. As such, simplifications have to be made. The aforementioned gating can help with this, to not consider very unlikely assignments. Track clustering is another mentioned approach, which involves segmenting the measurement space into local clusters and utilizing single-target tracking for each cluster, then switching to the joint formulating whenever validation gates overlap. Lastly, while the total number of hypotheses will grow exponentially, it is likely that most of the probability mass is contained within a small subset of these. Utilizing an algorithm such as the Murty's method allows for the top-k best hypotheses to be extracted, while the rest can be effectively discarded. Lastly, it is worth noting that while the mixture-reduction and hypothesis pruning steps are necessary to run the J(I)PDA, they also discard information which may impact performance.

2.4.4 Interacting Multiple Models

The previous sections have assumed that a single motion and measurement model is defined a-priori. However a single model cannot necessarily describe every possible maneuvering target. As such, multiple models can be defined, as well as a notion of how one model will transition to another. Firstly, a set of M models are defined

$$\begin{aligned} \mathbf{x}_k &= \mathbf{f}^{(s)}\left(\mathbf{x}_{k-1}\right) + \mathbf{v}_k; \quad \mathbf{v}_k \sim \mathcal{N}\left(\mathbf{0}, \mathbf{Q}^{(s)}\right) \\ \mathbf{z}_k &= \mathbf{h}^{(s)}\left(\mathbf{x}_k\right) + \mathbf{w}_k; \quad \mathbf{w}_k \sim \mathcal{N}\left(\mathbf{0}, \mathbf{R}^{(s)}\right) \end{aligned} \tag{2.28}$$

for $s = 1, 2, \cdots M$ as in (Brekke 2020). Then a vector containing the probabilities of each model at time k is defined as

$$\mathbf{p}_k = \left[p_k^{(1)}, p_k^{(2)}, \cdots, p_k^{(M)}\right]^\top \tag{2.29}$$

This probability vector evolves according to $\mathbf{p}_k = \boldsymbol{\pi}^\top \mathbf{p}_{k-1}$, where $\boldsymbol{\pi}$ is a matrix of transition probabilities. Specifically, an element $\pi_{i,j}$ of $\boldsymbol{\pi}$ contains the probability that the target evolves from model i at time $k-1$ to model j at time k.

This modelling approach is called interacting multiple models (IMM), and when designing Bayesian trackers, densities can additionally be conditioned on $s_{1:k}$, although the specifics of this is left out of this theory section. An example where it is very reasonable to use IMM is modelling the flight of commercial planes, as both the standard constant-velocity and coordinated-turn models accurately describe different sections of a flight. A JIPDA tracker that implements IMM can be found in (Brekke et al. 2021).

2.4.5 Transformer Trackers

While the Transformer networks were originally designed to process natural language, their impressive performance on these tasks has inspired adaptations to many other fields, including target tracking. The main approach for leveraging Transformers in object tracking is to do so in an end-to-end fashion (Carion et al. 2020). This refers to a network that can take raw inputs and process them in such a way to generate the final output estimates without any hand-crafted pre- or post-processing. Furthermore, a large majority of Transformer-based target trackers come from the computer vision field, and as such, it is common to see other network architectures such as convolutional or recurrent neural networks introduced in hybrid with a transformer, such as in (Zhu et al. 2020) and (Meinhardt et al. 2022). In these methods, the output will be a single or a set of images, and the output will be a set of bounding boxes for each input. The MT3v1 and MT3v2 from (Pinto et al. 2021a) and (Pinto et al. 2022) are Transformer-only networks which instead operate directly on points in some measurement space, and output a set of estimates in state space.

A key advantage of Transformer-based solution is that they are easily parallelizable, allowing for very efficient training. This feature is especially beneficial when training for complex tasks that require significant computational resources, such as object tracking. Compared to Bayesian tracking methods, which rely on prior assumptions and probabilistic models, Transformer trackers remove the task of explicitly modeling the tracking task. This offers an ease-of-use and the potential for avoiding the need to tune the models to specific tasks, but comes with the downside of needing to train a network offline and design its architecture in such a way as to solve the tracking problem in the first place. Removing the explicit modeling also reduces the explainability of the solution. The online inference time of a Transformer will also not be greatly affected by input length, something which may not be the case with some Bayesian methods.

An end-to-end Transformer tracker will not require any explicit modelling of state distributions or data association like the Bayesian trackers outlined previously. As such it will not require the use of simplification techniques in order to be computationally feasible. Instead, the goal is for the Transformer to be trained well enough to learn this implicitly. A substantial advantage that a Transformer brings is the use of attention, allowing the network to adaptively attend to certain measurements in the input set. However, this also comes with a drawback of being hard to interpret, and requiring a rich training process.

2.5 Performance metrics for tracking algorithms

To assess a tracker or compare multiple trackers against one another, some performance metric or metrics need to be used, which provides insight into what a tracker is doing through numerical data.

2.5.1 Generalized Optimal Sub-Pattern Assignment metric

The mathematical intricacies of the mapping between the estimated targets and the ground-truth targets is not immediately clear, which has lead to the development of various metrics in order to assess the performance of multitarget trackers. One of the most recent proposals is the Generalized OSPA (GOSPA) in (Rahmathullah et al. 2017) which, given two random sets, accounts for the localization errors for matches between the two sets, as well as cardinality differences between the two sets. It is worth noting that while some meaning can be extracted from the GOSPA metric on its own, it is arguably more useful as a comparison tool.

From the definition presented in (Rahmathullah et al. 2017), the GOSPA metric between the estimated set of targets $\mathbb{X}$ and the corresponding ground truth $\mathbb{Y}$ is

$$d_p^{(c,\alpha)}\left(\mathbb{X}, \mathbb{Y}\right) = \left(\min_{\pi \in \Pi_{|\mathbb{Y}|}} \sum_{i=1}^{|\mathbb{X}|} d^{(c)}\left(x_i, y_{\pi(i)}\right)^p + \frac{c^p}{\alpha}\left(|\mathbb{Y}| - |\mathbb{X}|\right) \right)^{\frac{1}{p}} \tag{2.30}$$

for $|\mathbb{X}| \leq |\mathbb{Y}|$, while the order of the input sets are swapped in the case of $|\mathbb{X}| > |\mathbb{Y}|$. Π_n contains all permutations of the set $\{k \in \mathbb{N} : k \leq n\}$ for any n, and any element of Π_n will be a sequence on the form $(\pi(1), \ldots, \pi(n))$. Furthermore, $d^{(c)}(x, y) =$

$\min(c, d(x,y))$ where d is some distance metric for any real x and y. $d^{(c)}(\cdot)$ is referred to as a *cut-off* metric, since its maximum value will be c, irrespective of the values of x and y and the choice of $d(\cdot)$.

GOSPA can be interpreted as the cost of localization error for the assignment between $|\mathbb{X}|$ and $|\mathbb{Y}|$ which minimizes the cut-off localization error plus the cost of cardinality mismatches between the two set. Since the sum of localization errors goes from 1 to $|\mathbb{X}|$, any missed or false target will have the cost $\frac{c^p}{\alpha}$.

In the definition of GOSPA there are three parameters that can be tweaked, c, p, and α, under the constraints $c > 0$, $1 \leq p < \infty$, and $\alpha \leq 2$. The parameter c can be interpreted as the distance at which a ground truth target should be considered missed and when an estimate should be considered a false detection. As p increases, the localization cost of target estimates that are far away from any ground truth object increases, akin to penalizing "outliers". This cost is somewhat limited since targets that are distance c from any ground truth target is deemed unassignable and will thus receive the cost c^p. The cost of the cardinality mismatch $|\mathbb{Y}|-|\mathbb{X}|$ decreases for increasing α, but increases for increasing c and p. (Rahmathullah et al. 2017) argues that $\alpha = 2$ is the most appropriate value for evaluating MTT trackers. In short the argument uses that the cost of a missed target and false detection should be the same, regardless if it has been associated with another element under the permutation set. Under the worst-case assumption that the false detection and missed targets fall outside of the cut-off c for all possible permutations of the input sets, they will both contribute with a cost of c^p. That is, a cost of c^p for an assignment between two input elements.

In doing so, the GOSPA metric in equation 2.30 is expressed using assignment sets instead of permutation sets:

$$d_p^{(c,2)}(\mathbb{X}, \mathbb{Y}) = \left(\min_{\gamma \in \Gamma} \left(\sum_{(i,j)\in\gamma} d(x_i, y_j)^p + \frac{c^p}{2}\left(|\mathbb{X}| + |\mathbb{Y}| - 2|\gamma|\right) \right) \right)^{\frac{1}{p}} \tag{2.31}$$

where γ is a set of pairs of indices on the form (i,j) representing the estimate with index i in $\mathbb{X}$ being associated with the ground truth target at index j in $\mathbb{Y}$. That is, a particular assignment set may be expressed as $\gamma \subseteq \{1, \ldots |\mathbb{X}|\} \times \{1, \ldots |\mathbb{Y}|\}$, while the set of all possible assignment sets is denoted Γ. Every index gets at most one assignment, in order to avoid a single estimate being associated with multiple ground truth targets, and vice versa. The cardinality of γ will maximally be $|\mathbb{X}| \cdot |\mathbb{Y}|$, i.e. the assignment set covering all possible assignments between estimates and ground truths.

2.5.2 Decomposing GOSPA

The elements inside minimization operation in equation 2.32 constitute the cost for one possible assignment between $|\mathbb{X}|$ and $|\mathbb{Y}|$, and can be written as

$$\mathcal{C}_p^c(\gamma) = \sum_{(i,j)\in\gamma} d(x_i, y_j)^p + \frac{c^p}{2}\left(|\mathbb{X}| - |\gamma|\right) + \frac{c^p}{2}\left(|\mathbb{Y}| - |\gamma|\right) \tag{2.32}$$

which makes interpretability immediate, as the overall assignment cost is decomposed into three distinct elements. The first element of this sum is the localization error, i.e. the sum of the p-th order distance between the associated estimates and ground truths. The second element is the cost of misdetections, since $|\mathbb{X}| - |\gamma|$ counts the number of ground truth objects that have not been matched to a target estimate. Similarly, the last element is the cost of false detections, since $|\mathbb{Y}| - |\gamma|$ counts the number of unmatched target estimates. Both missed and false detections contribute with a cost of $\frac{c^p}{2}$, due to the scaling factor used in the cost function. Note that this formulation of GOSPA using $\alpha = 2$ does not use a cut-off distance for localization error. Instead, the minimization over all elements in Γ will find an optimal γ^* which will be a tradeoff between localization error and the summed cost of misdetections and false detections. This tradeoff is weighted using c, meaning that the interpretation of c from before still holds. Lastly, it is worth noting that while $d(x, y)$ can be any measure between x and y, it is both common and reasonable to use euclidean distance for the point-object MTT problem.

2.5.3 Localization error in GOSPA

The localization error in GOSPA need to be properly interpreted, especially when comparing multiple trackers. Firstly, the maximum localization error increases with an increase in the number of detected targets in a scene, and by extension, the number of total targets in the scene. A tracker that performs worse in terms of localization error could still get a lower localization score than another, better performing tracker, if the former correctly identifies fewer targets in the scene. It is possible to normalize with the number of detected targets, but this is not always done. Secondly, if no targets are detected, the localization error will be zero. This can for example affect ensemble averages, by appearing as if the overall localization error is lower than it actually is. Lastly, a state vector often contains elements with different units, for example position in meters and velocity in meters per second. While the state can be considered a single point in state space, and thus GOSPA can be applied, the calculation of localization error no longer makes sense when mixing units. A simple work-around for this can simply be to split the state vector into new vectors, all of which contain elements of the state vector that have the same units, then calculating GOSPA for each of these.

2.5.4 Filter consistency

At the core of the Bayesian trackers, and implicitly in the neural trackers, lies some kind of filter. A filter is said to be consistent if the errors it makes are, on average, well described by the output of the filter. (Brekke 2020). That is, if the filter makes a large error, it should also be less certain about that output. Likewise, if the filter is very certain about a prediction, it should also have a small error. Furthermore, the errors should not be biased, i.e. be acceptable as zero mean and white. These are properties that should be tested for to ensure that the filter is well-behaved.

Normalized estimation error squared

Testing if the filter uncertainty is commensurate with the errors can be done using the normalized estimation error squared (NEES), which for any given timestep is calculated as

$$\boldsymbol{\epsilon}_k = \left(\hat{\mathbf{x}}_k - \mathbf{x}_k\right)^\top \mathbf{P}_k^{-1} \left(\hat{\mathbf{x}}_k - \mathbf{x}_k\right) \tag{2.33}$$

i.e. the square of the estimation error, scaled with the inverse of the uncertainty provided by the filter. NEES can be calculated across N ensemble runs per timestep for the average NEES (ANEES). The ANEES should follow a scaled χ^2 distribution, and confidence intervals at the level of α for the ANEES plotted over time can be expressed as

$$\mathrm{l} = \frac{1}{N} F^{-1}\left(\frac{\alpha}{2}; Nd\right), \quad \mathrm{u} = \frac{1}{N} F^{-1}\left(1 - \frac{\alpha}{2}; Nd\right) \tag{2.34}$$

where F^{-1} is the inverse χ^2 and d is the dimension of $\mathbf{x}$. Ideally, the ANEES plotted over time should generally lie within these limits.

Ljung-Box test

Testing the whole sequence of estimate errors $\mathbf{E} = [e_1, e_2, \cdots, e_T]$ for whiteness can be expressed as the hypothesis test

$$\begin{cases} \mathrm{H}_0 : \mathbf{E} \text{ is a white noise sequence} \\ \mathrm{H}_1 : \mathbf{E} \text{ is not a white noise sequence} \end{cases} \tag{2.35}$$

An approach to test for this is to use the Ljung-Box test of (Ljung and Box 1978). This utilizes the fact that the autocorrelation for a white-noise sequence, i.e. how well the sequence correlates with itself, is 1 using zero offset, and 0 for all other offsets. This offset is also referred to as the lag. For h number of lags, the Ljung-Box statistic is defined as

$$Q = T(T+2) \sum_{k=1}^{h} \frac{\hat{\rho}_k^2}{T-k} \tag{2.36}$$

where $\hat{\rho}_k$ is the sample autocorrelation at lag k. Q is χ^2-distributed, and thus H_0 can be rejected if $Q > \chi^2_{1-\alpha,h}$ for significance level α. This test can also be performed over a set of lags individually, and the hypothesis test can then be performed for each specific lag.

One-sample t-test

The filter can be tested for bias by testing if the sample mean of the state errors is significantly different from zero. The test is formulated as

$$\begin{cases} \mathrm{H}_0 : \mu_{\mathbf{E}} & \text{is } 0 \\ \mathrm{H}_1 : \mu_{\mathbf{E}} & \text{is different from } 0 \end{cases} \tag{2.37}$$

The t-statistic is calculated as

$$t = \frac{\bar{\mathbf{E}} - \mu}{\frac{s}{\sqrt{T}}} \tag{2.38}$$

where s is the sample standard deviation of $\mathbf{E}$ and μ in this case is 0. Again, a significance level α can be set, and H_0 is rejected if $|t| > t_{\alpha/2, T-1}$. The absolute value of t is used since this test is two-tailed, i.e. the mean error has the potential to both be greater than or smaller than zero. $t_{\alpha/2, T-1}$ is the critical t-value for the set significance level and with $T - 1$ degrees of freedom.

3

The Multitarget Tracking Transformers V1 and V2

The MT3v1 and MT3v2 were covered briefly in the previous chapter, but a more thorough examination of these trackers are in order, since these serve as the basis for the MT3v3. This work was partially conducted through the project thesis (Strøm 2022) and is extended further here. While the MT3v1 and MT3v2 are similar in many aspects, there are a few distinctions that are important to point out. This chapter starts by introducing the MT3v1 in detail, as well as its reported performance and how this was measured. Then, the changes that were made to the MT3v1 to produce the MT3v2 are presented, together with its reported performance. Lastly, an overview of limitations and possible improvements for each of these architectures are discussed.

3.1 The MT3v1

The first MT3 architecture was initially presented in (Hess and Ljungbergh 2021) as the Multi-Object Tracking Transformer (MOTT) and was later used in the publication (Pinto et al. 2021a) where it was given the name MT3. To use a consistent naming convention, this will henceforth be referred to as the MT3v1.

3.1.1 Architecture

The MT3v1 is at its core an adoption of the DETR architecture presented in (Zhu et al. 2020), modified to work with point-objects and to perform multitarget tracking. In short, it is a Transformer architecture that takes a set of measurements, as its input and produces an output set consisting of estimated target states. The input is denoted $\mathbf{Z}_{T-\tau:T} = \left[\mathbf{z}^1_{T-\tau}, \cdots, \mathbf{z}^i_{T-\tau}, \cdots, \mathbf{z}^1_T, \cdots, \mathbf{z}^j_T\right]$ and consists of all the measurements received between the current time T and τ time in the past. The superscript for each measurement denotes the index for a single point measurement received at some specific time, and note that the number of measurements received at any given timestep is in $\mathbb{N}$, and is in general not the same for every timestep. A single measurement is assumed to be on the form $\mathbf{z} = [x\ y]^\top$, which implies that the sensor generating these measurements operates in the cartesian plane. Denoting the MT3v1 as $f_{\text{MT3v1}}(\cdot)$, the output is $\hat{\mathbf{X}}_T = f_{\text{MT3v1}}(\mathbf{Z}_{T-\tau:T})$. Each element of the output is on the form $\hat{\mathbf{x}}_{T,i} = [x_i\ y_i\ p_i]^\top$ where $i \in \{1, \ldots, 16\}$. The x_i and y_i are the estimated cartesian coordinates of a single object, and p_i denotes the probability that this estimate belongs to a true target. All estimates with a $p_i > \alpha$ are considered to be true, and the default value for this is $\alpha = 0.9$.

Encoder

The mapping $f_{\text{MT3v1}}(\cdot)$ is a two-stage encoder-decoder process. The encoder maps the input set to some learned embedding vector $\mathbf{e} = f_{\text{enc}}(\mathbf{Z}_{T-\tau:T})$. The design of f_{enc} is similar to that of (Vaswani et al. 2017), where a stack of identical encoder layers are used to calculate $\mathbf{e}$. The raw input sequence is firstly embedded and injected with positional information, denoted $\mathbf{z}' = f_{\text{ie}}(\mathbf{Z}) + \mathbf{p}$. The input embedding f_{ie} is referred to as the *pre-processor* of the MT3v1, since it converts the input measurements to a format that is usable in the encoder, and consists of a learned linear transformation and a normalization factor. The normalization factor ensures that the x and y coordinates of each measurement are mapped to the range $[-1, 1]$, under the assumption that the maximum and minimum values x and y can attain are the same. The positional encoding $\mathbf{p}$ in the MT3v1 is a learned vector that does not depend on the input sequence. The preprocessed measurements are taken as input to the first encoder layer, which produces the first embedding $\mathbf{e}^1$ through the two stages

$$\begin{aligned} \mathbf{z}'' &= \|\mathbf{z} + f_{\text{mha}}(\mathbf{z}', \mathbf{z}', \mathbf{z}')\|_{\text{LN}} \\ \mathbf{e}^1 &= \|\mathbf{z}'' + f_{\text{nn}}(\mathbf{z}'')\|_{\text{LN}} \end{aligned} \tag{3.1}$$

where $\|\cdot\|_{\text{LN}}$ is the layer normalization function from (Ba et al. 2016), f_{mha} is the multihead attention function from equation 2.10, and f_{nn} is a two-layer feed-forward network with a ReLU activation in between Any subsequent encoder layer l will take $\mathbf{e}^{l-1}$ as its input instead of $\mathbf{z}'$. Lastly, dropout is employed in all layers of the encoder. The encoder is configurable using the parameters described in table 3.1, which also contains the specific values used in the MT3v1.

Decoder

The MT3v1 decoder is based on the DETR decoder of (Zhu et al. 2020) and is on the form $X = f_{\text{dec}}(\mathbf{e}, \mathbf{o})$, where $\mathbf{e}$ is the output of the final encoder layer, and $\mathbf{o}$ are a set of *object queries*. Like the encoder, the MT3v1 decoder consists of a stack of identical layers, and can be formulated in a similar manner, although with three

Parameter	**Value**	**Description**
d_model	256	Dimension of the embedding vector
n_heads	8	Number of attention heads in f_{mha}
n_layers	6	Number of encoder layers for f_{enc}
dim_feedforward	2048	Dimension of the hidden layer in f_{nn}
dropout	0.1	Dropout probability used in all layers of the encoder

Table 3.1: The configurable parameters of the MT3v1 encoder

stages instead of two:

$$\begin{aligned}\mathbf{y} &= \|\mathbf{o}_{1:k} + f_{\text{mha}}\left(\mathbf{o}_{1:k}, \mathbf{o}_{1:k}, \mathbf{o}_{1:k}\right)\|_{\text{LN}} \\ \mathbf{y}' &= \|\mathbf{y} + f_{\text{mha}}\left(\mathbf{y}, \mathbf{e}, \mathbf{e}\right)\|_{\text{LN}} \\ \mathbf{d}^1 &= \|\mathbf{y}' + f_{\text{nn}}\left(\mathbf{y}'\right)\|_{\text{LN}}\end{aligned} \tag{3.2}$$

Also note that the second multi-head attention function takes two arguments, meaning that the keys and values are set to $\mathbf{e}$, while the queries are set to $\mathbf{y}$. The first novelty of the MT3v1 architecture is found in how the object queries are calculated, and is referred to as the *selection mechanism.* While the DETR decoder learns a static set of object queries to use as the starting point of the decoder, the MT3v1 selection mechanism uses the encoder embeddings to select a subset of the input sequence to use as object queries. That is, object queries are calculated from $\mathbf{o} = f_{\text{sel}}\left(\mathbf{e}\right)$. The selection mechanism f_{sel} is structured as follows. Firstly, the encoder embeddings are used to calculate scores $\mathbf{m}$ and adjustments $\boldsymbol{\delta}$ as

$$\begin{aligned}\mathbf{m} &= \text{softmax}\left(f_{\text{nn}}\left(\mathbf{e}\right)\right) \\ \boldsymbol{\delta} &= f_{\text{nn}}\left(\mathbf{e}\right)\end{aligned} \tag{3.3}$$

Then, the scores are used to calculate the indices of which elements of the input sequence to extract:

$$\mathbf{Z}_{\text{top}} = \mathbf{Z}_{\mathbf{r}}, \quad \mathbf{r} = \text{Top-k}\left(\text{argsort}\left(\mathbf{m}\right)\right) \tag{3.4}$$

which are added to the adjustments in order to produce the final object queries

$$\mathbf{o}_{1:k} = \mathbf{Z}_{\text{top}} + \boldsymbol{\delta} \tag{3.5}$$

The idea of the selection mechanism is thus to provide the decoder with a good starting point by finding promising measurements in the input sequence. (Pinto et al. 2021a) explains the additional adjustments as allowing for more flexibility, in the sense that the object queries are not restricted to being a transformation of only the selected measurements. Figure 3.1 shows the selection mechanism used in the MT3v1.

Each decoder layer does not produce output estimates directly, but instead the adjustment or offset $\boldsymbol{\Delta}^l$ that can be applied to $\mathbf{Z}_{\text{top}}$. This is referred to as *iterative refinement*, and is a technique introduced in (Zhang et al. 2019). The adjustment

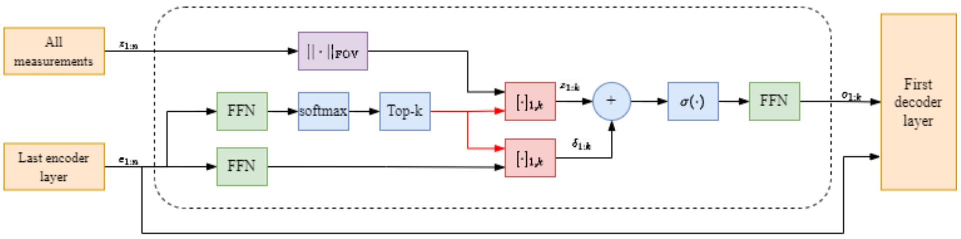


Figure 3.1: The selection mechanism of the MT3v1. Figured based on (Pinto et al. 2021a) but expanded with some details that were left out in their figure.

Parameter	Value	Description
d_model	256	Dimension of the embedding vector
n_heads	8	Number of attention heads for f^1_{mha} and f^2_{mha}
n_layers	6	Number of decoder layers for f_{dec}
dim_feedforward	2048	Dimensions of the hidden layer for all f_{nn}
dim_state_ref	128	Dimension of the single-layer network used to calculate state refinements
dim_obj_sel	128	Dimension of the single-layer network used to calculate object queries
dim_ext_prob	128	Dimension of the single-layer network used to calculate existence probabilities
dropout	0.1	Dropout probability used in all layers of the decoder

Table 3.2: The configurable parameters of the MT3v1 decoder. Note that d_model used here needs to be the same as in the encoder

at layer l is computed as $\boldsymbol{\Delta}^l = f_{\text{nn}}\left(\mathbf{d}^l\right)$ and is used together with $\mathbf{Z}_{\text{top}}$ and all previous adjustments in order to produce the estimates

$$\hat{\mathbf{X}}^l_T = \mathbf{Z}_{\text{top}} + \sum_{n=1}^{l} \boldsymbol{\Delta}^n \tag{3.6}$$

The output estimate of the final decoder layer is then used as the output $\hat{\mathbf{X}}_T$ of the MT3v1. The corresponding existence probabilities are not iteratively refined, but instead calculated directly at each layer as $\mathbf{p}^l = f_{\text{nn}}\left(\mathbf{d}^l\right)$.

An important distinction between the MT3v1 decoder and the original Transformer architecture lies in the way they generate outputs. While the original Transformer is autoregressive, producing outputs one element at a time conditioned on previously generated outputs, the MT3v1 decoder produces all outputs in parallel. The primary advantage of generating outputs in parallel is a speedup in both training and inferencing. Since all outputs are generated simultaneously, the decoding time remains approximately the same for all sequence lengths. Since the MT3v1 decoder is not autoregressive, it is like the encoder, permutation invariant in $\mathbf{e}$. Thus, the accuracy of the final output is conditioned on how good $\mathbf{o}$ and $\mathbf{Z}_{\text{top}}$ are. For example, if all of the measurements in $\mathbf{Z}_{\text{top}}$ lie far away from the true targets in a scene, it is unreasonable to expect that the output estimates $\hat{\mathbf{X}}_T$ are good.

3.1.2 Loss function

The loss function used by (Pinto et al. 2021a) is a the total sum of two different types of losses, a log loss and a contrastive loss

$$\mathcal{L} = \mathcal{L}_L + \mathcal{L}_C \tag{3.7}$$

The log loss is based on the localization error given the best matching between ground-truth targets and the MT3v1 output estimates, together with the probabilities of missed and false targets. In short, the optimal matching is solved using the Hungarian algorithm of (Kuhn 1955) and a matching loss is computed. This is combined in the final log loss function, which in addition to the matching loss penalizes missed and false targets.

As covered in detail in both (Khosla et al. 2020) and (Tian et al. 2020), contrastive loss is a method to of learning by comparing similar and dissimilar samples. In the context of the MT3v1, similar samples could mean samples that are closeby in measurement space, or two measurements that are both either clutter or originated from the same target. Dissimilar samples could be a clutter measurement and a target measurement. In the contrastive loss function, the ground-truth knowledge of which target a measurement came from, or if it is clutter, is used to define a loss that makes the measurements from the same target produce points in embedding space that are close to one another. This loss is weighted with a configurable parameter α which determined the overall importance of this loss. It is also referred to as contrastive *auxiliary* loss, since this loss introduces an auxiliary task aside from the main task of estimating the ground truth positions of targets, in the form of learning to separate which measurements are clutter, and which measurements originated from which target.

Also note that instead of computing the loss at the final decoder output, each intermediate decoder output produces its own loss, and the final loss for a given input is the sum of the losses from each decoder layer. This was done since it both accelerated learning and improved the final performance of the MT3v1.

3.1.3 Reported performance

The MT3v1 is tested by (Pinto et al. 2021a) on two different tasks, a simple "Task 1" and a more complex "Task 2". Higher complexity means more clutter, lower detection probability, more process and measurement noise, and a higher probability of new targets spawning. The parameters for these tasks are presented in table 3.3 compared against two other trackers, the PMBM and δ-GLMB. The other two trackers serve as a strong baseline for performance, but the intricacies of these trackers are not covered here. Scores for the three trackers across the two tasks are presented in table 3.4. These results were obtained in (Pinto et al. 2021a) using 1000 Monte Carlo simulations and averaging the scores. For each simulated scenario, measurements from 20 timesteps are simulated, and the GOSPA scores for the three trackers at the final timestep are reported. Note that confidence intervals are not provided. In (Pinto et al. 2021a) it is argued that the MT3v1 achieves good tracking performance when comparing against the other baselines,

Task	λ_0	p_d	λ_c	σ_q	σ_z
1	4	0.9	0.05	0.5	0.1
2	6	0.8	0.075	0.9	0.3

Table 3.3: Parameters for the tasks the MT3v1 is tested on. λ_0 is the average number of initial objects, p_d is the probability of detection, λ_c is the clutter intensity, σ_q is process noise, and σ_z is measurement noise.

Task	**Method**	**GOSPA**	**Localization**	**Missed**	**False**
1	PMBM	**1.267**	0.102	0.632	0.195
	δ-GLMB	1.863	**0.098**	1.137	0.335
	MT3v1	1.277	0.141	**0.528**	**0.094**
2	PMBM	4.075	0.3025	3.225	**0.163**
	δ-GLMB	4.450	**0.280**	3.515	0.323
	MT3v1	**3.662**	0.3767	**1.995**	0.364

Table 3.4: Performance of the MT3v1 on tasks 1 and 2, compared to baselines as reported in (Pinto et al. 2021a)

and that it excels in reasoning about probable associations between measurements and targets, while regressing the state estimates could be improved. The reported performance is also for two separate architectures, each trained on data generated using the task-specific parameters, although these pretrained architectures have not been made public by the authors.

3.2 The MT3v2

The MT3v2 was introduced by (Pinto et al. 2022) and is, as the name implies, an improvement over the MT3v1. This section will present these improvements.

3.2.1 Architecture

The architecture of the MT3v2 encoder is identical to that of the MT3v1. However, instead of assuming cartesian input measurements, it is assumed that each measurement in Z is on the form $\mathbf{z} = [r\ \dot{r}\ \theta]^\top$, i.e. range, doppler and bearing. This also changes the normalization factor in the input embedding to

$$\mathbf{s} = [r_{max} - r_{min} \quad \dot{r}_{max} \quad \theta_{max} - \theta_{min}]^\top \tag{3.8}$$

which is used for element-wise division for each measurement $\mathbf{z}$.

The selection mechanism introduced in the MT3v1 is expanded upon in the MT3v2 architecture. Firstly, the softmax function is replaced with an element-wise sigmoid. This is a sublte change, but increases the number of possible output configurations by removing the restriction that the outputs must sum to 1. Two feed-forward networks are also added. Where the MT3v1 selection mechanism calculates the decoder object queries as $\mathbf{Z}_{\text{top}} + \boldsymbol{\delta}$ where $\mathbf{Z}_{\text{top}}$ are used as the starting point for the decoder's iterative refinement, the MT3v2 selection mechanism computes the decoder object queries and corresponding positional encodings as

$$\begin{aligned} \mathbf{o} &= f_{\text{nn}}^{1}\left(\mathbf{e}_{\text{top}}\right) \\ \mathbf{q} &= f_{\text{nn}}^{2}\left(\mathbf{e}_{\text{top}}\right) \end{aligned} \tag{3.9}$$

while the starting point of the iterative refinement is $\mathbf{Z}_{\text{top}} + \boldsymbol{\delta}$. This approach makes the distinction between decoder object queries and initial estimates clear, while also injecting more information in the decoder in the form of the learned positional encodings which differ from those used in the encoder.

While the decoder structure of the MT3v2 is mostly kept the same as the MT3v1, its functionality is expanded in order to produce outputs on the form $\hat{\mathbf{x}} = [\boldsymbol{\mu},\ \boldsymbol{\Sigma},\ p]$ where $\boldsymbol{\mu} = [x,\ y,\ v_x,\ v_y]$, $\boldsymbol{\Sigma}$ is a diagonal covariance matrix, and p is the corresponding existence probability. Thus, each element of the estimate $\hat{X}_T$ can be interpreted as multi-Bernoulli. As such, the decoder becomes somewhat more complex since the measurement space and state-space differs from one another, and due to the fact that the decoder simply needs to predict more variables. In order to achieve this, the decoder firstly projects the initial estimates from measurement to state space, producing the initial state estimates

$$\boldsymbol{\mu}^0 = [\mathbf{r} \cdot \cos(\boldsymbol{\theta}),\ \mathbf{r} \cdot \sin(\boldsymbol{\theta}),\ 0,\ 0] \tag{3.10}$$

and it is these estimates that are iteratively refined in the decoder, in the same manner as in the MT3v1:

$$\boldsymbol{\mu}^l = \boldsymbol{\mu}^0 + \sum_{n=1}^{l} \boldsymbol{\Delta}^n \tag{3.11}$$

Unlike the state estimates, the associated covariance matrices and existence probabilities are not iteratively refined, but calculated directly from the decoder inputs

$$\begin{aligned} \boldsymbol{\Sigma}^l &= \text{Diag}\left(\text{Softplus}\left(f_{\text{nn}}^{l,1}\left(\mathbf{o}^l\right)\right)\right) \\ \mathbf{p}^l &= \text{Sigmoid}\left(f_{\text{nn}}^{l,2}\left(\mathbf{o}^l\right)\right) \end{aligned} \tag{3.12}$$

3.2.2 Loss function

Comparing to the MT3v1, the MT3v2 uses the same contrastive auxiliary loss function, but formulates the log loss in a more sophisticated manner using an approximation of the expected sum of the negative log-likelihood of the output estimates, which is based off of (Pinto et al. 2021b).

3.2.3 Reported performance

The MT3v2 is benchmarked against the same trackers as the MT3v1, but the testing scheme is somewhat more involved. Firstly, four tasks are tested instead of two, each with a separately trained architecture specific to a single task. The first two

Task	λ_0	p_d	λ_c	σ_q	λ_b
1	2	0.95	$4.4 \cdot 10^{-3}$	0.2	$1.3 \cdot 10^{-4}$
2	6	0.7	$2.6 \cdot 10^{-2}$	0.9	$3.5 \cdot 10^{-4}$
3	2	0.95	$4.4 \cdot 10^{-3}$	0.2	$1.3 \cdot 10^{-4}$
4	6	0.7	$2.6 \cdot 10^{-2}$	0.9	$3.5 \cdot 10^{-4}$

Table 3.5: Parameters for the tasks the MT3v2 is tested on. λ_0 is the average number of initial objects, p_d is the probability of detection, λ_c is the clutter intensity, σ_q is process noise, and λ_b is the birth intensity.

tasks use a diagonal measurement covariance matrix, while the final two use a more realistic radar model where measurement noise increases rapidly and non-linearly near the edges of the radar field-of-view. Aside from the different measurement noise model, tasks 1 and 3 otherwise use the same data generation parameters - likewise for task 2 and 4. Task 1 and 3 are considered to be simpler due to a higher probability of detection and survival , fewer clutter measurements and average number of starting targets, and a lower target birth intensity. These parameters are presented in table 3.5, while the reported GOSPA scores in (Pinto et al. 2022) are repeated here in table 3.6. The total GOSPA score for each tracker is reported with 95% confidence intervals. This allows for the conclusion that the MT3v2 performs as good or better than the baselines with significance. Furthermore, the MT3v2 evaluation pipeline includes the negative log-likelihood loss described in (Pinto et al. 2021b) for the MT3v2 and the best-performing baseline out of the two model-based trackers, which also shows the MT3v2 performing comparatively better than the baseline as the task complexity increases. The authors attribute this to the MT3v2 being better at handling complex associations many timesteps in the past more effectively. Lastly, the missed rates of the PMBM and MT3v2 are plotted in different sectors of the sensor FOV, which shows that the MT3v2 is better at handling the extreme nonlinearities that appear at the edges of the FOV for task 3 and 4. As was the case with the MT3v1, the MT3v2 performs the worst in terms of localization error, which in (Pinto et al. 2022) is attributed to the diagonal covariance matrices $\mathbf{\Sigma}^l$ and too little training.

The selection mechanism of the MT3v2 is somewhat more involved, as it generates adjusted measurements and position encodings, in addition to the object queries. As figure 3.2 shows, the adjusted measurements are instead used directly in the first decoder layer, while the object queries are generated directly from the top-k embeddings. The corresponding position encodings are also calculated from these.

In addition to the reported estimate performance, the MT3v2 is also benchmarked in terms of computational complexity, and is recited here in table 3.7. Due to the MT3v2 decoder operating in parallel, it is expected that inference time does not increase by any orders of magnitude as the task complexity increases. However, the much faster online inference times comes at the cost of offline training time. Individual MT3v2 architectures were trained for the four different tasks, each for

Task	Method	GOSPA	Localization	False	Missed
1	PMBM	3.44 ± 0.18	2.24	0.12	1.07
	δ-GLMB	3.84 ± 0.20	2.13	0.06	1.64
	MT3v2	3.46 ± 0.18	2.32	0.12	1.01
2	PMBM	19.03 ± 0.53	6.40	0.57	12.06
	δ-GLMB	20.63 ± 0.61	5.56	0.23	14.83
	MT3v2	17.03 ± 0.46	8.12	0.96	7.95
3	PMBM	7.27 ± 0.30	3.93	0.10	3.24
	δ-GLMB	7.42 ± 0.30	3.21	0.70	3.79
	MT3v2	6.01 ± 0.27	3.50	0.21	2.29
4	PMBM	26.72 ± 0.71	2.08	0.02	24.61
	δ-GLMB	27.43 ± 0.71	3.26	0.02	24.14
	MT3v2	22.82 ± 0.56	8.88	0.57	13.37

Table 3.6: The reported performance of the MT3v2 for each task, compared to baselines, as reported in (Pinto et al. 2022).

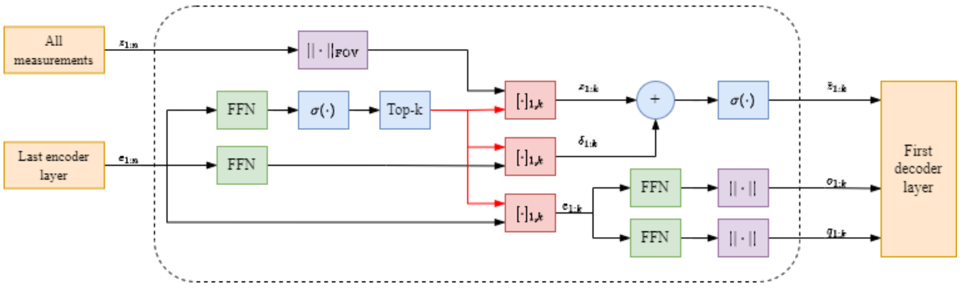


Figure 3.2: The selection mechanism of the MT3v2. Figured based on (Pinto et al. 2022) but expanded with some details that were left out in their figure.

Task	Algorithm	Inference time (s)
1	PMBM	4.92
	δ-GLMB	13.14
	MT3v2	0.03
2	PMBM	126.80
	δ-GLMB	217.66
	MT3v2	0.04
3	PMBM	13.90
	δ-GLMB	40.40
	MT3v2	0.04
4	PMBM	310.14
	δ-GLMB	781.00
	MT3v2	0.06

Table 3.7: Inference times for the MT3v2

approximately 4 days. Unlike the MT3v1, these pretrained architectures are made available online [1].

3.3 Problems and possible improvements

3.3.1 MT3v1 specific

A clear disadvantage of the MT3v1 architecture is that the decoder does not produce velocity estimates. This will prevent the MT3v1 from being used in any systems where velocity estimates are expected further down the pipeline, without having to resort to some ad-hoc solution using only the position estimates. Likewise, the MT3v1 comes with the disadvantage of not estimating uncertainties for the output components either. This limits confidence assessment available to the user of the MT3v1, whether that be a human or another system downstream.

The MT3v1 assumes that the input measurements are cartesian, which is not a problem or limitation in and of itself. However, it it is important to recognize that sensors like lidars, radars and cameras that are commonly used in tracking pipelines generally do not output raw measurements in the cartesian plane.

This could be remedied by simply transforming each measurement to the appropriate space, but this comes with its own problem, namely the normalization factor $\mathbf{s}$. Since this is defined entirely by the two parameters s_{upper} and s_{lower}, the sensor surveilance region is assumed to be square. If a sensor measuring range can detect targets further than $\frac{s_{\text{upper}} - s_{\text{lower}}}{2}$ range away, some measurements may be normalized outside of the expected range of -1 to 1, which in turn will lead to degraded performance in these scenarios. In the case of a sensor measuring range

[1]https://chalmersuniversity.app.box.com/s/kuf5wpjy8ufemy8ynpak7rrfh9p2exft

and bearing, the remedy for this will be to define the sensor field of view as an inscribed square defined by a circle with radius equal to the max sensor range and its origin at the sensor position. The drawback of this approach is that a section of the real sensor field of view will be left unused.

3.3.2 MT3v2 specific

In addition to range-bearing measurements, the MT3v2 assumes that the input measurements contain a doppler component. The distributions of range, doppler and bearing in the simulated scenarios used by (Pinto et al. 2022) for training and testing are shown in figures 3.3, 3.4 and 3.5 respectively. These histograms confirm that the data generation method outlined by the authors is correct. However, it also reveals that the doppler measurements for true targets and clutter are very different. It is thus likely that the MT3v2 quickly learns to rely on the doppler measurements when discriminating between true and false measurements, instead of through likely associations between tracks and measurements. The other trackers used as grounds for comparison are explained to have been modified to support doppler measurements following the suggestions of (Crouse 2014). However, there is no explicit measurement classification that is performed in the baselines. Thus, it is difficult to establish that the MT3v2 is a *tracker* that performs as well as the baselines, and it might be more correct to refer to it as a filter that distinguishes between true and false measurements. However, due to the MT3v1 performing well without the use of doppler measurements, it may still be interesting to design and test an architecture that uses range and bearing, but no doppler. This would also constitute a tracker that can be used with sensors that do not provide doppler measurements, and it is hypothesized that such a tracker would generalize better, since it could not rely on the distribution of the measurements at all in order to distinguish between targets and clutter.

The generalizability of the MT3v2 is something that was explored in (Strøm 2022), by crossing the four available pretrained architectures and the four tasks. Each architecture produced GOSPA scores that were significantly worse for the tasks it was not trained on. While this is a reasonable result for the tasks that used a different measurement model, it is a disadvantage of the MT3v2 that it is so sensitive to changing clutter and noise levels. There is also an interesting discussion to be had about how mismatch between tuning and the parameters used in simulated data for the baseline trackers should be used to compare mismatches between training data and simulated data, but this is delayed until chapter 7.

3.3.3 Common problems

Since the two architectures share many traits, and their training and evaluation loops are very similar, certain limitations and problems apply to both architectures.

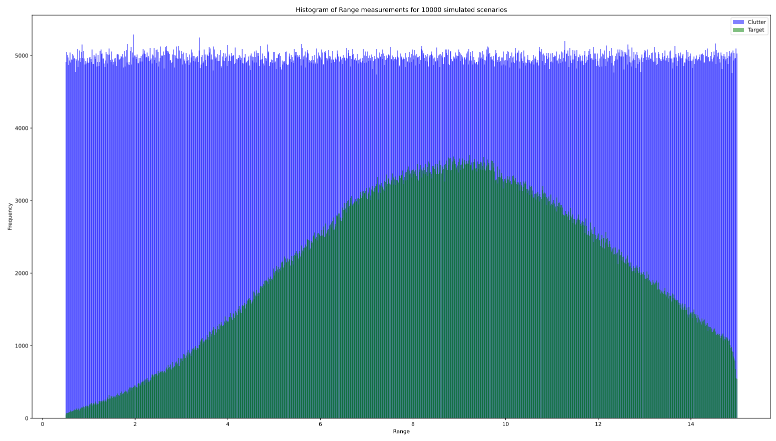


Figure 3.3: Histogram of range measurements from targets and clutter from the data generator used to train and test the MT3v2.

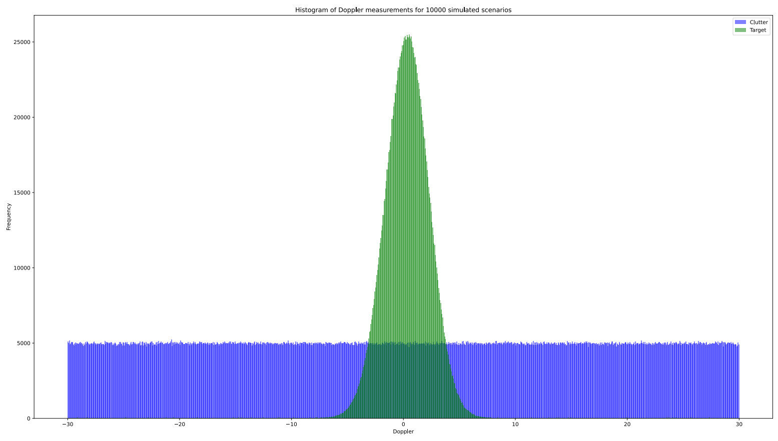


Figure 3.4: Histogram of doppler measurements from targets and clutter from the data generator used to train and test the MT3v2.

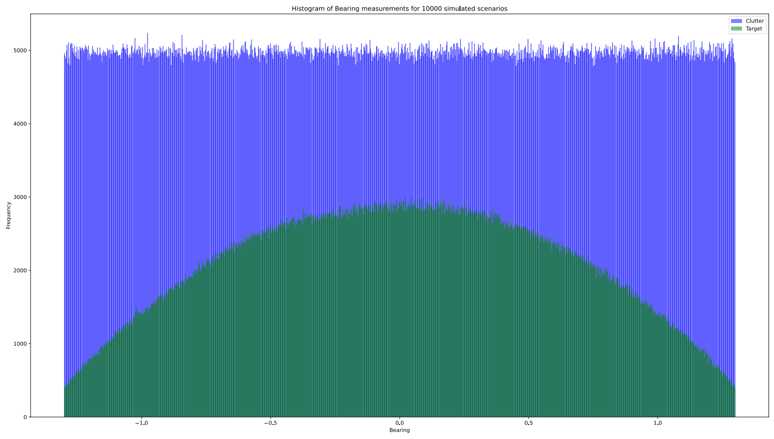


Figure 3.5: Histogram of bearing measurements from targets and clutter from the data generator used to train and test the MT3v2.

Architecture

Both the MT3v1 and MT3v2 use a learned position encoding vector that is added element-wise to the input embeddings along its first dimension. This enforces that the input sequence is of a fixed length τ, which for these architectures is set to 20. When there are not enough timesteps with measurements in the input, this problem may simply be alleviated by padding the input sequence and indicating where padding has occurred using a corresponding mask for the input. However, it is not possible to truncate sequences that are longer than τ without also losing information. A moving window across the measurement history must be used in order to use the MT3v1 and v2 as trackers. While this is mentioned briefly in the introduction sections of their respective papers, it is not further discussed. 20 timesteps may only account for a time window in the order of magnitude of seconds in real-world scenarios, and it is not unreasonable for a target to temporarily disappear for some seconds. For the MT3v1 and v2, such a scenario would result in the loss of all measurements that came from the target in question, effectively having to restart its track solution. Furthermore, the case of targets entering from the perimeter of the sensor field of view will rarely ever happen, although this is a case that is likely to occur often in real-world scenarios. As the figures 3.3, 3.4 and 3.5 show, the architectures will be provided significantly more measurements closer to the sensor, which may also impact generalizability.

The contrastive component $\mathcal{L}_C$ of the loss function utilized in both architectures is conceptually sound, as two measurements from the same target that are close to one another in time will be similar to one another. However, if the simulated data produces measurements that are *too easy* to distinguish from one another

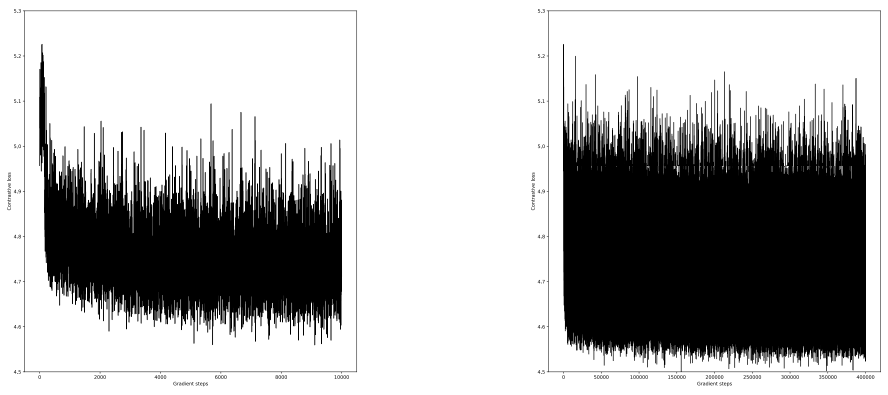


Figure 3.6: The contrastive loss component after 10000 and 400000 gradient steps. Figure was generated using data from the training metrics logger of the MT3v2 from the pretrained architecture for task 1 from (Pinto et al. 2022)

by simply observing their distributions, this component may dominate the NLL loss, resulting in an architecture that may get stuck in a local minimum where it is unable to learn more complex dependencies in areas such as data association and target maneuvering. Furthermore, it appears that the information that can be obtained by contrasting measurements is learned early on, leading to a significant portion of the total loss that cannot be reduced further through more training.

As described previously, the MT3v1 and v2 decoders are based on the DETR, and as such they both produce outputs in parallel instead of autoregressively. This in turn means that the output estimate for any given timestep in only conditioned on the available measurement set, and not the previous estimates. While the parallel approach offers lower training and inference times, it seems unconventional to design a tracker that does not use any information about the previous estimate, at least from the Bayesian and target-tracking assumption point of view. High localization error for both the Transformer trackers is a problem that is discussed in both papers, but it is not specifically attributed to the decoder architecture being parallel.

Evaluation

While the reported performance of the two architectures seem to perform as good or better than two very strong baselines in terms of GOSPA scores, the way these scores are computed still leaves much to be desired. Namely, the GOSPA score for a specific scenario of τ timesteps is calculated by comparing the ground truth targets and the tracker estimates at the very last timestep. This is problematic for several reasons. Firstly, it does not clearly indicate how well new tracks are initialized. This could have been shown by computing GOSPA scores for each timestep across an ensemble of scenarios, and providing the averages for each timestep. Secondly, by letting new targets spawn with a normally distributed state about a single point and only allowing one target to spawn per timestep, there will be more scenarios that appear as "single-target tracking in parallel" as compared to "challenging multitarget tracking". While it could be argued that the former is much more common in real-world scenarios, it would still be beneficial to include hand-crafted scenarios to demonstrate how capable the MT3v1 and v2 are in complex multitarget scenarios. This is especially true since the entire focus of the two papers revolve around the *multi* in multitarget tracking.

The reported GOSPA for the MT3v2 compares points in state-space, i.e. cartesian position and velocity, and a euclidean distance metric is used to calculate the localization error component. However, this is problematic since position and velocity have different units, and as such they cannot simply be added together. Doing this also makes it impossible to assess how well the trackers estimate positions and velocities separate from one another.

Complexity evaluation in this context is very difficult. The implementations of the baseline trackers are mostly CPU intensive. The Transformers however can almost entirely be ran on GPUs. The baseline trackers also do not have a learning component since they are model-based, which means that all computational complexity occurs at runtime. Meanwhile, the Transformer trackers require multiple days of GPU time on top-of-the-line hardware. Variation in time and memory complexity will also differ based on the input data for the model-based trackers, while the Transformers will stay approximately the same for all inputs. Evaluation of time and memory complexity is discussed in greater detail in chapter 7.

4 The Multitarget Tracking Transformer V3

Significant work has gone into the development of the MT3v1 and MT3v2 and has provided a solid foundation for point-object tracking using Transformers without being bound to the computer vision domain. Nevertheless, as described in the previous chapter, both architectures are limited by certain issues and constraints that must be resolved to enable further progress and facilitate usage outside of the highly specific environment they have currently been deployed into.

The specific limitations of the MT3v2, together with the shared problems between the MT3v1 and MT3v2, serve as the motivating factors behind the main contribution of this thesis, namely the MT3v3 architecture. This next generation of MT3 trackers introduces two major changes to the MT3v2. Firstly, the architecture itself is adjusted to allow for higher capacity, to produce more accurate decoder estimates, and to be flexible in the input sequence length. Secondly, the training procedure and data generator is altered to facilitate better generalizability and greater robustness to variations in the input data.

4.1 Architectural changes

The MT3v2 was used as the starting point for the MT3v3. The architectural changes that were made can roughly be divided in two – Changes made to the encoder side and changes made to the decoder side, i.e. changes in the input and the way the outputs are generated.

4.1.1 Input

The first architectural changes can be found in the input and the modules before the first encoder layer.

Firstly, the doppler component from the measurements has been removed. This comes with the benefit of supporting a larger set of sensors, since scanning sensors that provide doppler in addition to range and bearing is a subset of those that produce range and bearing only. In the case of cartesian position measurements, these can be converted to measurements on a format supported by the MT3v3 without requiring measurements of the target's velocity. Secondly, it provides for a more interesting tracking challenge. By removing doppler, the network must learn to attend to measurements in such a way as to estimate velocities without any explicit information about this in the measurements. Furthermore, as shown in the previous chapter, the MT3v2 may be using the doppler component to directly estimate which measurements are clutter and which originate from targets, and it is hypothesized that the removal of the doppler component forces the network to learn a different approach to this problem.

To remove the constraint of input sequences of size τ only, the learned absolute positional encoder was replaced. (Vaswani et al. 2017) shows good results using sinusoidal position encoding, and states that this was preferred over a learned encoding since it allows the network to use input sequences of arbitrary lengths. It is also possible to define a relative encoding scheme which simply uses the integer differences between the indices of the elements in the input sequence. While this is a very simple and efficient approach, it also runs the risk of introducing numerical instability for large input sequences. This can be somewhat alleviated by dividing each element in the matrix by the current sequence length. Lastly, the Time2Vec approach was considered due to its design that specifically focuses on encoding temporal sequences. This makes it align well with the MT3v3 input sequences, which consist of time-ordered data. All three of the aforementioned positional encoding schemes were implemented for use with the MT3v3, which allowed for future assessment of which method provides the best results for the MTT task.

4.1.2 Outputs

Another observation that was made in (Strøm 2022) and throughout the initial experimentation with the MT3v2 is that the existence probabilities in the output for a given pretrained architecture would fluctuate a lot across for longer tracks and for changing clutter intensity. This effect was in some cases noticeable in consecutive predictions for a target that generated measurements in both time steps, resulting in some tracks exhibiting a flickering-like effect. As described previously, the existence probabilities in the MT3v2 are calculated directly from the decoder output using a feed-forward network. However, an alternative approach could be to utilize the iterative refinement process that is done for the state estimates. For the MT3v3 this was implemented as a three-stage process,

1. Calculate the initial existence probabilities as $p_i^0 = f_{nn}\left(o_i\right)$
2. For each layer, calculate adjustments for the existence probability of each component using the state estimates at the current decoder layer, $\rho_i^l = f_{nn}\left(y_i^l\right)$

3. After the final decoder layer, calculate the final existence probability for each component as $p_i = p_i^0 + \sum_{l=1}^{L} \rho_i^l$

which is practically identical to the iterative refinement process done for the state estimates. It is hypothesized that iteratively refining the existence probabilities will incorporate more information from the encoded input and the decoder outputs, as opposed to simply letting the probability of existence for each output component be calculated directly from the final decoder layer.

As described by (Pinto et al. 2022), a possible improvement to the MT3v2 architecture could be to estimate the full covariance matrix at the output instead of limiting it to be a diagonal matrix. Doing so requires an increase in output size of the network that is used to generate the covariance matrix for each component from 4 to 16. To accommodate this, the capacity of the network was increased by adding another layer and using ReLU activations in between. Although this change will require more computations, it constitutes only a minor portion of the entire MT3v3 architecture. Given how little needed to be changed in order to facilitate this, it was deemed worth-wile to implement this proposed change.

4.2 Changes to the training pipeline

While all of the changes from MT3v1 to MT3v2 focused on the architecture itself, the changes made for the MT3v3 also involve the training loop. This is done since, no matter how sophisticated the architecture is, the overall performance will be commensurate with the quality of the training pipeline and the provided data.

4.2.1 Adaptive weighing of contrastive auxiliary loss

As shown in figure 3.6, the contrastive loss stagnates quickly, and will at some point in the training process dominate the total loss. This is in large caused by the scaling factor α being fixed. As the contrastive loss stagnates, all the information about similar and dissimilar samples has been used, and it would be reasonable to lower α at this point.

For the overall learning late, all MT3 trackers use the ADAM optimizer from (Kingma and Ba 2014), which among other things adapts the learning rate depending on the behavior of the gradient of the loss function. This concept has inspired the adaptive weighing of contrastive loss that is used in the MT3v3. Instead of setting α to a constant ahead of a training job, only the initial value α_0 is set. Then, for every gradient step, the weighting

$$\alpha_k = \frac{\alpha_0}{\sqrt{r_{k+1} + \delta}}, \quad r_{k+1} = \rho r_k + (1 - \rho)\nabla_c^\top \nabla_c \tag{4.1}$$

will be used. ∇_c is the gradient of the contrastive loss, while ρ is a hyperparameter that is used to determine how much the accumulated gradient should be weighed against the square of the current gradient. This update is identical to that of the ADAM optimizer, but using the gradient of the contrastive loss only, instead of the gradient of the total loss. δ is a small constant used to avoid poor numerical properties when the accumulated gradient is small, and is typically set to 10^{-6}. By accumulating the square of the gradient, the goal is to achieve a smooth decrease in α_k as k increases and the gradient changes.

4.2.2 Bimodal target birth model

Both the MT3v1 and MT3v2 employ a Gaussian birth model, which can be expressed as

$$\mathbf{x}_0 \sim \mathcal{N}(\boldsymbol{\mu}_0, \boldsymbol{\Sigma}_0), \quad \boldsymbol{\mu}_0 = \begin{bmatrix} x_0 \\ y_0 \\ v_{x,0} \\ v_{y,0} \end{bmatrix}, \quad \boldsymbol{\Sigma}_0 = \begin{bmatrix} \sigma^2_{x_0} & 0 & 0 & 0 \\ 0 & \sigma^2_{y_0} & 0 & 0 \\ 0 & 0 & \sigma^2_{v_{x_0}} & 0 \\ 0 & 0 & 0 & \sigma^2_{v_{y_0}} \end{bmatrix} \tag{4.2}$$

where the number of targets to spawn for any given timestep is sampled according to

$$n_b \sim \mathcal{P}(\lambda_b) \tag{4.3}$$

This birth model, together with the limited number of timesteps, leads to training data that does not cover a large area of the sensor view, especially as the maximum range increases. Furthermore, many targets in a real-world scenario will simply move into the sensor view from the outside, something that cannot be represented using this birth model.

In order to improve this, the MT3v3 uses a bimodal birth model. That is, in addition to the Gaussian birth model, henceforth referred to as M_1, a second birth model M_2 is defined. M_2 simulates targets moving into sensor view, and is defined by

$$\mathbf{x}_0 = \begin{bmatrix} \tilde{r}_{\max} \cdot \cos(\theta_0) \\ \tilde{r}_{\max} \cdot \sin(\theta_0) \\ v_a \cdot \cos(\psi_0) \\ v_a \cdot \sin(\psi_0) \end{bmatrix} \tag{4.4}$$

where

$$\tilde{r}_{\max} = r_{\max} + \epsilon, \quad \epsilon \sim \mathcal{U}(0,1), \quad \theta_0 \sim \mathcal{U}(\theta_{min}, \theta_{max}) \tag{4.5}$$

determine the initial position of the new target around the perimeter of the sensor view. The offset ϵ is used to emulate how targets may have entered some distance into the field of view in between two scans. It also serves as a way to prevent the MT3v3 from attending too much to measurements with ranges exactly at $r_{\max}$.

$$v_a = \sqrt{v_{0,x}^2 + v_{0,y}^2}, \quad \psi_0 = \operatorname{atan2}(x_{0,2},\ x_{0,1}) + \alpha_0, \quad \alpha_0 \sim \mathcal{U}(-\alpha, \alpha) \tag{4.6}$$

are used to sample an initial velocity for the target that ensures that it moves into the sensor view, and not away from it. The x and y component for the velocity are sampled according to the same distribution as in the first birth model, while α is a user-configurable parameter that defines a sector that the initial velocity can lie within. For example, letting $\alpha = \frac{\pi}{4}$ defines a quarter-circle, with its origin in the starting position of the target, and its center line pointing towards the origin.

Instead of a single birth intensity, both $\lambda_b^{M_1}$ and $\lambda_b^{M_2}$ are used to independently sample

$$n_{M_1} \sim \text{Poisson}\left(\lambda_b^{M_1}\right) \quad \text{and} \quad n_{M_2} \sim \mathcal{P}\left(\lambda_b^{M_2}\right) \tag{4.7}$$

which are then both used to spawn new targets. This approach was selected in favor of sampling a single n and then drawing which birth model to use, in order to have more fine-grained control of the average number of targets that enter the sensor view or appear around the sensor at any given time step.

4.2.3 Uniform distribution of data generator parameters

While the MT3v1 and MT3v2 define separate *tasks* of varying difficulty, and train architectures specifically for each task, the goal of the MT3v3 is to be more robust to changes in the underlying parameters used to generate data. Most notably, based on the cross-model validation results in (Strøm 2022), the following parameters were used when training the MT3v3

$$\begin{aligned}
k &\sim \mathcal{U}(10, 100) \\
p_d &\sim \mathcal{U}(0.7, 1.0) \\
\sigma_q &\sim \mathcal{U}(0.1, 5.0) \\
\lambda_c &\sim \mathcal{U}(5, 15) \\
\lambda_b^{M_1} &\sim \mathcal{U}(0.0, 0.2) \\
\lambda_b^{M_2} &\sim \mathcal{U}(0.0, 0.2) \\
p_s &\sim \mathcal{U}(0.8, 1.0) \\
n_{\text{start}} &\sim \mathcal{U}(1, 6)
\end{aligned} \tag{4.8}$$

The choice of uniform distributions for all parameters was to not inject any particular bias in the choice of parameters. Since the MT3v3 is already trained on so many samples, and new scenarios can be generated on the fly, it is reasonable to vary this data in an attempt to make the final trained architecture perform well across a wide selection of scenarios, without the need for any fine-tuning. However, too much variation in the data generator parameters may lead to intractably long training times to see good performance on any scenarios. Sensor measurement noise was specifically left out of this scheme, since it is a reasonable assumption that a single MT3v3 architecture will be trained for a single sensor, especially given that sensor field-of-view must be specified a priori.

4.3 Implementation

In terms of the practicality of using the MT3v1 and MT3v2, their implementations severely limit the operability of pretrained architectures, and setting them up for use with external applications was shown to be time-consuming. However, looking past the inputs and outputs, the MT3v1 and MT3v2 are implemented to a decent standard when it comes to configurability, scalability and readability.

The aim of the MT3v3 implementation was thus to keep the good parts of the prior versions, but improve upon the input and output, as well as providing examples which serve as a way for new users to get their own MT3v3 architectures running without much hassle. The approach of improving the architecture while simultaneously attempting to make the architecture more user-friendly is in part inspired by the various developments and versions of the YOLO architectures (Jiang et al. 2022).

Firstly, the changes described above were implemented on a fork of the MT3v2, with the intent of changing as few parts of the core implementation as possible, while cleaning up unused components. All three position encoders were implemented and are configurable as part of the encoder configuration. Secondly, the data generator was expanded to allow for passing of information that the CKF requires in testing, such as target death times. This was implemented to be backwards-compatible with the data generation used for training the MT3v3, which does not require this information. The issue of input and output formatting was addressed by creating a separate module with a set of conversion functions to and from standard Numpy ndarrays to the format that the MT3v3 outputs and the format that is required at the input. This module also ensures that all data is kept on the same device, either CPU or GPU, for sake of data compatibility. Separate examples and corresponding tutorials were made for the following:

1. Training an architecture from scratch
2. Loading a pretrained architecture
3. Incorporating the MT3v3 in an existing pipeline

As previously mentioned, the goal of this is to make the MT3v3 easy to use for new users, which in turn promotes future development. Lastly, the MT3v2 repository contains bloat and code that appears to have been used specifically for generating figures for the paper associated with it, and residuals from the MT3v1. These were all removed and cleaned up in the MT3v3 fork.

4.3.1 Network size

The removal of the doppler component of the input measurements reduces the total number of parameters of the embedding module by a third. However, the introduction of the Time2Vec encoding introduces two learnable parameters per embedding dimension, which is an increase from the 3 by 20 learned lookup table of the MT3v2. The rest of the encoder is unchanged. As for the decoder, the iterative refinement implementation replaces the feed-forward network used to estimate existence probabilities for each decoder layer. However, the size of this network remains the same and as such does not change the number of parameters in the MT3v3 when comparing to the MT3v2. This is not the case for the feed-forward networks used to estimate the state uncertainty for each decoder layer. As this is

Module	**learnable params**	**Description**
Input embedding	2×256	Model dimension is 256, 2 values per input
Position encoding	2×256	Single phase and frequency per embedding
Encoder: f_{mha}	$4 \times 256 \times (256 + 1)$	Self-attention, 4 square weight matrices plus bias
Encoder: $f_{\text{norm},1}$	2×256	First layer normalization
Encoder: f_{nn}	$2 \times 2048 \times (256 + 1)$	A single feed-forward network with 2 layers
Encoder: $f_{\text{norm},2}$	2×256	Second layer normalization
Selection mechanism: f_{nn,e_1}	256×256	Embedding projection for $m_{1:n}$. No bias.
Selection mechanism: f_{nn,e_2}	256×256	Embedding projection for $\delta_{1:n}$. No bias.
Selection mechanism: $f_{\text{nn},q}$	$16 \times (256 + 1)$	16 queries to generate
Selection mechanism: f_{nn,o_x}	$2 \times 256 \times (256 + 1)$	Starting points for state estimates
Selection mechanism: f_{nn,o_p}	$1 \times 256 \times (256 + 1)$	Starting points for existence probability
Decoder: $f_{\text{mha},1}$	$4 \times 256 \times (256 + 1)$	Self-attention, 4 square weight matrices plus bias
Decoder: $f_{\text{norm},1}$	2×256	First layer normalization
Decoder: $f_{\text{mha},2}$	$4 \times 256 \times (256 + 1)$	Cross-attention, 4 square weight matrices
Decoder: $f_{\text{norm},2}$	2×256	Second layer normalization
Decoder: $f_{\text{nn},x}$	$128 \times (256 + 1) + 128 \times (128 + 1) + 4 \times (128 + 1)$	Position and velocity delta
Decoder: $f_{\text{nn},p}$	$128 \times (256 + 1) + 128 \times (128 + 1) + 1 \times (128 + 1)$	Existence probability delta
Decoder: $f_{\text{nn},\Sigma}$	$128 \times (256 + 1) + 128 \times (128 + 1) + 16 \times (128 + 1)$	Uncertainty estimator
Decoder: $f_{\text{nn, oc}}$	$1 \times (256 + 1)$	Object classifier
Decoder: $f_{\text{nn},1}$	$2 \times 2048 \times (256 + 1)$	A single feed-forward network with 2 layers
Decoder: $f_{\text{norm},3}$	2×256	Third layer normalization
Decoder: $f_{\text{nn, cc}}$	256×256	Contrastive classifier. No bias.
Total	18 030 228	6 encoder and decoder layers

Table 4.1: The number of learnable parameters in the MT3v3 categorized by module and listed from input to output. The MT3v3 uses a single preprocessing step consisting of input embedding and positional encoding, 6 encoder layers, a single selection mechanism, and 6 decoder layers.

applied element-wise, the number of learnable parameters for this module is four times as many for the MT3v2 as compared to the MT3v2. In total, there are N learnable parameters in the MT3v3. An overview of each module and it's total number of learnable parameters is presented in table 4.1.

5 Experimental setup

The MT3v3 requires a significant amount of data for training, which facilitates the need for a stochastic data generator. When testing however, it might be interesting to consider a set of specific hand-crafted scenarios, which ideally should be done programmatically. All of this also requires powerful compute units in order to be done on a tractible timescale, which also requires its own setup. This chapter outlines the data generator, a simulator for hand-crafting scenarios, and the computational resources used for this. The setup for benchmarking the MT3v3 is also covered.

5.1 Data generator

A large number of samples are required to train the MT3v3 - in the order of magnitude of 10^6. For this reason, automatic data generation becomes a requirement. To allow for efficient training, this process must also be parallelized. By simulating data one has access to ground truths, object birth and death times, as well as knowing which measurements are true and which are false. The MT3v2 includes a simulator that solves the issues of automatic data generation and parallelization. However, as was shown in chapter 3, this generator has certain problems associated with it. The implementation of the data generator for the MT3v3 builds off the previous versions, but aims to address these problem. Table 5.1 contains all the user-configurable parameters that will be referenced throughout the following subsections.

5.1.1 Workflow

The MT3v3 data generation workflow in short, is as follows:

1. $t \leftarrow t + \Delta t$
2. Step alive targets once

Parameter	**Description**
n	Number of timesteps, upper and lower bound
max_objects	Maximum number of objects in the scene at any given instance
dt	Time step size
p_s	Probability of survival, upper and lower bound
p_d	Probability of detection, upper and lower bound
σ_q	Process noise, upper and lower bound
Σ_r	Measurement noise
λ_{clutter}	Clutter intensity, upper and lower bound
$\lambda_{\text{b, edge}}$	Birth intensity for targets entering FOV, upper and lower bound
$\lambda_{\text{b, in}}$	Birth intensity for targets appearing in FOV, upper and lower bound
n_{start}	Average number of starting objects, upper and lower bound
FOV	Sensor field of view
μ_0	Target starting state
Σ_0	Target starting variance

Table 5.1: The user-configurable data generation parameters.

3. Kill targets outside sensor field of view
4. Spawn new targets
5. Remove targets that do not survive
6. Generate true measurements from alive targets
7. Generate false measurements
8. Shuffle and timestamp measurements and return

and the details of each mechanism is described in the next subsections.

5.1.2 Targets

Ground truth targets use a near-constant velocity model in order to propagate the state of each object forward in time. Given the state vector $\mathbf{x}_k = [x\ y\ v_x\ v_y]$ at time k, then

$$\mathbf{x}_{k+1} \sim \mathcal{N}\left(\begin{bmatrix}\mathbf{I}_2 & \mathbf{I}_2\Delta_t \\ \mathbf{0}_{2x2} & \mathbf{I}_2\end{bmatrix}\mathbf{x}_k, \begin{bmatrix}\mathbf{I}_2\frac{\Delta_t^3}{3} & \mathbf{I}_2\frac{\Delta_t^2}{2}\Delta_t \\ \mathbf{I}_2\frac{\Delta_t^2}{2} & \mathbf{I}_2\Delta_t\end{bmatrix}\sigma_q^2\right) \tag{5.1}$$

where Δ_t is the time between samples and σ_q is the standard deviation for the process noise, which is also interpretable as the noise intensity in acceleration. If this is set to zero, all targets will move in a perfectly straight line, and increasing it from zero yields targets that maneuver more. The implementation of this model is left unchanged between the MT3v2 and MT3v3.

5.1.3 Sensor model

A sensor model contains models for measurements and how targets are detected. For the measurement model, the MT3v1 uses a cartesian model, while the MT3v2 uses polar measurements in conjunction with doppler. The MT3v3 strikes a middle-ground, providing options for both cartesian and polar measurements, but removes the doppler component that is used in the MT3v2. The two available measurement models are thus

$$\mathbf{z}_{k,c} = \mathbf{H}\mathbf{x}_k + \mathbf{w}_k, \quad \mathbf{H} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}, \quad \mathbf{w}_k \sim \mathcal{N}\left(\mathbf{0}_{2\times 1}, \mathbf{\Sigma}_c\right) \tag{5.2}$$

and

$$\mathbf{z}_{k,p} = \mathbf{h}(\mathbf{x}_k) + \mathbf{w}_k, \quad \mathbf{h}(\mathbf{x}_k) = \begin{bmatrix} \sqrt{\mathbf{x}_{k,1}^2 + \mathbf{x}_{k,2}^2} \\ \operatorname{atan2}\left(\mathbf{x}_{k,2}, \mathbf{x}_{k,1}\right) \end{bmatrix}, \quad \mathbf{w}_k \sim \mathcal{N}\left(\mathbf{0}_{2\times 1}, \mathbf{\Sigma}_p\right) \tag{5.3}$$

respectively, where $\mathbf{\Sigma}_c$ and $\mathbf{\Sigma}_p$ are diagonal matrices containing noise intensities.

The sensor field of view parameters can be set in order to limit the sensor's surveillance area. For the cartesian model this is given as the ranges

$$x_{\min} \leq x < x_{\max}, \qquad y_{\min} \leq y < y_{\max} \tag{5.4}$$

while for the polar model as

$$r_{\min} \leq r < r_{\max}, \qquad \theta_{\min} \leq \theta < \theta_{\max} \tag{5.5}$$

Furthermore, the parameter p_d is used to configure the probability of detection, i.e. if a target produces a measurement at a given timestep. The probability of detection together with the sensor field of view is collectively referred to as the detection model for the sensor. In the context of the simulator, a target producing a measurement is equivalent to that target being detected.

5.1.4 Clutter model

The clutter model is used to simulate clutter measurements, and is modeled as a PPP. That is, the number of clutter measurements for a given step is firstly sampled according to

$$n \sim \mathcal{P}(\lambda_c) \tag{5.6}$$

which is then used to draw n_{clutter} clutter measurements. When the cartesian measurement model is used, a clutter measurement will simply be sampled uniformly in the sensor field of view

$$\mathbf{z}_{k,i}^{\text{clutter}} = \begin{bmatrix} \mathcal{U}(x_{\min}, x_{\max}) & \mathcal{U}(y_{\min}, y_{\max}) \end{bmatrix}^{\mathrm{T}} \tag{5.7}$$

For the polar model, the clutter model becomes

$$\mathbf{z}_{k,i}^{\text{clutter}} = \begin{bmatrix} f_r\left(\mathcal{U}(0,1)\right) & \mathcal{U}(\theta_{\min}, \theta_{\max}) \end{bmatrix}^{\mathrm{T}} \tag{5.8}$$

where

$$f_r(u) = r_{\min} + (r_{\max} - r_{\min})\sqrt{u} \tag{5.9}$$

is used to sample range in order to avoid a difference in density of clutter measurements across the sensor view. If range was simply sampled uniformly as with bearing, there would be a surplus of clutter measurements closer to the sensor. This solves the problem of an increase in clutter intensity near the sensor that is seen in the MT3v2. In both the cartesian and polar case, $i \in (0, n-1)$

5.1.5 Target birth and death

A target is said to have *spawned* or been *born* when it first appears in the the sensor view. There are two ways a target can spawn – Either by crossing into the sensor's range from the outside or by appearing somewhere within the surveillance area of the sensor. The former is a more plausible scenario, especially for environments that do not have many obstacles that can occlude targets. The MT3v3 data generator employs the bimodal birth model as described in section 4.2.2.

Target death is, as the name implies, the opposite of target birth. Thus it follows that a target dies or *despawns* when it exits the sensor view. Targets may also at any random timestep simply disappear while still inside the sensor field of view. This could happen if a target is occluded by an obstacle. The mechanism of target death is simulated in a very simple manner, based on the probability of survival. For target i, $u_i \sim \mathcal{U}(0,1)$ is sampled, and target i is killed if $u_i < 1 - p_s$. Any target that exits the sensor view is also killed.

5.1.6 Detection model

The detection model used in the MT3v3 data generator is simply a constant probability of detection model. That is, for every ground-truth target, a measurement is generated if $i < p_d$, where $i \sim \mathcal{U}(0,1)$ and p_d is the user-defined probability of detection. The MT3v2 implements an additional detection model that is meant to emulate a specific automotive radar, but this is removed in the MT3v3 in order to simplify the implementation.

5.2 Traffic and detection simulator

While the data generator is good for producing a large quantity of training data, it does not allow for fine-grained control over what scenarios are produced. When testing the MT3v3, or any multitarget tracker for that matter, it is reasonable to test its performance on scenarios that are deliberately challenging. Examples of challenging scenarios can be multiple targets crossing close to one another, or targets moving in parallel.

To test such scenarios, a detection simulator was developed on top of an existing traffic simulator developed by Zeabuz AS. The traffic simulator is a library that simulates the motion of a set of vessels that can be configured with various attributes, such as vessel dynamics, control algorithm and maneuvering scheme. The aforementioned bona-fide multitarget tracking scenarios can thus be simulated by simply configuring vessels to behave in a certain manner in the traffic simulator. However, this simulator only produces ground truths, which are unsuitable for testing the MT3v3. For this reason, a detection simulator was developed[1], the purpose of which is to turn the ground truths from the traffic simulator into measurements from a simulated sensor.

5.3 Training and testing pipeline

The MT3v3 training and testing scheme was set up from scratch, and was done in order to validate that the new implementation was working as intended. The hardware originally used to train the MT3v1 and MT3v2 is also not available publicly, which meant that the training and testing setup would have to be redone regardless.

5.3.1 Hardware

Access to the IDUN compute cluster of NTNU's High Performance Computing Group (Själander et al. 2019) was granted in order to train the MT3v3 at a large scale. A single compute note with one NVIDIA A100 GPU with 24GB allocated VRAM was used, which allowed for 32 concurrent scenarios of 100 timesteps to be used for training. A data logger and evaluation tool was also scheduled to periodically generate data from the training process. All job details were specified using the SLURM workload manager, and all configuration parameters can be found in tables 5.3 and 5.2. The tools used for logging training data are the same for the MT3v2 and the MT3v3.

5.3.2 Training and validation data

The hardware setup described above was ran for a total of 12 days wall-clock time. This was chosen since it is about the same time as the four MT3v2 networks were trained for combined. This resulted in about $1.9 \cdot 10^6$ gradient steps, which totals to $6.0 \cdot 10^8$ generated scenarios that was processed by the MT3v3. However, since weights and biases are saved periodically, training can resume from periodic checkpoints if required. Periodic evaluation was performed by simply generating unseen scenarios and calculating a set of performance metrics. Since 32 batches were generated in parallel for each step, mini-batch gradient descent was used to update the weights, alongside the Adam optimizer.

5.3.3 Testing

Since testing does not require the large amounts of data as in the training scheme, it was decided to perform testing locally. A single laptop with an Intel(R) i7-

[1]The detection simulator was developed by the author and Odin Aleksander Severinsen for Zeabuz AS, and will thus not be made publicly available.

12800H CPU, an NVIDIA RTX A2000 8GB GPU, and 32GB of RAM was used. Inference time with the MT3v3 is similar to that of the IDUN setup, and although parallelization is substantially limited, it is not required for testing. Intelligent paging file management also allows for larger scenarios to be tested without running out of memory.

Parameter	**Description**
device	Device used for computation
n_{gs}	Number of gradient steps
n_{bs}	Number of batches to run in parallel
η_l	Learning rate
η_p	Patience parameter for learning rate reduction
η_f	Factor by which learning rate is reduced
α	Contrastive loss scaling factor
α_p	Patience parameter for contrastive loss scaling reduction
α_f	Factor by which contrastive loss scaling is reduced
$k_{\text{checkpoint}}$	Interval for saving model weights
k_{print}	Interval for printing progress updates
k_{plot}	Interval for generating plots
k_{save}	Interval for saving plots
k_{log}	Interval for logging updates
k_{eval}	Interval for evaluating GOSPA metric

Table 5.2: All user-configurable training parameters

Parameter	**Description**
partition	Either short, CPUQ or GPUQ, depending on the task
account	Which account to be billed for the job
time	The time limit for the SLURM job
nodes	The number of compute nodes to utilize
tasks-per-node	How many tasks to run per node
mem	Allocated memory in MiB

Table 5.3: The SLURM parameters for the NTNU HPC cluster used for training the MT3v3.

5.4 Benchmarking

While various performance metrics can be calculated for the MT3v3 alone, they can only provide a limited amount of information when presented in isolation. It is thus important to compare the MT3v3 to some baseline. A decision was made to use the CKF as a baseline for the optimal achievable performance, since it does not rely on data association as it is already given the target-to-measurement ground-truth information. The CKF also allows for sanity-checking the data generator and the other trackers. I.e., if the CKF has a false detection score that is not zero, there is something wrong with the data generator or CKF setup. Likewise, if one of the trackers perform better than the CKF, it is a sign that either the CKF is wrongly implemented, or that the tracker in question has access to information it should not have. Since the transformation from measurement to state space is non-linear, an Extended Kalman Filter was used at the core of the CKF. The IMM-JIPDA represents the other end of the baseline spectrum, i.e. a target for the MT3v3 to beat. This decision was made since the MT3v3 needs to show promising results to be considered for future research or use in a real application. "Promising results" can thus be defined as performing equally as good as or better than the IMM-JIPDA, since this is a tracker that performs reasonably well without the sophistication of the current Bayesian state-of-the-art trackers.

Since the measurements from the MT3v3 are polar, but the state estimates are assumed to be cartesian, the CKF is in reality implemented as a Clairvoyant Extended Kalman filter, but for sake of brevity, it is still referred to as the CKF. While the linearization process introduces errors, the CKF is still considered to be the strongest available baseline without directly accessing the ground-truth state.

5.4.1 Previous generation MT3

Since the MT3v1 uses cartesian measurements, it was deemed unfit for comparison with the MT3v3. The MT3v2 on the other hand, is used as grounds for comparison at certain points, especially for assessing the use/lack of doppler measurements. For the MT3v3 to be considered the "next-generation" of the MT3 trackers, it should also not perform drastically worse than its predecessors.

5.4.2 Tuning

The CKF and IMM-JIPDA each have a set of tunable parameters that can be changed on-the-fly. The MT3v3 on the other hand, would need to be retrained with a new set of parameters if a change is desired. For this reason, whenever the MT3v3 is tested against data generated using parameters that fall within the training range, the IMM-JIPDA is perfectly tuned to the data generator. In scenarios which the MT3v3 has not specifically been trained for, the median values for the data generator parameter ranges are used for tuning the IMM-JIPDA. This is somewhat of an arbitrary choice however, and is something that is discussed in chapter 7. The CKF is always given access to perfect tuning parameters.

5.4.3 Performance indicators

GOSPA for position and velocity serves as the go-to performance metric, since it does not require any ad-hoc assignment of targets and tracks, due to operating directly on point-patterns. As noted previously, GOSPA cannot be calculated for both position and velocity directly, due to the different units between the two.

For this reason, GOSPA is calculated using position only, while the quality of the velocity estimates is determined separately. Calculating GOSPA for more than just a single timestep is also used to assess various stages of the trackers for certain scenarios.

NEES is used to assess the consistency of the MT3v3 estimates, and is calculated only for the estimates that are matched to a ground-truth. State errors are also tested for bias and whiteness using a one-sample two-tailed t-test and a Box-Ljung test respectively.

6 Results

6.1 Data and scenario generation

In order to get an idea of what type of data that is automatically generated during training for the MT3v3, a set of randomly picked scenarios are presented in figure 6.1. In these figures, the data generator parameters have been fixed to those presented in table 6.1, but note that in the actual training pipeline, many of these are sampled for each training instance. The intention of these figures is simply to show how a scenario might look like, and for sanity-checking the data generator.

Parameter	**Value**
k	100
p_d	0.85
σ_q	1.0
λ_c	10
$\lambda_b^{M_1}$	0.1
$\lambda_b^{M_2}$	0.1
p_s	0.75
n_{start}	$\mathcal{U}(2, 4)$
$\boldsymbol{\sigma}_r$	$\text{Diag}(1.5, 0.044)$

Table 6.1: The parameters used to generate what is considered the "median" scenarios. The measurement noise is the same as that used in training, and corresponds to a variance of 2.25 meters and 6.25 degrees in range and bearing respectively. Fixing k is also fixed to the highest value attained in training in order to see effects that may only appear after sufficiently many timesteps.

Figure 6.2 shows a set of scenarios that would be generated for the MT3v2 during training. Note how due to the architecture, these scenarios are limited to 20 timesteps only. Also note that since that the positions of all new targets are

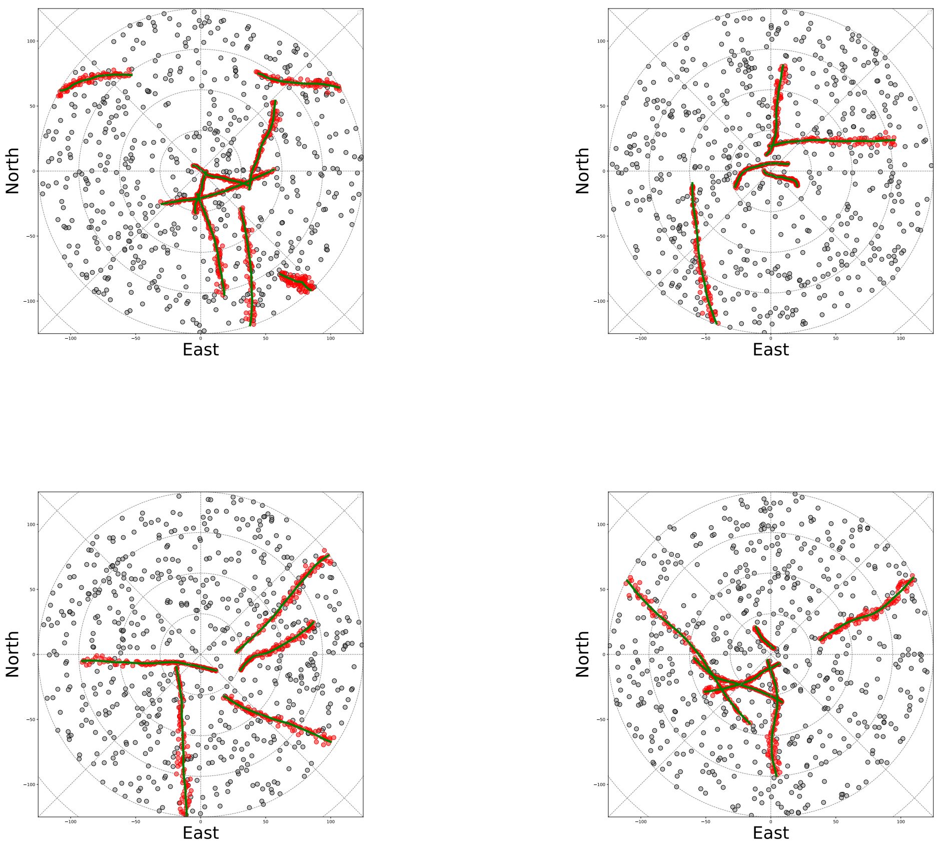


Figure 6.1: Four example scenarios from the MT3v3 data generator. Green tracks are the ground truths of the simulated targets, red dots are true measurements, and black dots are clutter.

normally distributed around the sensor, a large majority of the sensor view does not see any targets.

There are mainly five different parameters that are tweaked when designing a "simple" and "complex" scenario. These are the two noise components of the measurement model, the process noise of the targets, clutter intensity and probability of detection. A simple and complex scenario is plotted side by side in figure 6.3.

6.2 Training metrics

To assist in the selection of what positional encoding to be used, a set of MT3v3 architectures, each with their respective positional encoding module, were trained in parallel using the same data generation seeds, for 10^5 gradient steps. The total loss for each architecture is presented in figure 6.7, while the certainty distribution for each architecture is shown in figure 6.4. This figure shows two graphs for each position encoding scheme - average matched and unmatched certainties. The average matched certainty is the average existence probability for all predictions that have been matched with a ground truth target. Likewise, the average unmatched certainty is the average existence probability for all unmatched predictions. Lastly, the standard deviation for position and velocity estimates for the three networks are plotted in figures 6.5 and 6.6.

Two MT3v3 networks were trained using the t2v position encoding, but with two different methods for generating existence probabilities. The first uses the direct feed-forward network approach from the MT3v2, while the second one uses iterative refinement. The certainty distributions for the first 10^5 gradient steps is shown in figure 6.8. The training loss for both networks was practically the same, so this figure is left out.

As described previously, the final MT3v3 architecture was trained for $1.9 \cdot 10^6$ gradient steps. The total loss for this is shown in figure 6.9. GOSPA was evaluated periodically every 10^4 gradient steps, and is shown in figure 6.10

6.3 Benchmarking the MT3v3

The bulk of the results are from the MT3v3 benchmarking process, and aims to provide a detailed insight into how the MT3v3 performs compared to the IMM-JIPDA and CKF baselines. The MT3v2 is also tested in some aspects in order to get the full picture of how the new architecture performs. As described when introducing the CKF, it is the strongest possible baseline due to it accessing information about the ground-truth targets that is otherwise hidden in any real scenario. On the other hand, the IMM-JIPDA provides a "target to beat". Thus, when looking at performance in various scenarios, the MT3v3 is compared to the CKF and IMM-JIPDA. In the following section, the data generator parameters are fixed, and the CKF and IMM-JIPDA are provided with these parameter in order to obtain an optimal tuning.

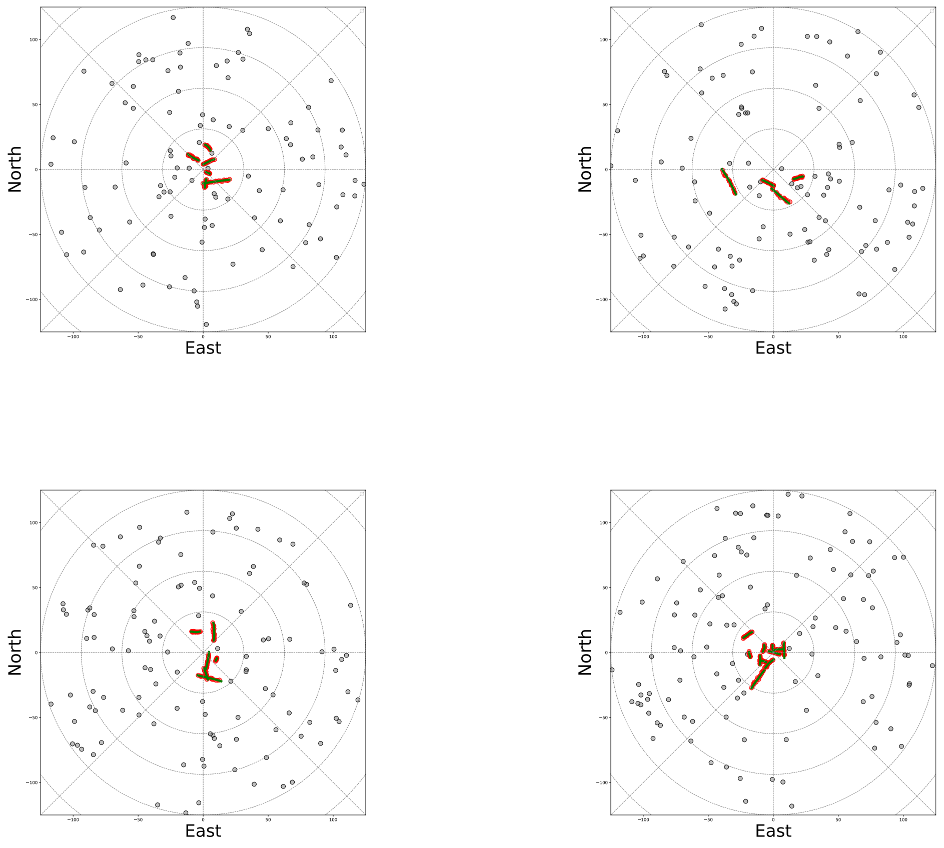


Figure 6.2: Four example scenarios from the MT3v2 data generator. Green tracks are the ground truths of the simulated targets, red dots are true measurements, and black dots are clutter.

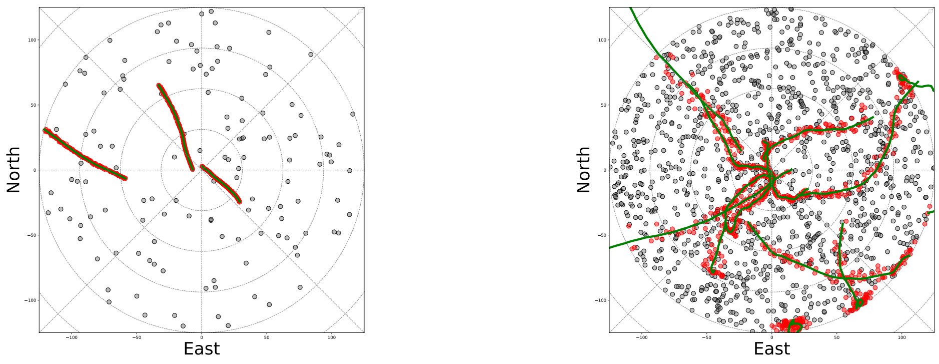


Figure 6.3: A simple and a complex autogenerated scenario. Green tracks are the ground truths of the simulated targets, red dots are true measurements, and black dots are clutter.

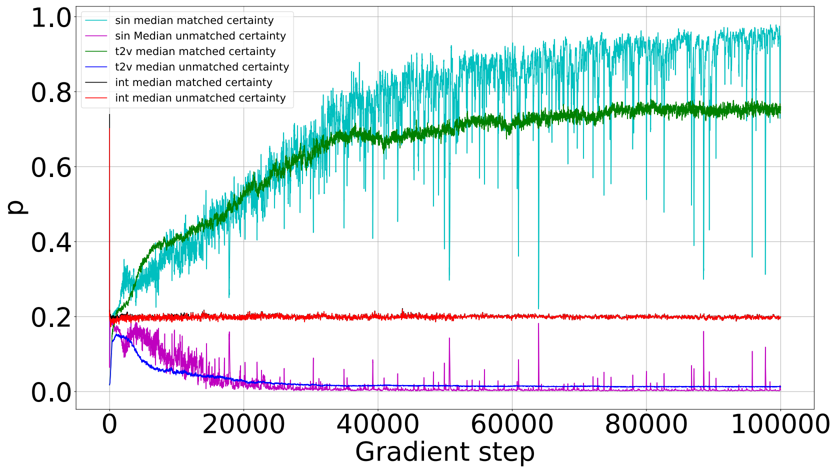


Figure 6.4: Certainty distributions for the first 100 000 gradient steps.

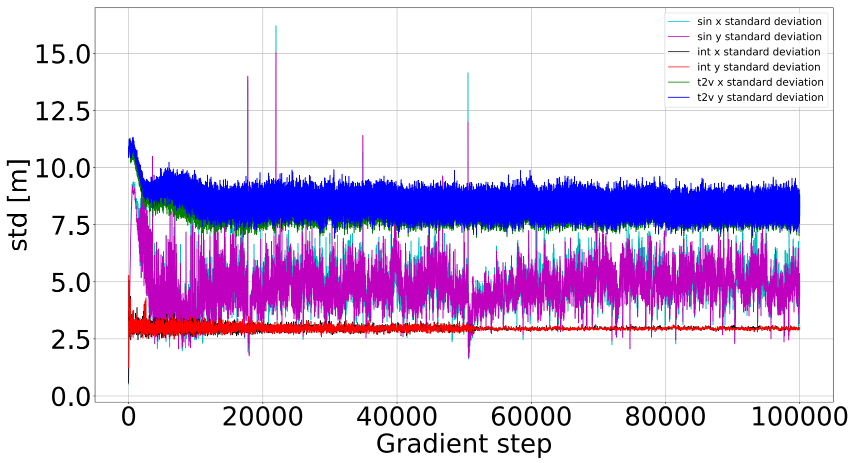


Figure 6.5: Position standard deviations for the first 100 000 gradient steps.

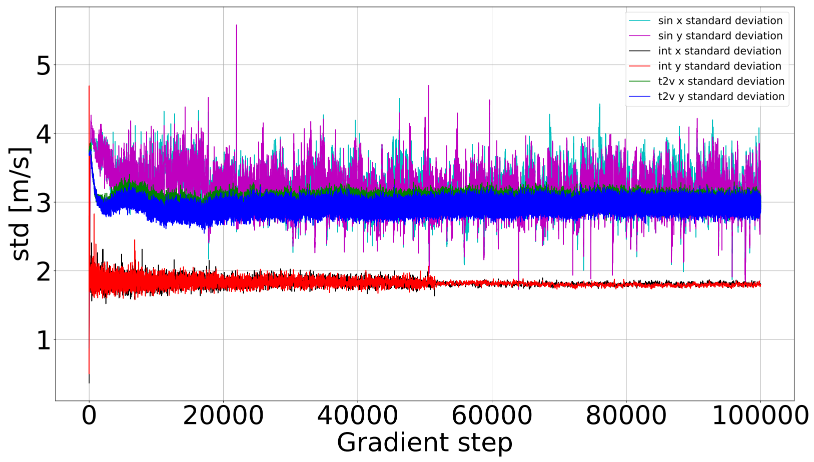


Figure 6.6: Velocity standard deviations for the first 100 000 gradient steps.

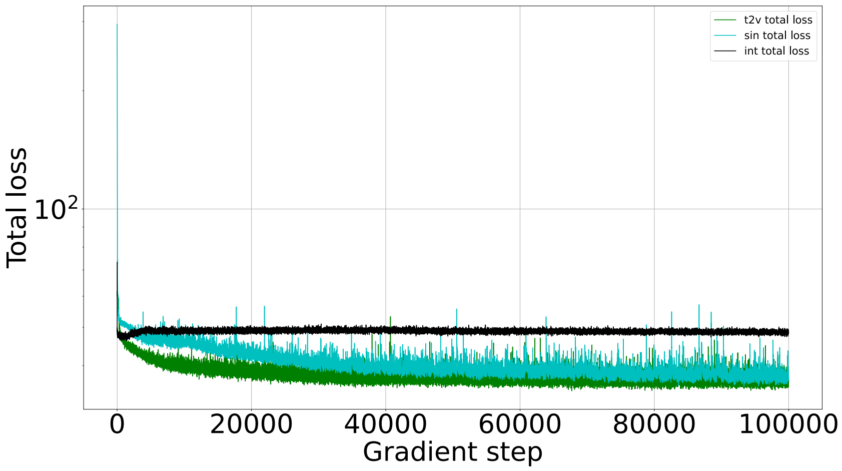


Figure 6.7: Total training loss for the first 100 000 gradient steps.

6.3.1 Initialization

Firstly, by setting both birth intensities to 0.0 and the probability of survival to 1.0, it is possible to gain insight into how the MT3v3 performs during startup and in the timesteps where new targets enter the sensor view, and how well these tracks are maintained over time. The position and velocity GOSPA for such a scenario with 100 timesteps is shown in figures 6.11 and 6.12. These results were obtained from the ensemble mean of 1000 Monte Carlo simulations. The corresponding ensemble averages are shown in tables 6.2 and 6.3 respectively. Note that the averages are shown both through the entire sequence and from $k = 15$ onwards. This is done to show the effect of the initialization process.

Architecture	GOSPA	Loc	False	Missed
CKF	0.346 ± 0.031	0.207	0.000	0.139
MT3v3	5.918 ± 1.553	0.402	0.647	4.869
JIPDA	6.020 ± 0.604	0.674	4.964	0.383
CKF*	0.287 ± 0.010	0.045	0.000	0.152
MT3v3*	3.074 ± 0.419	0.256	0.253	2.566
JIPDA*	5.445 ± 0.310	0.646	4.786	0.013

Table 6.2: Position GOSPA when birth intensities are set to 0.0 and probability of survival is set to 1.0. Stars indicate that the scores are calculated starting at timestep 15.

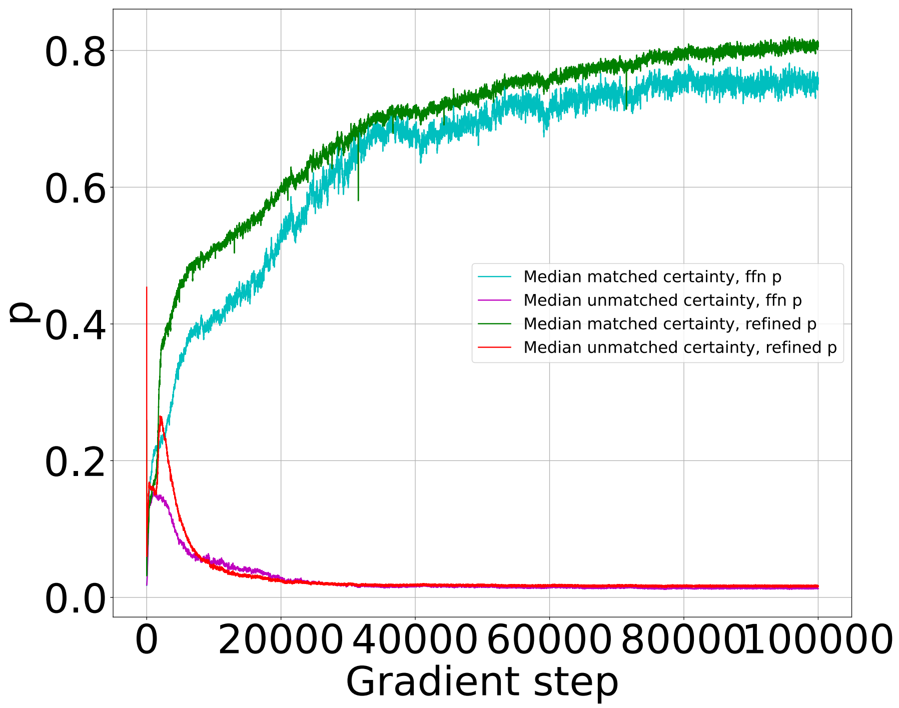


Figure 6.8: Certainty distributions for the first 100 000 gradient steps when comparing directly predicted and iteratively refined existence probabilities.

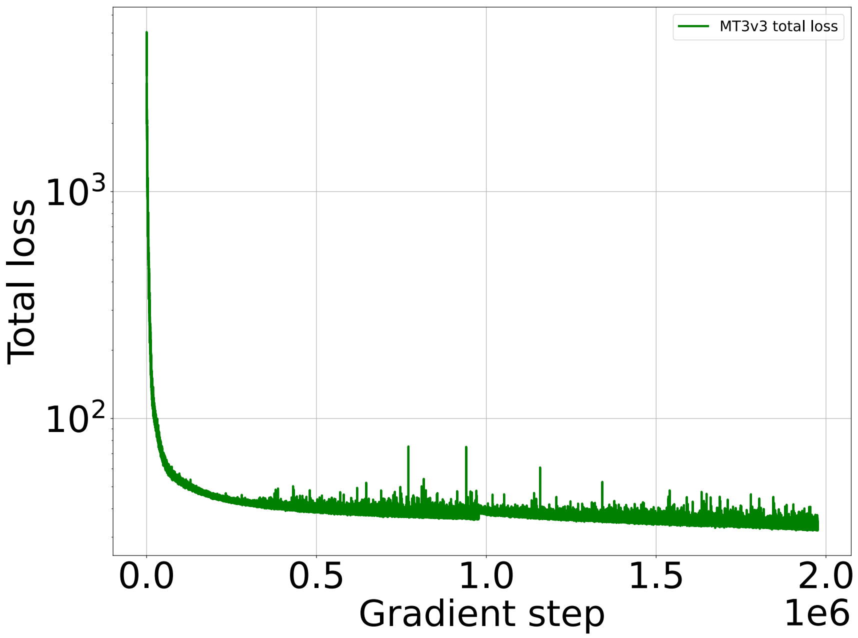


Figure 6.9: The total loss for the full MT3v3 training process.

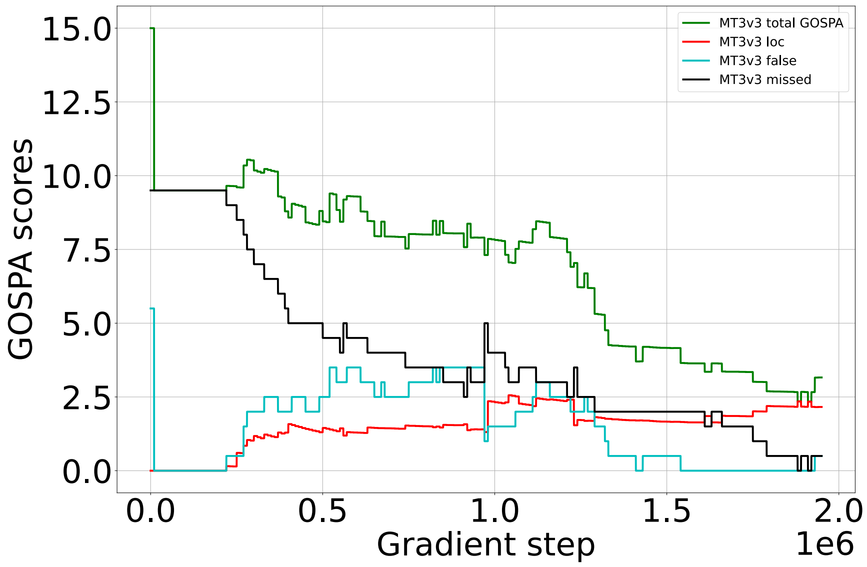


Figure 6.10: Periodically evaluated GOSPA scores for the MT3v3 during training.

Architecture	GOSPA	Loc	False	Missed
CKF	1.452 ± 1.323	1.159	0.063	0.230
MT3v3	6.789 ± 1.380	1.957	0.286	4.546
JIPDA	6.838 ± 0.689	1.525	4.948	0.365
CKF*	1.295 ± 0.247	1.110	0.000	0.186
MT3v3*	4.211 ± 0.476	1.515	0.177	2.518
JIPDA*	5.882 ± 0.345	1.083	4.786	0.012

Table 6.3: Velocity GOSPA when birth intensities are set to 0.0 and probability of survival is set to 1. Stars indicate that the scores are calculated starting at timestep 15

6.3.2 Doppler information and velocity estimates

Firstly, and perhaps most importantly, is the removal of doppler information in the input to the MT3v3. To benchmark how this affects the velocity outputs, the following was done: Firstly, the evaluation scheme used in (Pinto et al. 2022) was used, but expanded with velocity-GOSPA. Then, the original MT3v2 architecture using Task 1 was evaluated both with and without doppler information on the output. In the latter case, the doppler of all measurements was simply set to 0.0, and was done in order to assess how the pretrained MT3v2 is affected by the removal of doppler. Lastly, the MT3v3 using Time2Vec positional encoding is evaluated in the same way.

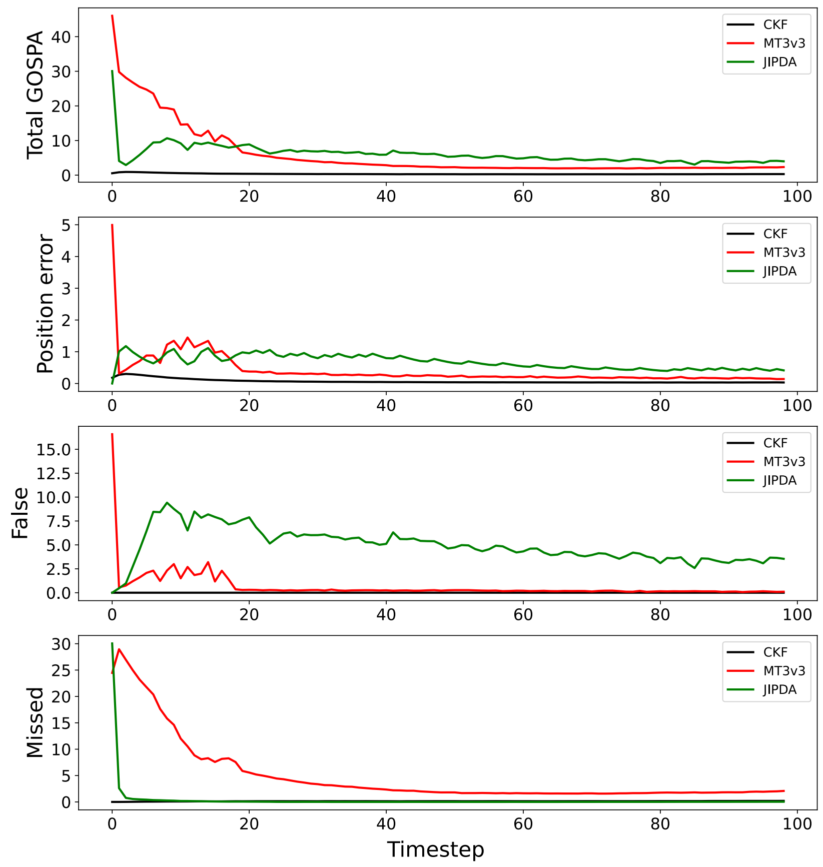


Figure 6.11: MT3v3 compared to the CKF and JIPDA through 100 timesteps. 1000 Monte Carlo simulations were used to produce ensemble averages for each timestep. The parameters in table 6.1 were used, except for birth intensities which were set to 0.0, and p_s was set to 1.0.

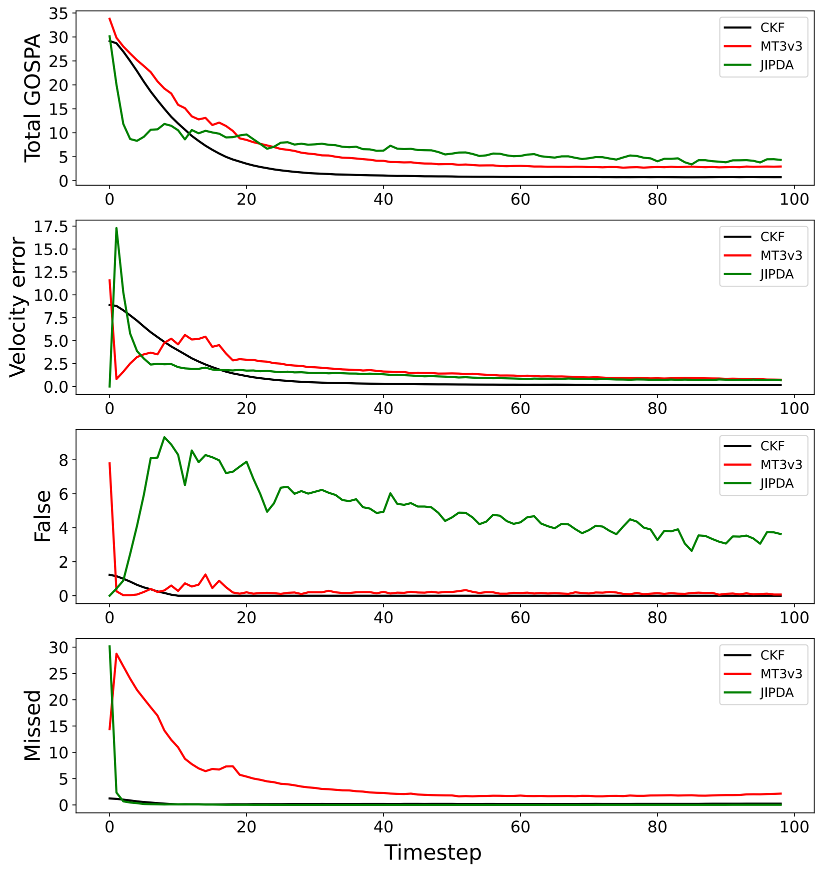


Figure 6.12: MT3v3 using velocity distance (magnitude only) compared to the CKF and JIPDA through 100 timesteps. 1000 Monte Carlo simulations were used to produce ensemble averages for each timestep. The parameters in table 6.1 were used, except for birth intensities which were set to 0.0, and p_s was set to 1.0.

Architecture	GOSPA	Loc	False	Missed
MT3v2	2.119 ± 0.161	0.949	0.120	1.050
MT3v2ND	69.561 ± 0.541	5.911	63.515	0.135
MT3v3	3.284 ± 1.104	0.234	0.917	2.133

Table 6.4: Position GOSPA. MT3v2ND is a pretrained MT3v2 tested on data where the doppler component of all measurements are 0.0.

Architecture	GOSPA	Loc	False	Missed
MT3v2	3.056 ± 0.175	1.906	0.110	1.040
MT3v2ND	67.588 ± 0.597	4.208	63.380	0.000
MT3v3	3.903 ± 1.342	1.820	0.433	1.650

Table 6.5: Velocity GOSPA. MT3v2ND is a pretrained MT3v2 tested on data where the doppler component of all measurements are 0.0.

6.3.3 Sliding window effects

In a practical implementation a sliding window will likely have to be implemented, in order to avoid an unbounded increase in memory consumption of the tracker as the measurement sequence increases in length. To test the extreme case of this, the MT3v3 was tested using short sliding windows. The results of this are presented in figure 6.13 which displays the result of three different sliding window lengths, $N = 15$, $N = 10$ and $N = 5$. Note that these tests are hot-started with 15 timesteps. The birth intensities and probability of survival are set such that any given target has a 0.05 probability of dying at any given moment and a target will spawn with probability 0.05. From the ensemble average GOSPA over time, it is clear that a smaller window size leads to more missed targets. It also appears to lead to somewhat more false estimates. This then leads to a higher total GOSPA score the smaller the sliding window.

Architecture	GOSPA	Loc	False	Missed
CKF	0.237 ± 0.018	0.162	0.000	0.075
MT3v3-5	6.266 ± 0.361	0.156	0.329	5.781
MT3v3-10	4.862 ± 0.210	0.182	0.271	4.409
MT3v3-15	4.178 ± 0.242	0.166	0.226	3.786
JIPDA	5.248 ± 0.219	0.439	4.352	0.458

Table 6.6: MT3v3 across 100 timesteps with sliding window sizes of 5, 10 and 15.

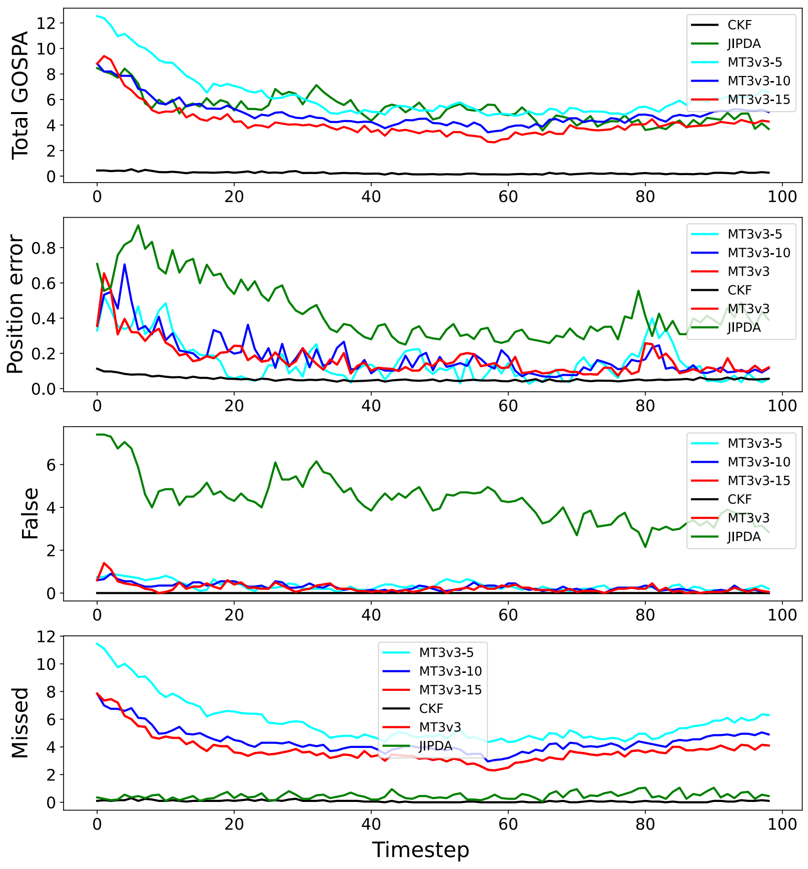


Figure 6.13: MT3v3 across 100 timesteps with sliding window sizes of 5, 10 and 15.

6.3.4 Variability in MT3v3 performance

Even while fixing the data generator parameters, there is a sizable gap between the best and worst GOSPA scores obtained by the MT3v3. In order to gain some insight into what these scenarios look like, a selection of scenarios in which the MT3v3 produces GOSPA scores that are both twice as high and twice as low as the JIPDA were studied in more detail.

When the MT3v3 performs poorly

A selection of four scenarios in which the MT3v3 performs very poorly are presented in figure 6.14. Note that each figure is zoomed in on the part of the sensor view in which the MT3v3 misses tracks the most, as it rarely ever produces false tracks. All scenarios have relatively low clutter and contains many ground-truth targets that move close to one another.

When the MT3v3 performs well

On the other hand, the MT3v3 performs much better than the JIPDA sometimes, and a selection of four such scenarios is presented in figure 6.15. The difference in performance between the two trackers in all of these cases can mostly be attributed to the JIPDA establishing false tracks. These scenarios contain comparatively more clutter and fewer targets than the scenarios in which the MT3v3 performs poorly.

6.3.5 Filter consistency

A metric that was not considered in previous assessments of the MT3 trackers is how consistent the final trained network is. This was tested for the MT3v3 using ensemble average NEES with the estimates and ground truth associations found when calculating GOSPA. Furthermore, the errors in x and y were tested for bias and whiteness.

Figure 6.16 shows the ensemble average NEES for 100 Monte Carlo simulations, and how it falls well below the lower χ^2 interval, with $\frac{\alpha}{2} = 0.025$, never peaking up into the confidence interval. The position errors were tested for whiteness using a Ljung-Box test with 10 lags. From this it was concluded that the position errors are sufficiently white. The errors were also checked for bias using a one-sample t-test, and it was also concluded that both errors are acceptable as zero-mean. The raw test results for whiteness and bias can be found in tables 6.8 and 6.7 respectively.

	Mean	t-statistic	p-value
x	-0.004	-0.326	0.745
y	0.031	1.544	0.126

Table 6.7: The results from the one-sample t-test to test the x and y errors for bias.

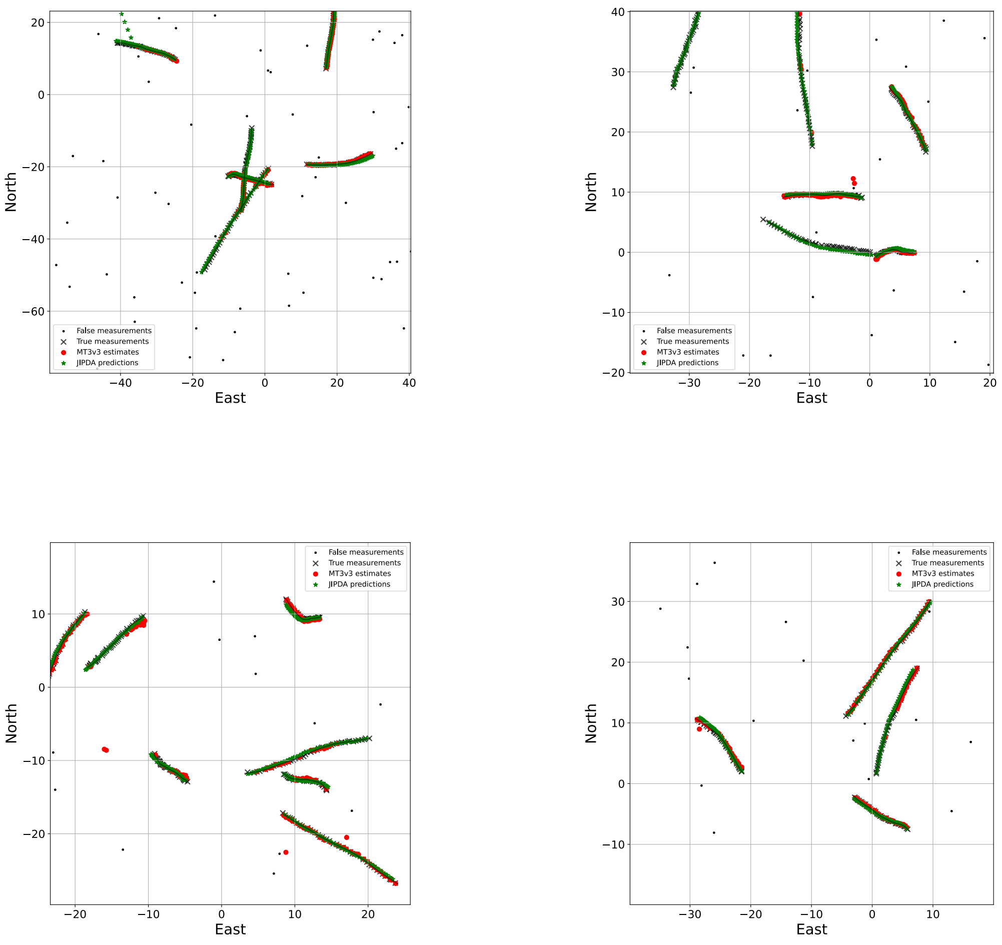


Figure 6.14: Four examples of the MT3v3 performing poorly. In all cases the MT3v3 is missing large sections of true tracks.

6.4 Generalizability tests

Generalizability was tested for the MT3v3 by simply generating scenarios with parameters outside of the ranges defined in equation 4.8. Specifically, the probability of detection, clutter intensity and measurement noise were altered to make new and more difficult scenarios. The JIPDA was not retuned for any of these scenarios, but instead was tuned once by giving it the median values of the parameter distributions.

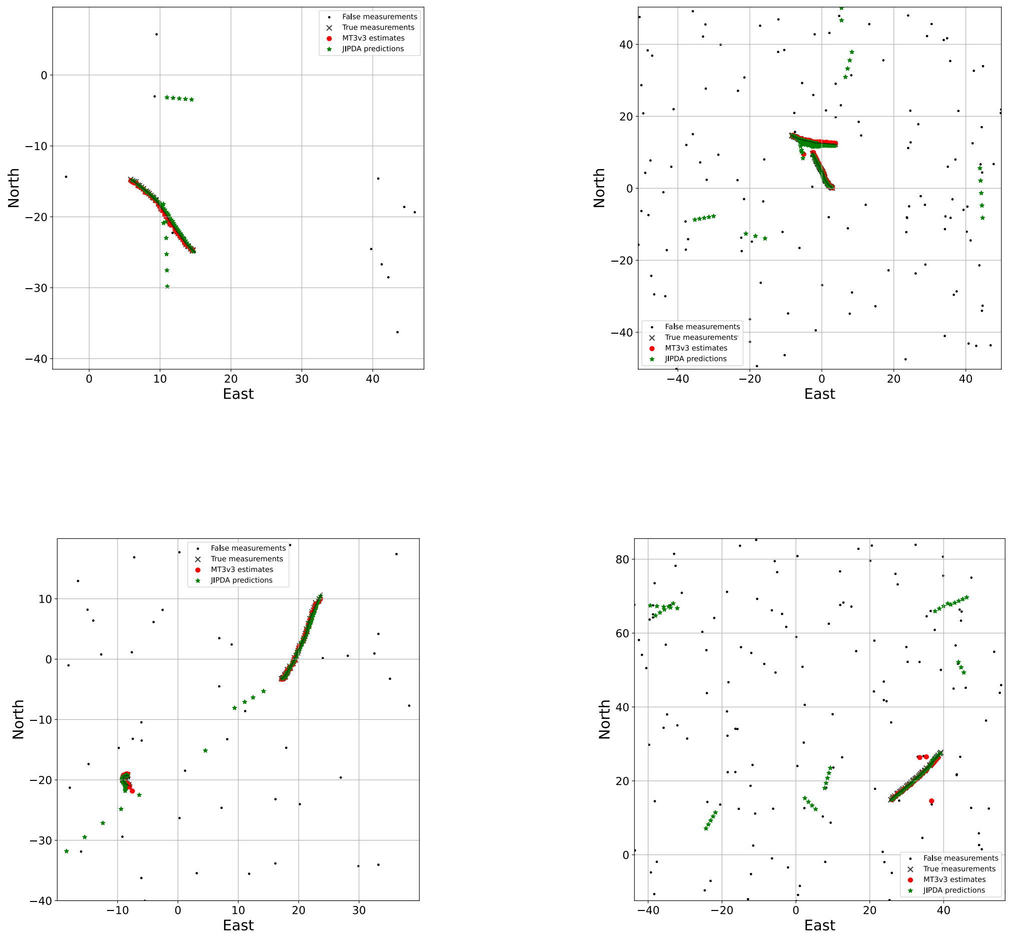


Figure 6.15: Four examples of the MT3v3 performing well. In all cases the JIPDA is making many false tracks.

Since the initialization phase has already been tested previously, all generalizability tests were carried out with hot-started trackers. That is, GOSPA estimates are calculated at the timesteps $n_h \leq k \leq n_{max}$ where $n_h > 0$. Unless explicitly stated, all ensemble averages below use $n_h = 25$, $n_{max} = 100$ and $n_{mc} = 1000$.

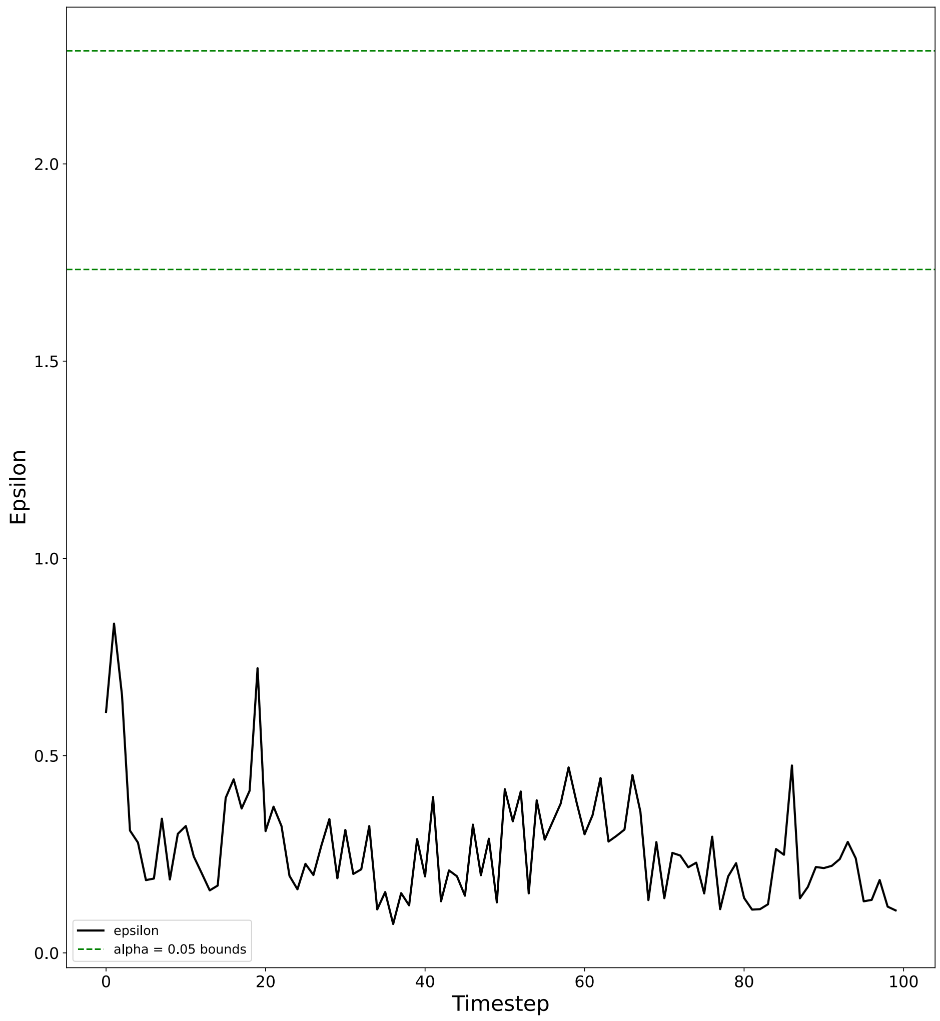


Figure 6.16: Ensemble average NEES for the MT3v3 across 100 timesteps. $\frac{\alpha}{2} = 0.025$ is used for the χ^2 intervals.

x			**y**		
Lag	Q	**p**	**Lag**	Q	**p**
1	0.094	0.759	1	2.874	0.090
2	0.710	0.701	2	3.136	0.208
3	0.834	0.841	3	3.191	0.363
4	1.300	0.861	4	3.343	0.502
5	2.330	0.802	5	3.347	0.647
6	2.874	0.825	6	3.420	0.755
7	3.101	0.875	7	4.198	0.757
8	3.145	0.925	8	4.236	0.835
9	3.235	0.954	9	4.247	0.894
10	4.092	0.943	10	5.244	0.874

Table 6.8: Ljung-Box test results for 10 lags. Q is the value of the Ljung-Box statistic, while **p** is the corresponding p-value.

6.4.1 Probability of detection

Lowering the probability of detection to values below what was seen in training for the MT3v3 resulted in significantly more missed targets, as can be seen in table 6.9. The JIPDA on the other hand, did not perform significantly worse when lowering p_d.

Algorithm	**GOSPA**	**Loc**	**False**	**Missed**
CKF	2.095 ± 0.465	1.062	0.000	1.033
MT3v3	14.528 ± 1.900	0.627	1.221	12.680
JIPDA	9.564 ± 0.860	1.751	3.419	4.394

Table 6.9: Ensemble averages for GOSPA with position localization errors over 1000 Monte Carlo simulations using the same parameters as in table 6.1 except for p_d which is set to 0.5. All scenarios were hot-started with 15 time steps.

6.4.2 Clutter intensity

Higher clutter intensities make for more challenging tracking scenarios. As such, a clutter intensity that resulted in an average of 30 false measurements for each timestep was selected. The resulting position GOSPA is shown in table 6.10. The MT3v3 is shown to be significantly less prone to establishing false tracks compared to the JIPDA, while only missing targets a bit more often.

Algorithm	GOSPA	Loc	False	Missed
CKF	0.457 ± 0.043	0.270	0.000	0.186
MT3v3	3.794 ± 0.655	0.610	1.061	2.123
JIPDA	10.954 ± 1.353	0.638	9.711	0.605

Table 6.10: Ensemble averages for GOSPA with position localization errors over 1000 Monte Carlo simulations using the same parameters as in table 6.1 except for λ_c which was multiplied by a factor of 5 from the highest value used in training. All scenarios were hot-started with 15 time steps.

6.4.3 Measurement noise

Measurement noise is assumed to be known to a certain extent a-priori, but the effect of increased measurement noise is tested regardless in order to get an idea of how the MT3v3 performs when there is a mismatch between the configured and actual measurement noise. Increasing both components by a factor of 10 does not yield significantly different results for either of the trackers as table 6.11 shows. However, increasing the measurement noises to $\sigma_r = 10$ and $\sigma_\theta = \frac{\pi}{4}$ yielded substantially more missed targets and an increase in localization errors, as table 6.12. While the MT3v3 yielded a lower localization score than both the CKF and JIPDA, it also missed more targets.

Algorithm	GOSPA	Loc	False	Missed
CKF	0.805 ± 0.132	0.238	0.000	0.091
MT3v3	3.206 ± 0.313	0.671	1.224	1.311
JIPDA	4.377 ± 0.167	0.644	3.303	0.430

Table 6.11: Ensemble averages for GOSPA with position localization errors over 1000 Monte Carlo simulations using the same parameters as in table 6.1, except for the measurement noise, which were set to $\sigma_r = 1.0$ $\sigma_\theta = 0.1745$. All scenarios were hot-started with 15 time steps.

Algorithm	GOSPA	Loc	False	Missed
CKF	1.377 ± 0.267	1.312	0.000	0.065
MT3v3	5.794 ± 0.243	0.921	0.402	4.471
JIPDA	9.821 ± 0.236	2.555	4.353	2.914

Table 6.12: Ensemble averages for GOSPA with position localization errors over 1000 Monte Carlo simulations using the same parameters as in table 6.1, except for the measurement noise, which were set to $\sigma_r = 10$ $\sigma_\theta = \frac{\pi}{4}$. All scenarios were hot-started with 15 time steps.

6.4.4 Birth and probability of survival

By increasing only the birth intensity, more targets will be present at any given timestep, while the opposite is true when decreasing probability of survival. If both birth intensities are increased and probability of survival is decreased, more and shorter tracks will appear. All three cases were tested for the MT3v3. The first of these is shown in table 6.13, and in terms of total GOSPA, the MT3v3 and JIPDA perform very similarly. However, most of the total GOSPA score for the MT3v3 is from missed targets, while it is mostly from false tracks for the JIPDA.

Architecture	**GOSPA**	**Loc**	**False**	**Missed**
CKF	0.594 ± 0.034	0.384	0.000	0.211
MT3v3	7.853 ± 0.320	0.610	1.933	5.310
JIPDA	7.831 ± 0.394	1.150	5.192	1.488

Table 6.13: Ensemble averages for GOSPA with position localization errors over 1000 Monte Carlo simulations using the same parameters as in table 6.1, except for birth intensity which was set to 0.25, and probability of survival was set to 0.75

Setting the probability of survival to 1.0 and the birth intensity to 0.25 yields scenarios with many targets, and the performance on these is shown in table 6.14. The MT3v3 ends up missing many more targets than the JIPDA, and thus performs significantly worse. On the other hand, by setting the birth intensity to 0.0 and the probability of survival to 0.75, there will quickly be very few targets in the scenario. As table 6.15 shows, the JIPDA performance is significantly better as compared to the previous scenarios in this subsection. While the MT3v3 performance is also better, it is still much worse than the "normal" scenarios that were tested earlier.

Architecture	**GOSPA**	**Loc**	**False**	**Missed**
CKF	1.730 ± 0.113	0.987	0.000	0.744
MT3v3	29.884 ± 1.189	1.717	0.878	27.290
JIPDA	8.462 ± 0.682	3.816	2.465	2.181

Table 6.14: Ensemble averages for GOSPA with position localization errors over 1000 Monte Carlo simulations using the same parameters as in table 6.1, except for birth intensity and probability of survival which were set to 0.25 and 1.0 respectively.

6.4.5 Long input sequences

The maximum sequence length that was used during training was 100 timesteps. Seeing how the MT3v3 performs on sequences much longer than this can reveal if the network is able to attend to a reasonable set of measurements, even when there is a lot of information in the overall input data. To test this, scenarios with 1000 timesteps were generated and used to evaluate the trackers in the same

Architecture	GOSPA	Loc	False	Missed
CKF	0.286 ± 0.111	0.246	0.000	0.040
MT3v3	7.240 ± 0.576	0.527	2.626	4.088
JIPDA	3.905 ± 0.355	0.496	2.557	0.852

Table 6.15: Ensemble averages for GOSPA with position localization errors over 1000 Monte Carlo simulations using the same parameters as in table 6.1, except for birth intensity and probability of survival which were set to 0.0 and 0.75 respectively.

way as before. To calculate the ensemble averages, $n_{mc} = 100$ was used, in order to keep testing times tractable. All other data generator parameters aside from sequence length were set to values inside the ranges used while training, in order to see the effects of longer input sequences in isolation. The ensemble averages are presented in table , and it appears that the increased sequence length does not impact performance negatively or positively.

Algorithm	GOSPA	Loc	False	Missed
CKF	0.491 ± 0.013	0.102	0.000	0.389
MT3v3	2.318 ± 0.218	0.196	0.539	1.583
JIPDA	3.937 ± 0.075	0.393	3.209	0.334

Table 6.16: Ensemble averages for GOSPA with position localization errors over 100 Monte Carlo simulations using the same parameters as in table 6.1, except for the sequence length, which was fixed to 1000 timesteps.

6.5 A selection of MTT scenarios

Aside from just using the data generator to benchmark the average performance of the MT3v3 on the scenarios produced by the data generator, a handful of scenarios were also generated using the traffic- and detector sim. These scenarios are hand-crafted to be difficult and test how well the tracker handles multiple targets in close proximity.

6.5.1 Target crossing

A subset of the handcrafted scenarios specifically test how well the MT3v3 handles targets that cross paths. Figure 6.17 shows three such scenarios. The first is a simple two-target crossing, while the second has two targets cross the first path in opposite directions. The third crossing scenario is similar to the first, but adds two stationary targets. The average GOSPA for the three crossing examples is shown in table 6.17. The MT3v3 across the board has a higher total GOSPA, but only the crossing scenario with stationary targets yields a higher total GOSPA with statistical significance – The confidence intervals are higher than before due to

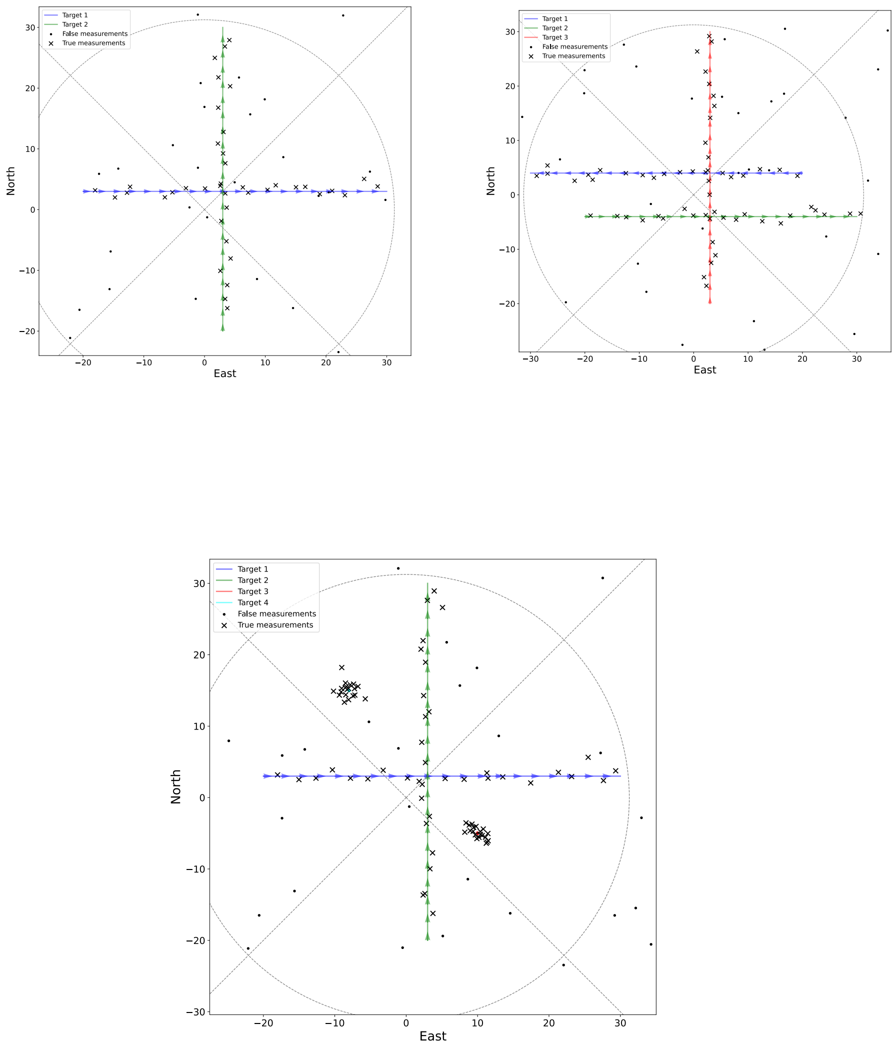


Figure 6.17: Three crossing examples. Arrows are used to indicate direction and magnitude of the targets' velocities.

not using ensemble averages from Monte Carlo simulations. As before, the JIPDA produces more false tracks, but the MT3v3 also produces a higher number of false tracks in these scenarios than before. The MT3v3 and JIPDA output estimates are shown in figure 6.18.

Scenario	**Algorithm**	**GOSPA**	**Loc**	**False**	**Missed**
Single Crossing	MT3v3	12.962 ± 3.310	1.942	5.306	5.714
	JIPDA	8.759 ± 1.692	0.800	7.653	0.306
Advanced Crossing	MT3v3	16.084 ± 3.422	1.594	4.898	9.592
	JIPDA	9.264 ± 2.549	0.815	8.143	0.306
Stationary Targets	MT3v3	21.324 ± 2.549	1.467	7.918	11.939
	JIPDA	11.835 ± 1.327	0.713	10.714	0.408

Table 6.17: The GOSPA scores for a single run of each handcrafted crossing scenario.

6.5.2 Parallel targets

Another subset of the handcrafted scenarios are meant to test how well the MT3v3 can separate two targets moving in parallel. These are shown in figure 6.19. The first two scenarios have two targets that approach one another and together with one another respectively. The third is the most complex of the three, as the targets maneuver heavily while also moving close to one another.

The average GOSPA scores for the three parallel scenarios are shown in table 6.18. The total GOSPA score for the MT3v3 is higher than the JIPDA in all three cases, but without statistical significance in the first two. As with the crossing examples, the MT3v3 makes more false tracks in all three parallel scenarios as compared to previous benchmarking. The MT3v3 and JIPDA output estimates are shown in figure 6.20.

Scenario	**Algorithm**	**GOSPA**	**Loc**	**False**	**Missed**
Parallel Approach	MT3v3	7.203 ± 1.512	1.186	4.661	1.356
	JIPDA	6.295 ± 1.481	0.617	5.508	0.169
Parallel Together	MT3v3	7.373 ± 1.780	1.186	4.831	1.356
	JIPDA	6.295 ± 1.481	0.617	5.508	0.169
Maneuvering Together	MT3v3	13.754 ± 2.330	2.738	3.983	7.034
	JIPDA	6.595 ± 1.061	4.306	1.949	0.339

Table 6.18: The GOSPA scores for a single run of each handcrafted parallel scenario.

6.6 Bearings only tracking

The problem of bearings-only tracking is, as the name implies, about tracking targets using nothing but bearings. Not having access to range measurements thus

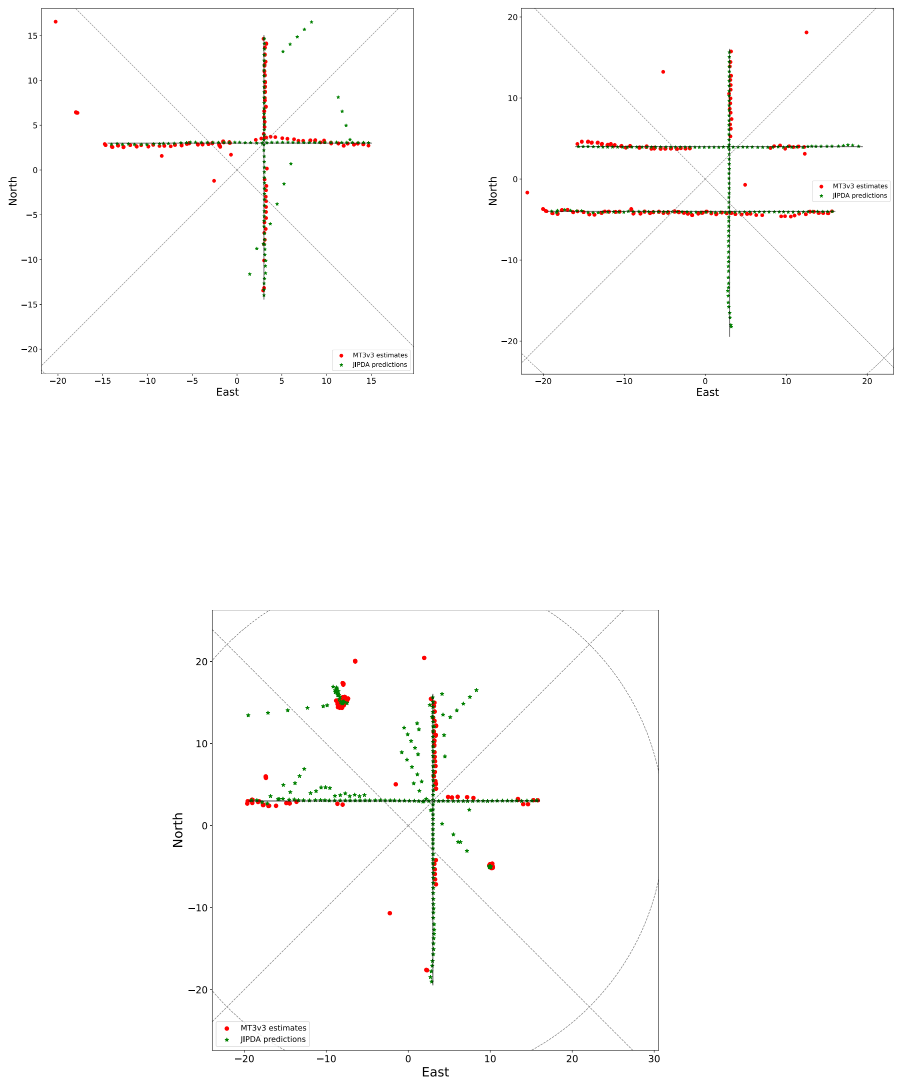


Figure 6.18: The tracker outputs for the three crossing scenarios.

makes this a much more challenging problem. While the MT3v3 was not designed with this in mind, it was tested regardless due to the flexibility of the data generator to produce data like this. The range component of the data generator was simply

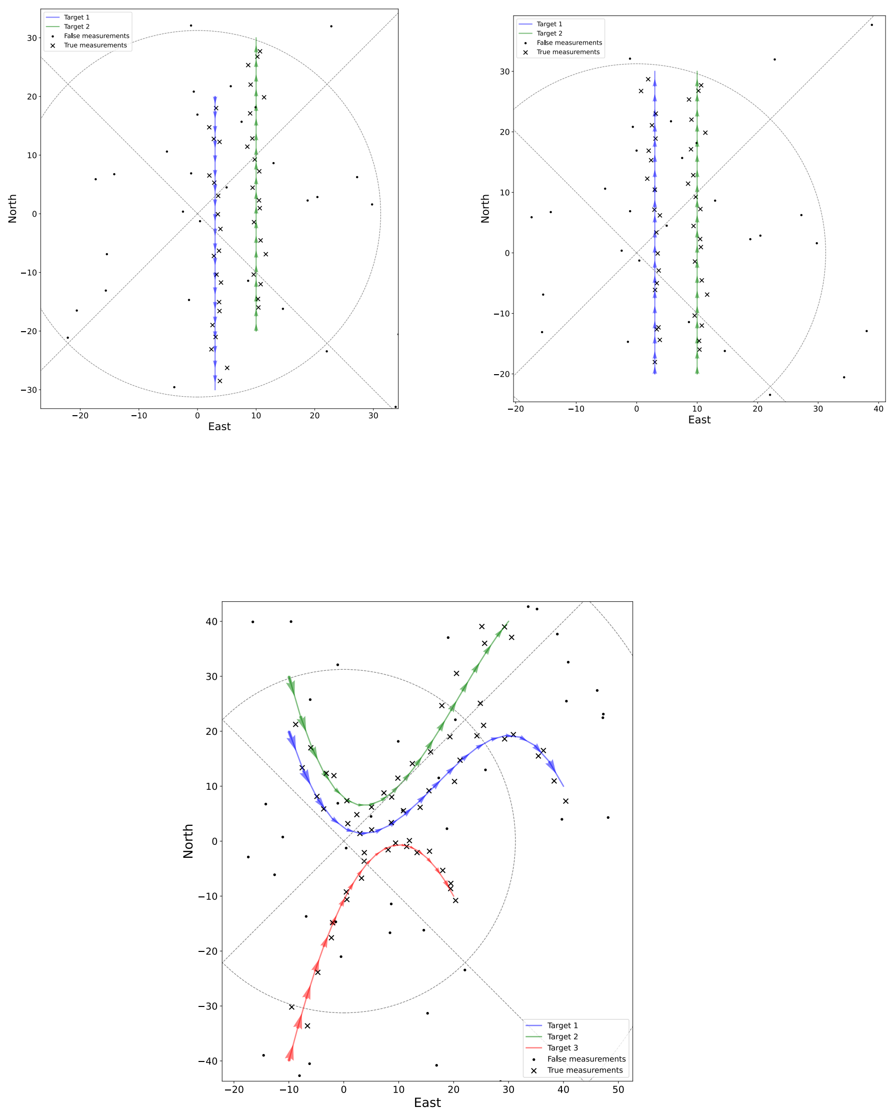
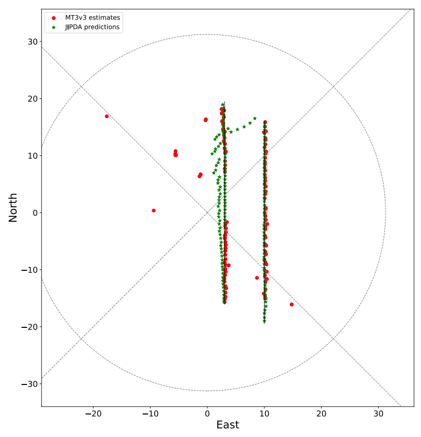
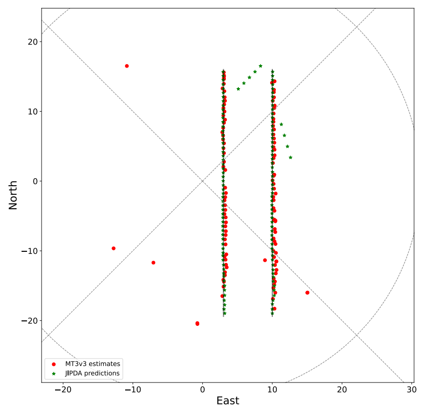


Figure 6.19: Three scenarios with targets moving in parallel. Arrows are used to indicate direction and magnitude of the targets' velocities.

removed, and the MT3v3 was configured to expect measurements on the form $[\theta\ t]$. The development of the total loss, the certainty distributions and a sample of the output hypotheses for the current scenario is shown in figure 6.21. The network





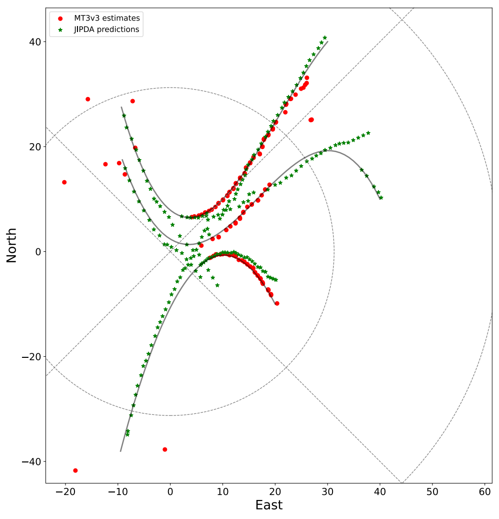


Figure 6.20: The tracker outputs for the three parallel scenarios.

quickly learns to place hypotheses on the radial line between the sensor and a target. However, the network is also unable to reduce the spread of the hypotheses, and the certainty distribution stagnates. As shown in figure 6.22, further training does

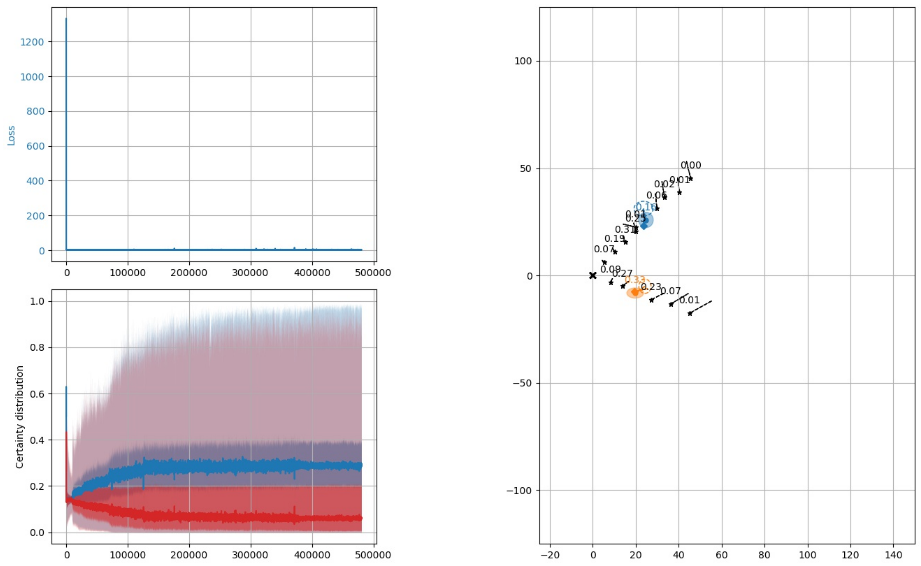


Figure 6.21: A selection of training data from the bearings-only MT3v3. Stars represent estimates produced by the network, and each corresponding number is the existence probability of that component. Clutter measurements are not displayed.

not appear to lead to any substantial improvements.

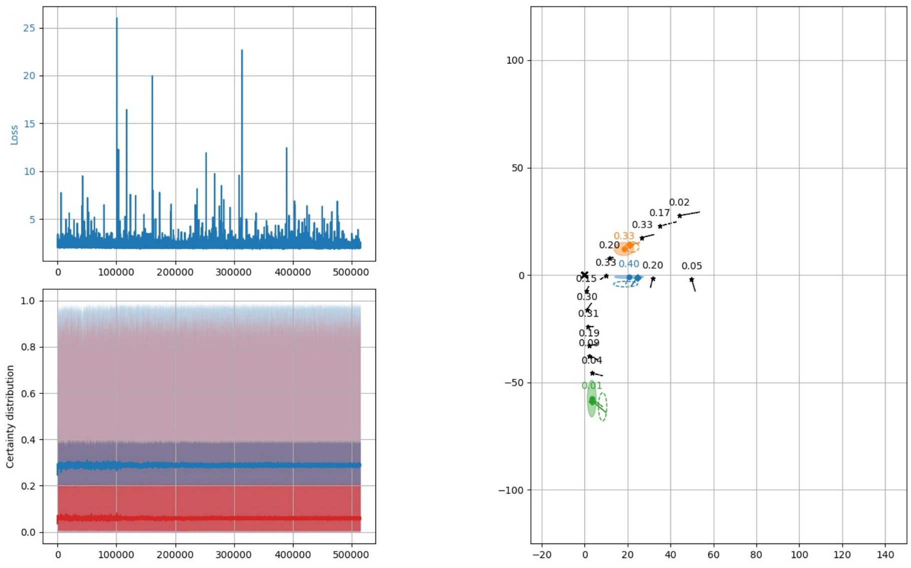


Figure 6.22: A selection of training data from the bearings-only MT3v3, continuing from the network that was already trained with about 500k steps shown in figure 6.21. Stars represent estimates produced by the network, and each corresponding number is the existence probability of that component. Clutter measurements are not displayed.

7 Discussion

7.1 MT3v3 architecture changes

The main architecture changes that were tested are the position encoding scheme, iterative refinement of existence probabilities, and full covariance estimates. Improving flexibility and robustness were the two main factors considered when determining how beneficial an architectural change is, since these were the motivating factors behind designing the MT3v3 to begin with.

7.1.1 Position encoding

The three encoding schemes that were considered were an integer difference approach, the sinusoidal encoding from (Vaswani et al. 2017), and the Time2Vec approach as described in (Kazemi et al. 2019). The latter was determined to be the best choice for the MT3v3, as a reasonable middle-ground between good numerical properties and evaluation performance.

A common factor between all the implemented position encoders is the requirement that they are able to handle input sequences with an arbitrary number of elements. This was made to be a hard requirement since the goal of the MT3v3 is to be more flexible than the previous generations of MT3 trackers,

Furthermore, using more timesteps means that the tracker has access to more information, which could lead to performance gains. The downside of accepting a variable number of input measurements is that there are no direct guarantees that the MT3v3 will produce good outputs when given an input sequence that is longer than the longest sequence used for training. It is hypothesizes that for very long sequences, the MT3v3 will attend to the latest N measurements, and that the upper limit of usable information is determined by d_{model}, since this defines the embedding dimension used throughout the network.

In terms of training performance for the MT3v3 using integer difference encoding, the standard deviations for state estimates, certainty distributions, and the total output loss all plateaued early in the training process, indicating that no useful learning took place. This may be caused by numerical problems with this approach: Since the position encoding is added to the input embeddings, and the integer differences between the indices of two elements increases linearly with the distance between the two elements in the input sequence, the position encoding will dominate the embedding for long sequences. It is hypothesized that normalizing the position encodings would yield better numerical properties for this approach. This could be done either by normalizing with the current sequence length, or with the maximum sequence length for all inputs. The former of these approaches would yield a different encoding for every sequence length, but could be used for arbitrary sequence lengths. The latter would produce the same encoding for all sequences, but would be limited to sequences defined by the maximum length used for normalization. In any case, this approach was deemed to be naive and subsequently discarded.

The matched and unmatched certainty distributions for the MT3v3 with sinusoidal encoding converged to 1 and 0 respectively within the first few hundred thousand gradient steps, which is a clear improvement over the integer encoding. The standard deviations of the state estimates are somewhat high, and also seem to plateau after about 20000 gradient steps. The total loss decreases evenly with increasing gradient steps. However, the certainty distributions and total loss appears somewhat unstable, as indicated by the random spikes in their respective graphs. The estimated standard deviations at the output appear to be more stable.

As for the Time2Vec approach, the average matched and unmatched certainty also converge towards 1 and 0 respectively, and appear to be more stable than the sinusoidal encodings. The unmatched certainty average converges somewhat more quickly than the sinusoidal, but the matched certainty average converges more slowly. It is hypothesized that since Time2Vec introduces a set of learnable parameters for frequency and phase to the sinusoidal embedding, as well as a linear dimension, the MT3v3 will potentially find a more optimal configuration compared to its sinusoidal counterpart. This is because the additional learnable parameters offer the network greater flexibility in adapting to the training scenarios, which in turn comes with the potential of better numerical properties for the architecture. However, it is also anticipated that due to the increased complexity introduced by these additional parameters, the network may require more time to attribute high existence probabilities to matched outputs. However, as the total loss graph shows, the overall convergence is quicker. Since all outputs with an existence probability higher than 0.5 are considered to be alive, the effect of a slower convergence in the average certainties will not be as apparent beyond a certain point. It is also worth noting that the standard deviations for the position estimates for the MT3v3 using Time2Vec are the highest out of the three position encoders. Meanwhile, the standard deviations for the velocity estimates are a little better than the sinusoidal encoding. In both cases, the Time2Vec standard deviations are more stable than the sinusoidal. While these both appear to stagnate early on, this is also something that was noted early in the training process for the MT3v2. It is hypothesized that the total loss at the output of the final decoder is more affected by differentiating true and false measurements and estimating the total number of targets in a scenario than refining the estimated state uncertainties. As such, it is assumed that these will improve with more training.

A weakness in this approach of architecture selection is that each architecture was trained for 10^5 gradient steps before analyzing the resultant data output, and making a decision about architecture. This is one order of magnitude lower than what was used for the final MT3v3. In an attempt to make more informed architecture decisions, the data generator parameters were fixed at their median values for this training. However, there may still be some interaction effects or other nuances to each approach that is only seen after training for longer or for a particular combination of data generation parameters. This is something that remains unknown for the sinusoidal and integer encoding. In either case, due to the nature of tracking data, i.e. sequentially arranged according to time of measurement, as well as the promising results presented in (Kazemi et al. 2019), Time2Vec appears to be a good choice of encoding for the MTT task.

7.1.2 Existence probability

The problem of introducing iterative refinement to the existence probabilities at the output was solved by starting each component at some learned point based on the object queries, then iteratively refining each existence probability for each decoder layer. The MT3v3 was not trained fully without this new addition, so a direct comparison between a single FFN and iterative refinement for estimating existence probabilities cannot be done. However, the MT3v3 was trained for 10^5 steps with the old and the new existence probability estimator, and the results are very close to one another, as figure 6.8 shows.

When the MT3v3 was trained for the full number of iterations determined earlier, iterative refinement of existence probability was used. It is hypothesized that the iterative refinement process does not add sufficient probability mass to the components that align with potential tracks, given how the MT3v3 missed track rate is much higher than the false track rate. Instead of learning what the initial probability existence should be, it may be a better approach to simply let $p_i^0 = 0.5$ for all i and let the iterative refinement stage adjust the components up and down from this middle-point.

7.1.3 Full covariance matrices

In (Pinto et al. 2022) it is hypothesized that estimating the full covariance matrices at the output could improve the relatively high localization errors of the MT3v2. This was the motivation for doing this in the MT3v3. While the MT3v3 sees substantially lower localization errors than the MT3v2, it is not possible to deduce that this is the result of adding full covariance matrix estimates. Since the covariance matrix for any given output estimate is the result of separate feed-forward networks, it does not directly affect the output estimate for that scenario. Instead, it affects the gradient of the loss function, which in turn will affect future outputs.

However, the network that predicts initial velocity from position estimates was increased in size. The network may also fundamentally learn tracking in a different manner to the MT3v2, given that it does not utilize doppler measurements. As such, it is unknown what the full covariance matrices actually contribute to.

Judging by the NEES shown in figure 6.16 it appears that the covariance estimates should be higher, in order to make the MT3v3 consistent. However, due to the covariance not directly affecting the output estimates of the next step, this would not actually improve any state predictions.

While the standard deviations for the output estimates plateau early in the training process in both the MT3v2 and MT3v3, it is more prominent in the latter. This may be caused by the fact that the covariance FFN has to estimate four times the number of outputs, and as such, learning becomes slower. There is also something fundamentally different between the covariance that comes from a Bayesian tracker like the JIPDA and the covariance estimates from the MT3v3. This is discussed in more detail in section 7.6.3.

7.1.4 Potential future architecture changes

Since the MT3v3 does not greatly alter the fundamental architecture of the MT3v2, it is also based on the DETR from (Zhu et al. 2020). Since this is a parallel architecture, it does not explicitly use the estimates of the previous timestep to generate the estimates for the current timestep. In terms of target tracking, and especially when comparing to the Bayesian approaches, this could be a sub-optimal approach when it comes to the accuracy of the tracker. Additionally, since the MT3v3 generates outputs in parallel, it is harder to compare it to a model-based approach, since they process the information in the input on two fundamentally different premises. For this reason, it is worth exploring an autoregressive decoder. The original Transformer architecture in (Vaswani et al. 2017) is autoregressive, and the literature on autoregressive Transformers is vast. An autoregressive Transformer can naturally model the joint probability of the entire output sequence given the input sequence as a product of conditional probabilities, since each output is conditioned on both the inputs and all previous inputs. This architecture approach is more in-line with how the model-based trackers are formulated.

Another future architecture change could focus on the bearings-only problem. As previously shown, the output estimates of the current MT3v3 stagnates fairly early in training when only using angle measurements. It is thus likely that a major change in the encoder or decoder would have to be made in order to achieve better results on this task.

7.2 Benchmarking the MT3v3

The performance of the MT3v3 in a ”standard configuration”, i.e. with data generator parameters within the limits used during training, was tested and compared to the CKF and JIPDA.

7.2.1 Track initialization

The false detection and missed detection components of the GOSPA shown in figure 6.11 shows how the MT3v3 performs in terms of track initialization and convergence. Firstly, the MT3v3 on average takes longer to produce outputs that are considered to be alive as compared to the JIPDA. This can be seen in how the missed score for the CKF converges after around 5 timesteps, while the MT3v3 only does so after around 20 timesteps. The MT3v3 also never converges to the JIPDA in terms of missed detections, As for false detections, the MT3v3 initially decreases in the first few timesteps, then increases again, before finally decreasing substantially from 25 timesteps and onwards. The false detections for the JIPDA initially increases, then slowly decays across the scenario, but with a value much higher than the MT3v3. Lastly, the error in position estimates for both trackers appear to converge fairly quickly. Overall, this leads to a total GOSPA that is lower for the JIPDA in the first 20 timesteps, before the MT3v3 catches up and performs better for the rest of the scenario.

It is desirable for a tracker to correctly identify true targets in as few timesteps as possible, especially when other parts of an autonomy system depends on the situational awareness a tracker provides. That is, the longer it takes to establish a track, the longer a potentially important target can exist in sensor view without any other parts of the system being informed. Since the time delta between measurements in the data generator is set to 0.1s, the 20 timesteps it takes for the MT3v3 to establish tracks corresponds to 2.0s of wall-clock time. This will however be different for every specific system the tracker will be part of, which means this consideration needs to be made on a case-by-case basis.

7.2.2 Position estimates

The position estimates of the MT3v3 are considerably better than the MT3v2. As discussed earlier, there could be numerous reasons for this – The full covariance matrix estimation and the increased velocity predictor network size to name a few. However, the training process itself could also have facilitated this. As indicated in (Strøm 2022), the MT3v2 seemed to almost exclusively base the position estimates on the values of the measurements, which is not the case for the MT3v3 to the same extent. It is hypothesized that since the network is trained on a more varied set of scenarios, that the network somewhat learns a general sense of motion model, and how to weigh measurements against the expected target motion. It is important to clarify that this type of weighing is not explicitly modelled, and is only assumed to have been learned implicitly. Since the MT3v3 does not use doppler measurements, it is also hypothesized that the network is forced into learn some concept of motion model in order to separate true and false measurements.

Furthermore, it is hypothesized that the adaptive contrastive loss helps the network during the training process, in the sense that localization errors will affect the gradient of the loss function to a greater extent as α decreases. This could in turn prevent stagnation in the learning process and allow for lower localization errors.

7.2.3 Velocity estimates

If the MT3v3 is to compete with other Bayesian trackers, is important that it can produce good velocity estimates without any direct information about velocity in the input (i.e. without doppler measurements). As shown in figure 6.12, the velocity error for the MT3v3 converges to the velocity error of the JIPDA. However, this takes about 80 timesteps which is very slow. Despite the slow convergence, the errors in velocity are still acceptable after around 20 timesteps. As table 6.3 shows, the velocity errors across the entire sequence is only about 0.5 meters per second higher for the MT3v3 compared to the JIPDA.

It is also worth noting that the errors in velocity estimates for the JIPDA initially decreases below the CKF, but converges to a higher value than the CKF. It is likely that this is caused by differing initial values for each filter's initial state uncertainty, as these were not kept the same throughout testing. When using points in velocity space, the CKF also produces some false target estimates. This is again likely caused by the way the CKF is initialized. Since there is no velocity information in the very first estimate, the first velocity estimate will in all likelihood also be far off the true velocity of the target. In velocity space, this estimate may thus be seen as false.

7.2.4 Sliding window

A smaller sliding window will reduce the available information in the input sequence to the network. Since the MT3v3 was shown to use about 15 timesteps to establish tracks, it is expected that using a window size smaller than this will lead to poor performance. This aligns well with what figure 6.13 shows in terms of missed targets. A smaller window size also produces more false estimates, which could indicate that the network becomes overconfident in estimates when a rich measurement history is not available. This could also be the reason for the spike in false estimates seen between timesteps 1 and 20 in figure 6.11. Since 20 timesteps was the lower bound on the number of timesteps for scenarios used for training, it is possible that this poor performance is also the result of poor generalization to short input sequences. Lastly, while the localization error is lower for all MT3v3 and sliding window combinations, this is mostly caused by zero localization error due to not having correctly detected any targets.

7.2.5 Variation in birth and death

The MT3v3 performs worse when setting birth intensities to zero and lowering the probability of survival. When the birth intensity is high, there are many more targets in the scenario, and as such, worse performance is to be expected. However, when setting lowering probability of survival, the MT3v3 also performs poorly. This may be due to most targets in the scene dying before the long initialization period required by the MT3v3 has passed.

7.2.6 Variation in performance

As seen in figure 6.14, the MT3v3 performs very poorly in certain scenarios, in the sense that it misses entire or parts of multiple tracks. This is observed for both high and low clutter intensity, and even with $p_d = 1.0$. By visual inspection there does not appear to be any recurring effects that triggers this, except for potentially the case where multiple targets maneuver close to one another. This performance loss was also not observed for any scenario with less than four targets present. In any case, missing tracks to this extent is detrimental to the usefulness of the MT3v3 as a multitarget tracker, and understanding why these situations occur is essential if the goal is to use this tracker in any real-world scenario. However, since the MT3v3 is an end-to-end neural approach, interpretability is a challenge, as is discussed in section 7.6.3.

On the other hand, there are scenarios in which the MT3v3 performs much better than the JIPDA. These are exclusively scenarios where the JIPDA produces many false tracks. This is consistent with the observations that the MT3v3 rarely produces false tracks but misses more targets, while the opposite is true for the JIPDA. All scenarios where the MT3v3 performs substantially better, including those shown in figure 6.15, have between one and three targets present, and a relatively high clutter density at certain regions of the sensor field of view.

The reason for the high variability between scenarios with few targets and scenarios with many targets is not certain. A possible reason for this is simply that scenarios like these are rarely generated during training. Concretely, finding the first four scenarios where the MT3v3 had a total average GOSPA score of more than two times that of the JIPDA took 2141 randomly generated scenarios. Similarly, finding the first four scenarios where the total average GOSPA score of the MT3v3 was less than half of the JIPDA took 9875 randomly generated scenarios. Regardless of how frequent they appear, the worst-case performance of a tracker is important to keep in mind, and for the MT3v3 this is particularly bad when comparing to its average performance.

Another indicator of the variation in performance are the confidence intervals throughout all MT3v3 results. In general, they tend to be higher than the CKF and JIPDA. However, noting the difference in confidence intervals in table 6.2, it appears that this variation is largely due to target initialization.

As figure 6.10 shows, the MT3v3 stops making false estimates after about 10^6 gradient steps, while the missed target loss remains higher than this. It is possible that the MT3v3 either has not learned how to properly assign existence probabilities, and that more training would improve this. It could also be that the MT3v3 has been trained for too many gradient steps, and that it becomes overconfident in the estimates it learns to produce in training, and that an earlier checkpoint would actually improve overall performance. The latter hypothesis would be in-line with the bias-variance tradeoff, which explains how, as the error the network makes decreases (bias), the greater the variation in the output will be for comparatively smaller changes to the input. This was however not explored further.

7.2.7 Handcrafted scenarios

In general, the JIPDA performs significantly better in the handcrafted scenarios. In all the handcrafted scenarios, the MT3v3 produces more false tracks than when using the data generator. The exact reason for this is uncertain, but it is hypothesized that there are some numerical differences between the data generator and the detector simulator that is causing the MT3v3 to make false estimates. It may also be due to the fact that the time delta used in the detector simulator is not the same as the data generator. Regardless of the cause of this difference, it highlights the importance of testing a neural network tracking solution using other data pipelines when determining its performance. Since only $n_{mc} = 1$ was used, the confidence intervals are large. These could be reduced by re-generating measurements from the same ground-truth targets $n_{mc} > 1$ times and calculating ensemble averages as before. Doing so may alter the conclusion drawn from the data presented in tables 6.17 and 6.18.

Crossing targets

In all the first crossing scenario where two targets cross paths, the JIPDA correctly tracks both targets throughout the entire sequence, although it establishes some false tracks throughout the sensor field of view, some of which intersect with the true targets' paths. The MT3v3 performs worse, as the targets are not correctly identified in the few meters before they cross. The crossing itself also impacts the localization error negatively for some timesteps after. Although not shown in the plots, the MT3v3 produces estimates that align well with both tracks throughout the entire scenario, but at certain points their existence probabilities drop below the fixed threshold. It thus appears that iteratively refining the existence probability does not necessarily help in scenarios like these.

The second crossing scenario is more complicated as two targets cross a third in parallel. However, the JIPDA handles this just as well as the simple crossing. The MT3v3 misses more targets in this scenario, and the target moving vertically is not correctly tracked until after both targets pass. One of the crossing targets is also not detected in the timesteps before and after crossing. It is possible that the extra target that crosses is in close enough proximity to the rest that the MT3v3 sees these measurements as clutter, and thus stops producing track estimates. It is also possible that employing an autoregressive decoder would alleviate these problems, as the previous estimates would be taken into consideration when producing the estimates at the current timestep.

Stationary targets are added to the first crossing scenario to create a third. Doing so makes the JIPDA produce even more false tracks, but all targets are still correctly identified and tracked. The MT3v3 estimates the stationary target positions more accurately than the JIPDA, but this comes at the cost of much worse performance on the moving targets. In total this amounts to an even higher missed target rate in this scenario as compared to the first. It is uncertain why adding stationary targets so drastically changes the estimates for the crossing targets, but it may be an indicator that the MT3v3 expects that the existence of a single target in a region decreases the likelihood that other measurements around it are also from other targets. This would also explain the track fragmentation that occurs around the region where two targets cross paths.

Parallel targets

When two target approach one another, the JIPDA correctly tracks both targets throughout the entire sequence. However, it also produces a third false track that evolves alongside one of the targets. This false track may be the result of a combination of clutter measurements aligning with true measurements, and that the measurement noise makes a single track appear as two separate ones. Where the JIPDA creates a third false track, the MT3v3 instead misses the true target, while the second true target is tracked well. There is no significant difference in the total GOSPA score for the two trackers in this scenario. However, the MT3v3 score is actually mostly composed of false tracks, and not missed targets.

The JIPDA does not produce the extra third false track for the scenario where the two targets move together, which may indicate that this was an one-off error caused by the specific combinations of true and false measurements. However, taking into consideration the whole sensor view, the JIPDA produces the same score in terms of false tracks for this scenario. The MT3v3 also manages to track both targets well without the fragmentation seen in the parallel approach scenario. In any case, the two trackers do not produce significantly different total GOSPA scores here either.

As for the three maneuvering targets, the JIPDA produces multiple tracks in the regions where all three targets are close to one another. It also does not manage to correctly track all turns that two of the targets make. However, it performs significantly better than the MT3v3, which struggles to establish all three tracks, and only partially tracks the targets once established. However, it appears to better adapt to the hard turns that they make. This indicates that the MT3v3 either has not learned a concept of ”motion model” and mostly follows the measurements, or that it has learned that targets can make sharp turns, and combines this knowledge with the measurements. Again, while not shown in the figures, the MT3v3 produces output components that align well with all targets throughout the entire scenario, but many of these components have very low existence probabilities.

Summary of the handcrafted scenarios

The goal of the handcrafted scenarios was both to test how the MT3v3 handles specific scenarios that can appear in the real world, but that have not been seen in training. There are also additional challenges with using data that originates from an entirely different simulation scheme. The JIPDA either performs as well or better than the MT3v3 in all scenarios, and the MT3v3 struggles to create output estimates that have a high existence probability for a large majority of timesteps in most scenarios. This indicates that more work need to be put into how the existence probabilities are generated for the MT3v3 to be more robust in such conditions.

7.2.8 Validity of the benchmark procedure

The CKF has not been considered much when explaining or discussing the results, since it is "cheating" the tracking task. Instead, it can be used to reveal implementation or testing errors, if the false components of its GOSPA score is not zero or any tracker performs better than it. It is also used as an indicator of how optimal the other trackers - the close to the CKF the other two trackers are, the closer they are to performing as good as they possibly could.

GOSPA and its decomposition was the only performance metric that was considered throughout the benchmarking process. This was done as an attempt to standardize how any given scenario was tested, but this has certain disadvantages. For example, while the missed target component offers some insight into how well the targets are tracked, it does not offer any insight into how fragmented tracks are. As discussed earlier, the MT3v3 has no notion of what a "track" is, and so further development would have to be made in order to facilitate the use of other performance metrics. It is thus important to keep in mind that this benchmarking procedure cannot be considered exhaustive, as it provides partial insight into how the trackers perform. A full discussion on how to exhaustively test a tracker is well outside the scope of this thesis, but is also an important consideration for future work.

A perfect comparison between the MT3v2 and MT3v3 cannot be made either, since they operate on fundamentally different information. As discussed in section 3.3.2, the doppler component especially for the MT3v2 can be used as a strong discriminant between true and false measurements, as well as providing some information about the cartesian velocity of the targets. The MT3v3 also sees more varied data, and is designed to operate on a wider set of scenarios, as opposed to the task-specific MT3v2. The MT3v3 will also use the entire measurement sequence when producing output estimates. For these reasons, the MT3v2 was left out of many of the benchmarks, especially when testing generalizability and on hand-crafted scenarios.

As previously mentioned, the IMM-JIPDA (which as a reminder is simply referred to as the JIPDA throughout the text) was selected as the "Bayesian tracker to beat", as opposed to the more sophisticated PMBM and δ-GLMB trackers that were used to benchmark the MT3v2. This was a conscious decision made early on, as there were doubts around the thoroughness of the reported MT3v2 performance, and how the MT3v3 would perform without doppler information and the other architectural changes. The VIMMJIPDA implementation that was used can be configured to not use visibility (V) and multiple models (IMM), which would reduce it to a normal JIPDA. However, since the MT3v3 was designed to operate on a range of different scenarios, which includes targets with different motion models, it was decided to keep the IMM component.

When presenting results, it is explicitly stated whether or not the JIPDA was tuned beforehand, as this is an important consideration to make, and will be further discussed in section 7.5. As described in section 7.2.7, the JIPDA makes significantly more false tracks than the MT3v3 across all tests, while missing fewer true targets. By increasing the track confirmation threshold, a more optimal balance between false and missed targets could potentially have been found for each scenario. If this threshold is set too high however, the very fast track initialization times seen in the JIPDA may be impacted negatively. The opposite is true for the MT3v3, as it misses significantly more targets than the JIPDA while simultaneously producing much fewer false tracks. The poor initialization time for the MT3v3 could thus also potentially be improved by tuning the existence threshold for specific tasks. As explained earlier, whenever scenarios were generated using parameters within the ranges that were used for training the MT3v3, the JIPDA was given the exact tuning parameters used. It is argued that this is a fair middle-ground for both trackers, since such scenarios are to be considered "known" to the MT3v3, and providing true scenario parameters is the only way to make a specific scenario "known" to the JIPDA as well. On the other hand, whenever parameters outside of the training ranges were used, it is argued that the JIPDA would be given an unfair advantage by being provided with the true scenario parameters. Tuning each tracker for every test scenario would also take up considerable time, and in a real-world scenario it is not reasonable to expect the tracker to be continuously tuned.

7.3 Training scenarios

As shown in figures 6.2 and 6.1, the old and new data generator respectively produce very different scenarios. By enriching the target birth process, the MT3v3 can train on a more varied dataset that covers a greater part of the sensor FOV. The method implemented for the MT3v3 specifically also scales with the FOV, unlike the MT3v2 which leads to a smaller portion of the sensor FOV being covered as the total surveilance area increases. Furthermore, in open-sea scenarios or scenarios with few occlusions, the majority of targets will enter into the sensor view at the edge of its field-of-view. As such, it is argued that the target birth model developed for the MT3v3 is an improvement over its last generation counterpart.

The standard way to organize data is to split it into training-testing or training-validation-testing. These names are very descriptive for what each data set is used for. Training data is used for training the network, while the test data is left out of the training procedure in order to assess the generalizability of the network, i.e. how well the network performs on unseen data. Validation data can be used in order to assess how the network is doing while training and give an estimate of the expected test performance. This is especially important for a fixed dataset that may be used multiple times throughout the training loop. However, all versions of the MT3 trackers use a data generator that can produce arbitrarily large datasets. As such, training and test data is simply generated on-demand.

Batch gradient descent is not possible when data is generated on-demand. This would instead require all data to be generated and stored ahead of time, but is infeasible in practice, which can be illustrated with some napkin math. Consider a data generator where two targets produce measurements and two clutter measurements are present at every timestep on average, and each scenario is on average 100 timesteps long. Storing these simulated measurements and their associated ids using 32 bit floating points would result in 8GB of total data. While this is not necessarily a large amount for the current generations of compute units. However, using the MT3v3 average inference time in the training loop of around 250ms, a single training step would take almost 16 days. A more reasonable approach is taken, utilizing the parallelizability of the transformer, by generating the maximum number of scenarios that can be kept in memory at once, treating it as a single batch and updating the learnable parameters after processing this batch. This approach of generating a batch of data on-demand and updating the network parameters after processing said batch indirectly constitutes mini-batch gradient descent.

7.3.1 Sensor placement

It is also worth noting that while the sensor can be placed anywhere in the cartesian plane, it will be fixed throughout all scenarios. A moving sensor would increase the variation in each scenario significantly, and would add an additional challenge that the MT3v3 would have to solve. Introducing a moving sensor platform would either require extra pre-processing to use the MT3v3 in its current state, but perhaps more interesting would be to adapt the input of the MT3v3 to also contain the pose of the sensor at the time of each measurement.

7.3.2 Scenario parameters

The bounds for the parameter distributions used for training scenarios as presented in table 5.2 were chosen through a subjective evaluation, where the goal was to achieve some variety in the training data while not requiring an infeasible long training period. However, such an approach is in a sense very arbitrary, and the effect of introducing such a variation in the training data was not tested. It would be possible to train two networks, one using the aforementioned distributions, and one with fixed parameters, and benchmark them against one another. This was however not done, as the MT3v3 was only trained using the full $1.9 \cdot 10^6$ gradient steps once. It is also unknown whether this way of training the MT3v3 is the most efficient. For example, it could be more computationally efficient to generate fewer scenarios, but reusing them multiple times.

7.4 Generalizability

Good generalizability is important for several reasons – Robustness, real-world application and efficiency to name a few. The importance of each of these will of course depend on the purpose of the tracker. For example, if the purpose of spending resources to set up and train the MT3v3 is to get a tracker that is to be used in the real-world as part of an autonomy pipeline, it is very important that the network performs well on real-world data without needing to see all potential input data during training. Real-world data will not necessarily follow the distributions defined by the MT3v3 data generator, and can be affected by effects that are not explicitly modelled by the data generator. One of the main motivating factors behind the design of the MT3v3 was, as previously mentioned, a desire for increased robustness to varying scenarios. In this regard, good generalizability implies good robustness.

Generalizability was tested by simply changing some of the parameters of the data generator. To limit the scope of the experiments, the parameters $\boldsymbol{\lambda}_c$, p_d and $\boldsymbol{\Sigma}_r$ were selected as the main parameters to change. In addition, the MT3v3 was tested with very long input sequences in order to assess the effect of input sequences much longer than those seen during training.

7.4.1 Clutter intensity

It was shown in (Pinto et al. 2022) that the MT3v2 outperformed the Bayesian state-of-the-art trackers used for benchmarking by a large margin in scenarios with a lot of clutter. However, since the doppler component used in the MT3v2 could be used to discriminate between a large majority of true and false measurements, it was not given that the MT3v3 would perform as well in scenarios with high clutter intensity. However, as shown in table 6.10, the MT3v3 does not miss any more targets in scenarios with a clutter intensity that is 5 times higher than the maximum intensity seen in training. The MT3v3 does end up making false tracks more often in these high-clutter scenarios, but nowhere near as many as the JIPDA. When increasing the clutter intensity past a certain point, in this case by 20 times the maximum intensity seen in training, the MT3v3 ends up missing all targets, while the JIPDA almost exclusively produces false tracks. It is unreasonable to expect great performance in such a case, since the ratio between false and true measurements is extremely high. While the MT3v2 managed to produce some true tracks while missing fewer targets through similar clutter intensities, the difference in distributions of doppler between true and false measurements are again a likely cause for this.

7.4.2 Probability of detection

As shown in table 6.9, the MT3v3 performs very poorly when provided with a scenario generated using a probability of detection that is much lower than that used for training. The MT3v3 ends up missing almost all tracks when $p_d = 0.4$, even while the rest of the data generator parameters are set to their median values, resulting in a total GOSPA that is much higher than the IMM-JIPDA. This is a result that is to be expected, given the relatively long track initialization time of the MT3v3. It is likely that the network does not attend to a set of true measurements well enough when there are unexpectedly large gaps between measurements, leading to either no outputs that land near a true target, or that the components that do have an existence probability that is too low. Testing a detection probability higher than the maximum value seen in training, that is > 0.95, was not done, since this case will only provide more information to the network, and thus it is highly unlikely that it produces worse results than with any lower p_d.

The probability of detection for any particular target for a given environment in the real world can vary quite heavily (Bocquet 2011), and since the MT3v3 cannot be tuned on-the-go, the poor performance when p_d is low may be a limiting factor. This is especially the case if the MT3v3 is to be used in any safety-critical operations.

7.4.3 Measurement noise

It can be argued that good generalizability in the measurement noise is not as important as other factors, since measurement noise can somewhat be known a-priori using information provided by the sensor manufacturer. However, it can still be used to get some insight into how output estimates are affected by the inputs. As with the previous experiments, measurement noises below the smallest values used in training were not tested. Since it is already shown that the MT3v3 is overconfident in the measurements when generating target estimates, lower measurement noise would simply lead to a lower localization error.

As shown in table 6.11, increasing the measurement noise by a factor of 10 did not significantly impact performance. The localization error is somewhat larger, which is reasonable given that the true measurements in these scenarios are of lower quality. When increasing the measurement noise by a factor of 100, both the number of false and missed targets increase significantly. This may indicate that the MT3v3 does not attend to some true measurements when they end up too far apart from one another, as a result of the largely increased measurement noise.

7.4.4 Input sequence length

As presented in table 6.16, increasing the sequence length to 1000 timesteps does not significantly affect performance. While the total GOSPA is somewhat lower, it is within the confidence interval of the scenarios with 100 timesteps. The total GOSPA may also be lower simply because the higher scores that are seen in the beginning of a scenario will have less impact on average GOSPA as the sequence length increases.

7.4.5 Improvements over the MT3v2

Since the MT3v2 was trained for 4 different tasks, while the MT3v3 is trained for a single, more varied "task", a direct comparison cannot be made for performance between the two. However, the very poor cross-model test results that were shown in (Strøm 2022) indicate that the MT3v2 will only perform well on scenarios that are seen during training. Since the MT3v3 is given more varied scenarios in training and performs well across the entire range of data generator parameters, while also providing reasonable performance for more difficult scenarios, it appears that enriching the training data yields a better performing tracker. This is not unexpected, as anything the network can learn is entirely dependent on the input data it is provided.

It is important to point out the difference between generalizability in the context of unseen data from the same distribution that was used to generate training data, and generalizability to data that comes from a different distribution. All MT3 architectures perform well on unseen data from the same distribution that was used to generate the training data. The task-specific MT3v2s outperform the MT3v3 in the specific tasks they were trained for, which is reasonable given that the MT3v2 was designed to be trained against a fixed task, and can use doppler information from the input measurements. However, there is no guarantee that data from, lets say, a real world scenario could be modelled adequately using the very same parameters that were used for one of these specific tasks. In this case, it is likely that the MT3v3 would perform better, simply because there is a greater probability that the data distribution from such a scenario could be adequately modelled using a combination of the parameters that were sampled during training. This performance would likely be a result of the MT3v3 having "seen" more data distributions during training. As demonstrated when setting the probability of detection far below the values used for training, or the clutter intensity very high above those used in training, the MT3v3 may also be flexible enough to handle scenarios that cannot be fully modelled by the training data.

7.5 Why use the MT3v3

Benchmarking the MT3v3 against JIPDA has provided valuable insights into the advantages of using MT3v3 for target tracking, while the additional development efforts that went into the MT3v3 implementation have also introduced additional benefits to using the MT3v3. It is important to consider this, as

> *A Transformer-based tracker must at the very least perform better than a JIPDA to be considered for future research.* – Edmund F. Brekke

Put in other words, if the MT3v3 does not offer any benefits over a "simple" multi-target tracker such as a JIPDA, then it is difficult to justify spending computational resources or additional research time on Transformers for point-object tracking.

7.5.1 Robustness and flexibility

The MT3v3 provides much lower position errors than the MT3v2 in the scenarios that were tested using both trackers, which is a clear cut improvement over the previous generation of MT3. Since the MT3v3 also provides good velocity estimates without doppler measurements, it is possible to use the MT3v3 in autonomy pipelines which utilize sensors that do not provide doppler, in contrast to the MT3v2 which in this case would perform very poorly. It can also support an arbitrary number of input measurements, which minimizes the pre-processing required to integrate the MT3v3 with existing systems.

It was also shown that the MT3v3 handles an increase in both clutter intensity and measurement noise significantly better than the JIPDA. It could thus be advantageous to use the MT3v3 in situations that cannot be reasonably be modelled as a constant, which would avoid having to estimate λ_c online. Since the GOSPA score of the MT3v3 is significantly better than the JIPDA after about 15 measurements, it could be reasonable to employ the MT3v3 to track long data sequences. This use case is further substantiated by the fact that the MT3v3 can utilize the information in every available measurement with a negligible change in inference time. While not tested, this implies that there is potential for performance gain over the JIPDA in the case that the first-order Markov assumption does not hold. Lastly, the MT3v3 does not explicitly perform any linearization in the measurement space to state space transformation, and so it is hypothesized that performance loss due to strong non-linearities can be mitigated.

7.5.2 Ease of use

While this relates more to the implementation and practical sides of the MT3v3, they are still important to consider. The MT3v3 was implemented with ease of use in mind. Support for measurements in common data structures such as Python lists and Numpy arrays was added through a utility library, and get-started examples were written to minimize the friction of getting started with training and inferencing with the MT3v3.

The MT3v3 is also fairly lightweight. The entire pretrained network is just over 200MB and achieves 50 inference steps per second on an NVIDIA RTX A2000 mobile GPU. There is also no extra memory overhead caused by the MT3v3 itself when the number of input elements increases, and the inference time does not change significantly with an increasing number of inputs.

Almost every tunable parameter is set before training, and as such, the task of tweaking and tuning can be performed almost entirely offline. This can be advantageous as it reduces deployment time given that the MT3v3 ”just works”, and is an advantage that most end-to-end neural solutions share. Since the data generator has been enriched in the MT3v3, it has seen more scenarios and can utilize this information to generalize somewhat well outside of known scenarios. The improved data generator is also flexible in that it is backwards compatible with both the MT3v1 and MT3v2, as both cartesian and polar measurement models are available. Doppler can also be turned on and off.

Lastly, there is very little modelling work required for the MT3v3, as it is assumed that the network will learn this automatically. This is especially advantageous for effects that are not necessarily straight-forward to model and/or incorporate in a Bayesian tracker. For example, while the bimodal birth model in the data generator can be explained with a few equations, more work would have to go into developing and implementing the Bayesian tracker, while the MT3v3 will instead likely learn these effects through the training process without any explicit instructions or clever ways of incorporation of this information.

7.6 Why *not* to use the MT3v3

While there are certain advantages to using the MT3v3, there are also significant downsides that must be taken into consideration.

7.6.1 Offline training

A downside that is immediately clear is that the MT3v3 has to be trained offline, and is one that all neural networks share. While a pretrained MT3v3 network is provided, any changes either requires additional training, or a complete retrain of the network. This process is very computationally expensive, and requires a GPU with a large quantity (over 20GB) of memory. Some rough estimates using the IDUN hardware setup as an example, training the MT3v3 for $3.5 \cdot 10^6$ takes around 200 hours (wall-clock) and consumes about 100 kWh. While this is insignificant when comparing to massive-scale models like the GPT4, it is still a cost that must be considered, and if this cost is not deemed to be worthwhile to spend, the MT3v3 cannot be used aside from the the pretrained network that is provided. The need for retraining when changing certain network configuration parameters thus also hinders flexibility.

7.6.2 Variation in performance

The good average performance that the MT3v3 achieves is plagued by high variation in performance, as discussed previously. While no upper bound for GOSPA score has been determined for the MT3v3, the poor performing scenarios show that this will be much higher than the JIPDA. At certain points, the MT3v3 will miss an entire track where it is reasonable to assume that it *should* have been able to. Given how poor the MT3v3 has been shown to perform, it cannot be trusted for any safety-critical application. This includes most autonomy pipelines that are deployed in the real-world outside of controlled experiments. As such, the MT3v3 must still be considered a research object.

7.6.3 Black box architecture

The variation in performance is hard to reason with due to how the MT3v3 by nature is a black box. Attention maps can offer some insight by indicating which inputs the network considers significant when generating outputs. However, these maps fail to provide explicit information about how the measurements are utilized to generate outputs. Furthermore, projecting measurements into feature space adds a lot of complexity which makes interpretation very challenging, or even impossible, as the features are entirely learned throughout the training process. Even if the features could be fully explained, which is highly unlikely, one would still need to describe how these features are combined with the selected anchor measurements in order to produce the final outputs. This would include an explanation of how the selection mechanism determines what measurements are ”best”.

Lastly, while the outputs contain uncertainty estimates, there is no way to guarantee that this reflects some inherent uncertainty that the network actually has when generating measurements. This adds another layer of obscurity when interpreting measurements, and can be detrimental to how the estimates are actually used in an autonomy pipeline that relies on this. As the NEES plots show, the state uncertainties do not correspond well to the estimate error of the MT3v3, and it is likely that this is caused by the disconnect between the uncertainty estimates and the inherent uncertainty in the network, if such a thing can be established. This lack of correspondence between state error and uncertainties that cannot be tuned away may deter users from the MT3v3.

As the MT3v3 is an end-to-end network, no information about the targets or environment is given a-priori. While this comes with the advantage of ease of use, it also discards a lot of potentially useful information that could be used to generate better estimates or learn quicker. This total lack of probabilistic modelling also contributes to the black-box effect.

7.7 The bearings-only MT3v3

The bearings-only tracking problem was tested with the MT3v3 due to how simple it was to set up. No architectural changes were considered and no existing literature was consulted. A single network was trained in two batches, which was shown in figures 6.21 and 6.22. The network appears to quickly learn to place all hypotheses in a straight line between the sensor and the estimated bearing of the target, which is reasonable, given that there is no information about range directly available in the measurements. However, it does appear that the components which are further away from the target receive less existence mass on average, as compared to the components that are closer to the target. In any case, as evident in the certainty distribution plot, the network eventually stagnates and is unable to be confident in what existence probability each output component should receive. It is not expected for this to happen without a major change to the network architecture, again due to the very limited information at the input.

8 Conclusion

With the rise of increasingly more intricate applications of autonomy, so does the requirement for good situational awareness systems. A tracker tracking system that is capable of tracking multiple targets over time is an integral part of many such systems. The current research field of multi-target tracking is roughly split in two, the purely Bayesian domain which utilizes explicit probabilistic modelling of the multi-target problem, and the neural network domain where the modelling effort is replaced by clever neural network architecture design and offline training. Transformer trackers is a subsection of the neural tracking domain and is in large dominated by camera tracking. There is a gap in the neural tracker research, as comparatively little effort has gone into Transformer trackers that use point-object as their input. This thesis serves as a contribution to the point-object Transformer tracking literature, by expanding upon the limited existing work in this field.

8.1 Addressing the title assertion

As per the title, the goal of this thesis was to contribute to the field of Transformer-based point-object multi-target trackers by providing a new architecture that is more flexible and robust than the existing alternatives. The main contribution of this thesis is thus new generation of MT3 tracker, namely the Multi-Target Tracking Transformer v3 (MT3v3). This is an end-to-end neural tracker for point-objects, and builds upon the previous MT3v1 and MT3v2 architectures of (Pinto et al. 2021a) and (Pinto et al. 2022). The MT3v3 produces estimates of targets' cartesian positions and velocity from polar measurements without requiring doppler information, and can support an arbitrary number of measurements at the input. By only utilizing range and bearing measurements, while still providing good velocity estimates at the output, the MT3v3 is flexible to be used in autonomy pipelines that utilize sensors such as low-cost radars or lidars that do not provide doppler information. Robustness was improved by altering the training scheme for the MT3v3 in numerous ways, and the generalizability of the fully trained architecture was tested. Compared to the MT3v2, the MT3v3 performs as good or better in terms of GOSPA on a set of generated scenarios. Using an IMM-JIPDA and a CEKF as the upper and lower baseline for error, the MT3v3 is also shown to produce good results in terms of localization and identifying targets in certain scenarios of varying complexity. However, the number of timesteps required to initialize tracks is comparatively high. The worst-case performance of the MT3v3 is also very poor, and it is significantly outperformed by the IMM-JIPDA in a set of handcrafted MTT scenarios. This variability in performance together with its black-box nature means that the MT3v3 should still be considered a research object.

8.2 Future work

There are many reasonable steps to take forward from the MT3v3, but based on the work done in this thesis, the most prominent future developments involve

1. Designing an autoregressive decoder for an MT3 architecture
2. Tackling other tracking problems, such as bearings-only tracking.
3. Using the family of MT3 architectures to design a hybrid neural-Bayesian tracker.

Firstly, it is hypothesized that an autoregressive decoder would allow the architecture to more easily learn to weigh an associated measurement with some learned relationship between the measurement history and the motion of the targets present in said measurements. This could further lead to a faster track initialization times and a further reduction in localization error, especially for increasing measurement noise and low probability of detection. Secondly, Transformers exceed in modelling dependencies that extend far into the past of the input sequence, and requires no explicit modelling of the task to be solved. This could be exploited in solving other problems in the tracking domain, such as bearings-only tracking. The MT3v3 could easily serve as a starting point for experimenting with this, by simply removing the range part of the input measurements. Lastly, there may exist a best-of-both-worlds architecture that uses both neural and Bayesian techniques. An example of this could be that modelling the motion of maneuvering targets is a well-known problem, and multiple different such models are used in the IMM scheme. However, measurement-to-track association gets increasingly complex as the number of measurements in the input sequence increases, and simplifications have to made in any real-time tracker that utilizes data association. In contrast, Transformer architectures can be designed to produce outputs at about the same speed regardless of the number of input elements. Utilizing a Transformer as a pre-processor for a Bayesian estimator could thus provide a good alternative to the end-to-end style that the MT3 trackers are designed in.

Bibliography

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

Yaakov Bar-Shalom and Edison Tse. Tracking in a cluttered environment with probabilistic data association. *Automatica*, 11(5):451–460, 1975.

Christopher M Bishop and Nasser M Nasrabadi. *Pattern recognition and machine learning*, volume 4. Springer, 2006.

S.S. Blackman. Multiple hypothesis tracking for multiple target tracking. *IEEE Aerospace and Electronic Systems Magazine*, 19(1):5–18, 2004. doi: 10.1109/MAES.2004.1263228.

Stephen Bocquet. Calculation of radar probability of detection in k-distributed sea clutter and noise. Technical report, DEFENCE SCIENCE AND TECHNOLOGY ORGANISATION (AUSTRALIA) JOINT OPERATIONS ..., 2011.

Edmund Brekke. Fundamentals of sensor fusion: Target tracking, navigation and slam. *Teaching material for TTK4250 - Sensor Fusion, NTNU*, 2020.

Edmund Førland Brekke, Audun Gullikstad Hem, and Lars-Christian Ness Tokle. Multitarget tracking with multiple models and visibility: Derivation and verification on maritime radar data. *IEEE Journal of Oceanic Engineering*, 46(4): 1272–1287, 2021.

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

Subhash Challa, Mark R Morelande, Darko Mušicki, and Robin J Evans. *Fundamentals of object tracking*. Cambridge University Press, 2011.

David Crouse. Basic tracking using nonlinear 3d monostatic and bistatic measurements. *IEEE Aerospace and Electronic Systems Magazine*, 29(8):4–53, 2014.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016. doi: 10.1109/CVPR.2016.90.

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

Zhiheng Huang, Davis Liang, Peng Xu, and Bing Xiang. Improve transformer models with better relative position embeddings. *arXiv preprint arXiv:2009.13658*, 2020.

Peiyuan Jiang, Daji Ergu, Fangyao Liu, Ying Cai, and Bo Ma. A review of yolo algorithm developments. *Procedia Computer Science*, 199:1066–1073, 2022.

Seyed Mehran Kazemi, Rishab Goel, Sepehr Eghbali, Janahan Ramanan, Jaspreet Sahota, Sanjay Thakur, Stella Wu, Cathal Smyth, Pascal Poupart, and Marcus Brubaker. Time2vec: Learning a vector representation of time. *arXiv preprint arXiv:1907.05321*, 2019.

Prannay Khosla, Piotr Teterwak, Chen Wang, Aaron Sarna, Yonglong Tian, Phillip Isola, Aaron Maschinot, Ce Liu, and Dilip Krishnan. Supervised contrastive learning. *Advances in neural information processing systems*, 33:18661–18673, 2020.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

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

Greta M Ljung and George EP Box. On a measure of lack of fit in time series models. *Biometrika*, 65(2):297–303, 1978.

Lu Lu, Yeonjong Shin, Yanhui Su, and George Em Karniadakis. Dying relu and initialization: Theory and numerical examples. *arXiv preprint arXiv:1903.06733*, 2019.

Minh-Thang Luong, Hieu Pham, and Christopher D Manning. Effective approaches to attention-based neural machine translation. *arXiv preprint arXiv:1508.04025*, 2015.

Tim Meinhardt, Alexander Kirillov, Laura Leal-Taixe, and Christoph Feichtenhofer. Trackformer: Multi-object tracking with transformers. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 8844–8854, 2022.

D. Musicki and R. Evans. Joint integrated probabilistic data association: Jipda. *IEEE Transactions on Aerospace and Electronic Systems*, 40(3):1093–1099, 2004. doi: 10.1109/TAES.2004.1337482.

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

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

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

Abu Sajana Rahmathullah, Ángel F. García-Fernández, and Lennart Svensson. Generalized optimal sub-pattern assignment metric. In *2017 20th International Conference on Information Fusion (Fusion)*, pages 1–8, 2017. doi: 10.23919/ICIF.2017.8009645.

Christian P Robert et al. *The Bayesian choice: from decision-theoretic foundations to computational implementation*, volume 2. Springer, 2007.

Magnus Själander, Magnus Jahre, Gunnar Tufte, and Nico Reissmann. EPIC: An Energy-Efficient, High-Performance GPGPU Computing Research Infrastructure. *arXiv:1912.05848 [cs]*, December 2019.

Roy L Streit and Roy L Streit. The poisson point process. *Poisson Point Processes: Imaging, Tracking, and Sensing*, pages 11–55, 2010.

Christopher Strøm. Can deep learning *really* be applied to model-based multi-object tracking? Specialization project, Department of Engineering Cybernetics, Norwegian University of Science and Technology, 2022.

Mengchu Tian, Yuming Bo, Zhimin Chen, Panlong Wu, and Cong Yue. Multi-target tracking method based on improved firefly algorithm optimized particle filter. *Neurocomputing*, 359:438–448, 2019.

Yonglong Tian, Chen Sun, Ben Poole, Dilip Krishnan, Cordelia Schmid, and Phillip Isola. What makes for good views for contrastive learning? *Advances in Neural Information Processing Systems*, 33:6827–6839, 2020.

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

Xuedong Wang, Tiancheng Li, Shudong Sun, and Juan M Corchado. A survey of recent advances in particle filters and remaining challenges for multitarget tracking. *Sensors*, 17(12):2707, 2017.

Chi Zhang, Guosheng Lin, Fayao Liu, Rui Yao, and Chunhua Shen. Canet: Class-agnostic segmentation networks with iterative refinement and attentive few-shot learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5217–5226, 2019.

Jinghao Zhou, Chen Wei, Huiyu Wang, Wei Shen, Cihang Xie, Alan Yuille, and Tao Kong. ibot: Image bert pre-training with online tokenizer. *arXiv preprint arXiv:2111.07832*, 2021.

Xizhou Zhu, Weijie Su, Lewei Lu, Bin Li, Xiaogang Wang, and Jifeng Dai. Deformable detr: Deformable transformers for end-to-end object detection. *arXiv preprint arXiv:2010.04159*, 2020.