CHALMERS
UNIVERSITY OF TECHNOLOGY

Deep Learning for Model-Based Multiobject Tracking

Citation for the original published paper (version of record):
Pinto, J., Hess, G., Ljungbergh, W. et al (2023). Deep Learning for Model-Based Multiobject Tracking. IEEE Transactions on Aerospace and Electronic Systems, 59(6): 7363-7379. http://dx.doi.org/10.1109/TAES.2023.3289164

(article starts on next page)

Deep Learning for Model-Based Multi-Object Tracking

JULIANO PINTO[1]
GEORG HESS[1]
WILLIAM LJUNGBERGH[2]
YUXUAN XIA[1], Member, IEEE
HENK WYMEERSCH[1], Senior Member, IEEE
LENNART SVENSSON[1], Senior Member, IEEE
[1] Department of Electrical Engineering, Chalmers University of Technology, 41296 Gothenburg, Sweden.
[2] Computer Vision Laboratory, Linköping University, 581 83 Linköping, Sweden.

***Abstract*— Multi-object tracking (MOT) is the problem of tracking the state of an unknown and time-varying number of objects using noisy measurements, with important applications such as autonomous driving, tracking animal behavior, defense systems, and others. The MOT task can be divided into two settings, model-based or model-free, depending on whether accurate and tractable models of the environment are available. Model-based MOT has Bayes-optimal closed-form solutions which can achieve state-of-the-art (SOTA) performance. However, these methods require approximations in challenging scenarios to remain tractable, which impairs their performance. Deep learning (DL) methods offer a promising alternative, but existing DL models are almost exclusively designed for a model-free setting and are not easily translated to the model-based setting. This paper proposes a DL-based tracker specifically tailored to the model-based MOT setting and provides a thorough comparison to SOTA alternatives. We show that our DL-based tracker is able to match performance to the benchmarks in simple tracking tasks while outperforming the alternatives as the tasks become more challenging. These findings provide strong evidence of the applicability of DL also to the model-based setting, which we hope will foster further research in this direction.**

***Index Terms*— Multi-object tracking, Deep Learning, Transformers, Random Finite Sets, Uncertainty Prediction.**

I. INTRODUCTION

Multi-object tracking (MOT) is the problem concerned with recursively estimating the state of an unknown and time-varying number of objects, based on a sequence of noisy sensor measurements. The objects of interest can enter and leave the field-of-view (FOV) at any time, they do not always generate measurements at every time step, and there can be false measurements originating from sensor noise and/or clutter. Methods capable of tracking objects under these conditions are required for a diverse set of important applications, including tracking animal behavior [1], pedestrian tracking [2], autonomous driving [3], oceanography [4], military applications [5], and many others. Methods to solve the MOT problem depend on whether they operate in the *model-based* or *model-free* setting. In the model-based setting, accurate and tractable models of the measurement likelihood, as well as the object dynamics, are available to the MOT designer. In contrast, under the model-free setting, such models are unavailable or intractable, e.g., due to high-dimensional measurements such as image or video data [2], [6].

In the model-based setting, filters based on the random finite set (RFS) formalism using multi-object conjugate priors [7], [8] can provide closed-form Bayes-optimal solutions to MOT and achieve state-of-the-art performance [9]. Yet, due to the unknown correspondence between objects and measurements, also known as the data association problem, the number of possible associations increases super-exponentially over time. Consequently, these methods must resort to approximations such as pruning/merging for remaining computationally tractable [8], [10], which inevitably leads to a deterioration of tracking performance, especially in challenging scenarios. Moreover, when the measurement and/or motion models are nonlinear, one must rely on Gaussian approximations or sequential Monte Carlo methods to handle the nonlinearity [11], which may further impact the tracking performance.

In contrast, DL methods can directly learn a mapping from sequences of measurements to state estimates in a data-driven fashion, thus sidestepping the complexity of dealing with data associations explicitly and therefore the need to resort to heuristics for maintaining computational tractability. However, almost all of the DL approaches have been designed for model-free MOT, especially video-based MOT, and are not straightforward to translate to the model-free setting. To the best of our knowledge, no prior work (except our preliminary analysis [12]) has developed a DL solution specifically tailored to the model-based setting, and compared their performance to traditional SOTA RFS-based methods such as [7], [8], [13], [14].

In this paper, we propose a DL-based tracker specifically tailored for **model-based MOT**, and provide a thorough comparison to SOTA Bayesian filters in this setting. As illustrated in Fig. 1, we leverage the insight that the available multi-object models can be used for generating unlimited synthetic training data. This allows for the use of modern high-capacity DL architectures such as the Transformer [15], which we further improve by using insights specific to the model-based setting. We show that our DL-based tracker is able to achieve

The work of William Ljungbergh was done during his time as a master's student at Chalmers. This work was supported, in part, by a grant from the Chalmers AI Research Centre Consortium. Computational resources were provided by the Swedish National Infrastructure for Computing at C3SE, partially funded by the Swedish Research Council through grant agreement no. 2018-05973. *(Corresponding author: Juliano Pinto, juliano@chalmers.se).*

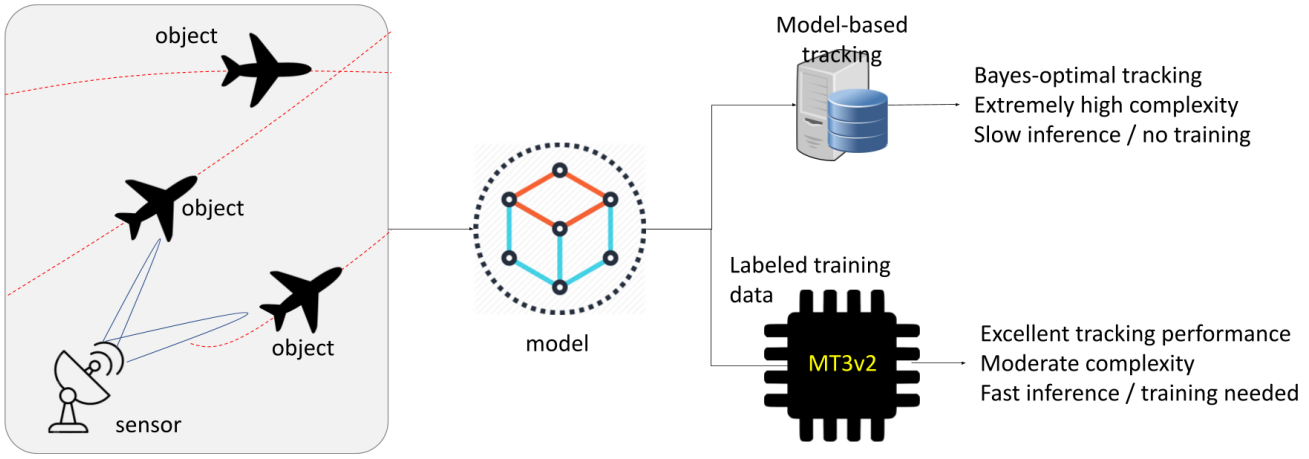

Fig. 1. Bayes-optimal trackers require accurate and tractable models of the environment for their correct functioning but result in solutions that are intractable for challenging tracking tasks. In our proposed solution, these mathematical models are instead used to generate unlimited training data for MT3v2, a DL-based tracker using the Transformer architecture.

comparable performance to the benchmarks in simple tasks while outperforming them in the presence of strong model nonlinearities or an intractable number of possible data association hypotheses. These findings provide strong evidence of the applicability of DL trackers also in the model-based setting. Our specific contributions are:

- A novel, high-performing DL-based tracker specifically tailored for the model-based MOT setting. The proposed architecture provides uncertainty estimates in addition to state estimates and uses multiple improvements compared to standard Transformers, including a selection mechanism for providing sample-specific object queries, a decoder that iteratively refines estimates, and a learned temporal encoding of the measurement sequences.
- Uncertainty-aware loss formulations suited for training general DL-based trackers in the model-based setting.
- An in-depth performance evaluation and comparison for SOTA Bayesian model-based MOT methods under a realistic radar measurement model with nonlinearities and finite FOV.

Notations

Throughout the paper we use the following notations: Scalars are denoted by lowercase or uppercase letters with no special typesetting (x), vectors by boldface lowercase letters ($\mathbf{x}$), matrices by boldface uppercase letters ($\mathbf{X}$), and sets by blackboard-bold uppercase letter ($\mathbb{X}$). Sequences are indicated by adding subscripts or superscripts denoting their ranges to the typesetting that matches their elements (e.g., $\mathbf{x}_{1:k}$ is a sequence of vectors, $\mathbb{X}_{1:j}$ of sets), and arrays by adding multiple such ranges (e.g., $\mathbf{x}_{1:k,1:n}$). The number of elements in a set $\mathbb{X}$ is denoted $|\mathbb{X}|$, and we further define $\mathbb{N}_a \doteq \{i \ : \ i \leq a \ , \ i \in \mathbb{N}\}$.

II. MULTITARGET MODEL AND PROBLEM FORMULATION

A. Measurement and Transition Model

For the analysis carried out in this paper, we use the standard multitarget transition and observation models for point objects [16, Chap. 5]. We denote the state vector of object i at time step t as $\mathbf{x}_i^t \in \mathbb{R}^{d_x}$, and the set of the states of all objects alive at time step t as $\mathbb{X}^t$. New objects appear according to a Poisson point process (PPP) parameterized with intensity function $\lambda_b(\mathbf{x})$, while object death is modeled as independent and identically distributed (i.i.d.) Markovian processes, with survival probability $p_s(\mathbf{x})$. The objects' motion models are also i.i.d. Markovian processes, where the single-object transition density is denoted as $f(\mathbf{x}^{t+1} \mid \mathbf{x}^t)$.

The single-object measurement likelihood is denoted $\mathbf{g}(\mathbf{z}^t \mid \mathbf{x}^t)$, $\mathbf{z}^t \in \mathbb{R}^{d_z}$, where the probability of detection in state $\mathbf{x}$ is $p_d(\mathbf{x})$ and each measurement is independent of all other objects and measurements conditioned on its corresponding target. Objects may generate at most one measurement per time step, and measurements originate from at most one object. Clutter measurements are modeled as a PPP with constant intensity λ_c over the field of view (FOV), and are independent of the existing objects and any other measurements. The set of all measurements generated at time step t (true measurements and clutter) is denoted $\mathbb{Z}^t$.

B. Problem formulation

In this investigation, we focus on the problem of multi-object estimation using a sequence of measurements of arbitrary length, i.e., estimating $\mathbb{X}^T$ given access to measurements from τ time steps in the past until the current time, i.e., $[\mathbb{Z}^{T-\tau}, \cdots, \mathbb{Z}^T]$. Although certain tracking applications require the estimation of the entire trajectories of the objects, many important cases do not. One such

example is perception in autonomous driving, where state estimates are the main requirement for optimal decisions, and trajectory estimation is only of marginal interest.

For applying a DL solution, we see this problem as a sequence-to-sequence mapping task, where a sequence of measurements $\mathbf{z}_{1:n}$ is to be mapped to a sequence $\mathbf{y}_{1:k}$. The sequence $\mathbf{z}_{1:n}$ is formed by first appending each measurement vector in the moving window $[\mathbb{Z}^{T-\tau}, \cdots, \mathbb{Z}^{T}]$ with its time of measurement, and then joining all measurements into a single sequence, in arbitrary order. Hence, $n = \sum_{t=T-\tau}^{T} |\mathbb{Z}^t|$. The sequence $\mathbf{y}_{1:k}$ specifies the predicted posterior density for $\mathbb{X}^T$ in the form of a Poisson multi-Bernoulli density[1] with k components. Each $\mathbf{y}_i \in \mathbb{R}^{d_y}$, $i \in \mathbb{N}_k$, contains the existence probability for that component and the parameters for describing its state density (e.g., mean and standard deviation).

III. BACKGROUND ON TRANSFORMERS

The DL method used in this paper is based on the Transformer architecture [15], which in recent years has shown great potential in complex sequence-to-sequence function approximation [17]–[19]. This section provides a background on this type of neural network when processing an input sequence $\mathbf{z}_{1:n}$ with $\mathbf{z}_i \in \mathbb{R}^{d_z}$, $i \in \mathbb{N}_n$, to an output sequence $\mathbf{y}_{1:k}$ with $\mathbf{y}_i \in \mathbb{R}^{d_y}$, $i \in \mathbb{N}_k$.

A. Overall Architecture

The Transformer architecture is comprised of two main components: an encoder and a decoder, as depicted in Fig. 2. These components together make for a powerful learnable mapping between an input sequence $\mathbf{z}_{1:n}$ and an output sequence $\mathbf{y}_{1:k}$, typically trained using stochastic gradient descent on a loss function $\mathcal{L}(\mathbf{y}_{1:k}, \mathbf{x}_{1:k})$ that compares the network predictions with a ground-truth sequence $\mathbf{x}_{1:k}$.

B. Multi-head Self-attention Layer

The main building block for the Transformer architecture is the *self-attention layer*, used multiple times inside both the encoder and decoder modules. The self-attention layer first computes three different linear transformations of the input:

$$\mathbf{Q} = \mathbf{W}_Q \mathbf{A}, \ \mathbf{K} = \mathbf{W}_K \mathbf{A}, \ \mathbf{V} = \mathbf{W}_V \mathbf{A}, \tag{1}$$

where $\mathbf{A} = \begin{bmatrix}\mathbf{a}_1, \cdots, \mathbf{a}_n\end{bmatrix} \in \mathbb{R}^{d \times n}$, and the matrices $\mathbf{Q}, \mathbf{K}, \mathbf{V}$ are referred to as queries, keys, and values, respectively. The matrices $\mathbf{W}_Q, \mathbf{W}_K, \mathbf{W}_V \in \mathbb{R}^{d \times d}$ are the learnable parameters of the self-attention layer. The output is then computed as

$$\mathbf{B} = \mathbf{V} \cdot \text{Softmax-c}\left(\frac{\mathbf{K}^\top \mathbf{Q}}{\sqrt{d}}\right), \tag{2}$$

[1]A Poisson multi-Bernoulli density is the disjoint union of a PPP and a multi-Bernoulli (MB) density. In turn, an MB density is the disjoint union of Bernoulli components, each described by an existence probability and a state density function [16].

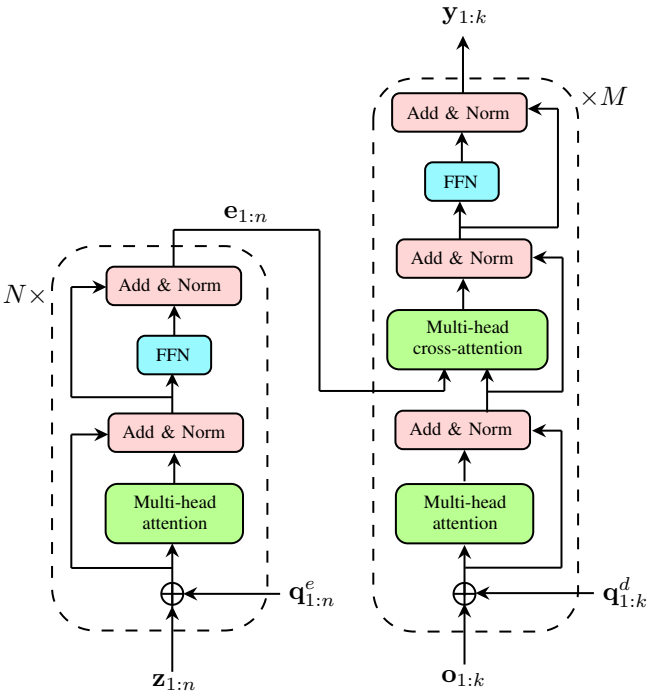

Fig. 2. Simplified diagram illustrating the Transformer architecture. Encoder on the left, containing N encoder blocks, processes the input sequence $\mathbf{z}_{1:n}$ into embeddings $\mathbf{e}_{1:n}$. Decoder on the right, containing M decoder blocks, uses the embeddings $\mathbf{e}_{1:n}$ produced by the encoder together with the object queries $\mathbf{o}_{1:k}$ to predict the output sequence $\mathbf{y}_{1:k}$. FFN stands for fully-connected feedforward neural network.

where $\mathbf{B} = \begin{bmatrix}\mathbf{b}_1, \cdots, \mathbf{b}_n\end{bmatrix} \in \mathbb{R}^{d \times n}$ and Softmax-c is the column-wise application of the Softmax function, defined as

$$[\text{Softmax-c}(\mathbf{Z})]_{i,j} = \frac{e^{z_{i,j}}}{\sum_{k=1}^{d} e^{z_{k,j}}}; \quad i, j \in \mathbb{N}_n$$

for $\mathbf{Z} \in \mathbb{R}^{n \times n}$, where $z_{i,j}$ is the element of $\mathbf{Z}$ on row i, column j. Because of this structure, each output $\mathbf{b}_i$ from a self-attention layer directly depends on all inner products of the type $\mathbf{a}_i^\top \mathbf{W} \mathbf{a}_j$, for $j \in \mathbb{N}_n$, with learnable $\mathbf{W}$, between the elements of the input sequence. This allows for the potential to learn an improved representation of each $\mathbf{a}_i$ that takes into account its relationship to all the other elements of the sequence.

In practice, Transformer-based architectures often use several self-attention layers in parallel and then combine the results, where this entire computation is referred to as a multi-head self-attention layer (shown in green in Fig. 2). For this, $\mathbf{A}$ is fed to n_h different self-attention layers (with separate learnable parameters) in parallel, generating n_h different outputs $\mathbf{B}_1, \cdots, \mathbf{B}_{n_h}$, all $\in \mathbb{R}^{d \times n}$. The final output $\mathbf{B}$ is then computed by vertically stacking the results and applying a linear transformation to reduce the dimensionality back to $\mathbb{R}^{d \times n}$:

$$\mathbf{B} = \mathbf{W}^0 \begin{bmatrix}\mathbf{B}_1 \\ \vdots \\ \mathbf{B}_{n_h}\end{bmatrix} \tag{3}$$

where $\mathbf{W}^0 \in \mathbb{R}^{d \times dn_n}$ is a learnable parameter of the multi-head self-attention layer. Finally, $\mathbf{B}$ is converted back to a sequence $\mathbf{b}_{1:n} = \text{MultiHeadAttention}(\mathbf{a}_{1:n})$.

C. Transformer Encoder

The Transformer encoder is the module in charge of transforming the input sequence $\mathbf{z}_{1:n}$ into the embeddings $\mathbf{e}_{1:n}$, where, after training, element $\mathbf{e}_i$ can potentially encode both the original value $\mathbf{z}_i$ and any important relationships it has to the other elements in the sequence.

A Transformer encoder is built from N encoder blocks in series, as shown in the left of Fig. 2. The output for encoder block $l \in \mathbb{N}_N$ is computed as

$$\tilde{\mathbf{z}}^{(l-1)}_{1:n} = \mathbf{z}^{(l-1)}_{1:n} + \mathbf{q}^e_{1:n} \tag{4}$$
$$\mathbf{t}^{(l)}_{1:n} = \text{MultiHeadAttention}(\tilde{\mathbf{z}}^{(l-1)}_{1:n}) \tag{5}$$
$$\tilde{\mathbf{t}}^{(l)}_{1:n} = \text{LayerNorm}(\tilde{\mathbf{z}}^{(l-1)}_{1:n} + \mathbf{t}^{(l)}_{1:n}) \tag{6}$$
$$\mathbf{z}^{(l)}_{1:n} = \text{LayerNorm}(\tilde{\mathbf{t}}^{(l)}_{1:n} + \text{FFN}(\tilde{\mathbf{t}}^{(l)}_{1:n})) \ , \tag{7}$$

where MultiHeadAttention is a multi-head self-attention layer, as described in Section III-B, FFN is a fully-connected feedforward neural network applied to each element of the input sequence separately, LayerNorm is a Layer Normalization layer (as introduced in [20]), and $\mathbf{z}^{(l)}_{1:n}$ is the input sequence after being processed by l encoder blocks. Hence, $\mathbf{z}^{(0)}_{1:n}$ is the original input sequence $\mathbf{z}_{1:n}$, and $\mathbf{z}^{(N)}_{1:n}$ is the output of the encoder module, also denoted $\mathbf{e}_{1:n}$. Importantly, $\mathbf{q}^e_{1:n}$ in (4), referred to as the positional encoding for the input sequence, is added to the input of every encoder block (as done in [21]), computed as $\mathbf{q}^e_i = f^e_p(i)$, where $f : \mathbb{Z} \to \mathbb{R}^{d_z}$, and f^e_p can either be fixed (usually with sinusoidal components [15]) or learnable [21]. Without it, the encoder module becomes permutation-equivariant[2], which is undesirable in many contexts. For instance, when processing images with Transformers the order of the elements in the input sequence is related to their location in the image, and therefore very important for correctly solving non-trivial tasks.

D. Transformer Decoder

Once the embeddings $\mathbf{e}_{1:n}$ are computed by the encoder, the decoder module is in charge of using them to predict the output sequence $\mathbf{y}_{1:k}$. Different structures for the Transformer decoder have been proposed for different contexts [15], [22], [23], and the one used for this paper is based on the DEtection TRansformer (DETR) decoder [21] using object queries $\mathbf{o}_{1:k}$ (illustrated on the right part of Fig. 2), due to its speed and capacity to generate outputs in parallel, instead of autoregressively. This type of decoder, just like the encoder module, is comprised of M decoder blocks, where the output for decoder block $l \in \mathbb{N}_M$ is computed as

$$\tilde{\mathbf{o}}^{(l-1)}_{1:k} = \mathbf{o}^{(l-1)}_{1:k} + \mathbf{q}^d_{1:k} \tag{8}$$
$$\mathbf{r}^{(l)}_{1:k} = \text{MultiHeadAttention}(\tilde{\mathbf{o}}^{(l-1)}_{1:k}) \tag{9}$$
$$\tilde{\mathbf{r}}^{(l)}_{1:k} = \text{LayerNorm}(\tilde{\mathbf{o}}^{(l-1)}_{1:k} + \mathbf{r}^{(l)}_{1:k}) \tag{10}$$
$$\tilde{\mathbf{e}}^{(l)}_{1:k} = \text{MultiHeadCrossAttention}(\tilde{\mathbf{r}}^{(l)}_{1:k}, \mathbf{e}_{1:n}) \tag{11}$$
$$\bar{\mathbf{e}}^{(l)}_{1:k} = \text{LayerNorm}(\tilde{\mathbf{r}}^{(l)}_{1:k} + \tilde{\mathbf{e}}^{(l)}_{1:k})) \tag{12}$$
$$\mathbf{o}^{(l)}_{1:k} = \text{LayerNorm}(\bar{\mathbf{e}}^{(l)}_{1:k} + \text{FFN}(\bar{\mathbf{e}}^{(l)}_{1:k}) \ , \tag{13}$$

where MultiHeadCrossAttention is a regular multi-head self-attention layer as described in Section III-B, with the difference that the matrices $\mathbf{K}$, $\mathbf{Q}$, and $\mathbf{V}$ in (1) are respectively computed as $\mathbf{W}_K\mathbf{e}_{1:n}$, $\mathbf{W}_Q\tilde{\mathbf{r}}^{(l)}_{1:k}$, and $\mathbf{W}_V\mathbf{e}_{1:n}$ (all of the subsequent self-attention computations are the same). The input to the first encoder block are the object queries $\mathbf{o}_{1:k}$, a sequence of learnable vectors trained jointly with the other model parameters. Once trained, each $\mathbf{o}_i \in \mathbb{R}^{d_o}$, $i \in \mathbb{N}_k$, will potentially learn to attend to the parts of the embeddings $\mathbf{e}_{1:n}$ that help predict $\mathbf{y}_i$. Similar to the encoder module, $\mathbf{o}^{(l)}_{1:k}$ represents the object queries after being processed by l decoder blocks (which also preserve the size of the input), where $\mathbf{o}^{(M)}_{1:k}$ denotes the output of the decoder module $\mathbf{y}_{1:k}$. Finally, to prevent the decoder module from being permutation-equivariant, a positional encoding $\mathbf{q}^d_{1:k}$ is added to the inputs of each layer, where $\mathbf{q}^d_i = f^d_p(i)$.

IV. RELATED WORK

In recent years, deep learning (DL) has been increasingly applied to the field of MOT, but mostly in the model-free setting. This section starts by reviewing the literature on DL for model-free MOT, up to recent state-of-the-art trackers using modern techniques such as Transformers. Then, a description of the challenges in doing the same for the model-free setting is presented, along with our contributions on this topic.

A. DL for model-free MOT

Deep learning (DL) has been widely applied to model-free MOT, resulting in multiple breakthroughs to the state of the art [6], [24], [25]. Some works use DL methods as aid for solving one or several specific MOT subtasks, such as object detection [26], [27], extracting high-level features from input data [28], [29], associating new measurements to existing tracks [30]–[32], managing track initialization/termination [33], predicting motion models [34], [35], and others. Others attempt to solve the entire MOT task using DL, with architectures based on extensions of object detectors [36], convolutional neural networks [37], [38], graph neural networks [39], [40].

More recently, and closer to our proposed work, Transformer-based architectures [41]–[45] have gained considerable prominence in the field, achieving excellent results in popular vision-based MOT benchmarks. All of them use encoder-decoder architectures with the concept

[2]A function f is equivariant to a transformation g iff $f(g(x)) = g(f(x))$.

of *track queries*. These queries are vectors predicted by the decoder and trained to summarize the history of an object. TrackFormer [41] and TransTrack [43] use these queries to determine which information from the current measurements will be utilized for generating predictions at the current time step. Additionally, the queries are used to create trajectories by linking predictions across time. MOTR [42] further expands upon this technique by incorporating a temporal aggregation network into the track queries, thereby enhancing the decoder's ability to base predictions on measurements from prior time steps. MeMOT [44] keeps an explicit memory bank of previous states to help with long-term associations and employs track queries to summarize an object's history, but it does not utilize them to link predictions over time (data association is addressed in a separate module). SegDQ [45] introduces the use of semantic segmentation as an auxiliary task in MOT, showcasing the benefits of multi-task learning.

B. DL for model-based MOT

Although DL approaches have been very successful in the model-free setting (especially video-based MOT [41]–[45]), the model-based setting has not received as much attention from practitioners. In principle, just as DL methods can often be easily extended to work with similar data in different settings, such model-free MOT trackers could be adapted to work for model-based MOT. Unfortunately, due to certain fundamental differences between these settings, several challenges arise when attempting this adaptation.

The first important difference between these contexts is in the dimensionality of the measurements obtained. Because vision-based MOT trackers must handle very high-dimensional measurements (images), their architectures have been developed with inductive biases specific to this challenge (e.g., stacks of convolutional layers, max pooling, etc.). In model-based MOT the measurements are typically low-dimensional, requiring these design decisions to be revisited and probably replaced by alternatives.

Second, these two settings have very different requirements regarding temporal associations. In vision-based MOT, it is very costly to process image data from multiple time steps jointly, and long-term temporal associations are not always essential (often a single image contains accurate information about the location of all objects). As a result, most [41], [43]–[48], if not all, of the existing vision-based MOT trackers are unable to process more than a few frames per time step. In contrast, long-term temporal associations are of utmost importance in model-based MOT, as data association uncertainties are significantly higher than for vision-based tasks. Adapting such methods to a context where long-range (considerably more than just a few time-steps) temporal associations are essential would require addressing this shortcoming in their design.

Third, vision-based trackers often leverage the image structure of the measurements for performing tracking in ways that do not have a straightforward counterpart in the model-based setting (e.g., [46], [47], which rely on learning appearance cues to handle occlusion properly). Naturally, such trackers cannot be used in the model-based setting without modifications, which would likely render some of their key novelties unusable.

Therefore, rather than handpicking a few of these approaches and attempting to tailor them to the model-based setting, we chose to draw inspiration from a variety of methods. To do so, we carefully picked the most promising aspects that we believe can work effectively in the model-based setting and incorporated other components that were not present in these approaches but that we believe can provide significant benefits for model-based MOT. This resulted in the novel loss formulations for training uncertainty-aware DL-based trackers described in Section V-D, and the proposed tracker, MT3v2. MT3v2 is specifically designed to excel in the low-dimensional model-based setting, with improvements such as the selection mechanism (Section V-B), iterative refinement (V-C), and the ability to directly leverage measurements arbitrarily far away in the past, this tracker surpasses the performance of current state-of-the-art Bayesian trackers in challenging, nonlinear tracking tasks.

V. MULTITARGET TRACKING TRANSFORMER V2

As mentioned in Section II, we see the model-based MOT problem as the task of mapping a sequence of measurements $\mathbf{z}_{1:n}$ to a sequence of predictions $\mathbf{y}_{1:k}$, corresponding to the parameters of a multi-Bernoulli density (state distribution and existence probability for each component) describing the objects present at time step T. Using the available transition and measurement models of the environment, we generate unlimited training data to train MT3v2 to learn this mapping. By approaching the problem as a sequence-to-sequence learning problem using a Transformer-based model, we are able to train a tracker that uses a constant number of parameters regardless of the number of time steps being processed. In this way, we maintain computational tractability by sidestepping the need for heuristics that often impact performance.

A. Overview of MT3v2 architecture

The MT3v2 architecture is similar to the Transformer encoder-decoder network described in Sec. III, with some important distinctions as shown in Fig. 3. First, instead of learning a set of fixed object queries $\mathbf{o}_{1:k}$, MT3v2 uses a selection mechanism to generate context-sensitive object queries (Sec. V-B), adapting to the measurements $\mathbf{z}_{1:n}$ for each sample. Second, the decoder structure uses iterative refinement (Sec. V-C) for improving regression performance. Third, the training of the architecture is improved by training with intermediate decoder predic-

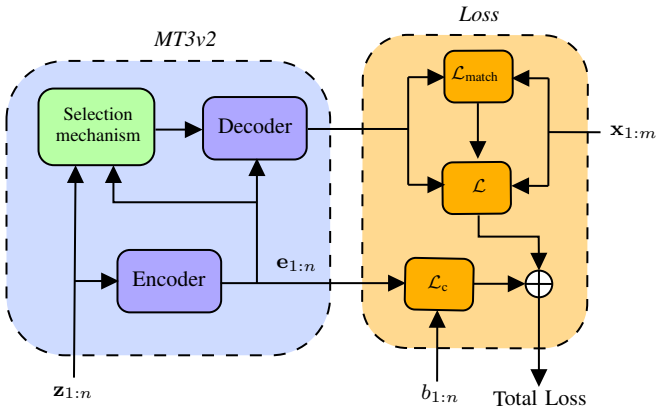

Fig. 3. Overview of the MT3v2 architecture. Input sequence of measurements $\mathbf{z}_{1:n}$ is processed by the encoder, generating the embeddings $\mathbf{e}_{1:n}$ and by the selection mechanism, generating the initial estimates $\tilde{\mathbf{z}}_{1:k}$, object queries $\mathbf{o}_{1:k}$, and positional encodings $\mathbf{q}^d_{1:k}$ for the decoder. The embeddings from the encoder, along with the output of the selection mechanism, are used by the decoder to output $\mathbf{y}_{1:k}$, describing a multi-Bernoulli density with k components.

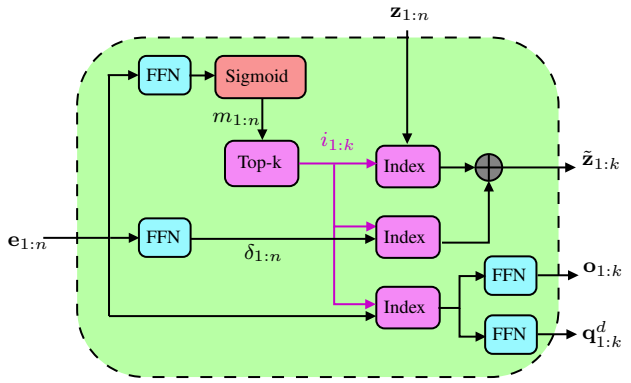

Fig. 4. MT3v2's selection mechanism: Embeddings $\mathbf{e}_{1:n}$ are fed to an FFN and then a sigmoid layer, producing scores $m_{1:n}$. The embeddings of the measurements with the top-k scores are fed to two FFNs, producing the object queries $\mathbf{o}_{1:k}$ and their corresponding positional encodings $\mathbf{q}^d_{1:k}$.

tions (Sec. V-D) and via the addition of a contrastive auxiliary loss (Sec. V-E). Lastly, instead of positional embeddings, we use embeddings based on the time-of-arrival of a measurement (Sec. V-F), allowing the model to leverage on this information without being distracted by the specific order of the elements in the measurement sequence $\mathbf{z}_{1:n}$.

The idea behind this specific structure is that the encoder can process the measurement sequence into a new representation $\mathbf{e}_{1:n}$ that summarizes relevant information to the MOT task, such as which are the clutter measurements, which measurements come from the same objects, etc. Then, the selection mechanism uses the generated embeddings and measurements to create object queries $\mathbf{o}_{1:k}$ (and corresponding positional embeddings) which are specifically suited for the current sequence $\mathbf{z}_{1:n}$. Furthermore, to relieve the decoder from the burden of having to generate predictions from scratch, the selection mechanism also generates potential starting points $\tilde{\mathbf{z}}_{1:k}$, which the decoder then uses for iterative refinement at each decoder block. The final output sequence from the decoder, denoted $\mathbf{y}_{1:k}$, represents the parameters of a k-component MB density. Each $\mathbf{y}_i, i \in \mathbb{N}_k$ is of the form $(\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i, p_i)$, containing respectively the mean and covariance for a Gaussian distribution, and the existence probability for that Bernoulli component.

Both the output sequence $\mathbf{y}_{1:k}$ from the decoder (along with outputs from the intermediate decoder blocks, see section V-D) and the embeddings $\mathbf{e}_{1:n}$ are used for training MT3v2. The output sequence is used to approximate the negative log-likelihood of the predicted multi-Bernoulli densities, while the embeddings are used for computing an auxiliary contrastive loss that accelerates learning. Training is then performed by optimizing the sum of these two different losses.

The rest of this section explains the selection mechanism and the iterative refinement process in the decoder in more detail, followed by a description of the negative log-likelihood loss and the contrastive auxiliary loss used, and finalizes with information about the most important preprocessing steps applied to the training data.

B. Selection Mechanism

The selection mechanism of MT3v2, illustrated in detail in Fig. 4, is in charge of producing the initial estimates for iterative refinement $\tilde{\mathbf{z}}_{1:k}$ (see Section V-C), and the object queries $\mathbf{o}_{1:k}$ along with their positional encodings $\mathbf{q}^d_{1:k}$, similar to the two-stage encoder proposed in [49]. It does so by learning to look for the measurements among $\mathbf{z}_{1:n}$ that are the best candidates to be used as starting points for the decoder (i.e., measurements that are likely to be close to the state estimates that the decoder is in charge of producing for that specific sequence $\mathbf{z}_{1:n}$) and basing its outputs on them. This simplifies the decoder's task and improves performance, as shown in Sec. VII-C.

First, scores $m_i \in [0, 1]$ for each of the embeddings $\mathbf{e}_i$ are computed according to $m_i = \text{Sigmoid}(\text{FFN}(\mathbf{e}_i)), i \in \mathbb{N}_n$. The indices $i_{1:k}$ of the top-k scores are then computed according to top-k$(m_{1:n}) = [i_1, i_2, \cdots, i_k]$, where

$$i_j = \arg\max_a m_a \quad \text{s.t. } a \notin \{i_l : l < j\}, \tag{14}$$

for $j \in \mathbb{N}_k$. These indices $i_{1:k}$ are used to index the sequences $\boldsymbol{\delta}_{1:n}$ (predicted adjustments), $\mathbf{z}_{1:n}$, and $\mathbf{e}_{1:n}$, by applying the Index function

$$\text{Index}(\mathbf{a}_{1:n}, i_{1:k}) = [\mathbf{a}_{i_1}, \mathbf{a}_{i_2}, \cdots, \mathbf{a}_{i_k}] \ , \tag{15}$$

which we will abbreviate as $\text{Index}(\mathbf{a}_{1:n}, i_{1:k}) = \mathbf{a}_{i_{1:k}}$. The initial estimates $\tilde{\mathbf{z}}_{1:k}$ are then computed by summing the top-k measurements and their corresponding predicted adjustments

$$\tilde{\mathbf{z}}_{1:k} = \mathbf{z}_{i_{1:k}} + \boldsymbol{\delta}_{i_{1:k}} \ , \tag{16}$$

where $\boldsymbol{\delta}_i = \text{FFN}(\mathbf{e}_i)$. At the same time, the object queries and decoder positional encodings are computed by feeding the top-k embeddings to separate FFN layers:

$$\mathbf{o}_{1:k} = \text{FFN}(\mathbf{e}_{i_{1:k}}), \quad \mathbf{q}^d_{1:k} = \text{FFN}(\mathbf{e}_{i_{1:k}}). \tag{17}$$

In contrast to MT3 [12], MT3v2's selection mechanism feeds $\mathbf{e}_{i_{1:k}}$ to FFN heads to produce the object queries and the positional encodings, not the indexed measurements $\tilde{\mathbf{z}}_{i_{1:k}}$. We found that this improves performance, as the embeddings are higher-dimensional and can therefore contain much more information about the task for producing useful queries and positional encodings.

C. Iterative Refinement

To further improve the performance of MT3v2, we adopt the idea of iterative refinement [49], [50] in the decoder. As stated previously, the decoder outputs the sequence $\mathbf{y}_{1:k}$ which represents the k components of an MB density, where each $\mathbf{y}_i = (\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i, p_i)$ contains respectively the mean, covariance, and existence probability for that Bernoulli. Instead of directly computing the sequence of predicted state means $\boldsymbol{\mu}_{1:k}$ from the output of the decoder's last layer (e.g., $\boldsymbol{\mu}_{1:k} = f(\mathbf{o}_{1:k}^{(M)})$, with some learnable f), we start with the initial state estimates $\tilde{\mathbf{z}}_{1:k}$ computed by the selection mechanism, and each decoder layer $l \in \mathbb{N}_M$ generates adjustments $\boldsymbol{\Delta}_{1:k}^l$ to it. Summing all adjustments to the initial estimates then yields the output $\boldsymbol{\mu}_{1:k}$ for the decoder.

Concretely, the initial estimates $\tilde{\mathbf{z}}_i$ are first transformed from measurement space to state-space, and are then denoted by $\boldsymbol{\mu}_i^0$. Then, the output $\mathbf{o}_{1:k}^{(l)}$ of each decoder layer l is fed to an FFN (each layer has an FFN with separate parameters), which then produces adjustments $\boldsymbol{\Delta}_{1:k}^l$ in the state space. New adjustments are added to the previous estimate at each decoder layer, resulting in predicted state means $\boldsymbol{\mu}_{1:k}^l$ for each layer l, where

$$\boldsymbol{\mu}_i^l = \boldsymbol{\mu}_i^0 + \sum_{l=1}^{M} \boldsymbol{\Delta}_i^l, \quad l \in \mathbb{N}_M \ , \tag{18}$$

Covariances and existence probabilities are not iteratively refined and are directly computed at each layer as

$$\boldsymbol{\Sigma}_{1:k}^l = \mathrm{Diag}\Big(\mathrm{Softplus}\big(\mathrm{FFN}(\mathbf{o}_{1:k}^{(l)})\big)\Big), \tag{19}$$

$$p_{1:k}^l = \mathrm{Sigmoid}\big(\mathrm{FFN}(\mathbf{o}_{1:k}^{(l)})\big), \tag{20}$$

where $\mathrm{Softplus}(\cdot)$ is applied element-wise as

$$\mathrm{Softplus}(x) = \log(1 + e^x), \tag{21}$$

and $\mathrm{Diag} : \mathbb{R}^n \to \mathbb{R}^{n \times n}$, also applied element-wise, is a function that constructs a diagonal matrix from its input. Predicting diagonal covariance matrices $\boldsymbol{\Sigma}_i^l$ may impact performance but improves training time considerably (avoids the need to invert a full positive definite matrix when computing the log-likelihood of the state density, see Sec. V-D). Putting these together, each decoder layer produces an MB density $\mathbf{y}_{1:k}^l = (\boldsymbol{\mu}_{1:k}^l, \boldsymbol{\Sigma}_{1:k}^l, p_{1:k}^l)$. The final output of MT3v2 is then the output at the last decoder layer, i.e., $\mathbf{y}_{1:k} = \mathbf{y}_{1:k}^M$, whereas the other outputs $\mathbf{y}_{1:k}^l, l \in \mathbb{N}_{M-1}$ are used only during training.

D. Loss

We train MT3v2 using an approximation of the expected sum of the negative log-likelihood (NLL) of the M MBs $\mathbf{y}_{1:k}^l, l \in \mathbb{N}_M$, evaluated at the ground-truth target states [51]. Training all the intermediate decoder block outputs instead of just the final predictions $\mathbf{y}_{1:k}$ from the final layer is shown to accelerate learning for deep Transformer decoder architectures and improve final performance [21], [52], and is confirmed by our ablations (see Sec. V-C). We sample a measurement sequence $\mathbf{z}_{1:n}$, and corresponding ground-truth targets $\mathbf{x}_{1:m}$ using the available models of the environment, where $\mathbf{x}_i \in \mathbb{R}^{d_x}, i \in \mathbb{N}_m$ are the states for the m objects which are alive at the last time step. Then the measurements are fed to MT3v2, which generates predictions $\mathbf{y}_{1:k}^l, l \in \mathbb{N}_M$ (one for each decoder layer, see Section V-C). The loss for this sample is then expressed as

$$\mathcal{L}(\mathbf{x}_{1:m}, \mathbf{y}_{1:k}^1, \cdots, \mathbf{y}_{1:k}^M) = -\sum_{l=1}^{M} \log f^l(\mathbf{x}_{1:m}), \tag{22}$$

where $f^l(\mathbf{x}_{1:m})$ is the MB density specified by $\mathbf{y}_{1:k}^l$, evaluated at $\mathbf{x}_{1:m}$.

Computing $f^l(\mathbf{x}_{1:m})$ directly is computationally intractable, since all possible associations between the MB components and the ground-truth object states must be accounted for, making the number of terms in this expression grow super-exponentially on k and m [51], [53]. However, in most cases all but one of the possible associations between Bernoullis and targets have negligible contribution, so we can approximate this NLL with only the contribution from the most likely association. To do so, we append '$\varnothing$' elements to the sequence $\mathbf{x}_{1:m}$ resulting in a new sequence $\tilde{\mathbf{x}}_{1:k}$ with the same number of elements[3] as each $\mathbf{y}_{1:k}^l$, and approximate the NLL as

$$-\log f^l(\mathbf{x}_{1:m}) = -\sum_{i=1}^{k} \log \sum_{\sigma} f_i^l(\tilde{\mathbf{x}}_{\sigma(i)}) \tag{23}$$

$$\approx -\sum_{i=1}^{k} \log f_i^l(\tilde{\mathbf{x}}_{\sigma^l(i)}), \tag{24}$$

where σ is a permutation function, $\sigma : \mathbb{N}_k \to \mathbb{N}_k \mid \sigma(i) = \sigma(j) \Rightarrow i = j$, corresponding to one possible association between MB components and ground-truth object states, and $f_i^l(\mathbf{x}_{\sigma(i)})$ is the Bernoulli density specified by $\mathbf{y}_i^l$ evaluated at the $\sigma(i)$-th element of $\tilde{\mathbf{x}}_{1:k}$, such that

$$\log f_i^l(\tilde{\mathbf{x}}_j) = \begin{cases} \log p_i^l + \log \mathcal{N}(\tilde{\mathbf{x}}_j; \boldsymbol{\mu}_i^l, \boldsymbol{\Sigma}_i^l) & \text{if } \tilde{\mathbf{x}}_j \neq \varnothing \\ \log(1 - p_i^l) & \text{otherwise.} \end{cases} \tag{25}$$

Finally, σ^l corresponds to the most likely association between objects and Bernoulli components predicted at the decoder layer l. Computing σ^l directly as described

[3] We choose a value of k which is large in comparison to the generative model and enforce $m \leq k$ by not adding more than k objects to any sample. This restriction is only enforced during training; during evaluation/inference this loss does not need to be computed.

in [51] resulted in unstable learning, and we instead approximate it similarly to [21], as

$$\sigma^l = \arg\min_{\sigma} \sum_{i=1}^{k} \mathcal{L}_{\text{match}}(\mathbf{y}_i^l, \tilde{\mathbf{x}}_{\sigma(i)}), \tag{26}$$

where

$$\mathcal{L}_{\text{match}}(\mathbf{y}_i^l, \tilde{\mathbf{x}}_{\sigma(i)}) = \begin{cases} 0, & \text{if } \tilde{\mathbf{x}}_{\sigma(i)}^l = \varnothing \\ \|\boldsymbol{\mu}_i^l - \tilde{\mathbf{x}}_{\sigma(i)}\| - \log p_i^l, & \text{otherwise,} \end{cases} \tag{27}$$

which can be solved efficiently using the Hungarian algorithm [54].

E. Contrastive Auxiliary Learning

Another improvement we add to the training process is an auxiliary task of trying to predict which of the measurements in $\mathbf{z}_{1:n}$ came from which objects (and which are clutter). Adding simpler auxiliary tasks often improves the initial part of the training process (when the main task is still too hard to solve, and might not provide much gradient information) and the generalization performance of the final model [55].

To implement this, we use an idea inspired by Supervised Contrastive Learning [56], where the model is trained to generate similar predictions for samples of the same classes, but dissimilar to samples of other classes. During the data generation, we annotate each measurement $\mathbf{z}_i, i \in \mathbb{N}_n$, with an integer b_i encoding from which object it came from, -1 if it is clutter . Let $\mathbb{P}_i$ be the set of indices of measurements that came from the same object as the measurement $\mathbf{z}_i$, $\mathbb{P}_i = \{j \in \mathbb{N}_n \mid j \neq i \ , \ b_i = b_j\}$, the auxiliary loss $\mathcal{L}_c$ is then defined similarly to [56], but using the object identifiers $b_{1:n}$ as the labels for the contrastive learning of the encoder embeddings:

$$\mathcal{L}_c(\mathbf{e}_{1:n}, b_{1:n}) = \beta \sum_{i=1}^{n} \frac{-1}{|\mathbb{P}_i|} \sum_{i^+ \in \mathbb{P}_i} \log \frac{e^{\mathbf{u}_i^\top \mathbf{u}_{i^+}}}{\sum_{j \in \mathbb{N}_n \setminus i} e^{\mathbf{u}_i^\top \mathbf{u}_j}} \tag{28}$$

$$\mathbf{u}_{1:n} = \frac{\text{FFN}(\mathbf{e}_{1:n})}{\|\text{FFN}(\mathbf{e}_{1:n})\|_2}. \tag{29}$$

where $\beta \geq 0$ is a hyperparameter controlling the trade-off between the auxiliary task and the main task. This loss can be intuitively understood as encouraging the processed embeddings $\mathbf{u}_i$ and $\mathbf{u}_j$ from different measurements $\mathbf{z}_i, \mathbf{z}_j$ to be similar if $b_i = b_j$ ($\mathbf{u}_i^\top \mathbf{u}_j$ will be large) or dissimilar if $b_i \neq b_j$ ($\mathbf{u}_i^\top \mathbf{u}_j$ will be small). Training the model on the sum of this auxiliary loss and the loss defined in Section V-D accelerated learning and improved MT3v2's final performance, especially in more challenging tasks, as shown in our ablations studies in Sec. V-C.

F. Preprocessing

1. *Preprocessing measurements*

Before feeding the measurement sequence to MT3v2, each measurement goes through three preprocessing steps. First, we divide a measurement $\mathbf{z_i}$ by a normalization factor to ensure that the values in each dimension (r, $\dot{r}$, θ) are in the range $[0, 1]$. The normalization factor is computed using the known FOV dimensions.

Second, we increase its dimensionality to a value $d' > 3$, by multiplying it with a learnable matrix $\mathbf{W} \in \mathbb{R}^{d' \times 3}$. This transformation allows the self-attention layers in the encoder to have dimensionality d' instead of 3, which increases the flexibility of its representational power.

Lastly, the time-of-arrival t of z_i is used to compute its encoder positional encoding $\mathbf{q}_i^e \in \mathbb{R}^{d'}$. This is done with the help of a learned lookup table $f_\lambda(t_i)$, where λ is trained jointly with the other parameters of the network. This allows the architecture to have direct access to the corresponding time of measurement for each $\mathbf{z}_i$, while at the same time sidestepping the need to learn that the position in the sequence $\mathbf{z}_{1:n}$ is irrelevant to the task.

2. *Postprocessing outputs*

As an additional step, we also post-process the model's outputs. The final predicted state means $\mu_{1:k}^l$ for each layer l (see (18)) are scaled using the FOV's known dimensions before computing the losses, so that an unscaled output ranging from 0 to 1 covers the entire FOV. This ensures that the decoder structure needs to produce at most unitary outputs regardless of the actual size of the FOV, which reduces the chance of exploding gradients due to large errors during training.

VI. EVALUATION SETTING

This section describes the setting used to evaluate the capabilities of the proposed DL tracker in model-based MOT. Specifically, we benchmark MT3v2 against two SOTA Bayesian MOT filters based on the random finite set formalism: the Poisson multi-Bernoulli mixture (PMBM) filter [10] and the δ-generalized labeled multi-Bernoulli (δ-GLMB) filter [8], in a simulated scenario with synthetic radar measurements. The PMBM filter provides a closed-form solution for MOT with standard multitarget models with Poisson birth, whereas the δ-GLMB filter provides a closed-form solution for MOT when the object birth model is a multi-Bernoulli (mixture). In what follows, we first describe the tasks the different algorithms were deployed on, along with their most relevant implementation details, and then we present the measures for evaluating the filtering performance.

A. Task Description

We compare the performance of the DL approach to the traditional Bayesian filters in four different tasks. task 1 is a baseline task, simpler than the other 3, where we expect traditional approaches to be a strong benchmark

for the DL tracker. We then investigate the impact of increasing the complexity of the data association (task 2), increasing the nonlinearity of the models (task 3), and both changes simultaneously (task 4).

The motion model used for all four tasks is the nearly constant velocity model, defined as:

$$f(\mathbf{x}^{t+1}|\mathbf{x}^t) = \mathcal{N}(\mathbf{x}^{t+1}\ ;\ \mu(\mathbf{x}^t)\ ,\ \Sigma\ ),$$
$$\mu(\mathbf{x}^t) = \begin{bmatrix}\mathbf{I} & \mathbf{I}\Delta_t \\ \mathbf{0} & \mathbf{I}\end{bmatrix}\mathbf{x}^t,\ \Sigma = \sigma_q^2\begin{bmatrix}\mathbf{I}\frac{\Delta_t^3}{3} & \mathbf{I}\frac{\Delta_t^2}{2} \\ \mathbf{I}\frac{\Delta_t^2}{2} & \mathbf{I}\Delta_t\end{bmatrix}$$

where $\mathbf{x}^{t+1}, \mathbf{x}^t \in \mathbb{R}^{d_x}$, $d_x = 4$ represents target position and velocity in 2D at time steps $t+1$ and t respectively, and $\Delta_t = 0.1$ is the sampling period, σ_q controls the magnitude of the process noise. The state for newborn objects is sampled from $\mathcal{N}(\boldsymbol{\mu}_b, \boldsymbol{\Sigma}_b)$ with

$$\boldsymbol{\mu}_b = \begin{bmatrix}7\\0\\0\\0\end{bmatrix}, \quad \boldsymbol{\Sigma}_b = \begin{bmatrix}10 & 0 & 0 & 0\\ 0 & 30 & 0 & 0\\ 0 & 0 & 3 & 0\\ 0 & 0 & 0 & 3\end{bmatrix},$$

values chosen to have an object birth model that covers a reasonable part of the field of view. The measurement model used is a non-linear Gaussian model simulating a radar system: $g(\mathbf{z}|\mathbf{x}) = \mathcal{N}\big(\mathbf{z}; H(\mathbf{x}), \boldsymbol{\Sigma}(\mathbf{x})\big)$, where H transforms the xy-position and velocity state-vector $\mathbf{x}$ into $(r, \dot{r}, \theta)$, respectively the range, Doppler and bearing of each target. For tasks 3 and 4, $\boldsymbol{\Sigma}(\mathbf{x})$ is computed according to the approach described in [57], with the hyperparameters detailed in Appendix A, resulting in a realistic radar measurement model with strong non-linearities close to the edges of the FOV (measurement noise increases quickly as the objects get closer to the edges). In contrast, in tasks 1 and 2 $\boldsymbol{\Sigma}(\mathbf{x})$ is set to the constant

$$\boldsymbol{\Sigma}(\mathbf{x}) = \begin{bmatrix}5.62\cdot10^{-3} & 0 & 0\\ 0 & 9.56\cdot10^{-1} & 0\\ 0 & 0 & 1.00\cdot10^{-2}\end{bmatrix}$$

where the values of the diagonal were chosen to make all tasks have similar measurement noise intensity in the central region of the FOV. The field of view for all tasks is the volume delimited by the ranges $(0.5, 150), (0, 30), (-1.3, 1.3)$ for r (m), $\dot{r}$ (m/s), and θ (radians), respectively.

All tasks use Poisson models with parameter λ_0 for the initial number of objects and have $\tau = 20$, and $p_s(\cdot) = 0.95$. To increase the data association complexity in tasks 2 and 4, certain hyperparameters of the generative model were changed, as shown in Table I. The simultaneous increase in the number of clutter measurements, process noise, and number of objects, along with a decrease in the detection probability, causes a substantial increase in the number of probable hypotheses that conventional MOT algorithms have to keep track of, making it considerably harder for them to perform well with a feasible computational complexity. To give an estimate of the data association complexity, we report that, for the PMBM filter, an average of 20 probable hypotheses were necessary to be kept track of in tasks 1 and 3 for achieving the reported performance. On the other hand, for tasks 2 and 4 the number of hypotheses maintained by PMBM was ∼140, almost an order of magnitude higher than for the simpler tasks.

TABLE I
Hyperparameters changed for increasing data association complexity.

Task	λ_0	p_d	λ_c	σ_q	λ_b
1	2	0.95	$4.4\cdot10^{-3}$	0.2	$1.3\cdot10^{-4}$
3	2	0.95	$4.4\cdot10^{-3}$	0.2	$1.3\cdot10^{-4}$
2	6	0.7	$2.6\cdot10^{-2}$	0.9	$3.5\cdot10^{-4}$
4	6	0.7	$2.6\cdot10^{-2}$	0.9	$3.5\cdot10^{-4}$

B. Implementation Details

For all experiments on MT3v2, the increased dimensionality of the measurements is $d' = 256$, we use $N = 6$ encoder blocks and $M = 6$ decoder blocks, multi-head self-attention layers with 8 attention heads and $k = 16$ object queries, embedding layers with $d^e = 256$, object queries with $d^o = 256$, and the contrastive loss hyperparameter β is set to 4.0. All FFNs in the encoder and decoder blocks have 2048 hidden units and are trained with a dropout rate of 0.1. All FFNs in the selection mechanism have 128 hidden units, while the one used for computing $\mathbf{u}_i$ in the contrastive loss has 256. To compute $\boldsymbol{\mu}_i^0$ in (18), measurements are mapped from measurement space to state-space as

$$\boldsymbol{\mu}_i^0 = \big(r_i\cos(\theta_i),\ r_i\sin(\theta_i),\ 0,\ 0\big)$$

(i.e., using 0 as initial estimates for the velocity dimensions). The model was trained using Adam [58] with a batch size of 32 and initial learning rate $5\cdot10^{-5}$, and whenever the loss did not decrease for 50k consecutive gradient steps the learning rate was reduced by a factor of 4. The training was performed on a V100 GPU for 1M gradient steps in tasks 1 and 2, 700k in task 3, and 400k in task 4, amounting to approximately 4 days of training for each task. MT3v2 was implemented in Python + PyTorch, and the code to define, train, and evaluate it is made publicly available at https://github.com/JulianoLagana/MT3v2, along with trained models for each of the tasks.

We proceed to describe the implementation details of PMBM and δ-GLMB. PMBM uses a Poisson birth model with Poisson intensity $\lambda_b\mathcal{N}(\boldsymbol{\mu}_b, \boldsymbol{\Sigma}_b)$, and the initial Poisson intensity for undetected objects is set to $\lambda_0\mathcal{N}(\boldsymbol{\mu}_b, \boldsymbol{\Sigma}_b)$. As for δ-GLMB, the multi-Bernoulli birth model is used, which contains a single Bernoulli component with probability of existence λ_b and state density $\mathcal{N}(\boldsymbol{\mu}_b, \boldsymbol{\Sigma}_b)$. In addition, to model undetected objects existing at time 0, the multi-Bernoulli birth model at time 1 contains $2\lambda_0$ Bernoulli components, each of which has probability of existence 0.5 and state density $\mathcal{N}(\boldsymbol{\mu}_b, \boldsymbol{\Sigma}_b)$.

To handle the nonlinearity of the measurement model, the iterated posterior linearization filter (IPLF) [59] is incorporated in both PMBM and δ-GLMB, see, e.g., [60].

The IPLF is implemented using sigma points with the fifth-order cubature rule [61] as suggested in [62] for radar tracking with range-bearing-Doppler measurements, and the number of iterations is 5. In IPLF, the state-dependent measurement noise covariance $\Sigma(\mathbf{x})$ is approximated as $\Sigma(\hat{\mathbf{x}})$ where $\hat{\mathbf{x}}$ is either the mean of the predicted state density or the mean of the state density at last iteration.

For PMBM and δ-GLMB, the unknown data associations lead to an intractably large number of terms in the posterior densities. To achieve computational tractability of both PMBM and δ-GLMB, it is necessary to reduce the number of parameters used to describe the posterior densities. First, gating is used to remove unlikely measurement-to-object associations, by thresholding the squared Mahalanobis distance, where the gating size is 20. Second, we use Murty's algorithm [53] to find up to 200 best global hypotheses. Third, we prune hypotheses with weights smaller than 10^{-4}. For PMBM, we also prune Bernoulli components with probability of existence smaller than 10^{-5} and Gaussian components in the Poisson intensity for undetected objects with weights smaller than 10^{-5}. Both PMBM[4] and δ-GLMB[5] implementations were developed in MATLAB.

C. Performance Measures

To evaluate the algorithms we use two performance measures: the generalized optimal sub-pattern assignment (GOSPA) metric [63], and the negative log-likelihood of the MOT posterior (NLL) [51]. The GOSPA metric is considered due to its widespread use, its computational simplicity, and for being a metric in the space of sets. The NLL performance measure is used to evaluate the algorithms further, taking into account all of the uncertainties available in the predicted MOT posterior. We compute a Monte Carlo approximation of the expected value of each performance measure by generating 1k samples of measurement sequences $\mathbf{z}_{1:n}$ and corresponding ground-truth object states $\mathbb{X}^T$ from the generative model. The measurement sequences are fed to each of the tracking algorithms, producing MOT posterior densities for each sample, which are then compared to the corresponding $\mathbb{X}^T$'s, using each of the performance measures.

1. *GOSPA metric*

To compute the GOSPA [63] metric, it is necessary to extract state estimates from the predicted MOT posterior for each algorithm, generating point-wise predictions for the states of objects alive at time step T: $\hat{\mathbb{X}} = \{\hat{\mathbf{x}}_1, \cdots, \hat{\mathbf{x}}_{|\hat{\mathbb{X}}|}\}$. For PMBM and δ-GLMB, we first process the MOT posterior by selecting the global hypothesis with the largest weight (method 1, as defined in [10]), generating a multi-Bernoulli distribution. Then, $\hat{\mathbb{X}}$ is formed by selecting the means of all Bernoulli components with existence probability greater than p_{cutoff}, where p_{cutoff} is chosen separately for each algorithm to minimize its GOSPA score. The MOT posterior defined by MT3v2's output $\mathbf{y}_{1:k}$ is already in the form of a multi-Bernoulli density, and we also form its $\hat{\mathbb{X}}$ by thresholding the components based on their existence probabilities.

Given a set of state estimates $\hat{\mathbb{X}}$, we compute the GOSPA metric between $\hat{\mathbb{X}}$ and the ground-truth target states $\mathbb{X}^T$, with $\alpha = 2$ and Euclidean distance, defined as

$$d_p^{(c,2)}(\hat{\mathbb{X}}, \mathbb{X}) = \min_{\gamma \in \Gamma} \Big(\underbrace{\sum_{(i,j)\in\gamma} \|\hat{\mathbf{x}}_i^T - \mathbf{x}_j^T\|_2^p}_{\text{Localization}} + \underbrace{\frac{c^p}{2}(|\hat{\mathbb{X}}| - |\gamma|)}_{\text{False detections}} + \underbrace{\frac{c^p}{2}(|\mathbb{X}| - |\gamma|)}_{\text{Missed objects}} \Big)^{\frac{1}{p}} \tag{30}$$

where the minimization is over assignment sets between the elements of $\hat{\mathbb{X}}$ and $\mathbb{X}$, such that $\gamma \subseteq \{1, \cdots, |\hat{\mathbb{X}}|\} \times \{1, \cdots, |\mathbb{X}|\}$, while $(i,j),(i,j') \in \gamma \implies j = j'$, and $(i,j),(i',j) \in \gamma \implies i = i'$. In all our experiments we use $c = 10$, $p = 1$. The cutoff parameter was chosen at 10 because the tracking task is 4-dimensional: 2D position and velocity. Higher-dimensional tasks make the Euclidean distance between states larger than for lower-dimensional tracking scenarios. See Appendix B for an analysis of the sensitivity of the results to the cutoff parameter c.

2. *NLL performance measure*

The NLL performance measure is computed by evaluating how well the MOT posterior explains the ground-truth data [51] in terms of its negative log-likelihood:

$$\text{NLL}(f_a, \mathbb{X}^T) = -\log f_a(\mathbb{X}^T) \tag{31}$$

where f_a is the MOT posterior computed by tracker a. As explained in [51], for an algorithm to obtain a good NLL score, its MOT posterior must be able to explain the set of objects $\mathbb{X}^T$ for all samples, including potential missed objects and false detections. Algorithms like δ-GLMB are therefore not suitable for being assessed with this performance measure, since the MOT posterior produced by them is unable to explain any missed objects: $f_{\delta-\text{GLMB}}(\mathbb{X}^T) = 0$ whenever $|\mathbb{X}^T| > n$, where n is the number of Bernoulli components in δ-GLMB's predicted posterior, therefore resulting in an average NLL score of ∞ (for NLL, lower values entail better performance). For PMBM we use method 1 as described in [10] to extract a PMB density, which is then able to explain any number of missed objects due to its PPP component.

To evaluate MT3v2 with the NLL performance measure, we add a PPP component with a piece-wise constant intensity function described as

$$\lambda_{\text{MT3v2}}(\mathbf{x}) = \begin{cases} \bar{\lambda}\,, & \text{if } \mathbf{x} \text{ is inside the FOV} \\ 0\,, & \text{otherwise} \end{cases} \tag{32}$$

to its posterior, resulting in a PMB MOT density for MT3v2:

$$f_{\text{MT3v2}}(\mathbb{X}^T) = \sum_{\mathbb{X}^D \uplus \mathbb{X}^U = \mathbb{X}^T} f_{\text{MB}}(\mathbb{X}^D) e^{-\bar{\lambda}} \prod_{\mathbf{x}\in\mathbb{X}^U} \lambda_{\text{MT3v2}}(\mathbf{x}) \tag{33}$$

[4] https://github.com/Agarciafernandez/MTT/tree/master/PMBM%20filter

[5] http://ba-tuong.vo-au.com/rfs_tracking_toolbox_updated.zip

where $f_{\text{MB}}(\cdot)$ is the MB density with k Bernoulli components described by MT3v2's output $\mathbf{y}_{1:k}$, and $\bar{\lambda}$ is tuned to minimize NLL over 1k samples from the generative model.

Lastly, directly computing the NLL for a PMB density is not computationally tractable except for the simplest cases, so we approximate it using the algorithm presented in [51], resulting in the following performance measure:

$$\text{NLL}(f, \mathbb{X}) \approx \min_{\gamma \in \Gamma} \underbrace{-\sum_{(i,j)\in\gamma} \log\left(p_i g_i(\mathbf{x}_j)\right)}_{\text{Localization}} \tag{34}$$

$$\underbrace{-\sum_{i\in\mathbb{F}(\gamma)} \log(1-p_i)}_{\text{False detections}} \underbrace{+\int \lambda(y')\mathrm{d}y' - \sum_{j\in\mathbb{M}(\gamma)} \log\lambda(\mathbf{x}_j)}_{\text{Missed objects}} ,$$

where p_i, g_i are the existence probabilities [6] and state densities for the i-th Bernoulli components of the PMB density f, and $\lambda(\cdot)$ is its PPP intensity function. Here Γ is the set of all possible assignment sets (as defined for GOSPA), while $\mathbb{F}(\gamma) = \{i \in \mathbb{N}_m \mid \nexists\, j : (i,j) \in \gamma\}$ is the set of indices of the Bernoullis not matched to any ground-truth (m is the number of Bernoulli components in f), and $\mathbb{M}(\gamma) = \left\{j \in \mathbb{N}_{|\mathbb{X}|} \mid \nexists\, i : (i,j) \in \gamma\right\}$ is the set of indices of ground-truths not matched to any Bernoulli component.

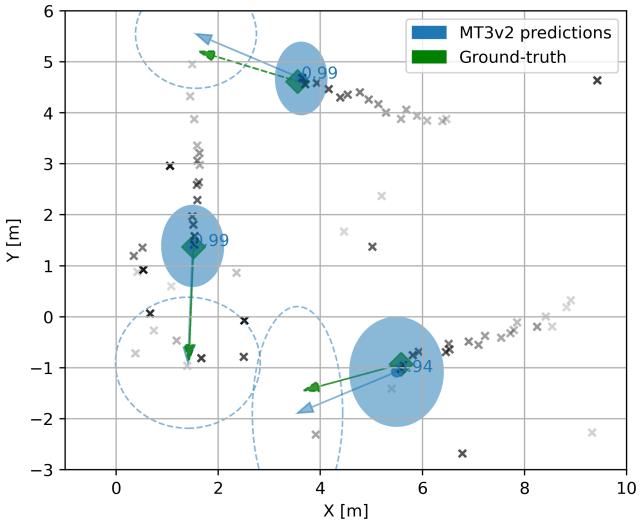

Fig. 5. Illustration of MT3v2's tracking performance in a simplified setting. Black crosses illustrate the measurements available to MT3v2 (Doppler component not illustrated to avoid clutter), and their transparency is determined by their relative time of measurement: more opaque crosses correspond to more recent measurements, closer to $t = T$. Dark blue circles and arrows show MT3v2's predicted object positions and velocities. Light blue ellipses illustrate the predicted position uncertainties and dashed ellipses the predicted velocity uncertainties. Ground-truth object positions and velocities are illustrated in green as diamonds and arrows, respectively.

VII. RESULTS

This section contains the results of the evaluations performed for assessing the tracking capabilities of the proposed deep learning tracker and is divided into four subsections. First, subsection A, describes an illustrative example of the performance of MT3v2 in a simple tracking task, depicting the predictions generated by the algorithm in this context and validating their soundness. Second, subsection B contains the results of a thorough comparison of MT3v2 to the model-based SOTA algorithms PMBM and δ-GLMB, in the four tasks described in Section A. Third, subsection C validates and quantifies the importance of each of MT3v2's most important design decisions via a set of ablation studies. Lastly, subsection D further evaluates each algorithm by investigating its running time complexity.

A. Illustrative Example

As a way to validate the soundness of the MT3v2 architecture, it is helpful to illustrate the predictions generated by it given a certain sequence of measurements. However, doing so using measurement sequences sampled from any of the four tasks described in section VI-A proved to be not optimal for this. The high measurement and motion noise in these tasks, along with a high number of clutter measurements makes it challenging to interpret measurement sequences visually.

Therefore, we resorted to creating a new, simpler task with fewer clutter measurements and lower measurement noise just for this purpose, and trained MT3v2 in it. MT3v2's tracking capabilities after training are illustrated in Fig. 5, which contains the measurement sequence fed to MT3v2, along with the predictions generated for this 2D tracking task. It shows MT3v2 being able to track three objects among clutter, estimating their position and velocities and also providing sensible uncertainty predictions for these quantities.

Evidently, although helpful as a sanity test for the approach, this type of analysis does not suffice for comparing MT3v2's performance to other approaches. Hence, we perform a thorough comparison in subsection B using the performance measures described in Section VI-C over a large number of samples instead.

B. Comparison to Model-Based SOTA

The performance of MT3v2 was compared to both SOTA Bayesian filters PMBM and δ-GLMB, in the four tasks[7] introduced in Section VI-A. For this purpose, MT3v2 was trained from scratch in each of the 4 tasks,

[6] Method 1 of extracting state estimates presented in [10] often generated Bernoulli components with existence probability equal to 1. To do a fair comparison with MT3v2, we cap all existence probabilities for both methods at a maximum of 0.99 instead, which prevents PMBM from obtaining too large of a penalty for some samples with false detections.

[7] We note that it is not possible to perform a comparison on datasets such as MOT17 and MOT20, since neither MT3v2, PMBM, nor δ-GLMB work with image inputs.

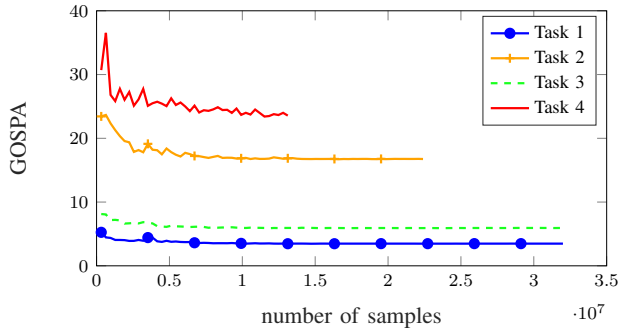

Fig. 6. GOSPA scores for MT3v2 during training for each of the four tasks. Performance improves steadily with more training, and most of the gains come from the first 1M training samples. Training on tasks 2 and 4 takes longer due to the more complicated measurement model, so fewer samples were processed in the same amount of allotted training time than for the other tasks.

and the GOSPA values during training for each of them are shown in figure 6. For all tasks, performance improved steadily with more training, and most of the gains in performance are obtained within the first 1M processed samples (30k gradient steps, 1-2 days of training time). The training continued for as long as possible within the computational constraints available, but more training in task 4 could have yielded better performance. Note that the GOSPA values for tasks 2 and 4 are considerably larger than for the other tasks primarily due to a higher average number of ground-truth objects to be predicted.

Once trained, we computed the average GOSPA and NLL scores over 1k samples for MT3v2 in each task. The resulting scores, together with those for the benchmark algorithms, are shown in Table II and III, along with the corresponding decompositions and 95% confidence intervals. NLL scores for δ-GLMB are not shown in Table III because, in the GLMB density representation, each global hypothesis consists of Bernoulli components with deterministic existence probabilities (either 0 or 1). After extracting the most likely global hypothesis, the resulting type of multi-object posterior assigns zero probability to any set of object states with cardinality larger than the number of Bernoulli components, incurring a penalty of infinity whenever false detections occur (all tasks). We refer to [51] for an in-depth explanation of this behavior.

In terms of GOSPA, we see that MT3v2 is able to match performance with the best benchmark in the simpler setting, task 1, while outperforming it in tasks 2, 3, and 4, supporting our hypothesis that DL trackers can outperform traditional model-based methods when the data association becomes more complicated and/or the models more nonlinear. In terms of NLL, the conclusion is similar, with the exception of task 2 where MT3v2 has similar performance to PMBM. Our results agree with a growing body of literature showing that performing estimation via a direct mapping from the sequence of measurements to the object states often yields better performance than recursively estimating the state at every time-step [64]–[66]. In a way, our approach is similar to

TABLE II

GOSPA scores for all tasks.

Task	Algorithm	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	**22.67 ± 0.54**	10.80	0.99	10.88

TABLE III

NLL scores for all tasks.

Task	Algorithm	NLL	Localization	False	Missed
1	PMBM	**1.78 ± 0.33**	1.24	0.14	0.39
	MT3v2	6.49 ± 0.35	5.76	0.25	0.48
2	PMBM	**31.40 ± 1.00**	23.57	1.47	6.35
	MT3v2	36.00 ± 0.94	29.75	2.62	3.62
3	PMBM	22.39 ± 1.01	10.85	2.39	9.16
	MT3v2	**12.90 ± 0.54**	10.80	0.66	1.43
4	PMBM	55.21 ± 1.49	25.23	1.18	28.79
	MT3v2	**45.55± 0.98**	18.55	5.49	21.52

those approaches, as we are learning a parametric function mapping the measurement sequence directly to the final object states.

Additionally, for most of the tasks, we notice that the localization error for MT3v2 (in both GOSPA and NLL) is higher than for the benchmarks, suggesting that the regression part of the network could be further improved. We theorize that further training plus predicting full state covariances (we only predict diagonal covariance matrices, see section V-C) would improve this, but leave it for future work. At the same time, we note the higher GOSPA-missed and NLL-missed cost for PMBM and δ-GLMB in almost all tasks, the gap to MT3v2 being the largest for task 4. As expected, the increase in the data association complexity for these tasks requires traditional approaches to aggressively prune hypotheses to remain computationally tractable, leading to more missed targets and worse performance.

To further investigate the high missed costs for the Bayesian benchmarks, we provide a failure case study of the algorithms in Task 4. To do so, we divide the (r, θ) dimensions of the FOV into 25 different sectors, and plot the missed ratio (number of missed objects / total number of objects) in each sector for tasks 2 and 3, along with those of MT3v2, in Figures 7 through 9. The missed rates for MT3v2 in task 2 are very similar to PMBM, so we omit it for conciseness.

Figure 7 shows that in task 2 the tracker's missed ratios are reasonably constant throughout the FOV, in

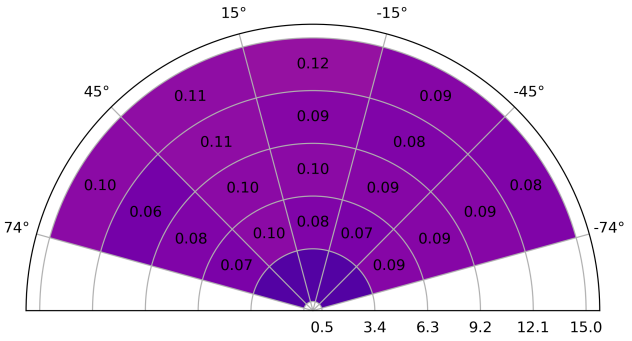

Fig. 7. PMBM's missed rate per FOV sector for task 2 (very similar to MT3v2 and δ-GLMB).

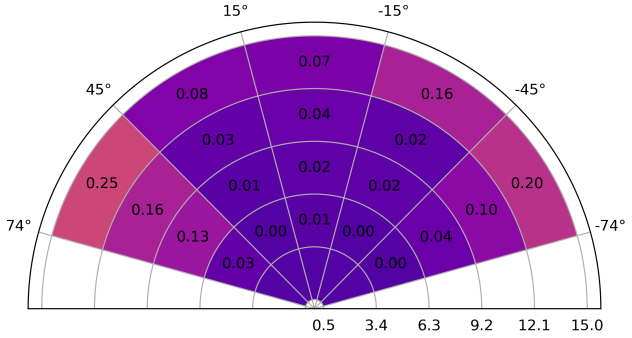

Fig. 8. MT3v2's missed rate per FOV sector for task 3.

line with our expectations, since the data association complexity is increased across the entire FOV, with no specific region being more or less complex than others. In task 3 on the other hand, Figs. 8 and 9 show that PMBM's (and δ-GLMB's) performance is considerably worse than MT3v2's for objects closer to the edges of the FOV. In these regions, the measurement model becomes highly non-linear, with the state-dependent measurement noise covariance $\mathbf{\Sigma}(\mathbf{x})$ (Section VI-A) increasing rapidly. We hypothesize that the Gaussian approximations in PMBM are not accurate enough in the presence of such strong non-linearities, and thus negatively impact the tracker's performance. In task 4 both of these changes (increased data association complexity and model nonlinearities) affect PMBM's performance, explaining its very high GOSPA and NLL's missed costs. In contrast, MT3v2's missed costs for all tasks are lower than for both benchmarks, especially in task 4 (3.94 vs 28.79 NLL-missed costs), suggesting that DL-based trackers indeed handle these challenges in a better way than traditional model-based approaches.

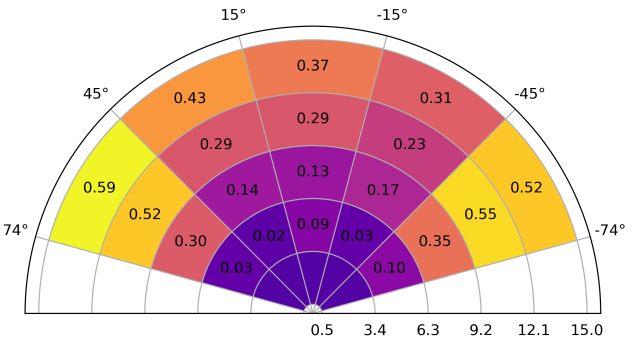

Fig. 9. PMBM's missed rate per FOV sector for task 3 (very similar to δ-GLMB).

C. Ablation Studies

In order to validate and quantify the importance of each of the most important design decisions in MT3v2, we ran extensive ablation tests evaluating all possible combinations of these variables. We altered four features in the ablations, referred to in Table IV as 1, 2, 3, 4, corresponding respectively to whether the training had:

1) Intermediate decoder losses: the additional loss components $f^l(\mathbf{x}_{1:m})$ for all $l \neq M$ in (22) (set to zero to turn off this feature).
2) Contrastive auxiliary loss: section V-E (we set β to zero to turn off this feature)
3) Selection mechanism: section V-B (the object queries $\mathbf{o}_{1:k}$, positional encodings $\mathbf{q}^d_{1:k}$, and initial estimates $\tilde{\mathbf{z}}_{1:k}$ in (16) and (17) are treated as additional parameters of MT3v2 and trained via gradient descent to turn off this feature).
4) Iterative refinement: section V-C (the predicted state means in (18) for each decoder layer are computed directly from that layer's output as $\mu^l_i = \text{FFN}(\mathbf{o}^{(l)}_{1:k})$ to turn off this feature).

With the exception of the changes described in the table, all models are identical in structure, hyperparameters, and initial weights, and were trained on the same data.

The average GOSPA for each ablation was computed using 1k samples from the models described for task 4 (all models were evaluated using the same data to reduce variance) and is shown in Table IV along with decompositions and 95% confidence intervals for the means. We provide all the trained models for these ablations in https://github.com/JulianoLagana/MT3v2.

TABLE IV
Ablation GOSPA scores for task 4.

1	2	3	4	**GOSPA**	Difference
				27.02 ± 0.60	+3.49
			✓	30.15 ± 0.76	+6.62
		✓		27.05 ± 0.61	+3.52
		✓	✓	28.07 ± 0.70	+4.54
	✓			29.11 ± 0.73	+5.58
	✓		✓	25.42 ± 0.58	+1.89
	✓	✓		26.68 ± 0.59	+3.15
	✓	✓	✓	28.36 ± 0.61	+4.83
✓				25.22 ± 0.58	+1.69
✓			✓	24.67 ± 0.59	+1.14
✓		✓		25.84 ± 0.60	+2.31
✓		✓	✓	24.13 ± 0.54	+0.6
✓	✓			25.86 ± 0.59	+2.33
✓	✓		✓	24.04 ± 0.57	+0.51
✓	✓	✓		25.89 ± 0.59	+2.36
✓	✓	✓	✓	23.53 ± 0.56	0.00

TABLE V
Complexity evaluation for each algorithm.

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

The results show that all the design decisions in MT3v2 are important for its performance: removing any of them results in worse GOSPA for this task. The most important design decision is training with intermediate decoder losses (1), which is known to be a design decision of great importance when training deep Transformer decoders [21], [52]. Putting (1) aside, the most impactful ablation is not performing iterative refinement (4). This is possibly due to the fact that Task 4 has the lowest signal-to-noise ratio of all of the tasks, making it challenging to learn to predict the object states from scratch instead of refining them from initial estimates. Interestingly, we also note that although performance is best when all proposed features are present, in isolation none of them improve performance.

D. Complexity Evaluation

As a further comparison of the algorithms considered, this section describes the inference times[8] for MT3v2, PMBM, and δ-GLMB in each of the 4 tasks from Section VI-A. MT3v2 was run on a V100 GPU, and PMBM and δ-GLMB on 32 Intel Xeon Gold 6130 CPUs. The average inference times are shown in Table V.

From the table, one can see that MT3v2's inference is orders of magnitude faster than the traditional methods during inference, and can directly be used for real-time tracking in many contexts. Additionally, inference times for it scale considerably better than for the benchmarks when increasing the complexity of the task (MT3v2 is more than 5000 times faster than the benchmarks in the most complicated task), highlighting another advantage of DL-based Transformer approaches. However, we also note that this comparison between approaches is far from perfect.

First, it is complicated to compare inference times between MT3v2 and the benchmarks, because these approaches are fundamentally different. MT3v2 is based on Transformers and deep learning and therefore benefits greatly from parallelization and specific hardware (such as GPUs) that has been perfected over recent years to increase inference and training speed. On the other hand, traditional Bayesian methods such as the ones we compare to rely on processing each time step in the sequence of measurements sequentially, therefore being harder to parallelize; benefitting more from faster CPUs instead.

Second, deep learning methods require training before they can be used for inference, which as noted previously can take a considerable amount of time. As shown in the table, MT3v2 requires 4 days of training in a task before being able to produce estimates, whereas the model-based benchmarks do not require any training.

Third, MT3v2 and the Bayesian methods were run on different hardware and implemented using different software, as described in Section VI-B. Our hardware choices were based on the available resources from C3SE, while our software choices mostly reflect what other open-source implementations used, rather than what would be optimal for each of the methods.

Nevertheless, the difference between inference times is so significant that we deemed it worth mentioning, even if the comparison is not perfect. We expect that even in the case that considerable effort is dedicated to speeding up the benchmarks, DL-based approaches that process the measurement sequence in parallel will continue to be more efficient, especially in challenging tasks.

VIII. CONCLUSION

In this paper, we propose what is to the best of our knowledge the first thorough comparison between DL trackers and Bayesian model-based filters in the model-based setting. To do so, we design a DL tracker specifically tailored for this setting, based on the Transformer architecture and with multiple improvements specific to model-based MOT.

Our results are the first to show two important findings. First, that deep learning trackers can match the performance of Bayesian algorithms in simple tasks, even though their performance is close to Bayes-optimal. Second, that DL can outperform these traditional approaches when the tracking task becomes more complicated, either due to increased complexity in the data association or stronger non-linearities in the models of the environment. This provides strong evidence for the applicability of deep learning to the multi-object tracking problem also in the model-based regime.

Interesting possibilities for future work are: (1) Adding more flexibility to the families of densities predicted by MT3v2 (e.g. Bernoulli components with non-diagonal covariances, more complicated state densities, normalizing flows [67], etc.); (2) Adapting MT3v2 to settings where trajectory estimates are needed; (3) Extending MT3v2 to work with higher-dimensional measurements

[8]The time required to process a complete sequence of measurements $\mathbf{z}_{1:n}$ and generate a predicted posterior density for $\mathbb{X}^T$.

such as images, and comparing it to existing model-free MOT methods in datasets such as MOT20 [68].

Appendix A
OFDM PARAMETERS

The parameters for the realistic RADAR model used as the measurement model for tasks 3 and 4 (see Section VI-A) are described in detail in Table VI, based on [69].

TABLE VI
OFDM parameters for realistic RADAR model

Parameter name	**Value**
Transmission power	0 dBm
Carrier frequency	140 GHz
Noise power spectral density	-174 dBm/Hz
Total bandwidth	2 GHz
Number of subcarriers	1000
Subcarrier spacing	2 MHz
Radar cross-section	0.1 m^2
Receiver noise figure	10 dB
Number of receive antennas	20
Number of OFDM symbols	2048
Cyclic prefix overhead	7%

Appendix B
Varying GOSPA cutoff parameter

The cutoff parameter c in GOSPA's definition (see (30)) determines the cost of missed and false detections in relation to the localization costs. This choice is nontrivial, and should reflect the design constraints of the application that tracking will be used for. Although in this paper we chose $c = 10$ for all experiments, we also ran an additional experiment evaluating GOSPA for all algorithms at different values of c for task 3. The results are shown in Fig. 10.

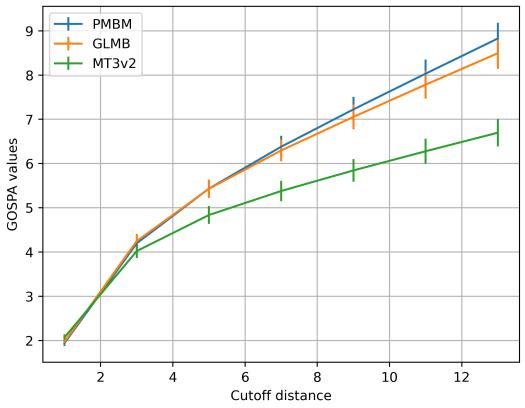

Fig. 10. GOSPA scores and 95% confidence intervals for task 3 at different cutoff levels.

The plot shows that all algorithms obtain similar performance at lower cutoff values, because they all have low localization errors (see II). However, as we increase the cutoff value, MT3v2 becomes clearly better than the benchmarks, primarily because of its lower false and missed rates (which receive higher penalties as c increases). We expect this to approximately hold in the other tasks as well, with MT3v2 becoming the clear winner as the cutoff values are increased (due to its significantly lower missed rates).

REFERENCES

[1] E. Itskovits, A. Levine, E. Cohen, and A. Zaslaver, "A multi-animal tracker for studying complex behaviors," *BMC Biology*, vol. 15, no. 1, p. 29, Apr 2017.

[2] Y.-C. Yoon, D. Y. Kim, Y.-M. Song, K. Yoon, and M. Jeon, "Online multiple pedestrians tracking using deep temporal appearance matching association," *Information Sciences*, vol. 561, pp. 326–351, 2021.

[3] A. Rangesh and M. M. Trivedi, "No blind spots: Full-surround multi-object tracking for autonomous vehicles using cameras and lidars," *IEEE Transactions on Intelligent Vehicles*, vol. 4, no. 4, pp. 588–599, 2019.

[4] D. Walther, D. R. Edgington, and C. Koch, "Detection and tracking of objects in underwater video," in *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, vol. 1, 2004.

[5] C. J. Harris, A. Bailey, and T. Dodd, "Multi-sensor data fusion in defence and aerospace," *The Aeronautical Journal (1968)*, vol. 102, no. 1015, p. 229–244, 1998.

[6] P. Dendorfer, A. Osep, A. Milan, K. Schindler, D. Cremers, I. Reid, S. Roth, and L. Leal-Taixé, "MOTchallenge: A benchmark for single-camera multiple target tracking," *International Journal of Computer Vision*, pp. 1–37, 2020.

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

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

[9] R. P. S. Mahler, *Statistical Multisource-Multitarget Information Fusion*. USA: Artech House, Inc., 2007.

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

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

[12] J. Pinto, G. Hess, W. Ljungbergh, Y. Xia, L. Svensson, and H. Wymeersch, "Next generation multitarget trackers: Random finite set methods vs transformer-based deep learning," in *24th International Conference on Information Fusion (FUSION)*. IEEE, 2021, pp. 1–8.

[13] D. Musicki and R. Evans, "Joint integrated probabilistic data association: JIPDA," *IEEE transactions on Aerospace and Electronic Systems*, vol. 40, no. 3, pp. 1093–1099, 2004.

[14] S. S. Blackman, "Multiple hypothesis tracking for multiple target tracking," *IEEE Aerospace and Electronic Systems Magazine*, vol. 19, no. 1, pp. 5–18, 2004.

[15] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," in *NIPS*, 2017, pp. 5998–6008.

[16] R. P. Mahler, *Advances in statistical multisource-multitarget information fusion*. Artech House, 2014.

[17] A. W. Senior, R. Evans, J. Jumper, J. Kirkpatrick, L. Sifre, T. Green, C. Qin, A. Žídek, A. W. Nelson, A. Bridgland *et al.*, "Improved protein structure prediction using potentials from deep learning," *Nature*, vol. 577, no. 7792, pp. 706–710, 2020.

[18] S. Karita, N. Chen, T. Hayashi, T. Hori, H. Inaguma, Z. Jiang, M. Someki, N. E. Y. Soplin, R. Yamamoto, X. Wang, S. Watanabe, T. Yoshimura, and W. Zhang, "A comparative study on transformer vs rnn in speech applications," in *IEEE Automatic Speech Recognition and Understanding Workshop (ASRU)*, 2019, pp. 449–456.

[19] N. Parmar, A. Vaswani, J. Uszkoreit, L. Kaiser, N. Shazeer, A. Ku, and D. Tran, "Image transformer," in *International Conference on Machine Learning*. PMLR, 2018, pp. 4055–4064.

[20] L. J. Ba, J. R. Kiros, and G. E. Hinton, "Layer normalization," in *Neural Information Processing Systems Deep Learning Symposium*, 2016.

[21] N. Carion, F. Massa, G. Synnaeve, N. Usunier, A. Kirillov, and S. Zagoruyko, "End-to-end object detection with transformers," in *European Conference on Computer Vision*, vol. 12346. Springer, 2020, pp. 213–229.

[22] D. R. So, Q. V. Le, and C. Liang, "The evolved transformer," in *ICML*, ser. Proceedings of Machine Learning Research, vol. 97. PMLR, 2019, pp. 5877–5886.

[23] H. Le, J. Pino, C. Wang, J. Gu, D. Schwab, and L. Besacier, "Dual-decoder transformer for joint automatic speech recognition and multilingual speech translation," in *COLING*. International Committee on Computational Linguistics, 2020, pp. 3520–3533.

[24] G. Ciaparrone, F. L. Sánchez, S. Tabik, L. Troiano, R. Tagliaferri, and F. Herrera, "Deep learning in video multi-object tracking: A survey," *Neurocomputing*, vol. 381, pp. 61–88, 2020.

[25] C.-Y. Chong, "An overview of machine learning methods for multiple target tracking," in *2021 IEEE 24th International Conference on Information Fusion (FUSION)*, 2021, pp. 1–9.

[26] S. Ren, K. He, R. B. Girshick, and J. Sun, "Faster R-CNN: towards real-time object detection with region proposal networks," in *NIPS*, 2015, pp. 91–99.

[27] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 779–788.

[28] F. Yu, W. Li, Q. Li, Y. Liu, X. Shi, and J. Yan, "Poi: Multiple object tracking with high performance detection and appearance feature," in *European Conference on Computer Vision*. Springer, 2016, pp. 36–42.

[29] J. Shen, D. Yu, L. Deng, and X. Dong, "Fast online tracking with detection refinement," *IEEE Transactions on Intelligent Transportation Systems*, vol. 19, no. 1, pp. 162–173, 2018.

[30] J. Chen, H. Sheng, Y. Zhang, and Z. Xiong, "Enhancing detection model for multiple hypothesis tracking," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, 2017, pp. 18–27.

[31] J. Shen, Z. Liang, J. Liu, H. Sun, L. Shao, and D. Tao, "Multiobject tracking by submodular optimization," *IEEE Transactions on Cybernetics*, vol. 49, no. 6, pp. 1990–2001, 2019.

[32] Y. Zhang, P. Sun, Y. Jiang, D. Yu, F. Weng, Z. Yuan, P. Luo, W. Liu, and X. Wang, "Bytetrack: Multi-object tracking by associating every detection box," in *ECCV (22)*, ser. Lecture Notes in Computer Science, vol. 13682. Springer, 2022, pp. 1–21.

[33] A. Milan, S. H. Rezatofighi, A. Dick, I. Reid, and K. Schindler, "Online multi-target tracking using recurrent neural networks," in *Thirty-First AAAI Conference on Artificial Intelligence*, 2017.

[34] H. Zhou, W. Ouyang, J. Cheng, X. Wang, and H. Li, "Deep continuous conditional random fields with asymmetric inter-object constraints for online multi-object tracking," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 29, no. 4, pp. 1011–1022, 2018.

[35] J. Yin, W. Wang, Q. Meng, R. Yang, and J. Shen, "A unified object motion and affinity model for online multi-object tracking," in *CVPR*. Computer Vision Foundation / IEEE, 2020, pp. 6767–6776.

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

[37] Y. Zhang, C. Wang, X. Wang, W. Zeng, and W. Liu, "Fairmot: On the fairness of detection and re-identification in multiple object tracking," *International Journal of Computer Vision*, pp. 1–19, 2021.

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

[39] X. Weng, Y. Wang, Y. Man, and K. M. Kitani, "Gnn3dmot: Graph neural network for 3d multi-object tracking with 2d-3d multi-feature learning," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 6499–6508.

[40] J. Li, X. Gao, and T. Jiang, "Graph networks for multiple object tracking," in *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, 2020, pp. 719–728.

[41] T. Meinhardt, A. Kirillov, L. Leal-Taixe, and C. Feichtenhofer, "Trackformer: Multi-object tracking with transformers," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022, pp. 8844–8854.

[42] F. Zeng, B. Dong, Y. Zhang, T. Wang, X. Zhang, and Y. Wei, "MOTR: End-to-end Multiple-Object tracking with tRansformer," in *European Conference on Computer Vision (ECCV)*, 2022.

[43] P. Sun, Y. Jiang, R. Zhang, E. Xie, J. Cao, X. Hu, T. Kong, Z. Yuan, C. Wang, and P. Luo, "Transtrack: Multiple-object tracking with transformer," *arXiv preprint arXiv:2012.15460*, 2020.

[44] J. Cai, M. Xu, W. Li, Y. Xiong, W. Xia, Z. Tu, and S. Soatto, "Memot: Multi-object tracking with memory," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2022, pp. 8090–8100.

[45] Y. Liu, T. Bai, Y. Tian, Y. Wang, J. Wang, X. Wang, and F.-Y. Wang, "Segdq: Segmentation assisted multi-object tracking with dynamic query-based transformers," *Neurocomputing*, vol. 481, pp. 91–101, 2022.

[46] G. Maggiolino, A. Ahmad, J. Cao, and K. Kitani, "Deep OC-SORT: multi-pedestrian tracking by adaptive re-identification," *CoRR*, vol. abs/2302.11813, 2023.

[47] Y. Du, Y. Song, B. Yang, and Y. Zhao, "Strongsort: Make deepsort great again," *CoRR*, vol. abs/2202.13514, 2022.

[48] J. Hyun, M. Kang, D. Wee, and D. Yeung, "Detection recovery in online multi-object tracking with sparse graph tracker," in *WACV*. IEEE, 2023, pp. 4839–4848.

[49] X. Zhu, W. Su, L. Lu, B. Li, X. Wang, and J. Dai, "Deformable DETR: Deformable transformers for end-to-end object detection," in *International Conference on Learning Representations*, 2021.

[50] Z. Teed and J. Deng, "RAFT: Recurrent all-pairs field transforms for optical flow," in *European Conference on Computer Vision*, vol. 12347. Springer, 2020, pp. 402–419.

[51] J. Pinto, Y. Xia, L. Svensson, and H. Wymeersch, "An uncertainty-aware performance measure for multi-object tracking," *IEEE Signal Processing Letters*, vol. 28, no. 1689–1693, 2021.

[52] R. Al-Rfou, D. Choe, N. Constant, M. Guo, and L. Jones, "Character-level language modeling with deeper self-attention," in *AAAI Conference on Artificial Intelligence*, 2019, pp. 3159–3166.

[53] D. F. Crouse, "On implementing 2D rectangular assignment algorithms," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 52, no. 4, pp. 1679–1696, 2016.

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

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

[56] P. Khosla, P. Teterwak, C. Wang, A. Sarna, Y. Tian, P. Isola, A. Maschinot, C. Liu, and D. Krishnan, "Supervised contrastive learning," in *Advances in Neural Information Processing Systems*, vol. 33. Curran Associates, Inc., 2020, pp. 18 661–18 673.
[57] Z. Abu-Shaban, X. Zhou, T. Abhayapala, G. Seco-Granados, and H. Wymeersch, "Error bounds for uplink and downlink 3D localization in 5G millimeter wave systems," *IEEE Transactions on Wireless Communications*, vol. 17, no. 8, pp. 4939–4954, Aug. 2018.
[58] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *International Conference on Learning Representations*, 2015.
[59] Á. F. García-Fernández, L. Svensson, M. R. Morelande, and S. Särkkä, "Posterior linearization filter: Principles and implementation using sigma points," *IEEE Transactions on Signal Processing*, vol. 63, no. 20, pp. 5561–5573, 2015.
[60] Á. F. García-Fernández, J. Ralph, P. Horridge, and S. Maskell, "A Gaussian filtering method for multitarget tracking with nonlinear/non-Gaussian measurements," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 57, no. 5, pp. 3539–3548, 2021.
[61] I. Arasaratnam and S. Haykin, "Cubature kalman filters," *IEEE Transactions on Automatic Control*, vol. 54, no. 6, pp. 1254–1269, 2009.
[62] D. Crouse, "Basic tracking using nonlinear 3D monostatic and bistatic measurements," *IEEE Aerospace and Electronic Systems Magazine*, vol. 29, no. 8, pp. 4–53, 2014.
[63] A. S. Rahmathullah, Á. F. García-Fernández, and L. Svensson, "Generalized optimal sub-pattern assignment metric," in *IEEE International Conference on Information Fusion (Fusion)*, 2017.
[64] T. Li, H. Chen, S. Sun, and J. M. Corchado, "Joint smoothing and tracking based on continuous-time target trajectory function fitting," *IEEE Transactions on Automation Science and Engineering*, vol. 16, no. 3, pp. 1476–1483, 2019.
[65] J. Zhou, T. Li, and X. Wang, "State estimation with linear equality constraints based on trajectory function of time and karush-kuhn-tucker conditions," in *2021 International Conference on Control, Automation and Information Sciences (ICCAIS)*, 2021, pp. 438–443.
[66] L. Chen, "From labels to tracks: it's complicated," in *Signal Processing, Sensor/Information Fusion, and Target Recognition XXVII*, vol. 10646. SPIE, 2018, pp. 8–13.
[67] D. Rezende and S. Mohamed, "Variational inference with normalizing flows," in *International Conference on Machine Learning*. PMLR, 2015, pp. 1530–1538.
[68] P. Dendorfer, H. Rezatofighi, A. Milan, J. Shi, D. Cremers, I. D. Reid, S. Roth, K. Schindler, and L. Leal-Taixé, "MOT20: A benchmark for multi object tracking in crowded scenes," *CoRR*, vol. abs/2003.09003, 2020.
[69] H. Wymeersch *et al.*, "Localisation and sensing use cases and gap analysis," Deliverable 3.1, 2022. [Online]. Available: https://hexa-x.eu/deliverables/

Juliano T. A. L. Pinto was born in Brazil in 1990, where he received his B.Sc. degree in mechatronics from Instituto Mauá de Tecnologia. After moving to Sweden, he received his M.Sc. degree in Systems, Control, and Mechatronics from Chalmers University of Technology, where he is currently a PhD student. His interests include reinforcement learning, supervised learning, and multiple object tracking.

Georg Hess is an industrial PhD student at Chalmers Univ. of Tech. and at Zenseact, an autonomous driving company. He received the B.S. in mechanical engineering (2018) from Chalmers Univ. of Tech., B.S. in business administration (2019) from Univ. of Gothenburg and the M.S. in electrical engineering (2021) from Chalmers Univ. of Tech. His research is focused on perception for autonomous driving using deep learning, with an emphasis on object detection and tracking.

William Ljungbergh received the B.Sc. in Mechanical Engineering and the M.Sc. degree in Systems, Control, and Mechatronics from Chalmers University of Technology in 2018 and 2021. He is currently persuing a Ph.D degree in Computer Vision in an industrial collaboration between Linköping University and Zenseact. His research interests include computer vision and deep learning with applications in autonomous driving. The focus of his Ph.D. is on vision-based motion forecasting for autonomous vehicles.

Yuxuan Xia received the M.Sc. degree in communication engineering and the Ph.D. degree in signal processing from the Chalmers University of Technology, Gothenburg, Sweden, in 2017 and 2022, respectively. He is currently a Postdoctoral Researcher with the Department of Electrical Engineering, Chalmers University of Technology. His main research interests include multi-object tracking and sensor fusion, especially for extended objects. He has co-organized tutorials on multi-object tracking at the Fusion 2020, 2021 and 2022 conferences.

Henk Wymeersch obtained the Ph.D. degree in Electrical Engineering/Applied Sciences in 2005 from Ghent University, Belgium. He is currently a Professor of Communication Systems with the Department of Electrical Engineering at Chalmers University of Technology, Sweden. He is also a Distinguished Research Associate with Eindhoven University of Technology. Prior to joining Chalmers, he was a postdoctoral researcher from 2005 until 2009 with the Laboratory for Information and Decision Systems at the Massachusetts Institute of Technology. Prof. Wymeersch served as Associate Editor for IEEE Communication Letters (2009-2013), IEEE Transactions on Wireless Communications (since 2013), and IEEE Transactions on Communications (2016-2018). During 2019-2021, he is a IEEE Distinguished Lecturer with the Vehicular Technology Society. His current research interests include the convergence of communication and sensing, in a 5G and Beyond 5G context.

Lennart Svensson was born in Älvängen, Sweden in 1976. He received the M.S. degree in electrical engineering in 1999 and the Ph.D. degree in 2004, both from Chalmers University of Technology, Gothenburg, Sweden. He is currently Professor of Signal Processing, again at Chalmers University of Technology. His main research interests include machine learning and Bayesian inference in general, and nonlinear filtering, deep learning and multi-object tracking in particular.