Crafted vs. Learned Representations in Predictive Models - A Case Study on Cyclist Path Prediction

Pool, Ewoud; Kooij, Julian F.P.; Gavrila, Dariu M.

DOI
10.1109/TIV.2021.3064253
Publication date
2021
Document Version
Accepted author manuscript
Published in
IEEE Transactions on Intelligent Vehicles

Crafted vs. Learned Representations in Predictive Models
A Case Study on Cyclist Path Prediction

Ewoud A. I. Pool [1], Julian F. P. Kooij [1] and Dariu M. Gavrila [1]

***Abstract*—This paper compares two models for context-based path prediction of objects with switching dynamics: a Dynamic Bayesian Network (DBN) and a Recurrent Neural Network (RNN). These models are instances of two larger model categories, distinguished by whether expert knowledge is explicitly crafted into the state representation (and thus is interpretable) or whether the representation is learned from data, respectively. They have shown state-of-the-art performance in previous work.**

In order to provide a fair comparison, we ensure that both models are treated similarly with respect to the use of context cues and parameter estimation. Specifically, we describe (1) how to integrate the context cues (used previously by the DBN) into the RNN, and (2) how to optimize the DBN with back-propagation similar to the RNN, while keeping an interpretable state representation.

Experiments are performed on a scenario where a cyclist might turn left at an intersection in front of the ego-vehicle. Results show that the RNN successfully leverages the context cues, and that optimizing the DBN improves its performance with respect to existing work. While the RNN outperforms the optimized DBN in predictive log-likelihood by a significant margin, both models attain similar average Euclidean distance errors (23–39 cm for DBN and 31–34 cm for RNN, predicting 1 s ahead).

***Index Terms*—Active safety, vulnerable road users (VRUs), motion prediction**

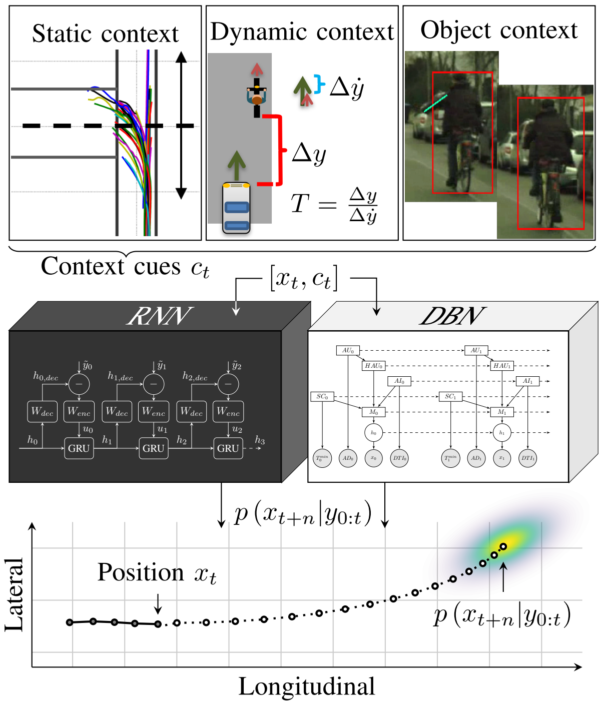


Fig. 1: Context-based cyclist path prediction with a RNN ("black box", i.e. learned representation) and a DBN ("white box", i.e. crafted representation). The context cues are: distance to the intersection (static context), time until the ego-vehicle overtakes (dynamic context), and a possible arm gesture (object context). Predictions involve distributions over future cyclist positions.

I. Introduction

Vehicle environment perception has made great strides over the past years, largely thanks to advances in neural networks such as Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs). These data-driven approaches learn an optimized state representation, rather than requiring a (hand) crafted state representation. At the core of all neural network methods is gradient descent-based optimization.

The prediction of the future path of Vulnerable Road Users (VRUs) (e.g. pedestrians, cyclists, and other riders) is a challenging remaining problem for vehicle environment perception, due to their high manoeuvrability. Context information, such as body gestures, road lay-out or the vicinity of other road users have been shown to improve the accuracy of path prediction compared to only using point target kinematics (e.g. [1]). Methods with learned representations have shown state of the art performance in specific scenarios.

The downside of purely data-driven approaches is they do not provide an intuitive explanation of their output: the learned state representation essentially renders them black-box models. The lack of interpretability complicates understanding why they fail when they do, which is disadvantageous for safety-critical domains such as intelligent vehicles. To counter this, a field of study emerged to make the reasoning of neural networks explicit (e.g. [2]), but this remains an open challenge [3]. The lack of interpretability is especially noticeable with path prediction, where the temporal aspect implies causality: a cyclist is predicted to turn left because of the outstretched arm, a pedestrian is predicted to cross the street because he failed to see the approaching vehicle, etc.

On the other hand, models with (hand) crafted state representations such as Dynamic Bayesian Networks (DBNs) [1] can capture causal relationships explicitly and are popular for interpretable probabilistic VRU path prediction. However, as their crafted representations are an abstraction of the real world, they might not encode all the useful information that is available in the data. Additionally, the parameters for these

1) Intelligent Vehicles group, TU Delft, The Netherlands

methods are often not optimized, but instead individually estimated from ground truth annotations (e.g. [1]) or tuned manually (e.g. [4]). Estimating parameters individually does not necessarily optimize the predictive performance of the complete model directly. Next to that, the additional context ground truth labeling is a timely investment that is not required for a learned state representation.

In this paper, we compare the context-based path prediction performance of a model with a learned state representation, an RNN, to that of a model with a crafted state representation, a DBN (fig. 1). For the comparison, we level the playing field in two ways with respect to the state of the art. First, we provide an RNN which can incorporate the context cues effectively, similar to the DBN. Second, we show that we can employ the same optimization strategy on the DBN as we employ for the RNN, namely gradient descent, while ensuring that the meaning of its crafted state representation is not lost.

II. Previous Work

VRU path prediction has attracted great attention in the previous decade, see recent surveys [5], [6]. Path prediction methods require VRU positions as input. Ground plane positions relative to a vehicle reference frame can be obtained from detections in various sensors (camera [7], radar [8], LiDAR [9]). If ground plane positions relative to a global reference frame are needed (e.g. this paper), then vehicle ego-motion compensation is necessary, as an additional preprocessing step. Following sub-sections describe various aspects of prediction methods.

A. Motion models

Two main categories of motion models are physics- and pattern-based [6]. In physics-based methods, motion is predicted by forward propagation of a set of explicit dynamics equations with a physical interpretation. This category contains the single-motion model case, as in Linear Dynamical Systems (e.g. plain Kalman filter) and extensions to the non-linear case (e.g. unscented/extended Kalman filter or particle filtering). This category also contains more advanced approaches with multiple motion models, either as a mixture [10] or with switching dynamics, e.g. Interacting Multiple Models (IMM). Context cues can guide the switch in dynamics, leading to a more general DBN [1], [11], [12].

Pattern-based methods instead derive predictions from previously seen data. One way of doing this is to match the current (partial) track to previously seen (complete) tracks in a database, and use the best matching exemplar for extrapolation [13]. An alternative is to perform non-linear regression by means of Gaussian Process Dynamic Models (GPDMs) [13], [14], Quantile Regression [15] or RNNs [16], [17], [18], [19], [19], [20], [21], [22], [23]. Popular instantiations of RNNs are Long Short Term Memory networks (LSTMs) and Gated Recurrent Units (GRUs). The latter uses fewer parameters as the former while it may keep a similar performance [24]. An RNN can predict not only a future state but also its uncertainty (e.g. Gaussian distribution [17], or similarly to an IMM filter, a mixture of Gaussians [18]). RNNs cannot inherently handle missing data (e.g. a frame where a VRU was not detected), and methods have been proposed to overcome this (e.g. [19]).

Some approaches blur the line between pure physics-based and pattern-based methods. Fraccaro *et al.* [20] model the dynamic latent state of an RNN with a Kalman filter, allowing them to use the exact inference, prediction, and smoothing of a Kalman filter for the dynamics. Li *et al.* [21] propose to make separate predictions with both a DBN and RNN, and fuse these afterwards in an online adaptive weighting scheme.

B. Context Cues

Object context cues are those that are directly linked to the object of interest, in addition to point target kinematics (positions, velocities). For example, Keller and Gavrila [13] use dense optical flow features to improve pedestrian path prediction. Xiong *et al.* [25] incorporate a learned feature representation of the VRU related cues, either through the feature representation of a re-identification network or through the last layer feature representation of the YOLO object detector [26]. Quintero *et al.* [14] recover a full 3D articulated pose of a pedestrian.

Static context cues refer to the influence of the world surrounding the VRU on their path. These are static effects such as an expectation on where VRUs plan to walk to [4], or the VRU's preference to traverse certain kinds of semantic areas (sidewalks, grass, zebra crossings, etc.). One way of implementing this is through Inverse Reinforcement Learning (IRL) [27], [28], or with neural networks [29]. Ballan *et al.* [30] learn preferred routes directly on top-down image data rather than on a semantic map and show that the learned knowledge is transferable to new locations. Saleh *et al.* [31] forego the need for a goal by using IRL only to learn the reward map of a static scene. Another approach is to directly encode the structure of the road ahead to limit the possible paths that the VRU can take [10], [32], or to predict the trajectory along the curvature of the road [33].

Dynamic context cues include whether the VRU is aware of his or her surroundings. Kooij *et al.* [1] incorporate both whether the vehicle and the pedestrian are on a collision course as well as the pedestrian's awareness thereof into a DBN to predict the future position of a pedestrian who might cross the road. Additionally, they show the same DBN structure can also be used to predict the future position of a cyclist who might turn left at a coming intersection. Neogi *et al.* [34] leverage the interaction between ego-vehicle and pedestrian for path prediction near an intersection as well. Other dynamic objects or VRUs can also influence the future path of VRUs. Social Force Models [22], [23] model the influence that nearby VRUs have on each other.

C. Parameter Estimation

Methods with learned state representations can optimize their parameters directly by performing gradient descent of an objective loss using training data. The main requirement is that this loss is differentiable. For example, it is possible to directly optimize the entire predicted trajectory [16] and all parameters at once [15]. The downside is that while the learned representation fits the data, it is not necessarily possible to interpret the hidden state of the learned representation. Being able to interpret *why* such a model predicts what it does is an active field, both in path prediction [19] as well as in detection [2]. Attentive neural networks [35] improve the interpretability of a neural network by forcing the network to make predictions on only a subset of all available information, such that the “attention” of the network points to specific areas or moments in time.

Methods with a crafted state representation on the other hand often explicitly fix certain parameters a priori which ensures that the latent state is interpretable. Kooij *et al.* [1] fix the dynamic models in a DBN to a constant-velocity model as well and estimate the other parameters for the context cues by annotating all context variables at each frame. A similar approach can be found in [21]. Hashimoto *et al.* [12] use a DBN and fix its dynamic model to be a constant-velocity model while optimizing the other parameters through maximum likelihood estimation. Batkovic *et al.* [4] specifically structure their model so the few parameters can be tuned by hand. If the goal is to optimize the DBN for estimating the current state (i.e. filtering) and the DBN only has discrete hidden variables, both the optimal parameters and structure can be computed [36]. If it has both discrete and continuous hidden variables, parameter optimization can be done by Expectation-Maximization (EM) [37] or gradient descent [38, p. 169].

D. Contributions

Our main contribution in this paper is a comparison of two state of the art models for context-based path prediction: one with a learned state representation, an RNN, and one with a crafted state representation, a DBN, at a level playing field. To ensure that both models are treated similarly with respect to the use of context cues and parameter estimation, we describe

- how to integrate the context cues (used previously by the DBN [1]) into an RNN [16][1], and conversely,
- how to optimize the DBN [1] with gradient descent by utilizing back-propagation (similarly to the RNN [16]), while keeping its state representation interpretable.

The comparison is made on a cyclist scenario. All relevant experimental data is made available to the scientific community for non-commercial benchmarking.[2]

III. Methodology

For the path prediction task, we consider models that predict at every time step t a probability distribution over the top-down 2D position x, n steps into the future, given all previous measurements $y_{0:t}$. A measurement $y_t = [x_t, c_t]$ contains the position x_t as well as multiple context cue measurements $c_t = [c_{1_t}, \dots, c_{N_{ct}}]$, where $c_t \in \mathbb{R}^{N_c}$. In general, the prediction task can be written as $p\left(x_{t+n}|y_{0:t}\right)$. This section covers the structure of the two approaches used to determine $p\left(x_{t+n}|y_{0:t}\right)$: one RNN-based (section III-A) and its training scheme (section III-B), and one DBN-based (section III-C) and its training scheme (section III-D).

[1] [16] is our earlier conference paper that this article builds upon.

[2] For the dataset, follow the links at www.intelligent-vehicles.org.

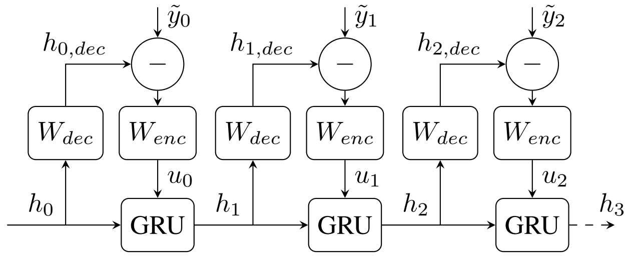


Fig. 2: The processesing of measurements over time by the RNN. This figure shows the incorporation of inputs over three time steps.

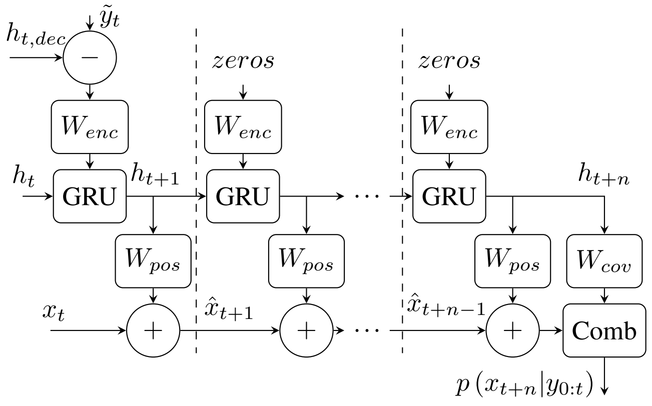


Fig. 3: The prediction of $p\left(x_{t+n}|y_{0:t}\right)$ at time step t by the RNN. The layers W_{enc} and W_{pos} are shared with the temporal update process (see fig. 2). The block labeled Comb is the combination of eqs. (4) to (9).

A. Recurrent Neural Network Model

For the RNN, the position is supplied as the difference in position between two time steps, $x_t - x_{t-1}$, as is done in [17]. The input for the RNN is then $\tilde{y}_t = [x_t - x_{t-1}, c_{1_t}, \dots, c_{N_{ct}}]^\top$. At t_0 the position difference is taken as zero. The architecture of the RNN can be split up into two parts: the first incorporates inputs $\tilde{y}_t$ into the hidden state over time (i.e. inference), and the second predicts a Gaussian distribution as the future trajectory based on the hidden state at a certain time step.

The first part, the inference architecture, is laid out schematically in fig. 2. The main component is a Gated Recurrent Unit (GRU), which is used because of its relatively low number of parameters. The hidden layer h_t, a vector with N_h elements, is decoded into an expected input, which is subtracted from the actual input, and the result u_t is fed into the GRU:

$$u_t = W_{enc}\left(\tilde{y}_t - W_{dec}(h_t)\right) \tag{1}$$

$$= W_{enc}\left(\begin{bmatrix} x_t - x_{t-1} \\ c_t \end{bmatrix} - \begin{bmatrix} W_{pos}(h_t) \\ W_{cues}(h_t) \end{bmatrix}\right), \tag{2}$$

where $W_{enc}(h_t) = w_{enc}h_t + b_{enc}$, a linear layer with w_{enc} and b_{enc} as trainable parameters. All other functions $W_{(\cdot)}()$ are linear layers as well, with parameters $w_{(\cdot)}$ and $b_{(\cdot)}$. The goal of the linear layers is solely to scale the internal representation of the GRU, and as such no nonlinear functions are added.

For prediction, the signal that is fed into the GRU is computed as:

$$u_t = W_{enc}\left(\mathbf{0}\right). \tag{3}$$

All future hidden states $h_{t+2}, \ldots, h_{t+n}$ are then computed as shown in fig. 3. The predicted Gaussian distribution over the future position $\mathcal{N}\left(\hat{x}_{t+n}, \Sigma_{t+n}\right)$ is computed as in [17]:

$$\hat{x}_{t+n} = x_t + \sum_{i=1}^{n} W_{pos}(h_{t+i}) \tag{4}$$

$$\begin{bmatrix} l^{[0]} & l^{[1]} & l^{[2]} \end{bmatrix}^\top = W_{cov}(h_{t+n}) \tag{5}$$

$$\sigma_1 = \exp\left(l^{[0]}\right) \tag{6}$$

$$\sigma_2 = \exp\left(l^{[1]}\right) \tag{7}$$

$$\rho = \tanh\left(l^{[2]}\right) \tag{8}$$

$$\Sigma_{t+n} = \begin{bmatrix} \sigma_1^2 & \rho\sigma_1\sigma_2 \\ \rho\sigma_1\sigma_2 & \sigma_2^2 \end{bmatrix}. \tag{9}$$

B. Recurrent Neural Network Training

The RNN is trained by minimizing the negative log-likelihood of the predicted Gaussian distribution on the known future position. To ensure that the output of each model is a consistent path the loss is averaged over each time step and the entire range of 1 time step up to and including n time steps ahead. The optimized parameters in the RNN are those of the GRU, the layers W_{enc}, W_{pos}, W_{cues}, W_{cov}, and h_0. Each of these parameters is initialized randomly using the default PyTorch strategy for such layers.

Two training strategies will be considered to reduce overfitting and improve convergence. Firstly, data normalization: The mean and variance of $\tilde{y}_t$ in the training data are computed. The input $\tilde{y}_t$ is scaled and translated accordingly before it is fed to the RNN. The inverse of the scaling and translation is applied to the output of each prediction step in eq. (4), i.e. $W_{pos}(h_{t+i})$. The covariance matrix predicted by the RNN is not scaled in any way. Secondly, during training, we reset the hidden state h_t back to the initial hidden state h_0 with a probability of 5% at every time step. This is to prevent the RNN from overfitting by recognizing a specific trajectory from just the first few measurements. Experiments showing the importance of these training strategies are given in section V-A2.

C. Dynamic Bayesian Network Model

This paper discusses the specific version of a DBN as described in [1], although the methodology can be used for alternative scenarios as well, e.g. [11]. At any time step t, the entire state of the DBN is defined by a partially observable continuous hidden state h_t and discrete hidden state $\mathcal{D}_t$. The discrete hidden state $\mathcal{D}_t = [M_t, C_{1_t}, C_{2_t} \ldots C_{N_C t}]$ specifies the current dynamic mode M_t as well as N_C discrete variables representing the state of the context cues. For a single time step, there are in total $|\mathcal{D}| = |M| \times |C_1| \times \cdots \times |C_{N_C}|$ possible combinations for the discrete state.

In the DBN, the discrete state at time $t = 0$ follows a categorical distribution $\mathcal{D}_0 \sim \text{Cat}\left(\mathcal{P}^0\right)$ with parameters $\mathcal{P}^0$, and can stochastically transition at subsequent time steps to a new value:

$$\mathcal{D}_t \sim \text{Cat}\left(\mathcal{P}^{(\mathcal{D}_{t-1})}\right). \tag{10}$$

Here, $\mathcal{P}^{(\mathcal{D}_{t-1})}$ is a $|\mathcal{D}|$-dimensional parameter vector conditioned on the past discrete state $\mathcal{D}_{t-1}$, i.e. the row from a $|\mathcal{D}| \times |\mathcal{D}|$ transition table. Of the N_C discrete variables C_{n_t}, N_c have corresponding measurements c_{n_t} and their probability distribution $p\left(c_{n_t}|C_{n_t}\right)$ is specific for that context cue.

The propagation of the continuous state h_t over time and the relation between the measurement x_t and the continuous state h_t are as follows:

$$h_t = A^{(M_t)} h_{t-1} + \epsilon_t, \qquad \epsilon_t \sim \mathcal{N}\left(\mu_\epsilon^{(M_t)}, Q^{(M_t)}\right) \tag{11}$$

$$x_t = C\, h_t + \eta_t, \qquad \eta_t \sim \mathcal{N}\left(0, R\right). \tag{12}$$

Similar to eq. (10), the superscript $^{(M_t)}$ indicates that there is a separate matrix/vector for each of the N_M models M_t. The matrices A and C are model parameters. Both the measurement and the state are perturbed by Gaussian noise that is not directly measurable, η and ϵ, respectively, with parameters μ_ϵ, Q, and R. Finally, the prior on the continuous state is normally distributed, $p\left(h_0\right) \sim \mathcal{N}\left(h_0, P_0\right)$ with parameters h_0 and P_0.

Inference and prediction with this model only apply matrix multiplications, inversions, and additions, so their gradient can be computed analytically (see appendix).

D. Dynamic Bayesian Network Training

The DBN is trained just as the RNN: by minimizing the negative log-likelihood of the predicted Gaussian distribution. The loss is again averaged over the entire range of 1 to n time steps ahead. During optimization, certain parameters are fixed such that the interpretability of the state is guaranteed. For example, assume the continuous measurements are the top-down 2D positions. If the first two items in the continuous state vector of length 4 should represent the 2D position, then fixing $C = \left[\begin{smallmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{smallmatrix}\right]$ during optimization ensures a correct state representation. Exact details for our case-study are given in section IV-B.

Additionally, to ensure that the covariance matrices are positive definite, they are reparameterized as upper-triangular matrices U during optimization, e.g. $Q = U^\top U$ [38, p. 169]. To improve numerical stability, all covariance matrices have a small epsilon 10^{-6} added to their diagonal.

The initial state distribution and the transition matrices for the discrete variables are also re-parameterized, using softmax functions [38, p. 169], since optimizing the values in the probability tables directly could result in invalid values. For example, a row $\mathcal{P}^{(\mathcal{D}_t)}$ from a probability table is re-parameterized with $|\mathcal{D}|$ learnable parameters $\tilde{\mathcal{P}}^{(\mathcal{D}_t)}$ as follows:

$$\mathcal{P}_i^{(\mathcal{D}_t)} = \frac{\exp\left(\tilde{\mathcal{P}}_i^{(\mathcal{D}_t)}\right)}{\sum_{j=1}^{|\mathcal{D}|} \exp\left(\tilde{\mathcal{P}}_j^{(\mathcal{D}_t)}\right)}. \tag{13}$$

Each parameter of the context measurement distributions $p\left(c_{n_t}|C_{n_t}\right)$ that has a limited domain can be re-parameterized as well. For example, the variance σ of a Gaussian can be kept positive by re-parameterizing it as $\sigma = \exp\left(\tilde{\sigma}\right)$.

For the initial value of all parameters, we can select a more reasonable initial estimate than random values specifically because each parameter and state variable has a certain interpretation assigned to it, unlike the RNN. For certain parameters this is done by defining them explicitly, such as the motion models $A^{(M_t)}$ and measurement model C. For the noise parameters of eqs. (11) and (12), the discrete measurement likelihoods $p(c_{n_t}|C_{n_t})$, and the discrete state transition probability $\mathcal{P}^{(\mathcal{D}_t)}$, there are two options. The first option, *annotation-based initialization* (as in [1]), is to estimate these using additional ground truth annotations for the discrete variables. Those annotations, together with the context measurements c_{n_t} are used to fit the $p(c_{n_t}|C_{n_t})$ distributions. The transition probability $\mathcal{P}^{(\mathcal{D}_t)}$ is estimated from the discrete state annotations as well. A downside is that annotating ground truth for these latent variables is laborious and often ambiguous. The second option, *annotation-free initialization*, is to forego the annotations and select initial values for the variables based on expert knowledge. This has become possible thanks to the optimization step afterward. The two options are described for our use case in sections IV-B2 and IV-B3.

IV. Case Study

We now describe the case study used to compare the RNN and DBN models. We first give an overview of the dataset, along with a description of the relevant context cues and their related measurements. We thereafter define the scenario-specific parts of the DBN: its crafted state representation, how it is trained and finally the two initial estimate strategies.

A. Dataset

The RNN and DBN from section III are trained and evaluated on the tracks from the cyclist scenario used for the original DBN [1]. This dataset contains 51 tracks of a cyclist approaching an intersection at a steady pace. These are recorded with a stereo-camera setup at 16 fps from a moving vehicle that drives behind the cyclist, resulting in 5744 frames total. There are no other traffic participants nearby. The cyclist is instructed beforehand to either raise their arm or not, and then either turn left or continue straight at the intersection.

The dataset contains the longitudinal and lateral position of the cyclist in a global reference frame, as well as measurements on the three context cues shown schematically in fig. 1. The first is distance to intersection (DTI), the distance between the cyclist and the intersection along the longitudinal axis of the road. The second, T^{min}, is the time it takes for the vehicle to overtake the cyclist if they would both keep moving with the same velocity. The third, Arm Detector (AD), indicates whether the arm of the cyclist is raised. This is given as a confidence score as computed by a Naive Bayes classifier.

The tracks are divided into several sub-scenarios, based on whether the cyclist turned left or went straight, whether the arm was raised or not, and on how critical the situation was. These sub-scenarios are divided into two categories, based on whether the overall combination of context cues refers to a typical ("normal") scene in real traffic or not. For instance, raising an arm in a critical situation before turning left is considered a typical combination of context cues in such a scenario, whereas not raising an arm in a critical situation is not. The number of tracks per sub-scenario is given in table I.

TABLE I: Breakdown of the number of tracks in the cyclist dataset for the sub-scenarios with normal (above the line) and anomalous contextual behavior (below the line) [1].

Sub-scenario			Occurrences
non-critical	arm not raised	straight	6
non-critical	arm not raised	turn	6
non-critical	arm raised	turn	6
critical	arm not raised	straight	10
critical	arm raised	turn	7
non-critical	arm raised	straight	5
critical	arm raised	straight	4
critical	arm not raised	turn	7

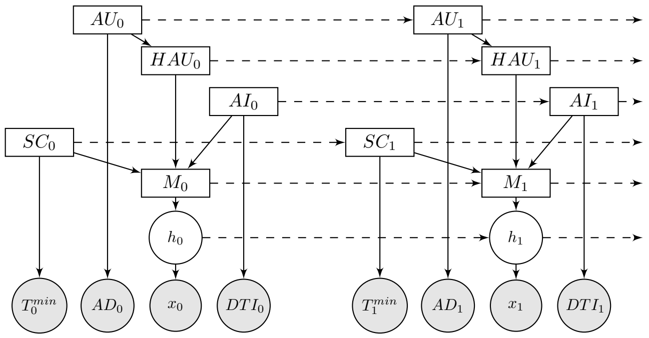


Fig. 4: The graph representation of the DBN from [1]. Rectangular nodes are discrete, round nodes are continuous. Gray nodes indicate measured values.

Each track involving turning has the frame where the cyclist first visibly starts to turn manually labeled as Time To Event (TTE) = 0. Frames before and after the labeled frame have negative and positive TTE values, respectively. In the experiments TTE is used to temporally align tracks in a meaningful way [13]. For the straight tracks, TTE = 0 is defined as the first frame on which the cyclist is past the point on the intersection where 25% of the turning tracks have already started their turn, according to the annotations.

Some of the tracks from the dataset contain frames without position information. Because the proposed RNN has no inherent way to handle missing data, we use the smoothed tracks as described in [1] for both training and evaluating the RNN and DBN.

B. DBN Scenario-specific Crafting

We first explain the DBN state representation from [1], together with what parts of the model we fix during training to keep the model interpretable. Next, we describe the annotation-based method from [1] to find the initial estimate for the remaining parameters. Finally, we explain the annotation-free method for selecting initial parameters. The training of this method is identical to the annotation-based method.

1) Model definition and training: The model has two constant-velocity models as dynamic modes: one for when the cyclist moves straight and one for when the cyclist turns left. Its graph representation is given in fig. 4. The elements in the continuous hidden state vector $h_t \in \mathbb{R}^6$ are the lateral and longitudinal position (referred to as x and y in this section), the x and y velocity of the cyclist if turning left, and the x and y velocity of the cyclist if moving straight. The discrete hidden state $\mathcal{D}_t = [M_t, AU_t, HAU_t, AI_t, SC_t]$ contains the current model M_t and four context-related binary variables: whether the cyclist's arm is raised, AU_t, whether the cyclist's arm has been raised, HAU_t, whether the cyclist is at the intersection, AI_t, and whether the situation is critical, SC_t.

For the discrete state, fig. 4 shows that each variable in the discrete state is assumed to only depend on parts of the previous discrete state. This leads to 5 separate transition tables, one for each discrete state: $\mathcal{P}_{AU}$, $\mathcal{P}_{HAU}$, $\mathcal{P}_{AI}$, $\mathcal{P}_{SC}$, and $\mathcal{P}_M$. For $\mathcal{P}_{HAU}$, to represent the notion whether the cyclist has had an arm up, it encodes the following rule:

$$p\left(HAU_t | HAU_{t-1}, AU_t\right) = \begin{cases} \text{true} & \text{if } (HAU_{t-1} \vee AU_t) \\ \text{false} & \text{otherwise.} \end{cases} \tag{14}$$

$\mathcal{P}_{HAU}$ is fixed during optimization, the others are optimized.

The optimizable continuous state parameters are shown in table II. When initialized, the A matrices encode two constant-velocity models. During optimization, the A matrices are constrained in such a way that the hidden state keeps the representation of position and velocity, but the constant-velocity assumption is removed. Instead, the velocity at the next time step can be any linear combination of the previous x and y velocity. In the initial parameter estimation, the process noise $\mathcal{N}(\mu_\epsilon, Q)$ is assumed to only affect the position and is assumed to be zero-mean. During optimization, $\mathcal{N}(\mu_\epsilon, Q)$ can affect both the position and the velocity, and is not assumed to be zero-mean. The measurement noise covariance R is not constrained. Finally, the continuous initial state distribution $\mathcal{N}(h_0, P_0)$ is defined with the assumption that the initial state of the cyclist is moving straight. Therefore, the position should initially not affect the mean and covariance of the latent turning speed when moving straight. As such, $\mathcal{N}(h_0, P_0)$ is set up so that the position only correlates with the velocity of the cyclist moving straight. During optimization, the same structure is kept.

2) Annotation-based initial estimate: The parameters from table II that require an initial estimate are Q, R, h_0, and P_0. In [1], these are found by running a Kalman smoother over the tracks, which gives a ground truth position and velocity at each time step. The transition tables for $\mathcal{P}_{AU}$, $\mathcal{P}_{AI}$, $\mathcal{P}_{SC}$, and $\mathcal{P}_M$ are estimated by first annotating their related discrete variables (i.e. arm up, at intersection, situation critical, and current model) for each frame. Then, the transition tables are computed by counting the number of occurrences where the discrete ground truth annotations relevant for that transition table switch from one discrete state another. Finally, $p\left(c_{n_t} | C_{n_t}\right)$, the distribution for each context feature measurement given their respective discrete variable, is fitted using either Mixtures of Gaussians (MoGs) or beta distributions, equal to [1].

3) Annotation-free initial estimate: For the initial state h_0, the initial position is taken from the mean position of all initial positions. The cyclist straight velocity is assumed to be 18 km/h. We assume the same velocity when turning left,

TABLE II: The initial estimation and optimization for all parameters in the DBN that relate to its continuous state. The state vector is, in order: the x and y position, the x and y velocity if turning left, and the x and y velocity if moving straight. $\mathbf{0}$ indicates that part is fixed to be zeros, $\mathbf{I}$ indicates that part is fixed to be identity. In the left column, $(\cdot)$ indicates values retrieved from the initial estimation step [1]. In the right column, it indicates which values are altered during optimization. The size of each $(\cdot)$, $\mathbf{0}$ or $\mathbf{I}$ is 2×2, except for the vectors μ_ϵ and h_0 where it is 2×1.

Initial estimate	Optimization
Kinematic parameters:	
$A^{(1)} = \begin{bmatrix} \mathbf{I} & \mathbf{I} & \mathbf{0} \\ \mathbf{0} & \mathbf{I} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{I} \end{bmatrix}$	$A^{(1)} = \begin{bmatrix} \mathbf{I} & \mathbf{I} & \mathbf{0} \\ \mathbf{0} & (\cdot) & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{I} \end{bmatrix}$
$A^{(2)} = \begin{bmatrix} \mathbf{I} & \mathbf{0} & \mathbf{I} \\ \mathbf{0} & \mathbf{I} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{I} \end{bmatrix}$	$A^{(2)} = \begin{bmatrix} \mathbf{I} & \mathbf{0} & \mathbf{I} \\ \mathbf{0} & \mathbf{I} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & (\cdot) \end{bmatrix}$
$C = \begin{bmatrix} \mathbf{I} & \mathbf{0} & \mathbf{0} \end{bmatrix}$	$C = \begin{bmatrix} \mathbf{I} & \mathbf{0} & \mathbf{0} \end{bmatrix}$
Noise parameters:	
$\mu_\epsilon = \begin{bmatrix} \mathbf{0} \\ \mathbf{0} \\ \mathbf{0} \end{bmatrix}$	$\mu_\epsilon^{(1)} = \begin{bmatrix} (\cdot) \\ (\cdot) \\ \mathbf{0} \end{bmatrix}, \mu_\epsilon^{(2)} = \begin{bmatrix} (\cdot) \\ \mathbf{0} \\ (\cdot) \end{bmatrix}$
$Q = \begin{bmatrix} (\cdot) & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} \end{bmatrix}$	$Q^{(1)} = \begin{bmatrix} (\cdot) & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & (\cdot) & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} \end{bmatrix}, Q^{(2)} = \begin{bmatrix} (\cdot) & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & (\cdot) \end{bmatrix}$
$R = \begin{bmatrix} (\cdot) \end{bmatrix}$	$R = \begin{bmatrix} (\cdot) \end{bmatrix}$
Initial state parameters:	
$h_0 = \begin{bmatrix} (\cdot) \\ (\cdot) \\ (\cdot) \end{bmatrix}$	$h_0 = \begin{bmatrix} (\cdot) \\ (\cdot) \\ (\cdot) \end{bmatrix}$
$P_0 = \begin{bmatrix} (\cdot) & \mathbf{0} & (\cdot) \\ \mathbf{0} & (\cdot) & \mathbf{0} \\ (\cdot) & \mathbf{0} & (\cdot) \end{bmatrix}$	$P_0 = \begin{bmatrix} (\cdot) & \mathbf{0} & (\cdot) \\ \mathbf{0} & (\cdot) & \mathbf{0} \\ (\cdot) & \mathbf{0} & (\cdot) \end{bmatrix}$

albeit at a 45-degree angle. The initial covariance P_0 is estimated as a diagonal matrix. The initial position variance is set to the variance of the lateral and longitudinal initial position. The initial covariance of the velocity in both directions and both modes is one-tenth of the initial velocity. The observation noise R is set to identity in meters. The process noise Q acting on the position is set to one-tenth of the initial velocity.

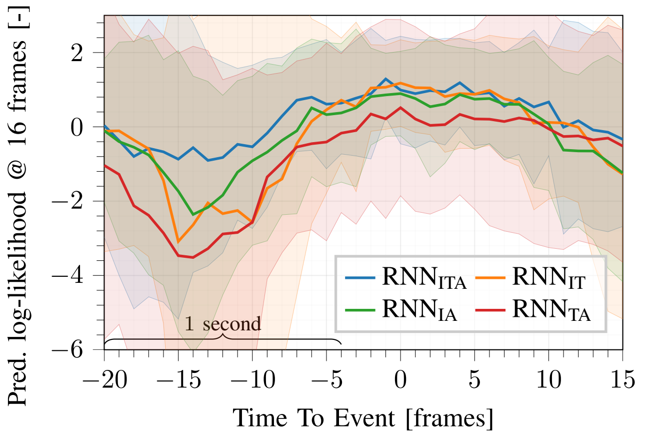
For the context parameters, the context transition matrices have a 0.01 probability of transitioning to another binary state,

$$\mathcal{P}_{AU} = \mathcal{P}_{AI} = \mathcal{P}_{SC} = \begin{bmatrix} 0.99 & 0.01 \\ 0.01 & 0.99 \end{bmatrix}. \tag{15}$$

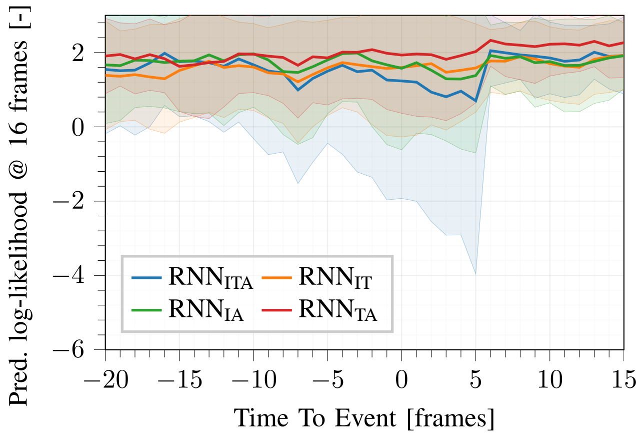
The model transition matrix $\mathcal{P}_M$ is set to have a transition probability from straight to turning of 0.01 when the conditions of a normal turning subscenario are met, as given in table I, otherwise it is 0. Finally, the parameters for the conditional probabilities $p\left(c_{n_t} | C_{n_t}\right)$ are selected in an intuitive sense: for example, the "at intersection" normal distribution is centered at the intersection, with the "not at intersection" MoG before and after the intersection. The same distributions types

TABLE III: The log-likelihood of the predictions 16 steps (one second) in the future, averaged over the period TTE $\in [-15, 15]$. The models are only trained on tracks from the normal sub-scenarios. For the RNNs, the letters indicate which context cues were available to the RNN: **I** if the RNN used the distance to **I**ntersection, a **T** if the RNN used the **T**ime until the vehicle could overtake, and an **A** if it used the probability that the **A**rm was up.

Evaluated on	RNN	RNN$_\text{I}$	RNN$_\text{T}$	RNN$_\text{A}$	RNN$_\text{IT}$	RNN$_\text{IA}$	RNN$_\text{TA}$	RNN$_\text{ITA}$
All normal subscenarios	−0.42	0.68	0.40	0.51	0.47	0.57	0.36	**0.81**
Normal turning subscenarios	−1.21	0.06	−0.58	−0.32	−0.28	−0.17	−0.70	**0.34**
Normal straight subscenarios	0.79	1.63	1.90	1.79	1.64	1.73	**2.00**	1.55
All anomalous subscenarios	−9.47	0.42	−5.54	−5.07	−1.22	−9.72	−8.76	**−11.48**



(a) Normal turning sub-scenarios



(b) Normal straight sub-scenarios

Fig. 5: One second ahead prediction log-likelihood mean (thick line) and one-sigma standard deviation (shaded area) of RNNs over time. When turning (fig. 5a), the RNN with all three context cues (RNN$_\text{ITA}$, blue line) performs the best.

from the annotation-based method are used for the annotation-free method.

V. Experiments

In section V-A, we evaluate the performance of the RNN and investigate whether incorporating context cues improves its predictive accuracy. In section V-B, we evaluate the performance impact of gradient-based optimization of the DBN parameters in comparison to the previously used annotation-based parameter estimation method. After having assured that both the RNN and DBN models can be trained on the same context cues with the same optimization strategy, we compare their performances in section V-C.

Given the measurements up to time step t, each model computes a distribution of the future position x_{t+n} at time step $t+n$: $p\left(x_{t+n}|y_{0:t}\right)$. As in [1], we evaluate this predictive distribution $n = 16$ steps (one second) into the future, around the point where the cyclist may turn left: the range TTE $\in [-15, 15]$. Let $\widehat{x}_{t+n}$ be the actual future position in the data. Following [1], we use two different performance metrics to evaluate a sequence, namely, the log-likelihood of this future position under the predictive distribution (higher is better) and the Euclidean distance between predicted expected position and this actual future position (lower is better):

$$ll\left(t+n|t\right) = \log p\left(x_{t+n} = \widehat{x}_{t+n}|y_{0:t}\right) \quad (16)$$

$$error\left(t+n|t\right) = \left|\mathbb{E}_{x_{t+n}}\left[p\left(x_{t+n}|y_{0:t}\right)\right] - \widehat{x}_{t+n}\right|. \quad (17)$$

The predictive distribution for the DBN is a mixture of N_M^2 Gaussians ($N_M = 2$) [1]. It is a single Gaussian for the RNN. All models are implemented in PyTorch [39] and evaluated using leave-one-out cross-validation on a Titan X Pascal GPU. For the RNN, after a preliminary hyperparameter search we select a hidden layer size of $N_h = 32$, and train using the Amsgrad [40] algorithm for 2000 iterations with a learning rate of 0.0015, taking 50 minutes per cross-validation fold. The DBN is trained for 1000 iterations with a learning rate of 0.0001 using the same algorithm, taking 130 minutes per fold. Both models run in real-time: 4 ms per frame for the RNN, and 10 ms per frame for the DBN.

A. RNN Evaluation

We analyze how well the RNN incorporates context cues in its prediction by looking at the performance of the RNN with every combination of context cues as input values. Next, we analyze the effectiveness of the training strategies of section III-B through an ablation study.

1) Incorporating context cues in an RNN: Table III shows the predictive log-likelihood of RNNs incorporating different combinations of context cues (the caption defines the naming convention). From left to right, the table shows RNNs with increasingly more information available to them. On the normal sub-scenarios, the addition of one cue (columns RNN$_\text{I}$, RNN$_\text{T}$, and RNN$_\text{A}$) improves the likelihood over the model without any context cues. We observe that using two cues does not improve performance over using a single cue. Apparently, the additional information does not outweigh the disadvantage of increasing the input dimensionality. Utilizing all three context cues (RNN_{ITA}), however, does result in the best performance.

All models perform better on the straight sub-scenarios than on the turning sub-scenarios, likely due to the more complex dynamics in the turning scenario.

The full model also attains the lowest log-likelihood of all models on the anomalous data. This further shows that it leverages the context information to inform on its predictions, as the only difference between the normal and anomalous sub-scenarios is the validity of the context cues. The full RNN model thus successfully discriminates between such sub-scenarios and has shifted the mass of its predictive distribution away from the anomalous cases.

For a more in-depth analysis, fig. 5 shows the log-likelihood over time, using the annotated TTE to temporally align the tracks. These graphs show the log-likelihood of a prediction made at that specific TTE, e.g. the point at $\text{TTE} = -10$ shows the likelihood of the prediction for $\text{TTE} = 6$. Figure 5a shows that the RNNs increase in accuracy starting around $\text{TTE} = -10$, a moment where the RNN predicts what happens after the turn. That means that the RNN detects that the cyclist will turn over half a second before the annotated point of turning, $\text{TTE} = 0$.

For the normal sub-scenarios where the cyclist continues straight (fig. 5b), all models perform relatively similar. The performance of the full model does decrease slightly over time. This is in line with the results of table III: the main reason for the overall better performance of the full model is the improved performance on the tracks of the normal turning sub-scenarios, without losing too much performance on the normal straight sub-scenarios.

When comparing the average Euclidean distance error of the predictions, the full model outperforms the other models as well, albeit only slightly: the average Euclidean distance error is 33 cm when evaluated on the tracks from the normal sub-scenarios (34 cm/31 cm on normal turning/straight sub-scenarios, respectively). The other RNNs with one or two context cues have an error between 34 cm and 35 cm, the RNN with no context cues has an error of 49 cm.

Overall, we find that the RNN exploits the additional context cues. This mirrors the results for the DBN found in [1]: both approaches benefit most from combining all distinct types of context in the normal sub-scenarios, while as expected both also assign a low probability to the designed anomalous cyclist responses to these context cues. We thus conclude that both models have homogenized context input.

TABLE IV: The categorization of all DBN parameters into distinct groups, to study the effect of optimizing related parameters. The superscript $(\cdot)^{(M_{t-1})}$ indicates that the parameter is distinct for each dynamic mode. The letter in brackets is used to specify what has been optimized in a DBN.

Name of group	Content of group
Context parameters (C)	$\mathcal{P}_{AU}$, $\mathcal{P}_{AI}$, $\mathcal{P}_{SC}$, $\mathcal{P}_{M}$, $\mathcal{P}^0$, $p\left(c_{n_t} \mid C_{n_t}\right)$
Noise parameters (N)	$Q^{(M_t)}$, $\mu_\epsilon^{(M_t)}$, R, P_0, h_0
Kinematic parameters (K)	$A^{(M_t)}$

2) RNN training strategies: Next we demonstrate the importance of the training strategies discussed in section III-B through an ablation study. The RNN is trained with all three context cues as additional input and evaluated on all tracks from the normal sub-scenarios. As shown before, the proposed RNN_{ITA} achieves an average prediction log-likelihood of 0.81. Without normalization, the prediction log-likelihood drops to -5.85. Without resetting the hidden layer during training, it drops to -0.61. This shows that both training strategies help improve the accuracy of the RNN.

B. DBN Evaluation

Turning to the DBN, we first verify that the optimization increases the performance compared to the annotation-based initial parameter estimation. Secondly, we show that the optimization improves the alignment of the latent turning probability with the annotated moment of turning. Finally, we compare the performance of the annotation-based initial estimate with the annotation-free initial estimate.

To better understand the effects of the optimization, we categorize the relevant parameters into three groups, see table IV. Various combinations of these groups are either fixed to their initial estimate or optimized. The letter within brackets in the table is used to specify what has been optimized in a DBN. For example, DBN^{CN} has both the **C**ontext and **N**oise parameters optimized. DBN (no superscript) refers to the original, unoptimized DBN from [1]. When optimized, constraints as mentioned in section III-D apply.

1) Optimizing the DBN: Table V shows the performance of the original and optimized DBNs. Every optimized DBN improves overall performance compared to the original DBN. Optimizing all parameters (DBN^{CNK}) results in the best overall performance.

The Euclidean distance error improves for each optimized model, save one. The unoptimized DBN has an error of 64 cm

TABLE V: The log-likelihood the DBNs on their predictions 16 steps (one second) in the future, averaged over the period $\text{TTE} \in [-15, 15]$. The models are only trained on tracks from the normal sub-scenarios. The names indicate which parameter groups were further optimized (see table IV).

Evaluated on	DBN	DBN^{C}	DBN^{N}	DBN^{CN}	DBN^{NK}	DBN^{CNK}
All normal subscenarios	-1.53	-1.38	-0.22	-0.20	-0.14	$\mathbf{-0.12}$
Normal turning subscenarios	-2.95	-2.63	-1.08	-1.04	-1.02	$\mathbf{-1.00}$
Normal straight subscenarios	0.84	0.69	1.15	1.14	1.25	**1.28**
All anomalous subscenarios	-2.40	-2.13	$\mathbf{-1.10}$	-1.12	-1.11	-1.14

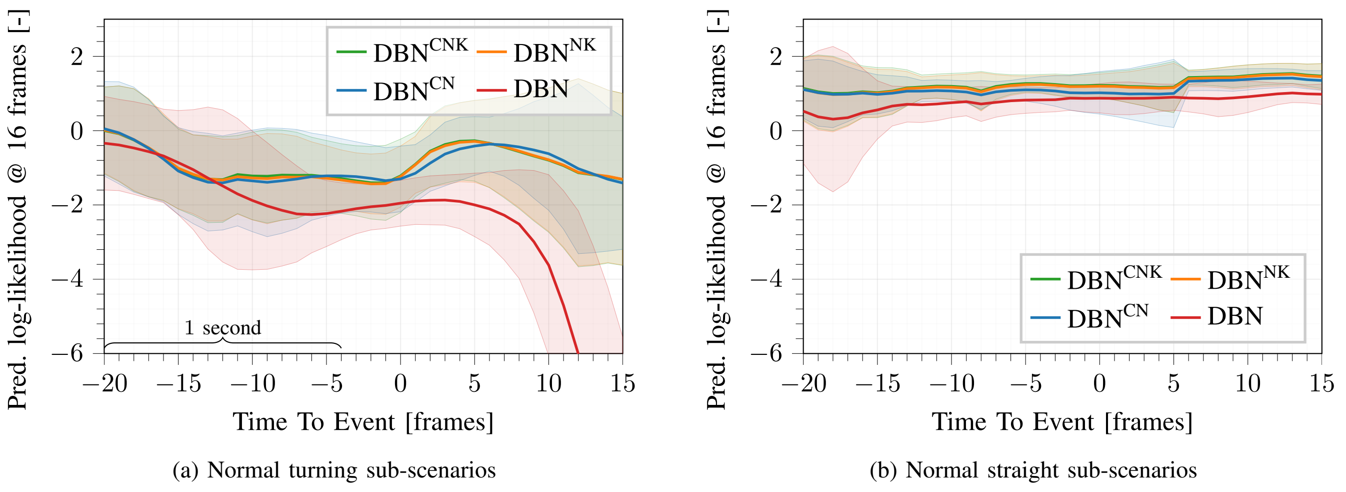


Fig. 6: One second ahead prediction log-likelihood mean (thick line) and one-sigma standard deviation (shaded area) of DBNs over time. In the turning sub-scenarios (fig. 6a), the performance of the DBN with no additional parameter optimization (DBN, red line) deteriorates after TTE = 5. The optimized DBNs do not see this deterioration. In the straight scenario (fig. 6b), optimization improves performance.

for the turning sub-scenarios, and 25 cm when moving straight. All optimized DBNs except DBN^{C} attain an error of 39-42 cm when turning, and 22-24 cm when moving straight. For DBN^{C}, the error increases to 67 cm when turning, and decreases to 19 cm when going straight.

To understand the performance over time, fig. 6 shows the prediction log-likelihood of the three best performing optimized DBNs alongside the unoptimized DBN. For the turning case (fig. 6a), the main improvement in performance stems from better modeling of the turning dynamics. Because the context cues only inform on the likelihood of *switching* rather than the likelihood of the current dynamic mode, the DBN can only infer the cyclist is turning from position information. For the sub-scenarios where the cyclist continues straight (fig. 6b), optimizing consistently improves the performance.

2) Detection of dynamics change: The probability of being in the turning dynamic mode should remain close to zero when the cyclist moves straight. This is indeed the case: our experimental records show that the average probability of turning on straight scenarios is less than 0.5% for all models.

Conversely, the turning probability should go up for the normal turning tracks around TTE $= 0$, the annotated moment of turning. Figure 7 shows how this probability changes over time for the turning scenario. The graph shows that optimizing the context group has no discernible effect on when the model switches to turning: DBN^{CNK} coincides with DBN^{NK}, DBN^{CN} with DBN^{N}, and DBN^{C} with DBN. This is because the context cues inform the model on when the switch from straight to turning is more likely to occur. Whether the cyclist is actually turning is determined by the likelihood of the position measurements and therefore by the dynamics.

The other parameter groups do affect the model's reaction to turning. Optimizing the noise parameter group moves the moment of turning closer to TTE $= 0$. Optimizing the kinematic parameter group moves it even closer.

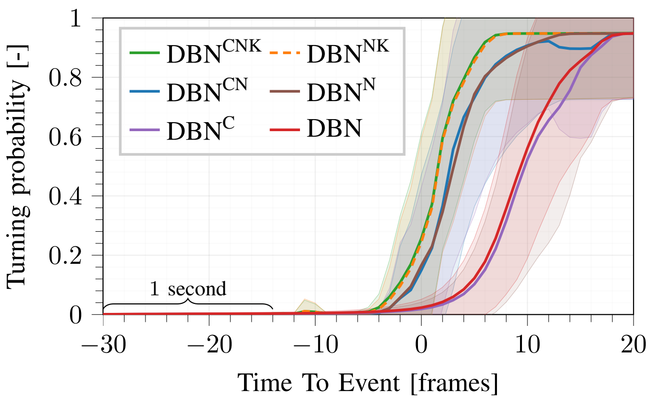
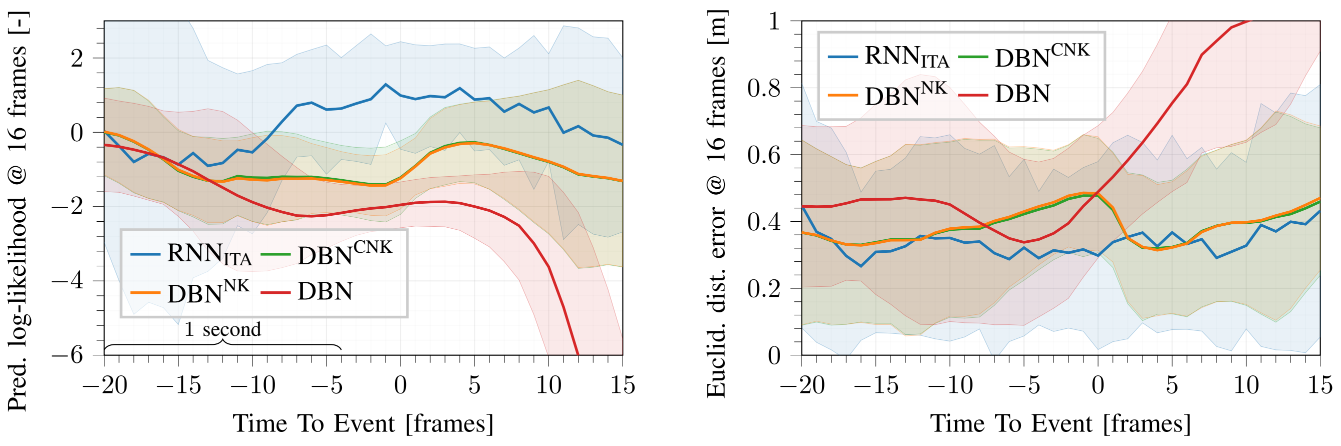


Fig. 7: The mean (lines) and one-sigma standard deviation (shaded area) of the turning probability for the normal turning tracks. The turning probability is most in line with the annotated moment of turning when all parameters are optimized (DBN^{CNK}, green line, and DBN^{NK}, dashed orange line).

3) Annotation-free initial estimation: To assess the need for annotations, we perform the annotation-free initial estimation scheme laid out in section IV-B3, and then optimize the model as before, i.e. like DBN^{CNK}. This leads to an average log-likelihood over all scenarios of -0.2, which still outperforms the unoptimized DBN (-1.53, see table V), but it is slightly worse than the log-likelihood of DBN^{CNK} with annotation-based initial estimation (-0.12). At the same time, the average Euclidean distance error over all normal scenarios did improve from 33 cm to 31 cm. We conclude that we can do without the laborious manual annotation step of the latent variables of the DBN and still obtain a competitive performance.

C. Comparison of DBN with RNN

After having established that both the RNN and the DBN can be trained on the same context cues using the same



(a) Log-likelihood of normal turning sub-scenarios (b) Euclidean distance error of normal turning sub-cenarios

Fig. 8: The mean (lines) and one-sigma standard deviation (shaded area) of the one second ahead prediction log-likelihood (fig. 8a) and Euclidean distance error (fig. 8b) for the normal turning sub-scenarios.

optimization strategies, we now compare both approaches to assess the performance impact for using either a crafted or a learned state representation. When comparing the average log-likelihood, the best RNN in table III outperforms the best optimized DBN in table V: 0.81 to -0.12. However, at the same time, the average Euclidean distance error over all normal sub-scenarios is 33 cm for both. The source of the Euclidean distance error is not equal for both models, however. The average Euclidean distance error made by the RNN on the turning sub-scenarios and the straight sub-scenarios is almost identical: 34 cm and 31 cm, respectively. Because the RNN is a generic model, it is reasonable that it has no bias towards either type of dynamics. In contrast, the linear models of the DBN can directly encode a cyclist going straight with a constant velocity model, whereas the varying radii of a cyclist turning left cannot be represented as well. The corresponding error values are 39 cm and 23 cm, for the best optimized DBN.

More in-depth, fig. 8a shows the predicted likelihood over time for all tracks from the normal turning sub-scenarios, centered around TTE $= 0$. The results are shown for the best performing RNN, RNN_{ITA}, as well as the unoptimized DBN and the two best performing optimized DBNs. From TTE $= -10$, the performance of the RNN (blue line) starts to diverge from the two optimized DBNs (green and orange line). While the gap narrows from around TTE $= 0$, it never fully closes. When comparing the Euclidean distance errors on the same tracks (fig. 8b), we see the same divergence between the RNN and the two optimized DBNs starting at TTE $= -10$ but find that the difference in Euclidean distance error returns to almost zero starting at TTE $= 2$. It seems that the DBN, when its parameters are optimized, can predict the average position almost as well as the RNN, thus differences in the log-likelihood are mostly due to larger variance in the predictive distribution required to compensate the DBN's linear dynamics.

As a last observation, both the RNN and the DBNs with optimized parameters show a dip in prediction log-likelihood (fig. 8a), but the RNN recovers around 10 frames earlier than the DBNs: TTE $= -10$ versus TTE $= 0$. This seems to indicate that the current context cues, together with the position information, already contain additional relevant information to predict when a cyclist will turn, but that the DBN is not yet properly capturing this aspect.

VI. Discussion

We examined two models for predicting the distribution over the future position of a cyclist: the RNN and DBN. They use completely different state representations for the dynamic state of the kinematics and context information. When performance is the only goal, the RNN is currently the best choice, as it attained the highest average log-likelihood. By using the right training strategies, the RNN was able to leverage the information present in the context cues (table III). However, because of the "black-box" nature of the RNN, it is difficult to inspect the model and explain how the context cues exactly affect its predictions, other than empirical validation and statistical arguments. On the other hand, the DBN has the benefit that we can ensure that its discrete latent state is interpretable by appropriately specifying the structure of the model (fig. 7) and its parameters. Interestingly, our results show that after gradient descent based optimization similar to the RNN, the performance gap is significantly reduced compared to previously reported results [1]. The optimized DBN even attains similar Euclidean distance error as the RNN (section V-C). Moreover, one can do without the laborious manual annotation step of all latent variables of the DBN (as is the norm in the state-of-the-art experimentation) and still obtain a similar performance.

An added value of investigating both an approach with a learned representation such as an RNN and an approach with a crafted representation such as the DBN is that they provide complementary insights: the former shows *if* certain measurements or context cues can help improve prediction, the latter shows *how well* our assumptions on the measurements and context cues hold. In our case, the similar Euclidean distance error but the worse log-likelihood of the DBN compared to the

RNN lead us to conclude that the DBN at times over-estimates the uncertainty in its predictions. This can be attributed to the DBN using only two linear dynamical models for varying turning behaviors. An important direction to improve the DBN is thus to allow for more varied motion dynamics. This could be achieved by loosening existing assumptions, e.g. that noise is constant over time, or by incorporating non-linear motion models with an extended or unscented Kalman filter or particle filter. Another direction is to learn more varied and specialized dynamic modes from the track data itself, e.g. by estimating the number of dynamics and their context with appropriate priors during model optimization [10], [41].

An open challenge is to create predictive methods that scale to a more diverse set of real-life traffic conditions (i.e. multiple scenarios, different road users) while remaining interpretable and incorporating a rich set of context cues. For the DBN, the computational complexity can be partially curbed by limiting the dependencies between discrete states (fig. 4), though it may be necessary to learn these dependencies from data instead of designing these relations manually as was done here. The interesting alternative is to take a learned representation method and encode expert knowledge in specific areas of the model, thereby making it interpretable and keeping its high performance. Possible directions include combining learned context representations to predict distributions over a fixed set of predefined dynamics [42], and incorporating agent interaction explicitly as a graph structure in the neural networks [23], [43]. In contrast, attentive networks [35] provide interpretability through inspection of node activations for specific inputs, rather than through explicit encoding of expert knowledge.

The advent of large-scale naturalistic datasets such as Argoverse [44] will be important to further these future research directions. Even so, our current findings on the impact of gradient-based optimization are also relevant to other scenarios where DBNs have already been successfully applied without such optimization strategies, such as signalized [12] and non-signalized [21], [45] pedestrian crossing, and in joint pedestrian-driver awareness collision risk estimation [11]. We also note that our approach of studying the representation in isolation may be useful for other applications too, such as surveillance with path prediction in crowds, where traditional expert-designed representations [46] have been fully replaced by learned representations [23]. Ideally, expert knowledge and semantic concepts can be seamlessly incorporated in the learned representation and optimized jointly, potentially resulting in the best of both worlds.

VII. Conclusion

We described two models for predicting a Gaussian-based distribution over the future position of a cyclist that incorporate various context cues and learn distinct dynamic modes. The main distinction between these models was their latent state representation: crafted vs. learned. For the RNN model with a learned state representation, we showed that it could leverage the context cues to improve its path prediction. For the DBN model with a crafted representation, we explained how to optimize it while keeping its latent state interpretable.

Comparing the two models thus at a level playing field, we found that the RNN attains the best predictive performance overall (significantly outperforming the optimized DBN on the log likelihood measure, while performing similarly on the Euclidean distance error measure, i.e. 31–34 cm vs. 23–39 cm for the DBN). This suggests, more broadly, that if performance is the only relevant metric (and sufficient data is available), a learned state representation is the preferable choice. On the other hand, results showed that optimizing the DBN did partially close the performance gap with the RNN, even without a laborious manual annotation of all latent variables. We conclude that crafted state representations remain suitable for safety-critical applications where it is important to understand why a model behaves the way it does, or for cases where one wishes to further knowledge of the underlying causalities.

Further work could focus on models that better combine data adaptation and expert knowledge. The DBN could be allowed more flexibility to adapt to the data by means of automatic motion model discovery including extensions to higher-order/non-linear motion models. Conversely, the RNN could be more strongly regularized by explicitly encoding physical models or relevant (infrastructure or otherwise) context known to a human expert.

Acknowledgment

This work was supported by the Dutch Science Foundation NWO-TTW within the SafeVRU project, nr. 14667.

Appendix

We provide an overview of the computational graph created for the inference algorithm of the DBN to aid reproducibility and to demonstrate that the inference algorithm is suitable for gradient-based optimization. A schematic overview of the data flow for one time step in the algorithm is shown in fig. 9. Inference consists of three main steps: *Predict*, *Update* and *Marginalize*, see [1]. The update step can only be applied when integrating past or current measurements; it cannot be used in future time steps. At each step, the algorithm computes a new distribution over the DBN's latent state. The probability of a discrete state $\mathcal{D}_t = [M_t, C_{1_t}, \ldots C_{N_C t}]$ is expressed with a scalar $d_t^{(\mathcal{D}_t)}$. The continuous state is represented by N_M means $h_t^{(M_t)}$ and covariances $P_t^{(M_t)}$, one for each model M_t.

The subsections below list the equations corresponding to each step for the latent states, and the figure also refers to these equations per step. All equations consist of basic operations such as matrix multiplications or additions, which are differentiable and straightforward to implement in frameworks such as PyTorch [39] and TensorFlow [47].

1) Predict: Given $p(h_{t-1}, \mathcal{D}_{t-1}|y_{0:t-1})$ from the previous iteration, compute $p(h_t, \mathcal{D}_t, \mathcal{D}_{t-1}|y_{0:t-1})$. With the model definitions of section III-C, prediction of the next continuous state is done using a Kalman filter, for every N_M^2 combination of current and previous model:

$$h_{t|t-1}^{(M_t,t-1)} = A^{(M_t)} h_{t-1}^{(M_{t-1})} + \mu_\epsilon^{(M_t)} \tag{18}$$

$$P_{t|t-1}^{(M_t,t-1)} = A^{(M_t)} P_{t-1}^{(M_{t-1})} A^{\top(M_t)} + Q^{(M_t)} \tag{19}$$

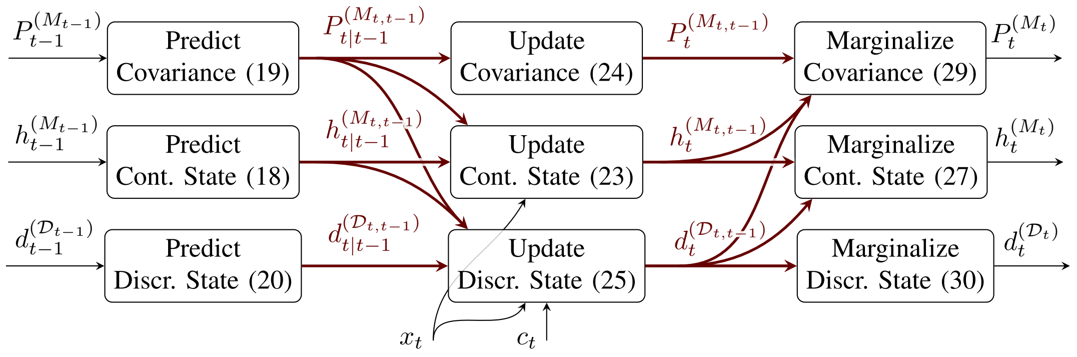

$P_{t-1}^{(M_{t-1})}$
Predict Covariance (19)
$P_{t|t-1}^{(M_{t,t-1})}$
Update Covariance (24)
$P_{t}^{(M_{t,t-1})}$
Marginalize Covariance (29)
$P_{t}^{(M_{t})}$
$h_{t-1}^{(M_{t-1})}$
Predict Cont. State (18)
$h_{t|t-1}^{(M_{t,t-1})}$
Update Cont. State (23)
$h_{t}^{(M_{t,t-1})}$
Marginalize Cont. State (27)
$h_{t}^{(M_{t})}$
$d_{t-1}^{(\mathcal{D}_{t-1})}$
Predict Discr. State (20)
$d_{t|t-1}^{(\mathcal{D}_{t,t-1})}$
Update Discr. State (25)
$d_{t}^{(\mathcal{D}_{t,t-1})}$
Marginalize Discr. State (30)
$d_{t}^{(\mathcal{D}_{t})}$
x_t
c_t

Fig. 9: A schematic overview of how the DBN incorporates contextual (c_t) and positional (x_t) measurements to infer the state over time. Each block corresponds to an equation, referenced in brackets. The computations are done in joint domains ($(M_{t,t-1})$ and $(\mathcal{D}_{t,t-1})$) for the bold colored lines and in single domain (e.g. $(\mathcal{D}_{t-1})$) for the thin black lines.

$$d_{t|t-1}^{(\mathcal{D}_{t,t-1})} = \mathcal{P}_{\mathcal{D}_t}^{(\mathcal{D}_{t-1})} d_{t-1}^{(\mathcal{D}_{t-1})} \tag{20}$$

The superscript $^{(M_{t,t-1})}$ indicates that the predicted state is computed for each possible model combination (i.e. their joint probability). Accordingly, the distribution of the hidden state is defined as a mixture of N_M^2 Gaussians [1]. Similarly, the discrete state probability $d_{t|t-1}^{(\mathcal{D}_{t,t-1})}$ is computed for the $|\mathcal{D}|^2$ joint discrete state combinations.

2) Update: Obtain $p(h_t, \mathcal{D}_t, \mathcal{D}_{t-1}|y_{0:t})$ by incorporating measurement $y_t = [x_t, c_{1_t}, \ldots, c_{N_{c_t}}]$, akin to a Kalman update:

$$S_t^{(M_{t,t-1})} = CP_{t|t-1}C^\top + R^{(M_t)} \tag{21}$$
$$K_t^{(M_{t,t-1})} = P_{t|t-1}C^\top S_t^{-1} \tag{22}$$
$$h_t^{(M_{t,t-1})} = h_{t|t-1} - K_t(x_t - Ch_{t|t-1}) \tag{23}$$
$$P_t^{(M_{t,t-1})} = (\mathbf{I} - K_tC)P_{t|t-1} \tag{24}$$
$$d_t^{(\mathcal{D}_{t,t-1})} = d_{t|t-1}^{(\mathcal{D}_{t,t-1})} p\left(x_t|h_{t|t-1}, P_{t|t-1}\right) \prod_{n=1}^{N_c} p\left(c_{n_t}|C_{n_t}\right) \tag{25}$$

Where $p\left(x_t|h_{t|t-1}, P_{t|t-1}\right)$ is the likelihood of the measurement x_t for the state $h_{t|t-1}^{(M_{t,t-1})}$ with covariance $P_{t|t-1}^{(M_{t,t-1})}$. For readability, the superscript $^{(M_{t,t-1})}$ has been omitted on the right-hand side for the variables $h_{t|t-1}$, $P_{t|t-1}$, S_t, and K_t.

3) Marginalize: Computing the full joint probability by iterating the previous steps would quickly become intractable, as there are $(N_M)^t$ motion model combinations after t steps. To make inference tractable, we marginalize $p(h_t, \mathcal{D}_t, \mathcal{D}_{t-1}|y_{0:t})$ over the past discrete state $\mathcal{D}_{t-1}$ to obtain approximation $p(h_t, \mathcal{D}_t|y_{0:t})$. The mixture of Gaussians over joint models $(M_{t,t-1})$ is therefore collapsed to a mixture over only the current motion models (M_t) through moment matching [1]:

$$d_t^{(M_{t-1})} = \sum_{\mathcal{D}_{t,t-1}/(M_{t-1})} d_t^{(\mathcal{D}_{t,t-1})} \tag{26}$$
$$h_t^{(M_t)} = \sum_{M_{t-1}} h_t^{(M_{t,t-1})} d_t^{(M_{t-1})} \tag{27}$$
$$e_t^{(M_{t,t-1})} = h_t^{(M_{t,t-1})} - h_t^{(M_t)} \tag{28}$$
$$P_t^{(M_t)} = \sum_{M_{t-1}} \left(P_t^{(M_{t,t-1})} + e_t^{(M_{t,t-1})} e_t^{\top(M_{t,t-1})}\right) d_t^{(M_{t-1})} \tag{29}$$
$$d_t^{(\mathcal{D}_t)} = \sum_{\mathcal{D}_{t-1}} d_t^{(\mathcal{D}_{t,t-1})} \tag{30}$$

Here, in eq. (26) the notation $\mathcal{D}_{t,t-1}/(M_{t-1})$ refers to all variables in the joint discrete state $\mathcal{D}_{t,t-1}$ except (M_{t-1}).

REFERENCES

[1] J. F. P. Kooij, F. Flohr, E. A. I. Pool, and D. M. Gavrila, "Context-Based Path Prediction for Targets with Switching Dynamics," *Int. Journal of Comp. Vision (IJCV)*, pp. 239–262, 2019.
[2] R. R. Selvaraju, M. Cogswell, A. Das, R. Vedantam, D. Parikh, and D. Batra, "Grad-CAM: Visual explanations from deep networks via gradient-based localization," in *Int. Journal of Comp. Vision (IJCV)*, 2017, pp. 618–626.
[3] T. Viering, Z. Wang, M. Loog, and E. Eisemann, "How to manipulate CNNs to make them lie: the Grad-CAM case," *Workshop on Interpretable and Explainable Machine Vision*, 2019.
[4] I. Batkovic, M. Zanon, N. Lubbe, and P. Falcone, "A computationally efficient model for pedestrian motion prediction," in *IEEE European Control Conf. (ECC)*, 2018, pp. 374–379.
[5] D. Ridel, E. Rehder, M. Lauer, C. Stiller, and D. Wolf, "A literature review on the prediction of pedestrian behavior in urban scenarios," *IEEE Int. Conf. on Intell. Transp. Syst. (ITSC)*, pp. 3105–3112, 2018.
[6] A. Rudenko, L. Palmieri, M. Herman, K. M. Kitani, D. M. Gavrila, and K. O. Arras, "Human motion trajectory prediction: A survey," *Int. Journal of Robotics Research (IJRR)*, vol. 39, no. 8, pp. 895–935, 2020.
[7] M. Braun, S. Krebs, F. Flohr, and D. M. Gavrila, "EuroCity Persons: A novel benchmark for person detection in automotive context," *IEEE Trans. on Pattern Anal. and Mach. Intelligence (TPAMI)*, vol. 41, no. 8, pp. 1844–1861, 2019.
[8] A. Palffy, J. Dong, J. F. P. Kooij, and D. M. Gavrila, "CNN based road user detection using the 3d radar cube," *IEEE Robotics and Automation Letters (RAL)*, vol. 5, no. 2, pp. 1263–1270, 2020.
[9] J. R. van der Sluis, E. A. I. Pool, and D. M. Gavrila, "An experimental study on 3D person localization in traffic scenes," in *IEEE Intelligent Vehicles Symposium (IV)*, 2020.
[10] E. A. I. Pool, J. F. P. Kooij, and D. M. Gavrila, "Using road topology to improve cyclist path prediction," in *IEEE Intelligent Vehicles Symposium (IV)*, 2017, pp. 289–296.
[11] M. Roth, F. Flohr, and D. M. Gavrila, "Driver and pedestrian awareness-based collision risk analysis," in *IEEE Intelligent Vehicles Symposium (IV)*, 2016, pp. 454–459.
[12] Y. Hashimoto, G. Yanlei, L.-T. Hsu, and K. Shunsuke, "A probabilistic model for the estimation of pedestrian crossing behavior at signalized intersections," in *IEEE Int. Conf. on Intell. Transp. Syst. (ITSC)*, 2015.
[13] C. G. Keller and D. M. Gavrila, "Will the pedestrian cross? A study on pedestrian path prediction," *IEEE Trans. on Intelligent Transportation Systems*, vol. 15, no. 2, pp. 494–506, 2014.

[14] R. Quintero, I. Parra, D. F. Llorca, and M. Sotelo, "Pedestrian intention and pose prediction through dynamical models and behaviour classification," in *IEEE Int. Conf. on Intell. Transp. Syst. (ITSC)*, 2015, pp. 83–88.

[15] B. Volz, H. Mielenz, I. Gilitschenski, R. Siegwart, and J. Nieto, "Inferring Pedestrian Motions at Urban Crosswalks," *IEEE Trans. on Intelligent Transportation Systems*, pp. 544–555, 2018.

[16] E. A. I. Pool, J. F. P. Kooij, and D. M. Gavrila, "Context-based cyclist path prediction using recurrent neural networks," in *IEEE Intelligent Vehicles Symposium (IV)*, 2019, pp. 824–830.

[17] A. Graves, "Generating sequences with recurrent neural networks," *arXiv preprint arXiv:1308.0850*, 2013.

[18] S. Becker, R. Hug, W. Hübner, and M. Arens, "An RNN-based IMM filter surrogate," in *Scandinavian Conf. on Im. Anal.*, 2019, pp. 387–398.

[19] P. Ondrúška and I. Posner, "Deep tracking: Seeing beyond seeing using recurrent neural networks," in *AAAI Conf. on Artificial Intell.*, 2016.

[20] M. Fraccaro, S. Kamronn, U. Paquet, and O. Winther, "A disentangled recognition and nonlinear dynamics model for unsupervised learning," in *Conf. on Neural Information Proc. Syst. (NIPS)*, 2017, pp. 3601–3610.

[21] Y. Li, X.-Y. Lu, J. Wang, and K. Li, "Pedestrian trajectory prediction combining probabilistic reasoning and sequence learning," *IEEE Trans. on Intelligent Vehicles*, 2020.

[22] N. Lee, W. Choi, P. Vernaza, C. B. Choy, P. H. Torr, and M. Chandraker, "DESIRE: Distant future prediction in dynamic scenes with interacting agents," in *IEEE Conf. on Comp. Vision and Patt. Recog. (CVPR)*, 2017, pp. 336–345.

[23] A. Alahi, K. Goel, V. Ramanathan, A. Robicquet, L. Fei-Fei, and S. Savarese, "Social LSTM: Human trajectory prediction in crowded spaces," in *IEEE Conf. on Comp. Vision and Patt. Recog. (CVPR)*, 2016, pp. 961–971.

[24] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, "Empirical evaluation of gated recurrent neural networks on sequence modeling," in *Neural Information Proc. Syst. (NIPS) Workshop on Deep Learning*, 2014.

[25] H. Xiong, F. B. Flohr, S. Wang, B. Wang, J. Wang, and K. Li, "Recurrent neural network architectures for vulnerable road user trajectory prediction," in *IEEE Intelligent Vehicles Symposium (IV)*, 2019, pp. 171–178.

[26] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You Only Look Once: Unified, Real-Time Object Detection," *IEEE Conf. on Comp. Vision and Patt. Recog. (CVPR)*, 2015.

[27] K. M. Kitani, B. D. Ziebart, J. A. Bagnell, and M. Hebert, "Activity forecasting," in *Eur. Conf. on Comp. Vis. (ECCV)*, 2012, pp. 201–214.

[28] V. Karasev, A. Ayvaci, B. Heisele, and S. Soatto, "Intent-aware long-term prediction of pedestrian motion," in *IEEE Int. Conf. on Robotics and Automation (ICRA)*, 2016, pp. 2543–2549.

[29] S. Huang *et al.*, "Deep learning driven visual path prediction from a single image," *IEEE Trans. on Image Processing*, vol. 25, no. 12, pp. 5892–5904, 2016.

[30] L. Ballan, F. Castaldo, A. Alahi, F. Palmieri, and S. Savarese, "Knowledge transfer for scene-specific motion prediction," in *Eur. Conf. on Comp. Vis. (ECCV)*, 2016, pp. 697–713.

[31] K. Saleh, M. Hossny, and S. Nahavandi, "Contextual recurrent predictive model for long-term intent prediction of vulnerable road users," *IEEE Trans. on Intelligent Transportation Systems*, 2019.

[32] ——, "Cyclist trajectory prediction using bidirectional recurrent neural networks," in *Australasian Joint Conf. on A.I.*, 2018, pp. 284–295.

[33] G. Raipuria, F. Gaisser, and P. P. Jonker, "Road infrastructure indicators for trajectory prediction," in *IEEE Intelligent Vehicles Symposium (IV)*, 2018, pp. 537–543.

[34] S. Neogi, M. Hoy, K. Dang, H. Yu, and J. Dauwels, "Context model for pedestrian intention prediction using factored latent-dynamic conditional random fields," *IEEE Trans. on Intelligent Transportation Systems*, 2020.

[35] A. Sadeghian, V. Kosaraju, A. Sadeghian, N. Hirose, H. Rezatofighi, and S. Savarese, "SoPhie: An attentive GAN for predicting paths compliant to social and physical constraints," in *IEEE Conf. on Comp. Vision and Patt. Recog. (CVPR)*, June 2019.

[36] N. X. Vinh, M. Chetty, R. Coppel, and P. P. Wangikar, "GlobalMIT: learning globally optimal dynamic bayesian network with the mutual information test criterion," *Bioinformatics*, vol. 27, no. 19, pp. 2765–2766, 2011.

[37] K. P. Murphy, "Switching Kalman filters," *Technical report, Department of Computer Science, UC Berkeley*, 1998.

[38] ——, "Dynamic Bayesian networks: representation, inference and learning," Ph.D. dissertation, University of California, Berkeley, 2002.

[39] A. Paszke *et al.*, "Automatic differentiation in PyTorch," in *Conf. on Neural Information Proc. Syst. (NIPS)*, 2017.

[40] S. J. Reddi, S. Kale, and S. Kumar, "On the convergence of Adam and beyond," in *International Conf. on Learning Representations*, 2018.

[41] J. F. Kooij, G. Englebienne, and D. M. Gavrila, "Mixture of switching linear dynamics to discover behavior patterns in object tracks," *IEEE Trans. on Pattern Anal. and Mach. Intelligence (TPAMI)*, vol. 38, no. 2, pp. 322–334, 2015.

[42] T. Phan-Minh, E. C. Grigore, F. A. Boulton, O. Beijbom, and E. M. Wolff, "Covernet: Multimodal behavior prediction using trajectory sets," in *IEEE Conf. on Comp. Vision and Patt. Recog. (CVPR)*, 2020, pp. 14 074–14 083.

[43] Y. Huang, H. Bi, Z. Li, T. Mao, and Z. Wang, "Stgat: Modeling spatial-temporal interactions for human trajectory prediction," in *IEEE Conf. on Comp. Vision and Patt. Recog. (CVPR)*, 2019, pp. 6272–6281.

[44] M. F. Chang *et al.*, "Argoverse: 3D tracking and forecasting with rich maps," in *IEEE Conf. on Comp. Vision and Patt. Recog. (CVPR)*, 2019.

[45] J. F. P. Kooij, N. Schneider, F. Flohr, and D. M. Gavrila, "Context-based Pedestrian Path Prediction," in *Eur. Conf. on Comp. Vis. (ECCV)*, 2014, pp. 618–633.

[46] M. Luber, J. A. Stork, G. D. Tipaldi, and K. O. Arras, "People tracking with human motion predictions from social forces," in *IEEE Int. Conf. on Robotics and Automation (ICRA)*, 2010, pp. 464–469.

[47] M. Abadi *et al.*, "TensorFlow: Large-scale machine learning on heterogeneous systems," *USENIX Symp. on OS Design and Implement.*, 2016.

Ewoud Pool received the Master's degree in System and Control engineering at the TU Delft in 2015. Since 2016, he works on obtaining the Ph.D. degree at the TU Delft, focusing on path prediction of vulnerable road users for use in automated vehicles. His research interests include the modeling of human dynamics, perception for automated vehicles, and machine learning.

Julian Kooij obtained the Ph.D. degree in artificial intelligence at the Univ. of Amsterdam in 2015, where he worked on unsupervised machine learning and predictive models. In 2013 he worked at Daimler AG on path prediction of vulnerable road users for highly-automated vehicles. In 2014 he joined the computer vision lab of the TU Delft, and since 2016 he is an Assistant Professor in the Intelligent Vehicles group at the same university. His research interests include novel probabilistic models and machine learning techniques to infer and anticipate critical traffic situations from multi-modal sensor data.

Dariu M. Gavrila received the Ph.D. degree in computer science from Univ. of Maryland at College Park, USA, in 1996. From 1997 until 2016, he was with Daimler R&D, Ulm, Germany, where he became a Distinguished Scientist. In 2016, he moved to TU Delft, where he since heads the Intelligent Vehicles group as a Full Professor. His research deals with sensor-based detection of humans and analysis of behavior, most recently in the context of the self-driving car in complex urban traffic. He was awarded the Outstanding Application Award 2014 and the Outstanding Researcher Award 2019, both from the IEEE Intelligent Transportation Systems Society.