

Received March 10, 2021, accepted April 7, 2021, date of publication April 9, 2021, date of current version April 19, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3072135

A Two-Block RNN-Based Trajectory Prediction From Incomplete Trajectory

RYO FUJII¹, JAYAKORN VONGKULBHISAL², RYO HACHIUMA¹, (Student Member, IEEE), AND HIDEO SAITO¹, (Senior Member, IEEE)

¹Department of Science and Technology, Keio University, Yokohama 223-8522, Japan

²IBM Research, Tokyo 103-8510, Japan

Corresponding author: Ryo Fujii (ryo.fujii0112@keio.jp)

This work was supported by JST-Mirai Program, Japan, under Grant JPMJMI19B2.

ABSTRACT Trajectory prediction has gained great attention and significant progress has been made in recent years. However, most works rely on a key assumption that each video is successfully preprocessed by detection and tracking algorithms and the complete observed trajectory is always available. However, in complex real-world environments, we often encounter miss-detection of target agents (e.g., pedestrian, vehicles) caused by the bad image conditions, such as the occlusion by other agents. In this paper, we address the problem of trajectory prediction from incomplete observed trajectory due to miss-detection, where the observed trajectory includes several missing data points. We introduce a two-block RNN model that approximates the inference steps of the Bayesian filtering framework and seeks the optimal estimation of the hidden state when miss-detection occurs. The model uses two RNNs depending on the detection result. One RNN approximates the inference step of the Bayesian filter with the new measurement when the detection succeeds, while the other does the approximation when the detection fails. Our experiments show that the proposed model improves the prediction accuracy compared to the three baseline imputation methods on publicly available datasets: ETH and UCY (9% and 7% improvement on the ADE and FDE metrics). We also show that our proposed method can achieve better prediction compared to the baselines when there is no miss-detection.

INDEX TERMS Trajectory prediction, recurrent neural network, Bayesian filter, miss-detection.

I. INTRODUCTION

Predicting future trajectory from video data is an indispensable technology for developing navigation systems that can be used in several scenarios, such as self-driving vehicles, social robots, and navigation systems for blind people. High-quality predictions guide the user to the appropriate path and avoid dangerous situations (e.g., collision). The most common setting of trajectory prediction is the surveillance setting from a fixed camera, where the position of the agent (e.g., pedestrian, vehicles) is often treated as a single point [1], [2].

Many approaches to trajectory prediction forecast the future state of the target agent conditioned on the history of past states [7]–[10]. The state of the agent is often represented as its position. To obtain the history of a position of the agents, we need to detect where the agents are in the current and past time steps (e.g., object detection) and to establish object

correspondences between time steps (e.g., object tracking). Therefore, the trajectory prediction task is a downstream task of object detection and object tracking, and greatly depends on the performance of the upstream tasks. Many trajectory prediction works rely on a key assumption that each sequence is successfully preprocessed by object detection and tracking algorithms, and that they can always access the complete trajectory for trajectory prediction. In other words, the detection algorithm has to successfully detect all of the agents in the scene, and the tracking algorithm has to successfully track all of the detected agents. In details, most trajectory prediction research from bird's-eye view images assumes that all positions of all pedestrians can be obtained during 6.4 or 8 seconds: observing the trajectory for 3.2 seconds and predicting the future trajectory for 3.2 or 4.8 seconds [1], [2]. However, this is not feasible for real-world applications. There often emerge cases wherein an object is miss-detected due to bad image condition such as motion blurs, illumination changes, occlusion by other agents, and cluttered backgrounds [11], as shown in Fig. 2. When the target agent is not detected

The associate editor coordinating the review of this manuscript and approving it for publication was Ramakrishnan Srinivasan¹.

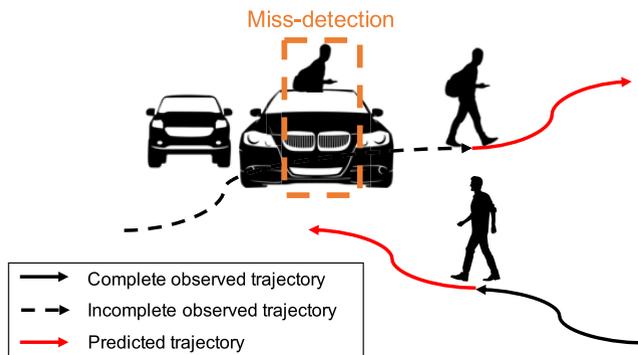


FIGURE 1. Our goal is to predict the future trajectory of agents from incomplete trajectory because of miss-detection. We develop a model which does not affect the trajectory prediction result from complete trajectory.

in several frames during the observation time, the future trajectory cannot be predicted against incomplete observed trajectory (Fig. 1).

The common solution for dealing with incomplete observed trajectory is to ignore these cases from the dataset as outliers. The agent who disappears even in one frame in a sequence is excluded from the dataset to handle incomplete data [1], [2], [12]. However, in real-world situations, an intelligent system must be able to continuously predict the future trajectory of all agents. This exclusion could cause the system to ignore the interaction between the excluded target agent and other agents. Furthermore, the system cannot consider the possibility of collision with the excluded agents and it could cause dangerous accidents. To avoid the exclusion of an undetected agent, we could attempt to impute the missing state of the agent. One possible solution is to impute a previously observed state when the miss-detection occurs [13], [14]. This can prevent the data exclusion of the agent whose state is not available. However, this can cause the model to interpret the data as the person keeping the previous state. Thus, compensating the states with an assumed value can lead to high error. In this paper, we investigate the problem of trajectory prediction from incomplete observations particularly due to miss-detection.

Bayesian filter-based methods [15] (particularly, Kalman filter [16]) are among the traditional approaches for trajectory prediction [17] and they are often used as baseline methods for comparison [1]. Bayesian filters recursively update the posterior distribution of predictions with the arrival of new data. The filtering process can be described as a cycle of two steps, the prediction and the update step. Due to simple structure, they do not often perform well on long-term prediction [18]. With the ability to learn and produce long sequences, Recurrent Neural Networks (RNNs), and in particular the Long Short-Term Memory (LSTM) networks [19], have recently become a widely popular modeling approach for predicting human motion [1], [20]–[28]. Recently, the connection between Bayesian filters and RNNs has been studied, and the observation that Bayesian filters are a special type of

RNNs has been proposed [29]. Inspired by the connection, we hypothesized that using the assumed state (e.g., the last observed state) for the missing time step causes the RNNs to fail to update their hidden state.

Using this intuition, to avoid wrong updates of the hidden state, we propose a simple two-block modification in which we add a new RNN block for the missing time step. We evaluate our two-block mechanism on an existing trajectory prediction model from a bird's-eye view against three baseline methods and show noticeable improvement to trajectory prediction from incomplete trajectories. Our proposed method does not affect trajectory prediction results from complete trajectories, even if the model is trained with incomplete data. Our contributions are as follows:

- We propose a two-block RNN that learns the inference step of Bayesian filters for trajectory prediction from incomplete observed trajectory due to miss-detection.
- We show that our model can outperform three baselines and that it performs better compared to the best baseline by 12% (ADE) and 4% (FDE) on ETH [5] and UCY [6]. Our model trained by incomplete trajectory does not affect the trajectory prediction result from the complete data.

II. RELATED WORK

The problem of trajectory prediction has received significant attention in recent years across various applications, such as self-driving vehicles, service robots, and advanced surveillance systems. A large body of research has addressed this problem. Many approaches define an explicit dynamical model based on Newton's laws of motion and use them as building blocks of a Bayesian filter (particularly, Kalman filter) [8], [18], [38], [39]. Many other works have investigated how to incorporate the concept of a rational agent when modeling human motions. Recently, a number of works explore approaches to approximate motion dynamics from training data using deep neural networks [1], [2], [20]–[28], [40]–[44]. In particular, Recurrent Neural Networks (RNNs) have recently become widely popular for modeling the dynamics approach [1], [20]–[28]. These sequential data-driven approaches assume N^{th} -order Markov models in which a limited state (e.g., position, velocity) history of N time steps is a sufficient representation of the entire state history. In this section, we only review RNN-based trajectory prediction literature. For a more extensive review, we refer the reader to the article in [45].

A. RNNs FOR TRAJECTORY PREDICTION

Trajectory prediction has been studied extensively in surveillance settings from bird's-eye views. Many studies have developed models to account for agent social interactions and social convention, in addition to scene semantics that may affect the trajectory. Social-LSTM [1] is a pioneer work to model pedestrian trajectories as well as their interactions in continuous space. It introduces the social pooling layer, which allows the LSTMs to share the hidden states of the

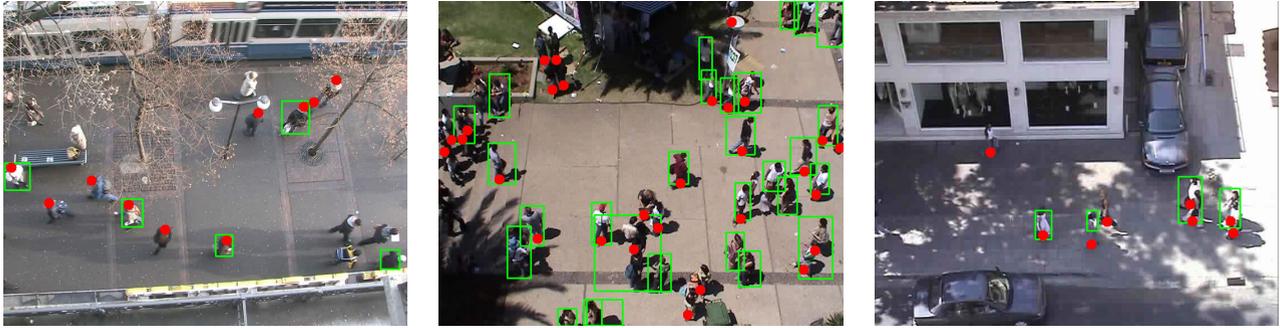


FIGURE 2. Examples of miss-detection annotated by Faster R-CNN [3] pretrained with MS-COCO dataset [4] from ETH [5] and UCY [6] datasets. The red points indicate the ground-truth positions attached to each dataset and green boxes indicate the predictions. Existing works on trajectory prediction often ignore miss-detected agents.

TABLE 1. A survey of approaches for handling missing entries in RNN-based time-series prediction tasks. See Sec. II-B for details.

Approach	Method to handle missing entries					Target application	
	Drop data	Impute data for the encoding stage			Use modified RNNs		
		Last Filling	Zero Filling	Learning-based	Internal		External
Alahi et al. [1]	✓					Trajectory prediction	
Gupta et al. [2]	✓					Trajectory prediction	
Styles et al. [12]	✓					Trajectory prediction	
Yao et al. [13]		✓				Traffic accident detection	
Malla et al. [14]		✓				Trajectory prediction	
Lipton et al. [30]		✓	✓			Medical data analysis	
Kim et al. [31]				✓		Medical data analysis	
Che et al. [32]		✓			✓	Medical data analysis	
Cao et al. [33]				✓	✓	General	
Tian et al. [34]				✓	✓	Traffic flow prediction	
Kim et al. [35]				✓	✓	Medical data analysis	
Luo et al. [36]				✓	✓	General	
Luo et al. [37]				✓	✓	General	
Ours						✓ Trajectory prediction	

agents that are nearby. Many methods [2], [23], [42], [46] follow the problem formulation used in [1], including the assumption that each scene is first processed to obtain the spatial coordinates of all people at different time instants. Recently, to achieve multi-modality in the prediction output, many works combine RNN and Generative Adversarial Networks (GANs) [2], [23], [43]. Gupta et al. [2] proposed Social GAN with a new pooling mechanism that does take into account social interactions between all people in the scene and a variety loss that encourages the network to produce diverse predictions. Sophie [23] proposed an LSTM-based GAN with two attention mechanisms. Kosaraju et al. [43] proposed Social-BiGAT, which utilizes GAN with graph attention networks (GAT) that captures the social interaction.

B. HANDLING MISSING DATA FOR RNN

In order to use RNNs for time-series prediction tasks, the observed data must be encoded by the RNNs before the prediction can be made. However, missing entries in the observed data are unsuitable for encoding. Many approaches have been developed to address this issue (see Table 1). If the task involves multiple time series (e.g., for trajectory prediction), a simple approach is to drop the missing data and perform the prediction using only the observed data [1],

[2], [12]. However, this strategy leads to loss of information and may not work if the missing rate is high or when there are individual time series to process (e.g., in medical applications). Another strategy is to impute the missing data with some default values and to perform the prediction over the imputed data. For example, the last observed values [13], [14], [30], [32] or zero [30] can be used to fill in the values of missing entries. Still, the filled values may cause the system to learn from them as if they are observed values, which could lead to lower performance. More recent works propose to use learning-based techniques to perform imputation, e.g., using RNNs [31], [33]–[35] or GANs [36], [37] to estimate the missing entries, and they generally also propose to modify the internal structure of the RNNs, for example, to include a decay mechanism [32], [34], [35] that puts different weights on data from different time steps. However, it may not be straightforward to apply the modification of one architecture to different architectures as this may involve different formulations or complicated implementations.

Unlike previous works that impute missing entries or modify internal structure of RNNs, in this work, we first look at the relation between Bayesian filter and RNNs, then derive an algorithm from the relation when there is a missing observation. This results in a simple-to-implement method that does

not modify any internal structure of RNNs and also does not require an explicit form of imputation. Instead, our approach modifies RNNs externally by using two RNNs instead of one. This external modification allows our approach to be used with any RNN models.

C. RNNs AND BAYESIAN FILTER

Recently, the relationship between Bayesian filter and RNNs has been discussed. Gu *et al.* [29] show Bayesian filter is a special type of RNN and propose an RNN-based model for a face landmark localization task in videos. Lim *et al.* [47] introduced Recurrent Neural Filter (RNF), which aligns network modules with the inference steps of the Bayesian filter. RNF uses separate neural network components to directly model the Bayesian filtering steps. In this paper, we align RNN-based trajectory prediction models with the Bayesian filtering steps and explore the architecture that is suitable for trajectory prediction from incomplete observed trajectory.

III. PRELIMINARIES

In this section, we briefly review the Bayesian filter [15] and its connection with RNNs. The Bayesian filter has been used in a wide range of applications, including target tracking [48], robotics [49], and economics [50]. The goal of the Bayesian filter is to find the probable posterior distribution of the hidden state x_t at time instant t , given all of the measurements $z_{1:t}$, which is characterized by $p(x_t|z_{1:t})$. We use $z_{1:t}$ to denote the sequence of measurements up to time instant t . Suppose there is a dynamical system represented by the following equations:

$$x_t = f_t(x_{t-1}, u_t), \quad (1)$$

$$z_t = h_t(x_t, v_t), \quad (2)$$

where u_t and v_t are the system process noise and observation noise, respectively. Both are assumed to have known probability distributions. The functions f and h are the state transition function and the observation function, which can be represented in a probabilistic form as $p(x_t|x_{t-1})$ and $p(z_t|x_t)$, respectively. When the state transition and the observation functions are linear and the process and measurement noises are Gaussian, the Bayesian filter becomes the Kalman filter.

The Bayesian filter consists of recursive prediction and the update steps. In the prediction step, it predicts a prior distribution of the current state x_t based on an old estimate using the state transition function, which is characterized by $p(x_t|z_{1:t-1})$. In the update step, it updates prior state distribution to obtain a posterior estimate with new arrival measurements, which is characterized by $p(x_t|z_{1:t})$.

A. PREDICTION STEP

The prior distribution $p(x_t|z_{1:t-1})$ of the state is obtained using the Chapman-Kolmogorov equation and the state transition function,

$$p(x_t|z_{1:t-1}) = \int p(x_t|x_{t-1})p(x_{t-1}|z_{1:t-1})dx_{t-1}. \quad (3)$$

Bayesian filter

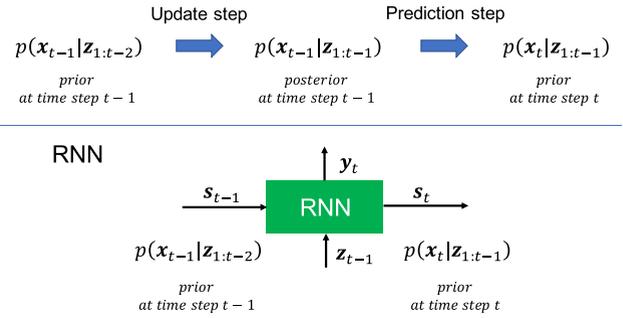


FIGURE 3. Relationship between Bayesian filter and RNNs. Given a sequence of measurement z_t , Bayesian filter recursively estimate the optimal hidden states x_t through two steps: prediction and update step. Similarly, given a sequence of input z_t , RNN updates the hidden state s_t every time step.

B. UPDATING STEP

When the new observation z_t is obtained, the prior distribution $p(x_t|z_{1:t-1})$ is updated according to Bayes' rule to estimate the posterior distribution $p(x_t|z_{1:t})$,

$$p(x_t|z_{1:t}) = \frac{p(z_t|x_t)p(x_t|z_{1:t-1})}{p(z_t|z_{1:t-1})}, \quad (4)$$

where $p(z_t|z_{1:t-1})$ can be expressed as,

$$p(z_t|z_{1:t-1}) = \int p(z_t|x_t)p(x_t|z_{1:t-1})dx_t. \quad (5)$$

RNNs bear resemblance to the Bayesian filters (see Fig. 3). Given a sequence of measurement, a Bayesian filter recursively estimates the optimal hidden states through two steps: prediction and update step, and optionally produce the target output every time step. Similarly, given a sequence of input, RNN updates the hidden state via a recurrent formula and optionally produces the output every time step.

The computation of RNNs is represented by the following equations,

$$s_t = \varphi_s(W_{ss}s_{t-1} + W_{zs}z_{t-1} + b_s), \quad (6)$$

$$y_t = \varphi_o(W_{sy}s_t + b_o), \quad (7)$$

where s_t represents the hidden state at time step t , φ_s and φ_o are activation functions, W_{ss} is the hidden-to-hidden transformation matrix, W_{zs} is the input-to-hidden transformation matrix, y_t is the output, W_{sy} is the hidden-to-output transformation matrix, and b_s and b_o are the bias terms.

Gu *et al.* [29] show Bayesian filters are a special type of RNNs with adaptive weights. While a Bayesian filter adapts its estimation models over time by changing weight, an RNN uses the fixed weight after training. Following this study, we assume an RNN performs the same procedure in updating the hidden state (6) as do the two steps of the Bayesian filter.

The simple solution with which to deal with incomplete measurements in the Bayesian filter is an imputation. If we can impute missing data, we can use the Bayesian filtering steps in the same way. However, in the update step, the prior distribution is updated with the imputed value in (4) and this might have a bad influence on the estimation of the

hidden state. Furthermore, the model in which parameters are learned with imputed assumed data will affect the update step with actual measurements. Therefore, we explore the method that does not rely on imputation.

IV. PROPOSED APPROACH

A. PROBLEM DEFINITION

We assume that the state of an agent is represented by its position and each scene is preprocessed by detection and tracking algorithms to obtain the position of each agent at each time instance. However, due to miss-detection, sometimes the obtained observed trajectory is incomplete. This includes the cases in which an agent is continuously detected in a sequence of frames but is not detected at the frame in the middle. The position at time step t is denoted as z_t , e.g., its 2D image coordinate in a bird's-eye view. When miss-detection does not occur, we can receive the complete trajectory $z_1, \dots, z_{T_{obs}}$. On the other hand, when miss-detection occurs, we can only access the incomplete observed trajectory. Our goal is to predict the probable future trajectory $z_{T_{obs}+1}, \dots, z_{T_{pred}}$ even from an incomplete observed trajectory. T_{obs} and T_{pred} denote the last observation and the last prediction time instance.

B. BAYESIAN FILTER WITH MISS-DETECTION

In the Bayesian filter, the future trajectory of the agent can be predicted with the appropriate hidden state and the observation functions. Thus, we can formulate the trajectory prediction task as the estimation of the hidden state x_t , from which an output can be optionally derived, from all the measurements $z_{1:t-1} = [z_1, \dots, z_{t-1}]$, which is represented by a probabilistic form: $p(x_t | z_{1:t-1})$. However, in the case of miss-detection, we do not have access to all of z_t , $t = 1, \dots, T_{obs}$. To prevent the ambiguity in the notation, let us define $\tilde{z}_{1:T_{obs}}$ to be the list of observed measurements until time step T_{obs} . Note that $\tilde{z}_{1:T_{obs}}$ differs from $z_{1:T_{obs}}$ since $z_{1:T_{obs}}$ assumes all z_t up to time T_{obs} to be observed, while for $\tilde{z}_{1:T_{obs}}$ some z_t may be missing. With this notation, our goal becomes the estimation of the hidden state x_t from the incomplete data $\tilde{z}_{1:t-1}$, which is represented in probabilistic form as $p(x_t | \tilde{z}_{1:t-1})$, and can be used to predict the future trajectory.

In this case, one cycle of prediction and update step can be represented by the following derivation,

$$p(x_t | \tilde{z}_{1:t-1}) = \int p(x_t | x_{t-1}) p(x_{t-1} | \tilde{z}_{1:t-1}) dx_{t-1}, \quad (8)$$

$$= \int p(x_t | x_{t-1}) p(x_{t-1} | \tilde{z}_{1:t-2}, z_{t-1}) dx_{t-1}, \quad (9)$$

$$= \frac{\int p(x_t | x_{t-1}) p(x_{t-1} | \tilde{z}_{1:t-2}) p(z_{t-1} | x_{t-1}) dx_{t-1}}{p(z_{t-1} | \tilde{z}_{1:t-2})}, \quad (10)$$

$$= \frac{\int p(x_t | x_{t-1}) p(x_{t-1} | \tilde{z}_{1:t-2}) p(z_{t-1} | x_{t-1}) dx_{t-1}}{\int p(z_{t-1} | x_{t-1}) p(x_{t-1} | \tilde{z}_{1:t-2}) dx_{t-1}}, \quad (11)$$

where in (9) we split the observed z_{t-1} from $\tilde{z}_{1:t-1}$ in (8), then we have applied the Bayes' rule to obtain (10).

Here, the prior distribution $p(x_{t-1} | \tilde{z}_{1:t-2})$ at time step $t-1$ is updated with the new observation z_{t-1} to obtain the posterior distribution $p(x_{t-1} | \tilde{z}_{1:t-1})$ at time step $t-1$. We can then estimate the prior distribution $p(x_t | \tilde{z}_{1:t-1})$ at time step t from this posterior distribution $p(x_{t-1} | \tilde{z}_{1:t-1})$ at time step $t-1$. Notice that (11) gives us a recurrent relation of computing $p(x_t | \tilde{z}_{1:t-1})$ as a function of $p(x_{t-1} | \tilde{z}_{1:t-2})$ and z_{t-1} , which we will use as the foundation of our two-block model in Section IV-C.

To utilize the above mechanism when the miss-detection occurs, i.e., the new measurement z_{t-1} is not available, we conventionally need to synthetically generate a measurement by imputation for the update step [13], [14]. However, updating the prior distribution with a synthetically generated measurement might have a bad influence on the estimation of hidden state x_t . To avoid the wrong update, we estimate the hidden state x_t from the measurements $\tilde{z}_{1:t-2}$ instead of compensating for the missing data with synthetically generated data,

$$p(x_t | \tilde{z}_{1:t-1}) = p(x_t | (\tilde{z}_{1:t-2}, z_{t-1} \text{ not observed})), \quad (12)$$

$$= p(x_t | \tilde{z}_{1:t-2}), \quad (13)$$

$$= \int p(x_t | x_{t-1}) p(x_{t-1} | \tilde{z}_{1:t-2}) dx_{t-1}. \quad (14)$$

We can see that the prior distribution $p(x_t | \tilde{z}_{1:t-1})$ at time step t can be directly estimated from the prior distribution $p(x_{t-1} | \tilde{z}_{1:t-2})$ at time step $t-1$ without the update step. This elimination of the update step prevents the model from accumulating the error that is caused by a wrong update every time step. Notice again that (14) provides a recurrent relation to compute $p(x_t | \tilde{z}_{1:t-1})$ as a function of $p(x_{t-1} | \tilde{z}_{1:t-2})$, but without the observation z_{t-1} . Using these recurrent relations, we can develop an algorithm for handling incomplete trajectories as described in the next section.

C. TWO-BLOCK MODEL FOR ENCODING MISS-DETECTION

Inspired by the connection between RNNs and the Bayesian filter [29], we apply the two above recurrent relations in the Bayesian filter with RNNs. Similar to the cases in the Bayesian filter, we use two RNNs depending on the detection result (see Fig. 4). One is used when the new measurement is available, and another is used when miss-detection occurs and the new measurement is not available for avoiding the wrong update. Fig. 5 shows a diagram of our two-block RNN model.

When the agent is successfully detected and a new measurement is available, we assume that one RNN works as a function that approximates a function approximating (11), which estimates the prior distribution at time step t from the prior distribution at time step $t-1$ and the new measurement,

$$s_t = g_c(s_{t-1}, z_{t-1}), \quad (15)$$

where g_c , implemented as an RNN, is a function that approximates the update and prediction steps with a new measurement in (11). Note the similarity between (11)

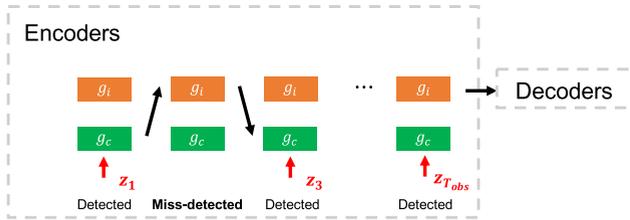


FIGURE 4. The flow of the proposed two-block RNN model. We use two functions, g_c and g_i for encoders depending on the object detection result. When miss-detection does not occur, we use g_c as an encoder, and when miss-detection occurs, we use g_i as an encoder.

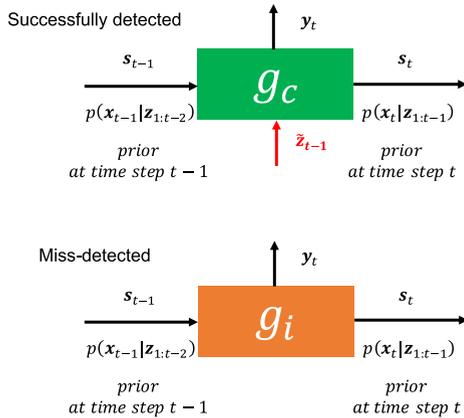


FIGURE 5. Two-block RNN diagram. The function g_c approximates the update and prediction steps with the new measurement, and the function g_i approximates the update and prediction steps without the new measurement.

and (15): the former is a recurrent relation between $p(x_t|\tilde{z}_{1:t-1})$, $p(x_{t-1}|\tilde{z}_{1:t-2})$, and z_{t-1} , while the latter is a recurrent relation between s_t , s_{t-1} , and z_{t-1} . Thus, we can interpret (15) as representing $p(x_t|\tilde{z}_{1:t-1})$ by s_t and g_c being a function that approximates the computation of the expression in (11).

When miss-detection occurs and the new measurement is not available, we assume another RNN works as a function approximating (14), which directly estimates the prior distribution at time step t from the prior distribution at time step $t - 1$ without a new measurement:

$$s_t = g_i(s_{t-1}), \quad (16)$$

where g_i , implemented as an RNN, is a function that approximates the prediction step without the new measurement and the update step, as in (14). Again, one could see a similar relationship between (14) and (16), which allows us to draw a similar interpretation to the case between (11) and (15).

D. PREDICTION

In the previous section, we encode the incomplete observed trajectory using the two-block RNN depending on the detection result. In order to predict the future trajectory $\hat{z}_{T_{obs}+1:T_{pred}}$ using the encoded information, we use another RNN:

$$s_t = g_p(s_{t-1}, \hat{z}_{t-1}), \quad (17)$$

$$\hat{z}_t = h_p(s_t), \quad (18)$$

Algorithm 1 Two-Block RNN Model for Future Trajectory Prediction

Input: Incomplete observed states of the agent $\tilde{z}_{1:T_{obs}}$

Output: Complete future states of the agent $\hat{z}_{T_{obs}+1:T_{pred}}$

- 1: **for** $t = 1$ to T_{obs} **do**
- 2: **if** the target agent is miss-detected **then**
- 3: $s_t = g_i(s_{t-1})$;
- 4: **else**
- 5: $s_t = g_c(s_{t-1}, z_{t-1})$;
- 6: **end if**
- 7: **end for**
- 8: **for** $t = T_{obs}+1$ to T_{pred} **do**
- 9: $s_t = g_p(s_{t-1}, \hat{z}_{t-1})$;
- 10: $\hat{z}_t = h_p(s_t)$;
- 11: **end for**
- 12: **return** $\hat{z}_{T_{obs}+1:T_{pred}}$

where g_p is a function that approximates a cycle of update and prediction steps, and h_p is a function that approximates an observation function that predicts the measurement from the hidden state (implemented as a Multilayer Perceptron (MLP) in our experiments). The pseudocode for the algorithm is provided in Alg. 1.

V. EXPERIMENTS

In this section, we perform experiments to evaluate our two-block RNN model against several imputation baseline approaches. All experiments are performed on a computer with an Intel Core Xeon CPU and an Nvidia Tesla K80 GPU. We use PyTorch [51] for our implementation.

A. EXPERIMENTAL SETTINGS

To evaluate our two-block RNN method, we experiment with the existing trajectory prediction model, Social GAN [2], a method to retrieve multiple possible future paths for multi-agents, on two publicly available datasets: ETH [5] and UCY [6]. We follow the same setting of Social GAN. For our approach, we only modify the encoders of Social GAN, which use a single agent encoder to encode the observed measurements of each agent independently into our two-block RNN.

1) BASELINES

We compare the performance of the proposed method with three baselines. The incomplete data due to miss-detection cannot be directly inputted into Social GAN. We firstly impute incomplete data for generating the synthetic complete data and then input the data to the model. We compare against the following imputation methods:

- **Last filling:** Impute the missing measurement with the last detected measurement.
- **Zero filling:** Impute the missing measurement with zero.
- **Linear filling:** Interpolate the missing measurement linearly using previous and next observed measurements. When the miss-detection happens at the end

TABLE 2. Quantitative results of all methods across different datasets. We present the results on two evaluation metrics (ADE and FDE).

Metric	Dataset	Trained to predict 8 future time steps					Trained to predict 12 future time steps				
		Last Fill.	Zero Fill.	Linear Fill.	Two-Block RNN (Ours)	Complete Data	Last Fill.	Zero Fill.	Linear Fill.	Two-Block RNN (Ours)	Complete Data
ADE	ETH	1.57	1.26	1.16	0.97	0.91	1.87	1.78	1.50	1.41	1.35
	HOTEL	4.41	4.70	3.57	3.49	3.10	5.12	4.84	4.60	4.56	6.63
	UNIV	1.17	1.33	1.11	1.02	0.95	1.61	2.04	1.66	1.53	1.40
	ZARA1	0.62	0.65	0.66	0.52	0.40	0.89	0.95	0.89	0.78	0.68
	ZARA2	0.68	0.71	0.78	0.52	0.40	1.00	1.07	1.10	0.83	0.63
	AVG	1.69	1.73	1.46	1.30	1.15	2.10	1.94	1.95	1.82	2.14
FDE	ETH	2.78	2.30	2.10	1.81	1.74	3.14	3.23	2.69	2.61	2.57
	HOTEL	5.17	5.86	4.66	4.39	3.89	5.94	5.61	5.00	5.50	8.46
	UNIV	2.06	2.18	1.92	1.82	1.67	2.92	3.35	2.97	2.80	2.56
	ZARA1	1.09	1.12	1.16	0.93	0.74	1.59	1.70	1.57	1.46	1.30
	ZARA2	1.21	1.25	1.43	0.95	0.74	1.98	2.03	2.14	1.56	1.23
	AVG	2.46	2.54	2.25	1.98	1.76	3.11	3.18	2.87	2.79	3.22

of the observation time, we extrapolate the missing measurement linearly.

2) IMPLEMENTATION DETAILS

Social GAN uses LSTMs as the RNNs in their model for encoders and sets 32 as the dimension of the hidden states for the encoder. We also use the LSTMs in our two-block RNN model. We halved the size of the hidden state of our two-block model, which consists of two RNNs for fair comparison in terms of the number of parameters. For other settings, we followed the original Social GAN.

As for the representation of measurement, Social GAN utilizes relative coordinates for translation invariance. However, when miss-detection occurs and the position at a time step is not available, we cannot compute both the previous and next relative coordinates around the time step. Therefore, we use absolute coordinates instead of relative coordinates. We normalize the pedestrian positions by subtracting the mean and dividing by the standard deviation of the training set.

3) DATASETS

We evaluate our two-block RNN model on two datasets: ETH [5] and UCY [6], which are commonly used for the trajectory prediction task. The ETH dataset contains two scenes named ETH and HOTEL. UCY dataset includes three scenes named ZARA-01, ZARA-02, and UCY. They are collected from bird's-eye views containing thousands of real-world pedestrian trajectories, and covering numerous challenging situations. We observe trajectories for 3.2 seconds (8 frames) and predict for 3.2 seconds (8 frames) or 4.8 seconds (12 frames), and use a leave-one-out approach, i.e., train on four sets and test on the remaining set for evaluation.

Unfortunately, ETH and UCY do not provide a miss-detection label. Therefore, we synthetically generate miss-detection masks. To generate the miss-detection masks, we randomly choose a miss-detection ratio from [0.2, 0.8] for every sequence both in training and evaluation.

4) EVALUATION METRICS

We follow the prior works [1], [2] for evaluation metrics. We use the Average Displacement Error (ADE) and the Final Displacement Error (FDE) metrics. ADE is the average $L2$ distance between the predictions and the ground truth overall predicted time step, and the FDE is the $L2$ distance between the prediction and ground truth of final destination.

B. EXPERIMENTAL RESULTS

1) MAIN RESULTS

In Table 2, we evaluate our model against all baseline models. We see that the linear imputation baseline that linearly estimates the missing state outperforms the other baselines, which do not estimate the missing state. Our two-block RNN model outperforms baselines on almost all of the datasets on both metrics. The best result among the baselines on the ADE metric is linear imputation with an error of 1.46 ($T_{pred} = 8$) and 1.95 ($T_{pred} = 12$). Our model has the ADE error of 1.30 ($T_{pred} = 8$) and 1.82 ($T_{pred} = 12$), which is 11% and 6% less than the best baseline, respectively, and the effect becomes the most prominent on ZARA2 (33% in $T_{pred} = 8$ and 25% in $T_{pred} = 12$). In terms of the FDE metric, we can also get the improvement of 12% ($T_{pred} = 8$) and 2% ($T_{pred} = 12$), compared to the best baseline.

2) CHANGING MISS-DETECTION RATIO

We compare the robustness of the two-block RNN model and baselines when the miss-detection ratio changes. We test the model, which is trained with a random miss-detection ratio chosen from [0.2, 0.8] under various fixed miss-detection ratios. In Fig. 6, we vary the miss-detection ratio, from 0 (complete data) to 0.7. Overall, our two-block model can get better results compared to baselines in every miss-detection ratio setting. The performance becomes better as the miss-detection ratio becomes lower. When no miss-detection occurs, i.e., the miss-detection ratio is 0, our model achieves the best performance compared to the baseline model on both ADE and FDE metrics, in both 8 and 12

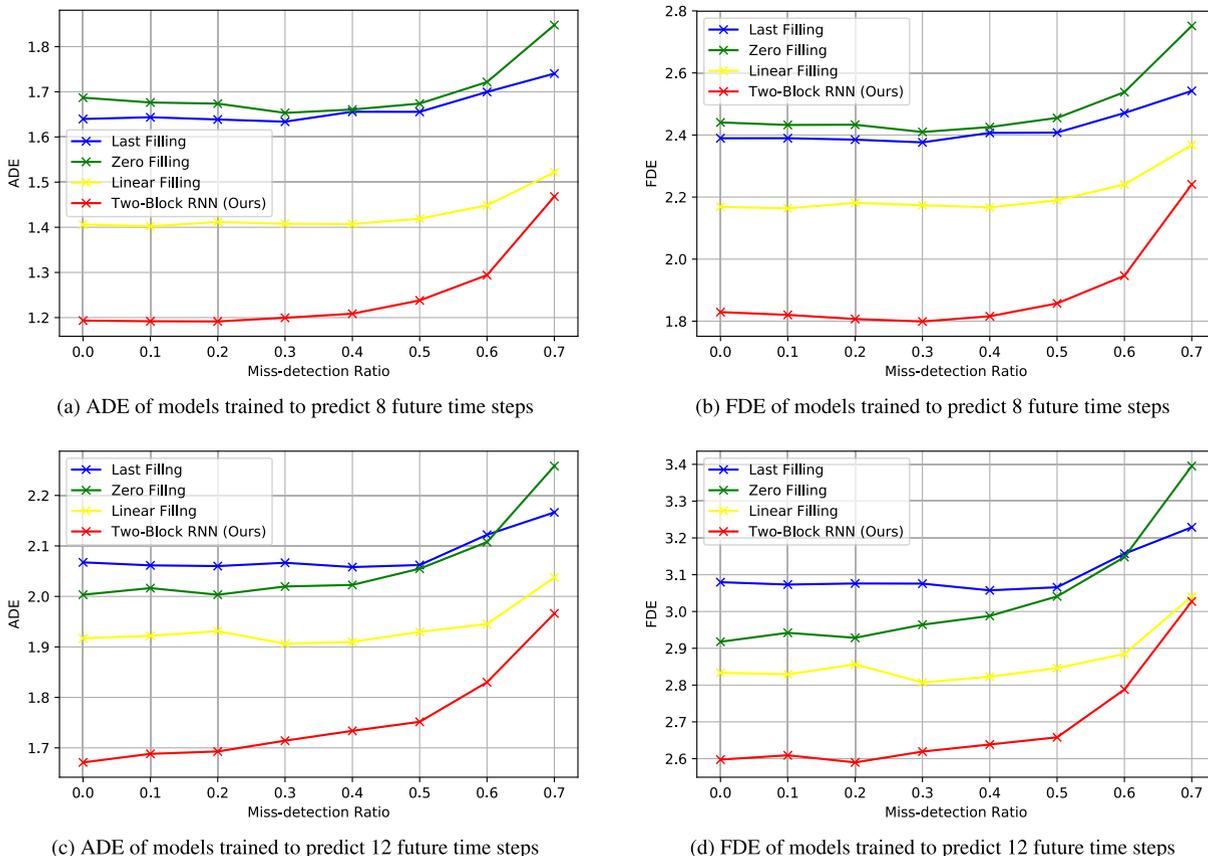


FIGURE 6. We compare the robustness of our proposed model and baselines against the miss-detection ratio changes on ADE and FDE metrics. Our two-block RNN model outperforms the baselines in every miss-detection ratio. Our method also achieves superior performance when no miss-detection occurs. The results are averaged over all datasets.

prediction time-step settings. In the baselines, a single RNN is trained to encode both real data and imputed data. As imputed data may be confused with measurement, this could cause the model to learn the incorrect values, leading to high-error prediction even when there is no miss-detection in test time. In the 8-time-step prediction, our model outperforms the best baseline model, linear filling by 15% (ADE) and 15% (FDE). In 12-time-step prediction, our model achieves superior performance against the best baseline model, linear filling by 12% (ADE) and 8% (FDE). On the other hand, in our two-block RNN model, one RNN, namely, g_c is trained to encode the real data and another RNN, namely, g_i is trained to update the hidden state without a new measurement. Therefore, when no miss-detection occurs in test time, we can use g_c which is trained only with real data as encoders, and this leads to better performance compared to the baselines.

3) DISPLACEMENT ERROR (DE) AT EACH TIME STEP

In Fig. 7, we report the displacement error at each prediction time step of all methods. Our model performs better every time step in both 8 and 12 prediction time step settings. This result suggests better prediction quality over all the time instants.

4) CHANGING PREDICTION LENGTH

To show the stability in predicting longer temporal horizons, we present the ADE and FDE for the prediction of 12, 16,

and 20 future time steps in Table 4. Here, we use the same models trained to predict 8 and 12 time steps as in the main results, and simply extend their prediction time steps. The performance becomes worse as the prediction length becomes longer. Still, our model has a consistent advantage at every prediction time-step setting.

5) ALIGNING THE HIDDEN STATE CHANNELS

To provide a fair comparison of our model, in the main results, we halve the number of channels of the hidden state of our two-block model, which consists of two RNNs (each having 16 channels), so that the total number of channels of our two-block model is the same as that of the baselines (32 channels). In this section, we run an additional experiment in which we align and vary the number of channels of both our model and the baseline RNNs, so that all RNNs, both of our model and the baselines, have the same number of channels, in order to confirm that the performance of our model is not caused by the difference in the number of channels. The results are shown in Table 4. We can see that our model can still outperform the baselines. Hence, we show that the performance of our model is not caused by the difference in the number of hidden state channels.

6) COMPUTATION TIME

Table 5 shows the comparison of the computation time of our method and the baseline methods. The total time is broken

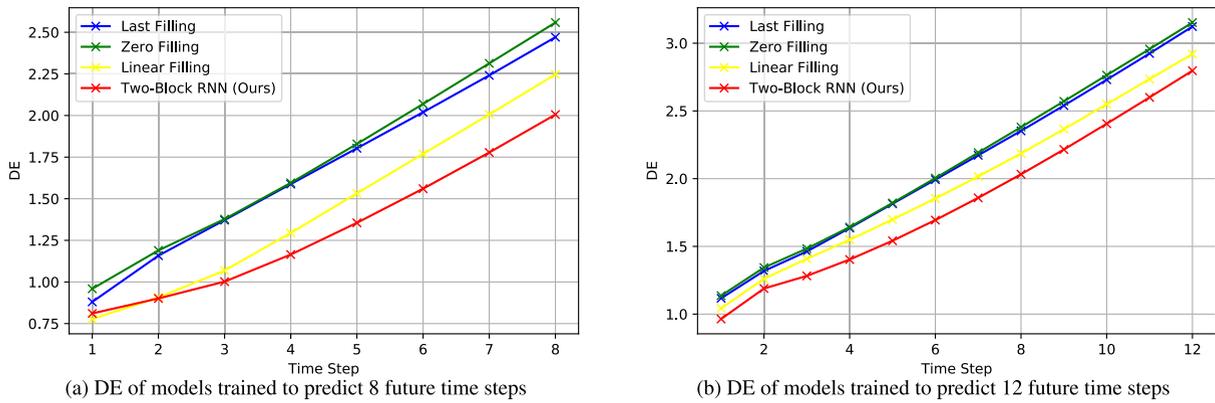


FIGURE 7. Displacement error at each time step. Our model provides better prediction quality over all time steps. The results are averaged over all datasets.

TABLE 3. Results of the evaluation on different prediction lengths from models trained to predict 8 and 12 future time steps. Best results are highlighted in bold. The results are averaged over all datasets.

Metric	Pred. Length	Trained to predict 8 future time steps				Trained to predict 12 future time steps			
		Last Fill.	Zero Fill.	Linear Fill.	Two-Block RNN (Ours)	Last Fill.	Zero Fill.	Linear Fill.	Two-Block RNN (Ours)
ADE	12	3.03	3.20	2.73	2.47	2.09	2.13	1.97	1.83
	16	4.85	5.04	4.47	3.97	3.33	3.19	3.02	2.81
	20	7.48	7.54	6.98	6.13	4.96	4.72	4.41	4.17
FDE	12	3.10	3.32	2.97	2.69	3.10	3.16	2.92	2.80
	16	3.95	4.13	3.84	3.37	3.82	3.70	3.50	3.32
	20	5.16	5.27	5.16	4.35	4.80	4.56	4.30	4.13

TABLE 4. Results for the experiment with varying number of hidden channels for the RNNs. Best results are highlighted in bold. The results are averaged over all datasets.

Metric	Num. Channels	Trained to predict 8 future time steps				Trained to predict 12 future time steps			
		Last Fill.	Zero Fill.	Linear Fill.	Two-Block RNN (Ours)	Last Fill.	Zero Fill.	Linear Fill.	Two-Block RNN (Ours)
ADE	16	1.48	1.58	1.56	1.30	2.10	2.18	2.37	1.82
	32	1.69	1.73	1.46	1.27	2.10	1.94	1.95	2.16
	64	1.79	1.46	1.52	1.28	2.14	2.03	1.93	1.81
FDE	16	2.23	2.41	2.40	1.98	2.88	3.13	3.67	2.79
	32	2.46	2.54	2.25	1.97	3.11	3.18	2.87	3.28
	64	2.63	2.18	2.30	2.00	3.16	2.97	2.93	2.80

TABLE 5. Computation time comparison (in milliseconds) for 8-time-step prediction. The means and the standard deviations of the computational time are reported.

Method	Computation time (milliseconds)			
	Filling	Encoding	Prediction	Total
Last Fill.	0.22 ± 0.07	0.49 ± 0.10	3.94 ± 0.55	4.66 ± 0.72
Zero Fill.	0.03 ± 0.01	0.49 ± 0.09	4.01 ± 0.53	4.55 ± 0.60
Linear Fill.	16.0 ± 17.5	0.47 ± 0.08	3.96 ± 0.37	20.3 ± 17.7
Ours	-	8.02 ± 0.86	3.72 ± 0.24	11.74 ± 0.98

down into the time for filling missing values, encoding the trajectories, and making predictions. Note that the baselines require the filling time to fill the missing values, while our method does not. In terms of total time, last filling and zero filling are the fastest, followed by our method and then by linear filling. Looking at the breakdown, we can see all methods

require roughly the same time for prediction, since they all use the same implementation. On the other hand, linear filling requires more time because it takes longer to compute the imputed values. Our method requires more time than the baselines do for encoding the trajectories. This is because our method requires selecting one LSTM from the two LSTM blocks to pass the data to in each time step,¹ and thus it does not receive the speed-up benefit from the optimized LSTM implementation used by the baselines. However, since the selection only requires an additional `if` statement, an optimized implementation of our two-block method should be able to achieve almost the same computation time as those of the baselines.

¹For our method’s implementation in this work, we simply put the two-block model in a `for` loop and iterate over the time steps.

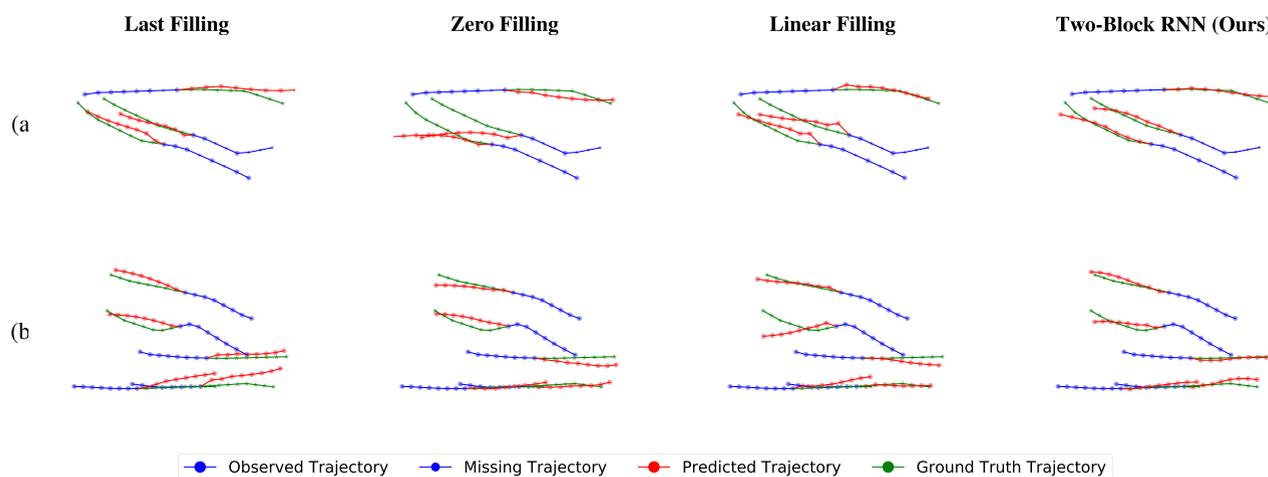


FIGURE 8. Qualitative comparison between our two-block RNN and the baseline models. (a) Trajectory prediction from observed trajectories with miss-detection and (b) trajectory prediction from the complete observed trajectories.

7) QUALITATIVE EVALUATION

As a demonstration, we visualize the prediction results of our two-block RNN and those of the baseline methods in Fig. 8. Here, we draw the average predicted trajectory of 100 samples. In Fig. 8(a), we can observe that the prediction of our two-block RNN is more accurate than those of the baselines. Moreover, even when the complete trajectories are available (Fig. 8(b)), our two-block RNN can still predict the trajectories that are closer to the ground truth.

VI. DISCUSSION

In this section, we discuss some limitations. To begin with, we used the absolute coordinates, instead of relative coordinates. To compute the relative coordinates, we need the current position and the next position. Thus, it is not possible to compute both the previous and the next relative coordinates at one time step, when there is missing data. However, using absolute coordinates leads to a bad influence on overall performance. It is especially the case that all models perform much more poorly than those trained with relative coordinates for the task of trajectory prediction from complete data in all datasets (64% worse in $T_{pred} = 8$ and 74% worse in $T_{pred} = 12$ on the ADE metric). If we could use relative coordinates for trajectory prediction with incomplete data, the performance might get better. In future work, we will work on this problem.

Next, we discuss other problems of trajectory prediction in real-world environments. We focus on predicting the future trajectory from incomplete observed trajectory due to miss-detection. Our model cannot handle the wrong detection, and tracking is out of focus, but if we can classify the detection or tracking result as valid or not, we can apply our model to these situations.

VII. CONCLUSION

In this paper, we address the problem of trajectory prediction from incomplete observed trajectory because of miss-detection. We proposed a two-block RNN-based model,

which takes advantage of the connection between the Bayesian filter and RNNs. Extensive experimental results on standard datasets, including ETH [5] and UCY [6], show that our approach outperforms baselines that are commonly used. Since we do not use imputed data for training, our model does not affect the performance of trajectory prediction from the complete observed trajectory.

REFERENCES

- [1] A. Alahi, K. Goel, V. Ramanathan, A. Robicquet, L. Fei-Fei, and S. Savarese, "Social LSTM: Human trajectory prediction in crowded spaces," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2016, pp. 961–971.
- [2] A. Gupta, J. Johnson, L. Fei-Fei, S. Savarese, and A. Alahi, "Social GAN: Socially acceptable trajectories with generative adversarial networks," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, Jun. 2018, pp. 2255–2264.
- [3] S. Ren, K. He, R. Girshick, and J. Sun, "Faster R-CNN: Towards real-time object detection with region proposal networks," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 39, no. 6, pp. 1137–1149, Jun. 2017.
- [4] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, "Microsoft COCO: Common objects in context," in *Proc. Eur. Conf. Comput. Vis. (ECCV)*, 2014, pp. 740–755.
- [5] S. Pellegrini, A. Ess, and L. Van Gool, "Improving data association by joint modeling of pedestrian trajectories and groupings," in *Proc. Eur. Conf. Comput. Vis. (ECCV)*, 2010, pp. 452–465.
- [6] L. Leal-Taixe, M. Fenzi, A. Kuznetsova, B. Rosenhahn, and S. Savarese, "Learning an image-based motion context for multiple people tracking," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jun. 2014, pp. 3542–3549.
- [7] J. F. P. Kooij, F. Flohr, E. A. I. Pool, and D. M. Gavrila, "Context-based path prediction for targets with switching dynamics," *Int. J. Comput. Vis.*, vol. 127, no. 3, pp. 239–262, Mar. 2019.
- [8] V. Karasev, A. Ayvaci, B. Heisele, and S. Soatto, "Intent-aware long-term prediction of pedestrian motion," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, May 2016, pp. 2543–2549.
- [9] S. Pellegrini, A. Ess, K. Schindler, and L. van Gool, "You'll never walk alone: Modeling social behavior for multi-target tracking," in *Proc. IEEE 12th Int. Conf. Comput. Vis.*, Sep. 2009, pp. 261–268.
- [10] K. M. Kitani, B. D. Ziebart, J. A. Bagnell, and M. Hebert, "Activity forecasting," in *Proc. Eur. Conf. Comput. Vis. (ECCV)*, 2012, pp. 201–214.
- [11] Y. Hosoya, M. Suganuma, and T. Okatani, "Analysis and a solution of momentarily missed detection for anchor-based object detectors," in *Proc. IEEE Winter Conf. Appl. Comput. Vis. (WACV)*, Mar. 2020, pp. 1399–1407.

- [12] O. Styles, T. Guha, and V. Sanchez, "Multiple object forecasting: Predicting future object locations in diverse environments," in *Proc. IEEE Winter Conf. Appl. Comput. Vis. (WACV)*, Mar. 2020, pp. 679–688.
- [13] Y. Yao, M. Xu, Y. Wang, D. J. Crandall, and E. M. Atkins, "Unsupervised traffic accident detection in first-person videos," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst. (IROS)*, Nov. 2019, pp. 273–280.
- [14] S. Malla, B. Dariush, and C. Choi, "TITAN: Future forecast using action priors," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2020, pp. 11183–11193.
- [15] J. V. Candy, *Bayesian Signal Processing: Classical, Modern and Particle Filtering Methods*. Hoboken, NJ, USA: Wiley, 2009.
- [16] R. E. Kalman, "A new approach to linear filtering and prediction problems," *J. Basic Eng.*, vol. 82, no. 1, pp. 35–45, Mar. 1960.
- [17] N. Schneider and D. M. Gavrila, "Pedestrian path prediction with recursive Bayesian filters: A comparative study," in *Proc. German Conf. Pattern Recognit.*, 2013, pp. 174–183.
- [18] A. Barth and U. Franke, "Where will the oncoming vehicle be the next second?" in *Proc. IEEE Intell. Vehicles Symp. (IV)*, Jun. 2008, pp. 1068–1073.
- [19] S. Hochreiter and J. Uger Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, 1997.
- [20] F. Bartoli, G. Lisanti, L. Ballan, and A. Del Bimbo, "Context-aware trajectory prediction," in *Proc. 24th Int. Conf. Pattern Recognit. (ICPR)*, Aug. 2018, pp. 1941–1946.
- [21] A. Vemula, K. Muelling, and J. Oh, "Social attention: Modeling attention in human crowds," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, May 2018, pp. 4601–4607.
- [22] N. Bisagno, B. Zhang, and N. Conci, "Group LSTM: Group trajectory prediction in crowded scenarios," in *Proc. Eur. Conf. Comput. Vis. (ECCV) Workshops*, 2019, pp. 213–225.
- [23] 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 *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2019, pp. 1349–1358.
- [24] K. Saleh, M. Hossny, and S. Nahavandi, "Intent prediction of pedestrians via motion trajectories using stacked recurrent neural networks," *IEEE Trans. Intell. Veh.*, vol. 3, no. 4, pp. 414–424, Dec. 2018.
- [25] L. Sun, Z. Yan, S. M. Mellado, M. Hanheide, and T. Duckett, "3DOF pedestrian trajectory prediction learned from long-term autonomous mobile robot deployment data," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, May 2018, pp. 5942–5948.
- [26] M. Pfeiffer, G. Paolo, H. Sommer, J. Nieto, R. Siegwart, and C. Cadena, "A data-driven model for interaction-aware pedestrian motion prediction in object cluttered environments," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, May 2018, pp. 5921–5928.
- [27] X. Shi, X. Shao, Z. Guo, G. Wu, H. Zhang, and R. Shibasaki, "Pedestrian trajectory prediction in extremely crowded scenarios," *Sensors*, vol. 19, no. 5, p. 1223, Mar. 2019.
- [28] P. Zhang, W. Ouyang, P. Zhang, J. Xue, and N. Zheng, "SR-LSTM: State refinement for LSTM towards pedestrian trajectory prediction," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2019, pp. 12077–12086.
- [29] J. Gu, X. Yang, S. De Mello, and J. Kautz, "Dynamic facial analysis: From Bayesian filtering to recurrent neural network," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jul. 2017, pp. 1531–1540.
- [30] Z. C. Lipton, D. Kale, and R. Wetzel, "Directly modeling missing data in sequences with RNNs: Improved classification of clinical time series," in *Proc. Mach. Learn. for Healthcare Conf. (MLHC)*, vol. 56, 2016, pp. 253–270.
- [31] H.-G. Kim, G.-J. Jang, H.-J. Choi, M. Kim, Y.-W. Kim, and J. Choi, "Recurrent neural networks with missing information imputation for medical examination data prediction," in *Proc. IEEE Int. Conf. Big Data Smart Comput. (BigComp)*, Feb. 2017, pp. 317–323.
- [32] Z. Che, S. Purushotham, K. Cho, D. Sontag, and Y. Liu, "Recurrent neural networks for multivariate time series with missing values," *Sci. Rep.*, vol. 8, no. 1, Dec. 2018, Art. no. 6085.
- [33] W. Cao, D. Wang, J. Li, H. Zhou, L. Li, and Y. Li, "BRITS: Bidirectional recurrent imputation for time series," in *Proc. Adv. Neural Inf. Process. Syst. (NeurIPS)*, vol. 31, 2018, pp. 6775–6785.
- [34] Y. Tian, K. Zhang, J. Li, X. Lin, and B. Yang, "LSTM-based traffic flow prediction with missing data," *Neurocomputing*, vol. 318, pp. 297–305, Nov. 2018.
- [35] Y. J. Kim and M. Chi, "Temporal belief memory: Imputing missing data during RNN training," in *Proc. 27th Int. Joint Conf. Artif. Intell.*, Jul. 2018, pp. 2326–2332.
- [36] Y. Luo, X. Cai, Y. Zhang, J. Xu, and Y. Xiaojie, "Multivariate time series imputation with generative adversarial networks," in *Proc. Adv. Neural Inf. Process. Syst. (NeurIPS)*, vol. 31, 2018, pp. 1603–1614.
- [37] Y. Luo, Y. Zhang, X. Cai, and X. Yuan, "E²GAN: End-to-end generative adversarial network for multivariate time series imputation," in *Proc. 28th Int. Joint Conf. Artif. Intell.*, Aug. 2019, pp. 1–7.
- [38] A. Elnagar, "Prediction of moving objects in dynamic environments using Kalman filters," in *Proc. IEEE Int. Symp. Comput. Intell. Robot. Autom.*, 2001, pp. 414–419.
- [39] T. Batz, K. Watson, and J. Beyerer, "Recognition of dangerous situations within a cooperative group of vehicles," in *Proc. IEEE Intell. Vehicles Symp.*, Jun. 2009, pp. 907–912.
- [40] N. Rhinehart, R. Mcallister, K. Kitani, and S. Levine, "PRECOG: PREDiction conditioned on goals in visual multi-agent settings," in *Proc. IEEE/CVF Int. Conf. Comput. Vis. (ICCV)*, Oct. 2019, pp. 2821–2830.
- [41] T. Zhao, H. Xu, M. Monfort, W. Choi, C. Baker, Y. Zhao, Y. Wang, and Y. N. Wu, "Multi-agent tensor fusion for contextual trajectory prediction," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2019, pp. 12118–12126.
- [42] Y. Huang, H. Bi, Z. Li, T. Mao, and Z. Wang, "STGAT: Modeling spatial-temporal interactions for human trajectory prediction," in *Proc. IEEE/CVF Int. Conf. Comput. Vis. (ICCV)*, Oct. 2019, pp. 6271–6280.
- [43] V. Kosaraju, A. Sadeghian, R. Martín-Martín, I. Reid, H. Rezatofighi, and S. and Savarese, "Social-BiGAT: Multimodal trajectory forecasting using bicycle-GAN and graph attention networks," in *Proc. Adv. Neural Inf. Process. Syst. (NeurIPS)*, vol. 32, 2019, pp. 137–146.
- [44] B. Ivanovic and M. Pavone, "The trajectron: Probabilistic multi-agent trajectory modeling with dynamic spatiotemporal graphs," in *Proc. IEEE/CVF Int. Conf. Comput. Vis. (ICCV)*, Oct. 2019, pp. 2375–2384.
- [45] A. Rudenko, L. Palmieri, M. Herman, K. M. Kitani, D. M. Gavrila, and K. O. Arras, "Human motion trajectory prediction: A survey," *Int. J. Robot. Res.*, vol. 39, no. 8, pp. 895–935, Jul. 2020.
- [46] J. Liang, L. Jiang, J. C. Niebles, A. Hauptmann, and L. Fei-Fei, "Peeking into the future: Predicting future person activities and locations in videos," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. Workshops (CVPRW)*, Jun. 2019, pp. 5718–5727.
- [47] B. Lim, S. Zohren, and S. Roberts, "Recurrent neural filters: Learning independent Bayesian filtering steps for time series prediction," in *Proc. Int. Joint Conf. Neural Netw. (IJCNN)*, Jul. 2020, pp. 1–8.
- [48] A. J. Haug, *Bayesian Estimation Tracking: A Practical Guide*. Hoboken, NJ, USA: Wiley, 2012.
- [49] T. D. Barfoot, *State Estimation for Robotics*, 1st ed. Cambridge, U.K.: Cambridge Univ. Press, 2017.
- [50] G. K. Pasricha, "Kalman filter and its economic applications," Univ. Library Munich, Munich, Germany, Tech. Rep., Oct. 2006, vol. 22734.
- [51] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "PyTorch: An imperative style, high-performance deep learning library," in *Proc. Adv. Neural Inf. Process. Syst. (NeurIPS)*, 2019, pp. 8024–8035.



RYO FUJII received the B.E. degree in information and computer science from Keio University, Japan, in 2020, where he is currently pursuing the M.Sc.Eng. degree in information and computer science. His research interests include machine learning and computer vision.



JAYAKORN VONGKULBHISAL received the B.Eng. degree in information and communication engineering from Chulalongkorn University, Bangkok, Thailand, in 2011, the M.Sc. degree in electrical and computer engineering from Carnegie Mellon University, Pittsburgh, PA, USA, in 2016, and the dual Ph.D. degree in electrical and computer engineering from Carnegie Mellon University and Instituto Superior Técnico, Lisbon, Portugal, in 2018. Since 2018, he has been a

Research Scientist with IBM Research, Tokyo, Japan. His research interest includes optimization and machine learning in computer vision.



HIDEO SAITO (Senior Member, IEEE) received the Ph.D. degree in electrical engineering from Keio University, Japan, in 1992. Since 1992, he has been with the Faculty of Science and Technology, Keio University. From 1997 to 1999, he joined the Virtualized Reality Project with the Robotics Institute, Carnegie Mellon University, as a Visiting Researcher. Since 2006, he has been a Full Professor with the Department of Information and Computer Science, Keio University.

...



RYO HACHIUMA (Student Member, IEEE) received the B.E. and M.Sc.Eng. degrees in information and computer science from Keio University, Japan, in 2016 and 2017, respectively, where he is currently pursuing the Ph.D. degree in science and technology. His research interests include 3D object recognition, 3D object tracking, and simultaneous localization and mapping.