Interactive Trajectory Prediction for Autonomous Driving via Recurrent Meta Program Induction Network

Chiyu Dong[1], Yilun Chen[2] and John M. Dolan[2]

Abstract— **Interactive driving is challenging but essential for autonomous cars in dense traffic or urban areas. A proper interaction requires understanding and prediction of future trajectories of all neighboring vehicles around a target vehicle. Current solutions typically assume a certain distribution or stochastic process to approximate human-driven cars' behaviors. To relax this assumption, a Meta Induction Network (IN) framework is developed. The original Conditional Neural Process (CNP) on which this is based does not consider the sequence of the conditions, due to the permutation invariance requirements for stochastic processes. However, the sequential information is important for the driving behavior estimation. Therefore, in the proposed method, a recurrent neural cell replaces the original demonstration sub-net. The behavior estimation is conditioned on the historical observations for all related cars, including the target car and its surrounding cars. The method is applied to predict the lane change trajectory of a target car in dense traffic areas. The proposed method achieves better results than previous methods and can use a smaller dataset, putting fewer demands on autonomous driving data collection.**

I. INTRODUCTION

As autonomous driving techniques advance, they face more complex scenarios than highway car following (i.e., Adaptive Cruise Control or ACC) and lane keeping. These simple scenarios only require the sensing of a single target vehicle's dynamic or static environment, where there is less ambiguity and fewer intention estimates are needed. However, in more complex interactive scenarios, such as lane change and ramp merging, the autonomous car needs to generate a path according to the interactions among other cars. In addition to robust perception and control, these scenarios require the autonomous car to interact appropriately with neighboring vehicles. In order to react properly, it is important to correctly understand human drivers' behaviors or intentions and then make a prediction of the human-driven cars' trajectory. For example, in a lane-change scenario, as Fig. 1 depicts, where the lane-changing trajectory of the target vehicle (Veh-s) highly depends on the surrounding vehicles' behaviors and recent movements. Therefore, in order to accurately predict the lane-change trajectory of one target vehicle, all surrounding trajectories and their interactions should be taken into account. Once knowing other vehicles' intentions and future trajectories' estimation,

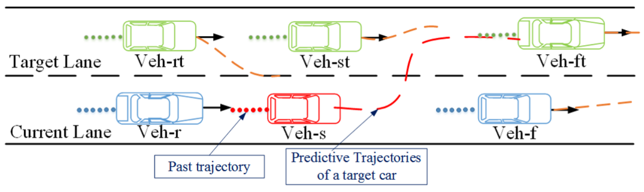

Fig. 1: A left lane change scenario. Veh-s is the target car whose future trajectory is the goal of the estimation (shown as the red dash line). At most five surrounding cars can be handle simultaneously in lane change scenarios.

current trajectory planners can generate proper reactions and paths to cooperative with surrounding cars. In Fig.1, Veh-s is the target vehicle whose future trajectory is desired. Even if its lane-change intention can be indicated by turn signals, knowing this information is inadequate for autonomous driving cars to cooperate and perform socially. One should estimate when and how the target car (Veh-s) will make the lane change, and then plan a proper trajectory to react. If there is a predictive engine which gives surrounding vehicles' future movement (the red dashed lines), trajectory planners of the autonomous driving car can generate the corresponding path to perform safely with the target vehicle (veh-s) and also all surrounding vehicles.

The difficulties of the trajectory estimation are: 1) modeling the mutual interactions among surrounding cars; 2) predicting the continuous trajectories based on the interactions. 3) There are no enough data in complex driving scenarios and interactions for traditional deep learning learning methods. Our previous method [1] considers the interaction of surrounding vehicles as a non-parametric regression and just outputs discrete lane-changing start/end points. Then, it is extended to the prediction of continuous trajectories [2]. The analysis of the interaction and the future movement is based on a short period of observations of all surrounding cars' movements. Thought these approaches consider the interaction and historical trajectories, it cannot handle massive data and efficiently take the full advantage of the data set.

In this paper, an induction neural network framework is proposed to tackle all of these three difficulties, i.e., making trajectory estimation based on surrounding vehicles' interactions by using small dataset. As the result, it can make accurate trajectory estimation of a target vehicle, in both longitudinal and lateral direction.

II. RELATED WORK

In the last decade, researchers have proposed numerous interactive intention or trajectory estimation algorithms for

[1]Chiyu Dong is with Department of Electrical and Computer Engineering, Carnegie Mellon University, Pittsburgh, PA 15213 USA. chiyud@andrew.cmu.edu

[2]Yilun Chen and John M. Dolan is with the The Robotics Institute, Carnegie Mellon University, Pittsburgh, PA 15213 USA. yilunc1, jdolan@andrew.cmu.edu

autonomous driving. Those algorithms can be categorized into three major categories of methods to address the social cooperation problem among cars:

A **Rule/Control-based** methods, represented by earlier slot-based lane-change decision making. In addition, control-based algorithms are also considered.

B **Optimization-based** approaches, which optimize specific cost functions to guarantee proper behaviors.

C **Probabilistic and Learning** approaches, most of which use the Markov Decision Process (MDP) and its extensions. Learning approaches mainly uses kernel methods or deep neural networks.

A. Rule-based methods

The rule-based methods are the most understandable and straightforward approaches. They have been applied on test vehicles since the 2007 DARPA Urban Challenge. For example, a slot-based approach was implemented for the CMU Boss vehicle merge behavior planner [3] . In the planner, kinematic information is used to check merge-in feasibility of each slot. Then the target slot is selected from the set of feasible slots according to the context of the maneuver, and predictions of others. The slot-based approach is straightforward to implement and robust in obvious or simple scenarios. However, the lack of prior knowledge of surrounding vehicles' intentions makes it hard to estimate or predict their movements and corresponding behaviors. Naranjo et al. [4] use fuzzy logic to make lane-change decisions. The method is also straightforward and simple to implement. However, it also does not consider prior knowledge, reaction of other cars, or prediction. Lu and Hedrick [5] formulated the cooperative behavior as a platoon control problem. But the algorithm still requires a "coordination layer" to select a pair of main-road cars to interact.

B. Optimization-based methods

Liu et al. [6] and Nilsson et al. [7] applied Model Predictive Control(MPC) to solve the cooperative planning problem. In order to improve computational efficiency, the MPC is converted to a convex optimization over its manifold. The method is theoretically sound for performing cooperative trajectory planning for robots. However, it still needs a predictive engine to provide an initial estimate of the other agents' possible future trajectories and the associated uncertainty. The Intelligent Driver Model (IDM) [8] based algorithms optimized the ego-vehicle's reactions in interaction-required scenarios such as freeway entrances and turns in intersections. The algorithms assumed that all other vehicles applied the identical behavioral model (IDM). Given a proposed intention for the ego-vehicle, the optimization algorithms converge to a proper cooperative trajectory regarding the reactions from the IDM-driven agents. This approach has several problems. Firstly, the assumption of IDM does not necessarily fit all vehicles, especially in interactive scenarios. Secondly, the IDM relies on several parameters which require hand-tuning. Thirdly, IDM is one of the most robust distance-keeping models, but interactive scenarios involve various interactions other than car-following, such as yield or not-yield reactions.

C. Probabilistic methods

Probabilistic methods form the largest percentage of solutions to lane changing or cooperative driving. Montemerlo et al. [9] integrated lane-changing behavior into Stanford Junior's global path planner, which is an instance of dynamic programming (DP). In fact, the problem is formulated as optimizing a variant Bellman equation, which implicitly follows the MDP framework and value iteration. Each action is assigned a penalty cost. The lane changing behavior is a penalty term in the cumulative cost function which is optimized by the DP. However, the algorithm does not consider other traffic participants. Yao et al. [10] search for k-nearest-neighbors in a lane-change scenario database to generate a trajectory. Measuring differences between trajectories and scenarios remains a problem. And if the dataset contains a large number of samples, searching for the k-nearest-neighbors is time-consuming. Galceran et al. [11] make decisions depending on the probability of past trajectories of all traffic participants. Both of them report discrete actions such as left-lane-change, right-lane-change etc., which can be used as an upper-level module in our method. Dong et al. [12] detect whether the other car will merge in by using PGM. However, this method only provides binary output of whether a car is likely to yield or not yield.

Ulbrich et al. [13] and Wei et al. [14] proposed an online Partial Obervable Markov Decision Process (POMDP) for lane-change using real-time belief space search [15]. However, to achieve real-time performance and use a simple POMDP framework, they discretized state and action spaces. To avoid discrete states, Bai et al. [16] proposed a continuous-state POMDP using a belief tree, and the model was applied to navigating intersections. However its actions are discrete and represented by a generalized policy graph (GPG). Seir et al. [17] proposed an online and approximate solver for a continuous-action POMDP, but only tested on toy problems. The POMDP solutions above still need manually designed probabilistic transition models and reward functions. Kuefler et al. [18] used Generative Adversarial Networks to mainly imitate and estimate single-lane behaviors. Qiao et al. [19] used reinforcement learning to model the interaction behaviors among vehicles and decision making for autonomous driving cars in intersections. Sadigh et al. [20] establish the transition models by (inverse) reinforcement learning, but their solutions are limited to the specific scenario. Chen et al. [21] solve the sequential prediction of the trajectory by using Long Short-Term Memory (LSTM).

Our method serves as a predictive engine which provides trajectory estimations. The estimated trajectory then helps the motion planner to make a desired path in dynamic environments. The trajectory estimation is based on an interactive model that captures mutual influences among all surrounding cars and sequential information. The model is optimized from real training data.

III. Method

A. Meta Induction Program and Conditional Neural Process

The meta induction architecture takes a set of demonstration examples and an additional observation as the inputs, i.e., $\mathcal{D} = \big((X_1, Y_1), ..., (X_n, Y_n)\big)$ and $\hat{X}$, and then it outputs the the corresponding estimation $\hat{Y}$. The size of D can be as small as $1 \sim 5$, i.e., $n \in [1, 5]$. In each training step, n demonstration examples as well as a new observations $\hat{X}$ are used to approximate the corresponding output $\hat{Y}$. The main difference between the Meta Induction Program with the plain deep learning approaches (e.g., the plain LSTM trajectory prediction [21]) is the demonstration part. In stead of directly looking for the mapping between the new observed input $\hat{X}$ to its corresponding output $\hat{Y}$, the demonstration part encodes a higher level description of prior condition of the task. More specifically, most of the previous method is looking for $P(\hat{Y}|\hat{X})$, but the Meta Induction Program uses one more condition: $\mathcal{D}$ (i.e., the demonstration examples), which results in $P(\hat{Y}|\hat{X}, \mathcal{D})$.

The lane change problem can be considered as a set of trajectory prediction tasks that have various number of surrounding cars, different type of dynamic changes and etc. The condition $\mathcal{D}$ and the network is used to generalize tasks in the training process, and "categorizes" the current task from demonstration examples when evaluating new inputs. Therefore, the condition $\mathcal{D}$ can better guide the training and evaluation of the network to achieve sound performance.

Conditional Neural Process is one of the meta-learning induction [22] methods. There are three sub-modules in the original Conditional Neural Process (CNP) [23], shown in Equation 1:

$$\begin{aligned} r_i &= h(X_i, Y_i) \\ r &= \bigoplus_{i}^{n} r_i \\ \phi_i &= g(X_t, r) \end{aligned} \tag{1}$$

where h is the demonstration network, which can be considered as a encoder for the historical input, and g is the generator network, which can be considered as a decoder. The middle line is the condition which is extracted from the historical input. Using the result from the demonstration network, it generates a set of intermediate results r_i. Along with the latest demonstration, the result $\mathbf{r}$ which is aggregated from r_i is used to generate factorized parameters for a stochastic process. Due to the induction and the conditioning framework, it can be considered as an adaptive kernel. The kernel that defines the stochastic process can change according to the recent observations. For example, if the CNP's results are tied to a Gaussian Process (GP), its kernel, i.e., mean μ_t and variance σ_t, will be the function of the observation data, and determined by the CNP.

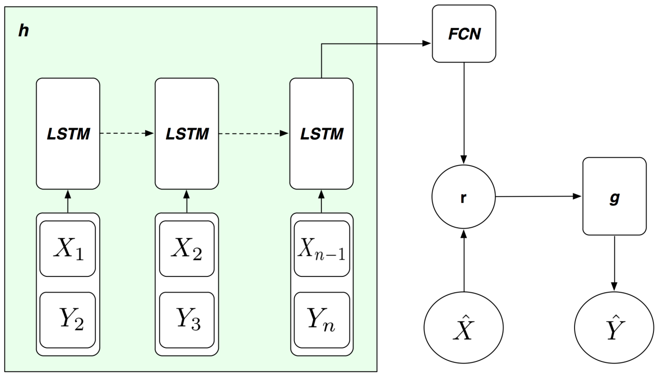

Fig. 2: The proposed recurrent structure. The green box indicates the observer sub-net. The LSTM is a asynchronous setup, which gives outputs until obtaining enough demonstrations. This output is then passed through a fully connected network (FCN) to obtain r, the intermediate condition tensor. $\hat{X}$ is the current observation right before the prediction, $\hat{Y}$ is the expected trajectory. and $\mathbf{g}$ is the generator.

B. Recurrent Induction Network

In the vanilla CNP, the result of each demonstration is summarized by r_i. For N demonstrations, $r_0, ...r_{n-1}$ will be obtained from the observation sub-net. The total condition which is generated from the N demonstrations is represented by the aggregation of r_i, i.e.,

$$\mathbf{r} = r_0 \bigoplus r_1 .... \bigoplus r_{n-1} \tag{2}$$

The main purpose of doing the aggregation is to ensure permutation invariance, which further satisfies the requirement for CNP being a stochastic process. In most cases, the aggregation is implemented by mean, i.e., Eq. 2 becomes $\mathbf{r} = \frac{1}{n} \sum\limits_{0}^{n-1} r_i$.

The permutation invariance property is theoretically sound for most static tasks, for example, function regression or image completion, whereas in applications where sequence is more important, the aggregation eliminates the sequential information. However, in most dynamic scenarios and tasks, such as trajectory prediction, the observation and result are highly serialized, and permutation invariance cannot be satisfied.

To retain the sequential information, it is intuitive to introduce recurrent neural networks into the demonstration sub-net. In Fig. 2, the green box indicates the demonstration sub-net, which consists of an asynchronous LSTM network. The figure shows a roll-out description of the LSTM. The dashed-arrow lines between LSTMs indicate its inner variables, which are passed inside the LSTM cell over time. The LSTM's results remain internal until it obtains enough demonstrations. The result of LSTM is passed through a fully connected network before being used as a conditioning tensor. Therefore, the how process can be expressed as:

$$\begin{aligned} \mathbf{r} &= h(\mathcal{X}_n, \mathcal{Y}_n) \\ \hat{Y} &= g(\hat{X}, \mathbf{r}) \end{aligned} \tag{3}$$

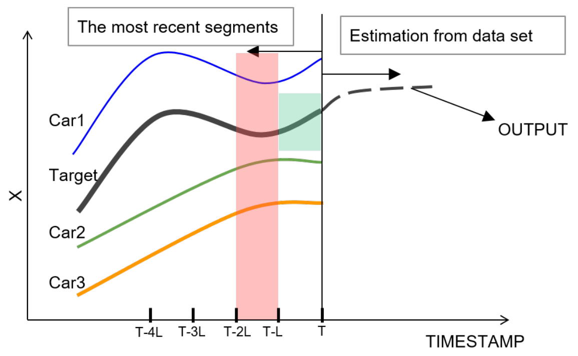

Fig. 3: Illustration of the input and output of the network in a lane change scenario. The figure shows the idea, does not correspond to real trajectories. x-axis is timestamp; y-axis is the lateral position, labeled as 'X'.

where h is the LSTM layer and followed by a fully connected layer. Comparing with the original CNP's expression, shown in Equation (1), here the aggregation operation $\bigoplus$ is replaced by the recurrent network (which mainly retains the sequential information) and a fully connected network. $\mathcal{X}_n$ contains n segments of observed trajectories. $\mathcal{Y}_n$ contains n corresponded trajectories for the target vehicle. In detail, $X_i \in \mathcal{X}_n$ is the observation of historical trajectories from time t of all vehicles at including the target and its surrounding cars; $Y_{i+1} \in \mathcal{Y}_n$ is one segment of the trajectory of the target vehicle from time $i+1$. A pair of $(X_i, Y_{i+1}$ is taken as one demonstration. Note that for the input of one step for the LSTM cell, X_i and Y_{i+1} are asynchronous (X_i contains all cars' trajectories in the i-th time interval, Y_{i+1} contains the target car's trajectory in the (i+1)-th time interval). X_{n+1} is the new input who is ready to be evaluated.

To avoid confusion, the inputs to the observer and generator sub-nets will be described separately. To better demonstrate the idea, only lateral translation vs timestamp plot is shown and discussed, (Fig. 3, x-axis is timestamp, y-axis is lateral position). Solid lines indicate the observed changes of lateral position over time of all related cars, i.e., one target and its surrounding cars. At time T, the goal is to predict the future trajectory of the target car, which is shown as the dark dashed line. L is the observation segment's time interval.

1) Input of the observer sub-net: The input of the demonstration network is the ordered set of examples $\mathcal{D} = \big((X_0, Y_1), (X_1, Y_2), ...(X_{n-1}, Y_n)\big)$, where the input set here $(\cdot)$ retains the order of the demonstration (X_i, Y_{i+1}) for each time interval. At each timestamp, the single observation X_i is a segment of past trajectories of length L for all cars. In Fig. 3, X_i corresponds to the pink region. Y_{i+1} is the trajectory of the target vehicle in the next time interval of Y_i. Y_{i+1} corresponds to the green region in Fig. 3.

2) Input of the generator sub-net: The input of the generator sub-net has two parts: a) The condition representation, which is generated from the observer sub-net. b) The segments of all vehicles' trajectories in the time interval immediately before the time of prediction T. More specially, this input contains all vehicles' trajectories in the time interval [T-L, T).

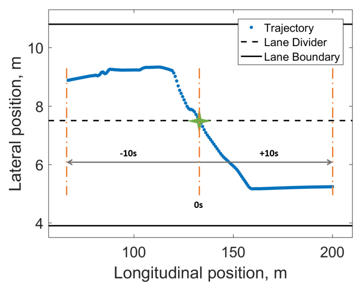

Fig. 4: An illustration of the labeled data for an individual lane-changing vehicle.

TABLE I: Number of the trajectories in the dataset for each vehicle.

Vehicle	s	f	r	rt	ft	st
No.	870	705	764	214	233	43

The main difference between the Conditional Meta induction and the plain Conditional Neural Process induction is that, instead of using an aggregator to remove the sequential effects, the proposed method uses a recurrent sub-network (LSTM) to extract conditions for the generator sub-network.

The loss function is a combination of the loss in two directions:

$$C = C_{lon} + \lambda C_{lat} \tag{4}$$

where C is the total loss, and C_{lon}, C_{lat} are the losses in the longitudinal and lateral directions, respectively. The main reason for this separation is that the longitudinal translation and lateral drifting are not in the same range. Empirically, the ratio is set to $\lambda = 10$ in the lane changing scenario. In detail, since we consider the sequence of the trajectory, a mean-squared-error is applied for both longitudinal and lateral losses.

IV. Experimental Results

A. Dataset Description

Real trajectory data from NGSIM[24] (US-101 and I-80 subsets) are used for training and testing in the experiments. Lane-change scenarios are extracted from the real data, and organized into groups. As shown in Fig. 1, each group contains one host car (Veh-s) and five surrounding cars, i.e., Veh-f, Veh-r, Veh-rt, Veh-ft, Veh-st. As shown in Fig. 4, the trajectory of each car in the group is recorded from 10 seconds before to 10 seconds after the target car (Veh-s) crossing the lane-marking.

TABLE I shows the total number of trajectories for each car type in the lane-change scenarios in the dataset. Segments of trajectories from all participants before the host car starts turning towards the target lane. Three seconds of historical trajectories are used for input. The prediction is the future trajectory in the next 5s after obtaining the historical observations.

B. Baseline Models

Three baseline models are implemented to demonstrate the features and improvements of the proposed algorithm.

1 Continuous trajectory estimation in RKHS [1];
2 Conditional Neural Process [23];
3 Plain LSTM trajectory estimation.

Note that all of the baseline algorithms predict the trajectory by considering the interactions between the target and its surrounding cars. Historical trajectories of all involved cars also contribute to the prediction in these baseline methods.

Conditional Neural Process (CNP) is implemented as a baseline model. The main difference between CNP and MIN is that, to retain the property of a stochastic process, CNP should be permutation-invariant. Therefore, there is an aggregation operation $\bigoplus$ in an intermediate layer between the observer and generator sub-net. The implementation also retains this property, and as is recommended, the aggregation operation $\bigoplus$ is implemented by a mean operator, which means that $\mathbf{r} = \frac{1}{n}\sum_{1}^{n} r_i$. In addition, the dimension of the intermediate condition representation $\mathbf{r}$ is 128. The intermediate condition $\mathbf{r}$ is the output of the observer sub-network h. The observer h contains a three-layer fully connected network, with increasing dimensions of 32, 64, and 128. Note that the last layer of the fully connected network outputs the condition representation $\mathbf{r}$. Followed by the $\mathbf{r}$, a new observation input x_t is combined. A concatenated tensor which contains $\mathbf{r}$ and x_t is then fed into the generator network g. The generator g also consists of a three-layer fully connected network. The CNP baseline shows the performance of the induction network. However, it does not consider the sequential information. This lack inspired the proposed method, which takes the sequentiality of trajectories into consideration.

A plain LSTM sequence-to-sequence trajectory estimation is also implemented, one-stack LSTM model is used. Instead of using a whole observed trajectory as the input, it feeds individual (x, y) coordinates once a step.

In the design of the proposed method, only one LSTM is stacked for the observer sub-net. The LSTM cell uses an asynchronous structure, which means that it will not output the intermediate condition representation $\mathbf{r}$ until it has processed enough data. We set the number of the demonstration to three, so that the observer sub-net will generate a result $\mathbf{r}$ only once it has processed three segments of past trajectory. However, the output dimension of the LSTM cell is the same as its input. With a possible variable dimension of the input, its output dimension is desired to be fixed. Then a fully connected network follows immediately after the LSTM observer network. To make the structure more comparable to the CNP implementation, the fully connected network only has one layer, containing 128 hidden nodes. The dimension of the condition representation $\mathbf{r}$ is equal to the one in CNP. The generator architecture uses the same design as that in the CNP implementation, i.e., a three-layer fully connected network. NOte that in the implementation of the continuous trajectory estimation in RKHS (the first baseline method),

TABLE II: Mean error and its standard deviation of the trajectory prediction comparing with the ground-truth, in the lateral direction. The unit is meter (m).

Methods		1s	2s	3s	4s	5s
RKHS	μ	0.052	0.251	—	—	—
	σ	0.051	0.250	—	—	—
LSTM	μ	0.286	-0.330	-0.588	-0.776	-1.209
	σ	0.776	0.880	0.919	1.020	**1.160**
CNP	μ	0.085	0.181	0.382	0.379	0.444
	σ	0.476	1.268	1.995	2.325	2.472
RMIN	μ	0.019	0.090	0.195	0.235	**0.248**
	σ	0.501	1.299	1.997	2.343	2.492

TABLE III: Mean error and its standard deviation of the trajectory prediction comparing with the ground-truth, in the longitudinal direction. The unit is meter (m).

Methods		1s	2s	3s	4s	5s
RKHS	μ	1.425	9.925	—	—	—
	σ	0.871	5.794	—	—	—
LSTM	μ	-2.100	-4.429	-6.050	-9.851	-17.932
	σ	6.356	8.225	10.303	14.608	20.553
CNP	μ	-0.229	-0.565	-0.782	-0.979	1.126
	σ	2.184	4.846	7.685	10.764	**13.995**
RMIN	μ	0.169	0.485	0.666	0.831	**1.091**
	σ	2.606	5.476	8.611	11.946	15.510

scenarios that have different numbers of surrounding vehicles are separately trained and the model is updated by a selection matrix. Otherwise, the result of the RKHS will be poor. However, the induction models (including CNP and the proposed RMIN) are more robust than other methods when facing the uncertainty of the surrounding traffic, as long as the total number of surrounding cars does not exceed the design value. In the current implementation, the upper bound on the number of surrounding cars is five, excluding the target car itself.

C. Comparison Results

TABLEs II and III show the prediction errors of the different methods at each timestamp. Mean errors and standard deviation are used in the comparison. To better demonstrate the results, errors are separately shown in longitudinal and lateral directions. All of these methods have increasing mean errors and standard deviations. The RKHS method can only predict over a 2s horizon within an acceptable accuracy.

In the lateral direction, the proposed method achieved the lowest mean error at each examined timestamp. The mean error of the RKHS at 2s is at the same level as that of RMIN at the 5-second mark. At 2s, RMIN has a mean error that is half of CNP's and 36% of the error of RKHS. However, RMIN has the largest standard deviation at that timestamp. At 5s, RMIN has a mean error of 0.248 meter, which is 55% of the CNP error at the same timestamp. Their standard deviations are at roughly the same level. Though the LSTM approach has the lowest standard deviation in the lateral direction, its absolute mean error is almost five times larger than that of RMIN.

In the longitudinal direction, the proposed method also achieves the lowest (absolute) mean error at each examined timestamp. RKHS is significantly worse than other methods in terms of mean error, even looking at the first two seconds. At 2s, the standard deviations are at the same level. However, the absolute mean errors of CNP and RMIN are around 0.5, which is 1/20th of the error of RKHS. One possible reason is that the RKHS uses one kernel function to estimate the whole trajectory with 2 dimensions. In detail, only one kernel function is used for the estimates in both lateral and longitudinal directions. At 5s, the mean error of RMIN is 27.8% smaller than that of CNP. The standard deviation of RMIN is only 9.8% larger than CNP's. The LSTM approach perform badly in longitudinal approach, though its lateral performance is comparable to that of the other methods. Note that its absolute mean error at 1s is already larger than that of CNP or RMIN at 5s. There are several reasons that contribute to the differences: 1) The input of the plain LSTM directly takes individual coordinates at each time step. However, RMIN and CNP take a segment of trajectories at each time step respectively as input to either an LSTM or a fully connected layers-based observer network. The latter provides more data and relations to the observer network. The induction models also use both historical observations and their expected output as inputs to extract the correlations between the observation and output, and the relationship is encoded as the condition tensor. This design makes it easier for induction models to accurately extract the tendency of the trajectory changes. In addition, the speed along the longitudinal direction is larger than that of the lateral direction. 2) The output (generator) network of the induction models uses a fully connected network, whereas the plain LSTM uses one LSTM cell. Since the fully connected network (FCN) describes the relationships among positions in the predicted trajectory more explicitly than the LSTM cell does, with a small set of training data, the FCN in the induction models can work better. 3) Therefore, any subtle changes of the observation (especially for the longitudinal positions) and model prediction error can result in a larger derivation in the longitudinal direction than in the lateral one.

V. CONCLUSIONS

This paper proposed a recurrent meta induction neural network for the trajectory prediction of human-driven cars considering all surrounding traffic. It uses an LSTM structure for demonstration and condition to capture sequential information in the historical trajectories. The method significantly outperforms traditional kernel methods in terms of mean error. In addition, compared to the original CNP aggregation which eliminates the effect of sequence, the use of the recurrent network in the demonstration and condition helps to obtain a lower level of mean error for the trajectory prediction, in both longitudinal and lateral directions. In the future, a more advanced generator and observer structure will be developed to further reduce error. The framework will also be extended to be more general, i.e., to apply to additional scenarios, such as turns in intersections and highway merging.

REFERENCES

[1] C. Dong, Y. Zhang, and J. M. Dolan, "Lane-change social behavior generator for autonomous driving car by non-parametric regression in reproducing kernel hilbert space," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2017, pp. 4489–4494.

[2] C. Dong, J. M. Dolan, and B. Litkouhi, "Continuous behavioral prediction in lane-change for autonomous driving cars in dynamic environments," in *2018 IEEE 21th International Conference on Intelligent Transportation Systems (ITSC) (ITSC2018)*, 2018.

[3] C. R. Baker and J. M. Dolan, "Traffic interaction in the urban challenge: Putting boss on its best behavior," in *Intelligent Robots and Systems, 2008. IROS 2008. IEEE/RSJ International Conference on*. IEEE, 2008, pp. 1752–1758.

[4] J. E. Naranjo, C. Gonzalez, R. Garcia, and T. De Pedro, "Lane-change fuzzy control in autonomous vehicles for the overtaking maneuver," *IEEE Transactions on Intelligent Transportation Systems*, vol. 9, no. 3, pp. 438–450, 2008.

[5] X.-Y. Lu and K. Hedrick, "Longitudinal control algorithm for automated vehicle merging," in *Decision and Control, 2000. Proceedings of the 39th IEEE Conference on*, vol. 1. IEEE, 2000, pp. 450–455.

[6] C. Liu, C.-Y. Lin, Y. Wang, and M. Tomizuka, "Convex feasible set algorithm for constrained trajectory smoothing," in *American Control Conference (ACC), 2017*. IEEE, 2017, pp. 4177–4182.

[7] J. Nilsson, M. Brännström, J. Fredriksson, and E. Coelingh, "Longitudinal and Lateral Control for Automated Yielding Maneuvers," *IEEE Transactions on Intelligent Transportation Systems*, vol. 17, no. 5, pp. 1404–1414, may 2016.

[8] M. Treiber, A. Hennecke, and D. Helbing, "Congested traffic states in empirical observations and microscopic simulations," *Physical review E*, vol. 62, no. 2, p. 1805, 2000.

[9] M. Montemerlo, J. Becker, S. Bhat, H. Dahlkamp, D. Dolgov, S. Ettinger, D. Haehnel, T. Hilden, G. Hoffmann, B. Huhnke *et al.*, "Junior: The stanford entry in the urban challenge," *Journal of field Robotics*, vol. 25, no. 9, pp. 569–597, 2008.

[10] W. Yao, H. Zhao, P. Bonnifait, and H. Zha, "Lane change trajectory prediction by using recorded human driving data," in *Intelligent Vehicles Symposium (IV), 2013 IEEE*. IEEE, 2013, pp. 430–436.

[11] E. Galceran, A. G. Cunningham, R. M. Eustice, and E. Olson, "Multipolicy decision-making for autonomous driving via changepoint-based behavior prediction: Theory and experiment," *Autonomous Robots*, 2017, in Press.

[12] C. Dong, J. M. Dolan, and B. Litkouhi, "Intention estimation for ramp merging control in autonomous driving," in *2017 IEEE 28th Intelligent Vehicles Symposium (IV'17)*, Jun. 2017, pp. 1584 – 1589.

[13] S. Ulbrich and M. Maurer, "Probabilistic online POMDP decision making for lane changes in fully automated driving," in *16th International IEEE Conference on Intelligent Transportation Systems (ITSC 2013)*. IEEE, oct 2013, pp. 2063–2067.

[14] J. Wei, J. M. Dolan, J. M. Snider, and B. Litkouhi, "A point-based mdp for robust single-lane autonomous driving behavior under uncertainties," in *Robotics and Automation (ICRA), 2011 IEEE International Conference on*. IEEE, 2011, pp. 2586–2592.

[15] S. Paquet, L. Tobin, and B. Chaib-draa, "Real-time decision making for large pomdps," in *Conference of the Canadian Society for Computational Studies of Intelligence*. Springer, 2005, pp. 450–455.

[16] H. Bai, D. Hsu, and W. S. Lee, "Integrated perception and planning in the continuous space: A POMDP approach," *The International Journal of Robotics Research*, vol. 33, no. 9, pp. 1288–1302, 2014.

[17] K. M. Seiler, H. Kurniawati, and S. P. N. Singh, "An online and approximate solver for pomdps with continuous action space," in *2015 IEEE International Conference on Robotics and Automation (ICRA)*, May 2015, pp. 2290–2297.

[18] A. Kuefler, J. Morton, T. Wheeler, and M. Kochenderfer, "Imitating driver behavior with generative adversarial networks," in *Intelligent Vehicles Symposium (IV), 2017 IEEE*. IEEE, 2017, pp. 204–211.

[19] Z. Qiao, K. Muelling, J. M. Dolan, P. Palanisamy, and P. Mudalige, "Automatically generated curriculum based reinforcement learning for autonomous vehicles in urban environment," in *Intelligent Vehicles Symposium (IV), 2018 IEEE*. IEEE, 2018, pp. 1233–1238.

[20] D. Sadigh, S. S. Sastry, S. A. Seshia, and A. Dragan, “Information gathering actions over human internal state,” in *Intelligent Robots and Systems (IROS), 2016 IEEE/RSJ International Conference on*. IEEE, 2016, pp. 66–73.

[21] Y. Chen, “Learning-based lane following and changing behaviors for autonomous vehicle,” Master’s thesis, Carnegie Mellon University, Pittsburgh, PA, May 2018.

[22] J. Devlin, R. R. Bunel, R. Singh, M. Hausknecht, and P. Kohli, “Neural program meta-induction,” in *Advances in Neural Information Processing Systems*, 2017, pp. 2080–2088.

[23] M. Garnelo, D. Rosenbaum, C. Maddison, T. Ramalho, D. Saxton, M. Shanahan, Y. W. Teh, D. Rezende, and S. M. A. Eslami, “Conditional neural processes,” in *Proceedings of the 35th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, J. Dy and A. Krause, Eds., vol. 80. Stockholmsmssan, Stockholm Sweden: PMLR, 10–15 Jul 2018, pp. 1704–1713.

[24] NGSIM, “U.S. Department of Transportation, NGSIM - Next generation simulation,” http://www.ngsim.fhwa.dot.gov, 2007.