2021 IEEE Intelligent Transportation Systems Conference (ITSC)
Indianapolis, USA. September 19-21, 2021

Trajectory Prediction via Learning Motion Cluster Patterns in Curvilinear Coordinates

Aotian Wu, Tania Banerjee, Anand Rangarajan and Sanjay Ranka

Abstract— **A high proportion of crashes happen at or near intersections. To improve intersection safety, trajectory prediction of vehicles has been studied intensively, mostly for automated vehicle (AV) and advanced driver assistance system (ADAS) applications. These approaches with vehicle-mounted sensors still suffer from limited detection range or occluded views. In this work, we propose to perform trajectory prediction on surveillance cameras. As Vehicle-to-Infrastructure (V2I) technology enables low-latency wireless communication, warnings from our prediction algorithm can be sent to vehicles in real-time. Our approach consists of an offline learning phase and an online prediction phase. The offline phase learns common motion patterns from clustering and finds prototype trajectories for each cluster, and updates the prediction model. The online phase predicts the future trajectories for incoming vehicles assuming they follow one of the motion patterns learned from the offline phase. We adopted a long short-term memory encoder-decoder (LSTM-ED) model for the task of trajectory prediction. We also explored the usage of a Curvilinear Coordinate System (CCS), which utilizes the learned prototype and simplifies the trajectory representation. Our model is also able to handle noisy data and variable-length trajectories. Our proposed approach outperforms the baseline Gaussian Process (GP) model, and shows sufficient reliability when evaluated on collected intersection data.**

I. INTRODUCTION

Road intersections, where two or more roads meet and cross, show complex geometry and traffic rules, and require drivers' timely maneuvers. It was reported by the Federal Highway Administration (FHWA) that more than 50% of fatal and injury causing crashes occur at or near intersections. Of all the intersection-related crashes, only about 4% is caused by vehicle or environmental reasons with 96% attributed to drivers [1]. Possible driver-attributed crashes at intersections include inadequate surveillance, false assumptions regarding other driver actions, obstructed views, illegal maneuvers, internal distractions and misjudgments of gaps, speed, etc. [1]. Potential solutions for reducing crashes tend to be focused on providing driver assistance of vehicle control or giving timely warnings of potential risks. To this end, predictive algorithms are proposed. For example, by tracking the surrounding vehicles, one can estimate their intended maneuvers, predict their future motion and estimate the risk of collision. Previous studies [2], [3], [4], [5] have focused on automated vehicle (AV) and advanced driver assistance system (ADAS) solutions. These approaches rely on on-board sensors, such as LiDAR, radar, and vision sensors. However, the on-board sensors have certain limitations, such as limited detection range or field of view, low resolution, drifting position estimation, and sensor imprecision. Therefore we propose to incorporate assistance from existing and planned intersection video infrastructure to improve intersection safety.

Vehicle-to-Infrastructure (V2I), as the next generation of Intelligent Transportation Systems (ITS), exchanges data from vehicles and infrastructures through wireless communication, and enables real-time assistance and warning from infrastructure to vehicles on the road. With the rapid development of wireless communication technologies, especially in the recent progress in 5G networks, algorithms running on the infrastructure side are ready to be delivered to road participants, and will be needed in the future.

In this work, we tackle the problem of robust and flexible vehicle trajectory prediction at intersections from surveillance videos, aiming to give early warnings to vehicles about possible collisions and abnormal behaviors. Our proposed approach only requires vision sensors, no signal data is needed, and can be applied to both signaled and unsignaled intersections. Our approach requires a setup stage, namely, i) google-map alignment, ii) learning typical motion patterns from historical data, and iii) training the trajectory prediction model. After the setup, our model can make real-time predictions of all the vehicles' trajectories within the intersection. We also take into account the noise introduced by the automatic vehicle tracking algorithms and varying lengths of trajectories captured.

The main contributions of the paper are as follows:

- We present a real-time capable vehicle trajectory prediction from surveillance cameras without requiring traffic signal or GPS data.
- Our algorithm automatically learns the typical motion patterns from historical data, and representatives of them are used in the online phase to speed up and boost the performance of trajectory prediction.
- We design a deep-learning-based trajectory prediction model, which is capable of dealing with variable-length observations and prediction periods.
- Multiple reasonable predictions are given by the model if multiple motion patterns are likely to describe the observed trajectory.

The rest of this paper is organized as follows. Section II presents the related works on trajectory prediction methods. Section III gives an overview of our proposed approach, consisting of an offline learning phase and an online prediction

Aotian Wu, Tania Banerjee, Anand Rangarajan, Sanjay Ranka are with University of Florida, Gainesville, FL 32611. This work is supported by NSF CNS 1922782, by the Florida Dept. of Transportation (FDOT) and FDOT District 5. The opinions, findings and conclusions expressed in this publication are those of the author(s) and not necessarily those of the Florida Department of Transportation or the National Science Foundation.

2021 IEEE International Intelligent Transportation Systems Conference (ITSC) | 978-1-7281-9142-3/21/$31.00 ©2021 IEEE | DOI: 10.1109/ITSC48978.2021.9564800

phase. Section IV describes the methodology of the offline trajectory clustering and prototype generation. Section V presents the CCS transformations and the trajectory prediction model. Section VI reports our experimental settings, and quantitative and qualitative results. We conclude with a few observations in Section VII.

II. Related Work

Vehicle motion or trajectory prediction has been studied extensively over the last two decades and is mainly used in ADAS and AV. The work in [6] organized the motion prediction approaches into three categories: physics-based, maneuver-based, and interaction-aware motion models. Our approach falls into the maneuver-based category. The work in [6] further divides the maneuver-based approaches into prototype-based and intention estimation. Our approach can be classified as prototype-based.

1) Prototype-based Trajectory Prediction: Prototype-based approaches assume that vehicle motion can be grouped into a finite set of clusters. A prototype trajectory is learned for each cluster as its representative. In the case of four-way intersections, there are mainly twelve motion patterns, namely *left turn*, *right turn*, and *going straight* from the four directions. For road sections, typical motion patterns are *lane keeping*, *lane change to the left* and *lane change to the right*. An early work [7] adopts a statistical approach, where clusters of motion patterns are represented by the mean and standard deviation. Later works [8], [9], [10] represent motion patterns with Gaussian Processes (GP) due to its ability to capture spatio-temporal characteristics of traffic situations. One drawback of GP models lies in its computational complexity. As a non-parametric approach, the number of parameters grows as more training samples are provided. In a surveillance setting, to leverage the huge amount of data captured from cameras, a non-parametric approach will be slow and expensive. In this work, our proposed approach not only fits a compact model to the training data, it also incorporates the prototype trajectories to make more informed predictions.

2) Recurrent Neural Network (RNN) for Trajectory Prediction: Trajectories are essentially sequences of locations, velocities, etc. Recurrent Neural Networks (RNNs), especially LSTMs and Gated Recurrent Units (GRUs) have been explored and broadly evaluated for the task of sequence generation and prediction. Over the last five years, RNNs and its variants have been gaining popularity in trajectory prediction. The work in [11] first applied an LSTM model to learn general human motions and predict their future trajectories. Later works [12], [13], [14], [15] extended the idea to vehicle trajectory predictions, enabling vehicles to reason about future motion of other surrounding vehicles. To the best of our knowledge, there is no work that utilizes a deep learning-based trajectory prediction model from surveillance vision systems. Unlike the above approaches for autonomous vehicles or ADAS where modeling interaction is important, our approach will be applied to give early warnings to vehicles entering intersections; hence, we focus on typical motion pattern finding and efficient online functioning.

III. System Overview and Pipeline

The paper is divided into two main components: an offline phase in which trajectories are clustered and good prototypes found. This is followed by an online phase in which different LSTMs are constructed to perform trajectory prediction. The second phase necessarily uses the prototype information from the first phase. In this section, we give a procedural description of our algorithm workflow. Fig. 1 concisely summarizes the workflow.

A. Offline Phase

The offline phase can be done periodically to capture the dynamics of traffic evolving (e.g., road construction that causes vehicle detour) or done once for setup if the intersection geometry remains unchanged. There are two main tasks in the offline phase: i) finding common motion patterns and their representatives and ii) training the LSTM-ED trajectory prediction model.

The trajectories are captured by a vision-based tracking algorithm, and are stored in the database. We refer to the trajectories in the database as historical trajectories. We use a portion of historical trajectories to learn common motion patterns by a course-to-fine clustering algorithm, followed by finding prototypes to represent each motion cluster. In order to generate longer and smoother prototype trajectories, a filtering and smoothing algorithm is applied before clustering. Lastly, the trajectory prediction model to be used in the online phase is trained in the offline phase.

B. Online Phase

The online phase receives real-time captured trajectories from the online tracking algorithm. We first detect whether the vehicle is waiting for traffic signals using the stay point detection algorithm from [16]. We start making predictions when the vehicle starts moving. For a partial trajectory from a moving vehicle, we match with prototype trajectories from the offline phase based on distance measurements. We then feed the partial trajectory, matching prototypes, and duration to be predicted into the trajectory prediction model to output its future trajectory.

IV. Trajectory Clustering and Prototype Trajectories

In this section, we introduce the methods used to cluster trajectories and find prototypes.

A. Historical Trajectory Clustering

Vehicles at an intersection exhibit common motion patterns, and their trajectories can be grouped into clusters. Within each cluster of the same moving direction, one or more finer clusters can be found, as there might be multiple entering and exiting lanes of the same motion. Our clustering approach is therefore a two step process where we first cluster the trajectories based on their direction of motion and next cluster the trajectories for a given motion using

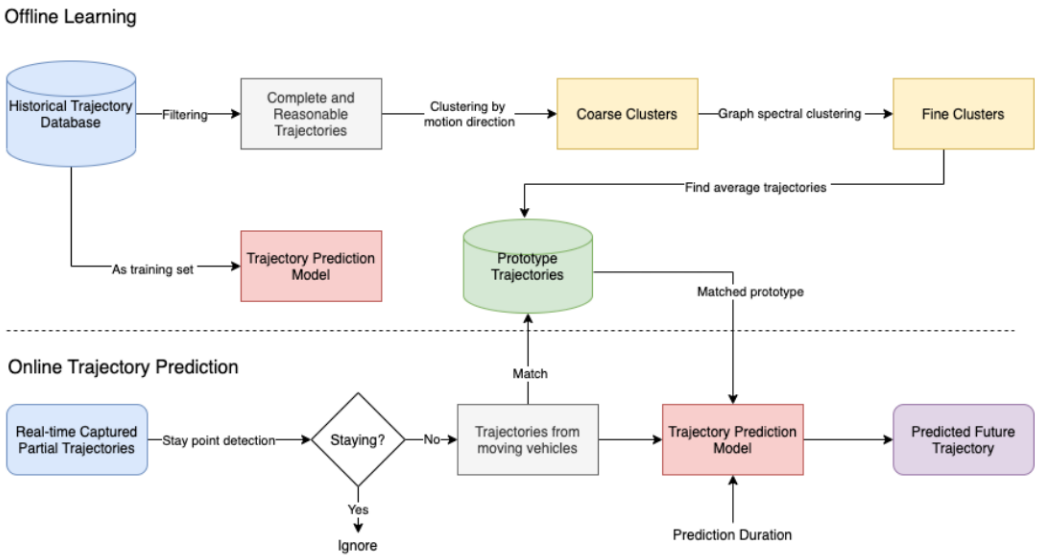

Fig. 1. The overall workflow of trajectory prediction in offline and online phases.

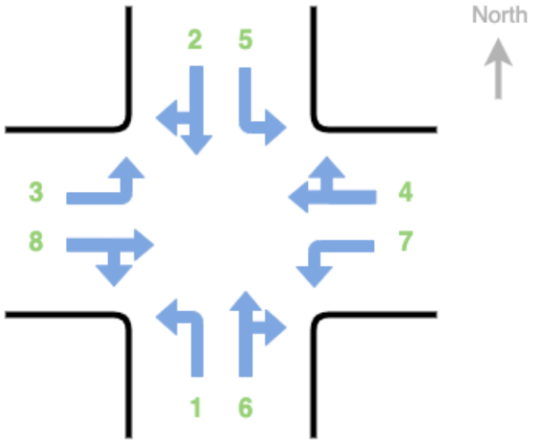

Fig. 2. Phase Diagram showing vehicles at four way intersections as defined in [17].

Fig. 3. Pictorial representation of information in configuration file

spectral clustering. In this section, we describe the clustering based on motion direction, followed by the description of a new distance measure for spectral clustering, and finally, we describe our spectral clustering approach.

1) Clustering by Motion Direction: When we analyze a collection of trajectories, one of the first steps is to automatically identify the phase of the trajectory. As shown in Fig. 2, the vehicle phase system assigns a single number for both through and right turn movements. We find it useful to first have a separate category for the right turn vehicular movement. So, in addition to the eight bins, each storing trajectories of a certain phase, we append four more bins for storing the trajectories doing right turns for phases 2, 4, 6, and 8.

The motion direction of a trajectory can be represented as the vector connecting the start and end point. Reference motion directions are obtained from annotation using CAD tools. Fig. 3 depicts a configuration file that contain these reference vectors for each direction, namely, South-to-North, North-to-South, East-to-West and West-to-East. These vectors are drawn approximately connecting the lanes of opposite side. The comparison of the direction of the trajectory vector with respect to each reference vector is done by computing the cosine of the acute angle between these two vectors. The cosine is very close to 1.0, when the direction of the trajectory vector aligns with the direction of a reference vector, and then we assign the phase of the reference vector to the trajectory.

After partitioning the trajectories by motion direction into bins, we apply spectral clustering to the trajectories of each bin. We used a new distance measure for computing the distance between two trajectories in the same bin. This is described next.

2) Distance Measure: The warping path between two trajectories T_i and T_j is found by applying a time warping algorithm. We use Dynamic Time Warping (DTW) which is a well known algorithm for finding similarities between two temporal sequences that vary in speed. As shown in

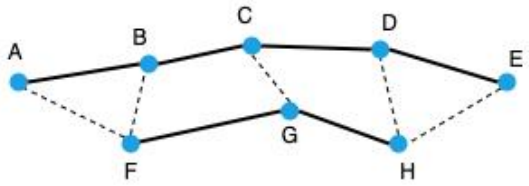

Fig. 4. Two trajectories represented by ABCDE and FGH. The dashed lines show the point correspondence (warping path) obtained using DTW.

Fig. 4 there are two trajectories $ABCDE$ and FGH. The warping path for these two trajectories is represented by the dashed lines and is $(A,F),(B,F),(C,G),(D,H),(E,H)$. Using geometric coordinates of the points, it is possible to determine the approximate area between the two paths. This is done by determining the area of the triangles $\triangle ABF$, $\triangle BCF$, $\triangle FGC$, $\triangle CDG$, $\triangle GHD$, and $\triangle DEH$, using coordinate geometry area formula for triangles when the coordinates of their corners are known. The sum of the area of all triangles approximates the area between the trajectories. Finally, the distance between the trajectories is approximated as the total computed area divided by the average length of the trajectories [18].

3) Clustering by Graph Spectral Clustering: Spectral clustering is applied next to the trajectories in the same direction bins and we get clusters of trajectories that have the same movement. Spectral clustering may also be used to identify the outliers or anomalous trajectories that are not similar to any of the other trajectories in the same bin. The details of our spectral clustering implementation may be found in [18].

B. Prototype Trajectory Generation

From our clustering algorithm, clusters of motion patterns are found. In this section, we present our approach to generate prototype trajectories for each cluster.

1) Complete Trajectory Determination: A large number of trajectories extracted from the traffic video are incomplete, meaning that they either appear from the middle of the intersection or disappear within it. These trajectories are as useful as complete ones in the LSTM-ED model training. However, we prefer complete trajectories for prototype generation for two reasons: 1) complete trajectories are spatially aligned as they start and end in the same regions, and 2) the distance measure we use performs better with longer reference trajectory.

Trajectories that starts at an intersection entrance and ends at an intersection exit are considered complete. We annotated the intersection boundary as a polygon. For each motion pattern, we annotate its enter line and exit line. We consider a trajectory complete only when its starting point is close to the entering line and its ending point is close to the exit line. The left part of Fig. 5 shows the complete trajectories from one cluster.

2) Averaging Complete Trajectories: Given a set of complete trajectories from a cluster, we aim to find a representative for the cluster by averaging the trajectories. We represent

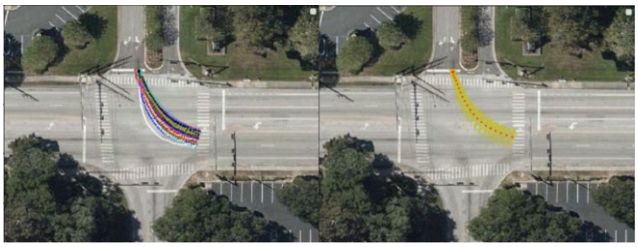

Fig. 5. Complete trajectories in a cluster and its prototype.

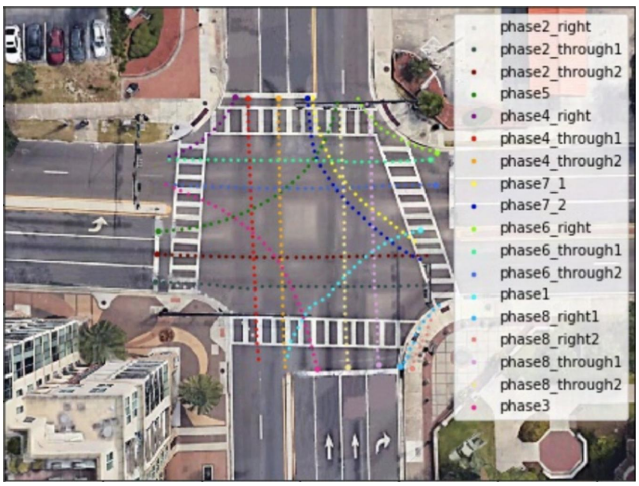

Fig. 6. The resulting prototypes. Note that the slight deviation of some prototypes is caused by imperfect Google Map alignment.

the complete trajectories as 2-D cubic splines, denoted as S, where x and y coordinates of a trajectory are parameterized by t:

$$\begin{aligned} S(t) &= (x(t), y(t)), \\ x(t) &= a_x t^3 + b_x t^2 + c_x t + d_x, \\ y(t) &= a_y t^3 + b_y t^2 + c_y t + d_y, \end{aligned} \tag{1}$$

where t's are equally-spaced break points in the range of $[0,1]$. As the complete trajectories are roughly spatially aligned, an average trajectory for each cluster can be found by

$$S_c(t) = \frac{1}{m_c} \sum_{j}^{m_c} S_j(t), \tag{2}$$

where S_j is a complete trajectory from the given set from cluster c, and m_c is the number of trajectories. After obtaining the average trajectories, the last step we do is to reparametrize and re-scale the splines. We adopt the arc-length parametrization algorithm from [19], which results in equally-spaced control points. We then re-scale the splines so that consecutive points of a spline have a distance of one meter. Fig. 6 shows the prototypes we obtained for an intersection.

V. Trajectory Prediction Model

In this section, we present our trajectory prediction model. A brief summary of the approach follows. The trajectories originally represented in Cartesian coordinates are first

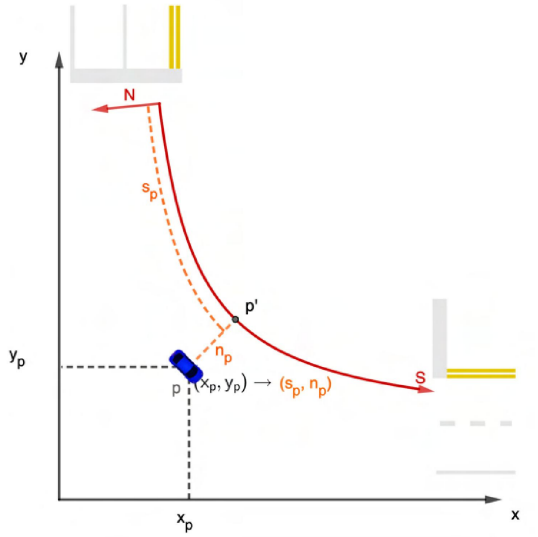

Fig. 7. Illustration of Curvilinear Coordinate System.

transformed to CCS. These CCS trajectories are then fed to the LSTM-ED model for training and inference. We first define the trajectory prediction problem in section V-A, introduce the CCS and transformations from and to Cartesian coordinates in section V-B, and describe the architecture, training and inference using LSTM-ED in section V-C.

A. Problem Definition

Given an observed partial trajectory of the i-th vehicle in CCS:

$$\boldsymbol{X}_i = [(x_i^{(1)}, y_i^{(1)}), (x_i^{(2)}, y_i^{(2)}), \ldots, (x_i^{(t_{obs})}, y_i^{(t_{obs})})], \quad (3)$$

the model predicts the future trajectory:

$$\boldsymbol{Y}_i = [(x_i^{(t_{obs}+1)}, y_i^{(t_{obs}+1)}), (x_i^{(t_{obs}+2)}, y_i^{(t_{obs}+2)}), \ldots, (x_i^{(t_{obs}+t_{pred})}, y_i^{(t_{obs}+t_{pred})})], \quad (4)$$

where t_{obs} is the original trajectory time cardinality with t_{pred} being the predicted temporal duration.

B. Curvilinear Coordinate System

Curvilinear coordinate systems (CCS) are a natural fit for our problem of trajectory prediction. Consider the problem of predicting left turn trajectories at intersections. In standard Euclidean coordinates, the velocity vector changes its orientation continuously through the turn whereas a reparametrization of the velocity along the curve has the advantage of better inertia representation. CCS are also akin to the use of Lagrangian frames in fluid mechanics (or the viewpoint from the boat in a river) as opposed to the Eulerian frame (or the viewpoint from the riverbank watching the boat float by). The work in [20] uses curvilinear coordinates to impose roadway geometry constraints to motion tracking and behavior reasoning algorithms. We extend the idea to intersection geometry, which implicitly constrains the vehicle trajectories to a set of standard trajectory templates. We propose to use the CCS defined as the shape of a prototype trajectory. Partial trajectories matching with the prototype are assumed to move along with it with an offset. The prediction is largely simplified in the CCS, as the model only needs to learn the difference between a new trajectory and the average of historical trajectories conforming to the same motion pattern.

For a vehicle entering an intersection, the possible motion patterns can be found by matching with trajectory prototypes. Most of the motion patterns can be ruled out, as their distances to the query trajectory are too far to be considered as potential matches. In this work, we only consider the closest 2 prototype trajectories. Each prototype trajectory defines a curvilinear coordinate system with s and n axes—essentially the tangent and normal at each point along the curve. The s-axis is defined as along the shape of a prototype with arc length coordinates used while the n-axis is defined to be perpendicular to the s-axis at every point on the curve giving us a measure of how far a point is from the curve, as shown in Fig. 7. In the rest of this section, we describe the transformations between the Image Coordinate System (ICS) and the Curvilinear Coordinate System (CCS), where Image Coordinates (IC) refer to Google Map aligned intersection image coordinates. We denote a point in IC as (x_p, y_p), and its corresponding point in CC as (s_p, n_p), as shown in Fig. 7.

1) ICS to CCS transformation: Our prototype trajectories are sampled from continuous splines, as in section IV-B.2, and the determination of the closest point on a spline from a query point requires the use of optimization methods. These are now described. Given p with coordinates x_p, y_p in the ICS, we aim to find the corresponding s_p, n_p in the CCS. We first find s_p by finding the closest point on spline S to the point p, which can be formulated as the following minimization problem:

$$\min_{s_p} \|p - S(s_p)\|_2. \quad (5)$$

We use a standard limited-memory (BFGS) algorithm (available in Python scientific libraries) to solve for s_p. Once s_p is determined, we turn our attention to n_p. First, $|n_p|$ is defined as the distance between p and p'. The sign of n_p is determined by:

$$\begin{aligned} &\text{sgn}(\vec{pp'} \times \vec{q'p'}), \\ &q' = S(s_p - \epsilon), \end{aligned} \quad (6)$$

where sgn is the sign of a number, $\times$ is the cross product, q' is a point near p' with a slightly smaller s-value and ϵ denotes a small number. $\vec{q'p'}$ estimates the growing direction of S at point p'. So far we have given the procedure of transforming from ICS to CCS. We now examine the opposite direction.

2) CCS to ICS transformation: Given s_p, n_p in the CCS, we then aim to find the corresponding x_p, y_p in the ICS. This is a relatively easy process, since the transformation has a convenient closed-form expression:

$$\begin{aligned} (x_p, y_p) &= S(s_p) + \vec{p'p}, \\ \vec{p'p} &= n_p \vec{e_\perp}, \end{aligned} \quad (7)$$

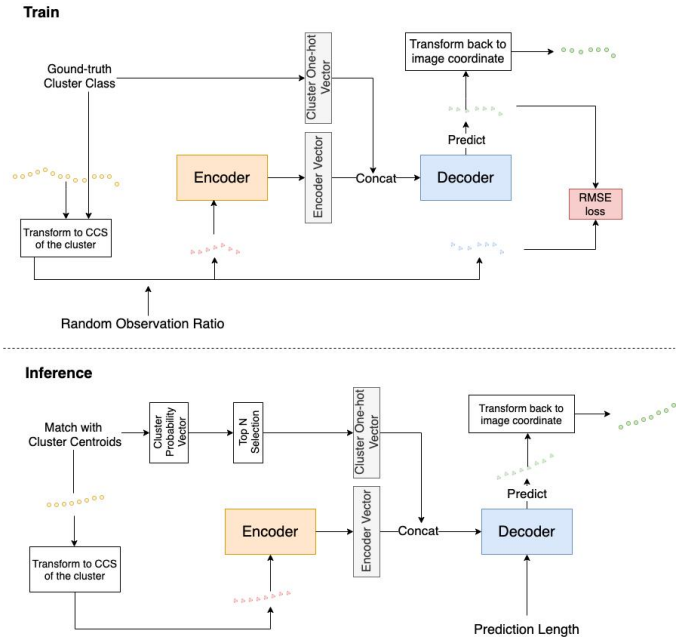

Fig. 8. LSTM encoder-decoder training and inference behavior.

where $\vec{e_\perp}$ is the unit vector perpendicular to S at point p', which can be found from the derivative of S. In summary, the use of CCS simplifies the task for trajectory prediction as it decouples the two coordinates to some extent. The S-coordinate mainly captures the speed along the road while the N-coordinate mainly captures the speed off the road (e.g. lane changes or abnormal behavior). The coupling of prototype trajectories to a "Lagrangian" coordinate system is therefore a vital contribution of this work.

C. LSTM Encoder-Decoder Model

LSTM networks are designed and proven effective for sequence modeling and prediction tasks. Thus, they are well-suited for trajectories represented as a sequence of coordinates. We adopt the encoder-decoder architecture to cope with variable-length observation and prediction period, as the trajectories captured at a intersection are of variable length. The encoder encodes observed partial trajectories to a fixed-length internal representation, and the decoder decodes the state and predicts possible future motions for a given period of time. Besides the internal vector from encoder, cluster belonging is also provided as input to the decoder. In this way, the decoder learns to predict differently considering its cluster.

1) Network Architecture: Both the encoder and decoder consist of two layers: a fully connected (FC) layer and an LSTM layer. The FC layer serves as an embedding function that embeds locations into a fixed-length vector. The embedding will then be fed to the LSTM layer. The encoder encodes the observed trajectory into the LSTM's last hidden state $\boldsymbol{h}_i^{(t_{obs})}$. The decoder takes $\boldsymbol{h}_i^{(t_{obs})}$ concatenated with the one-hot-encoded cluster class vector as input, and is trained to generate its future trajectory $\boldsymbol{Y}_i$.

2) Training: We adopt the L_2 loss for training, which measures the distance between the predicted and the ground-truth trajectories. As vehicles pass an intersection at different speeds, the number of trajectory points captured in the intersection vary over a wide range. For example, a *left turn* trajectory usually has more trajectory points captured than a *straight heading* trajectory, as a *left turn* vehicle will slow down as it enters the intersection, while a *straight heading* vehicle tends to be at the maximum speed limit. For this reason, we enable the model to encode variable-length observations and to decode variable-length predictions. We achieve this by splitting a training sample into an observation and prediction sequence with a random ratio t_{obs}:t_{pred}, so that the model is trained from mixed length data, and will learn to make variable-length predictions.

3) Inference: At inference time, the cluster class of a trajectory i is unknown. The model first infers cluster class by matching with all prototype trajectories. The degree of belonging of trajectory tr_i to cluster class $C^{(m)}$ is calculated by inverse distance weighting:

$$u_{C^{(m)}}(tr_i) = \frac{1/d(tr_i, tr_{C^{(m)}})}{\sum_{j=1}^{n^c} 1/d(tr_i, tr_{C^{(j)}})}, \tag{8}$$

where d is the distance measure introduced in section IV-A.2, and $tr_{C^{(j)}}$ represents the prototype trajectory of class $C^{(j)}$. If the trajectory has a similar degree of belonging to M classes, the model will output M predictions. For each possible cluster class, the model will make the corresponding prediction. To be more specific, for each possible $C^{(j)}$, the corresponding representation in CCS as well as the one-hot cluster vector will be fed to the LSTM-ED model, which then makes its future prediction with "probability" $u_{C^{(j)}}(tr_i)$. We set M to 2 based on intersection geometry, as each lane of an intersection usually allows 2 or less motion patterns (e.g., heading straight and right turn).

4) Implementation Detail: The embedding dimension is set to 64 for both the encoder and decoder. The encoder's LSTM layer has a hidden state of dimension 64 and the decoder's LSTM layer has a hidden state of dimension 64 + n_c. We use the Adam optimizer with learning rate 1×10^{-4}. The t_{obs}:t_{pred} ratio is randomly chosen in the range $[0.3, 0.7]$.

VI. Experiments

We evaluated the proposed approaches on the collected trajectory dataset from surveillance fisheye cameras at 3 intersections. Section VI-A describes our data collection methods as well as pre-processing steps to obtain google-map aligned trajectories. Section VI-B compared our trajectory prediction model with baseline methods. Finally, section VI-C evaluates the trajectory prediction pipeline for variable-length observation and prediction period.

A. Data Collection and Pre-processing

Our intersection trajectory dataset is obtained from three busy intersections in Gainesville, Florida. The surveillance cameras have a circular fisheye lens with 10 fps frame rate.

From our previous work [21], we have the mapping from fisheye video pixels to google map locations. A detection and tracking pipeline is running periodically to automate the trajectory capture process. We clean the dataset using rule-based filtering. We impose speed limits, within the intersection and non self-intersecting constraints on trajectories. We also applied a trajectory smoothing algorithm to compensate for detection imprecision. In addition, as the traffic signal phase is unknown in our setting, we exclude the stay points of trajectories before entering the intersection. After the above process, we obtain roughly 15,000 trajectories for each intersection which are then split it into training, validation and testing set with a roughly 7:1:2 ratio. The validation set is used in LSTM-ED training to avoid over-fitting. We refer to the 3 intersections simply as Intersection 1, Intersection 2 and Intersection 3.

B. Evaluation of trajectory prediction model

In this section, we evaluates the performance of the proposed trajectory prediction model and compare it with baseline models. This experiment is performed on Intersection 1, and we assume the ground truth cluster class of each trajectory is known. We compare the proposed model (LSTM-ED CCS) with two baseline models:

1) LSTM-ED ICS: an LSTM encoder-decoder model without coordinate transformation, otherwise the same as LSTM-ED CCS.
2) GP: a Gaussian process regression model used in [10]

We adopt the following two metrics for prediction evaluation:

1) Average displacement error (ADE): the average of Euclidean distance between ground truth and prediction over all trajectory points.
2) Final displacement error (FDE): the Euclidean distance between the end positions of ground truth trajectory and predicted trajectory.

We show the prediction errors of different observation and prediction periods. To ensure fair comparison for different periods settings, we set a threshold for minimum trajectory length for evaluation. In our case, only trajectories with more than 40 timestamps are chosen, as the maximum of observation timestamps plus prediction timestamps is 40, as shown in Table I. In this way, the same set of trajectories are used for each setting, as opposed to some trajectories (less than 40 timestamps) only used in shorter period settings. Thus, the experimental result is a true reflection of the model's performance for different observation and prediction periods. The prediction errors are reported in Table I.

We find that LSTM-ED CCS outperforms the two baseline models in almost every setting. GP tends to have a large FDE and the prediction result also looks unsmooth. LSTM-ED ICS performs similarly on shorter prediction periods as LSTM-ED CCS, but worse on longer prediction periods.

C. Evaluation of trajectory prediction pipeline

Our pipeline consists of cluster class determination and trajectory prediction. In other words, unlike the previous experiment, the ground truth cluster class is unknown to the LSTM-ED model. We evaluated the pipeline on all three intersections. The quantitative result is reported in Table II. The qualitative result is shown in Fig. 10. As explained in section V-C.3, our model produces multiple outputs if the observed trajectory matches with multiple prototypes. Fig. 9 shows one example of multiple outputs. From the observed trajectory, *going straight* and *right turn* are both likely.

TABLE I

COMPARISON OF TRAJECTORY PREDICTION APPROACHES GIVEN GROUND-TRUTH CLUSTER CLASS

Observation Length		10			20		30
Prediction Length		10	20	30	10	20	10
GP	ADE	0.75	1.86	2.76	0.45	1.20	0.35
	FDE	2.76	4.96	6.08	2.34	4.58	2.69
LSTM-ED ICS	ADE	0.61	1.24	1.96	0.47	0.91	0.45
	FDE	1.13	2.51	4.17	0.86	1.97	0.84
LSTM-ED CCS	ADE	0.51	1.10	1.76	0.45	0.91	0.47
	FDE	0.97	2.32	3.61	0.81	1.87	0.87

* We utilize two metrics: average displacement error (ADE) and final displacement error (FDE) (in meters) to compare the three approaches. The lower the error, the more accurate the approach.

Fig. 9. Two predictions at inference time. The red, blue and green points represent observed, ground-truth future, and predicted points of a trajectory respectively.

TABLE II

PREDICTION ERRORS FOR INTERSECTIONS 1, 2, AND 3

Observation Length		10			20		30
Prediction Length		10	20	30	10	20	10
Intersection 1	ADE	0.54	1.21	2.09	0.52	1.16	0.61
	FDE	1.02	2.70	4.83	0.97	2.61	1.15
Intersection 2	ADE	0.54	1.23	2.05	0.46	0.91	0.49
	FDE	1.03	2.72	4.29	0.79	1.90	0.82
Intersection 3	ADE	0.66	1.40	2.19	0.58	1.18	0.54
	FDE	1.24	2.89	4.38	1.03	2.44	0.99

VII. CONCLUSION

A real-time trajectory prediction approach coupled with aligned google map information is proposed in this paper. Our approach makes use of a historical trajectory database and finds typical motion patterns as a guide for future prediction. Given this prior information, our approach is able to make reasonable predictions based on variable starting position, observation period and prediction period. Experimental results on three intersections show the effectiveness

Fig. 10. Samples of the prediction results.

and extensibility of our approach. In the immediate future, we plan to integrate our trajectory prediction module to an early warning system. We believe our work will help increase the intersection safety.

REFERENCES

[1] E.-H. Choi, "Crash factors in intersection-related crashes: An on-scene perspective," Nat. Highway Traffic Safety Admin., Washington, DC, USA, Tech. Rep., 2010.

[2] A. Houenou, P. Bonnifait, V. Cherfaoui, and W. Yao, "Vehicle trajectory prediction based on motion model and maneuver recognition," in *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2013, pp. 4363–4369.

[3] J. M. Scanlon, R. Sherony, and H. C. Gabler, "Injury mitigation estimates for an intersection driver assistance system in straight crossing path crashes in the united states," *Traffic injury prevention*, vol. 18, no. sup1, pp. S9–S17, 2017.

[4] X. Huang, S. G. McGill, B. C. Williams, L. Fletcher, and G. Rosman, "Uncertainty-aware driver trajectory prediction at urban intersections," in *2019 International Conference on Robotics and Automation (ICRA)*. IEEE, 2019, pp. 9718–9724.

[5] J. M. Scanlon, R. Sherony, and H. C. Gabler, "Preliminary potential crash prevention estimates for an intersection advanced driver assistance system in straight crossing path crashes," in *2016 IEEE Intelligent Vehicles Symposium (IV)*. IEEE, 2016, pp. 1135–1140.

[6] S. Lefèvre, D. Vasquez, and C. Laugier, "A survey on motion prediction and risk assessment for intelligent vehicles," *ROBOMECH journal*, vol. 1, no. 1, pp. 1–14, 2014.

[7] D. Vasquez and T. Fraichard, "Motion prediction for moving objects: a statistical approach," in *IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA'04. 2004*, vol. 4. IEEE, 2004, pp. 3931–3936.

[8] J. Joseph, F. Doshi-Velez, A. S. Huang, and N. Roy, "A Bayesian nonparametric approach to modeling motion patterns," *Autonomous Robots*, vol. 31, no. 4, p. 383, 2011.

[9] Q. Tran and J. Firl, "Online maneuver recognition and multimodal trajectory prediction for intersection assistance using non-parametric regression," in *2014 IEEE Intelligent Vehicles Symposium Proceedings*. IEEE, 2014, pp. 918–923.

[10] S. A. Goli, B. H. Far, and A. O. Fapojuwo, "Vehicle trajectory prediction with Gaussian process regression in connected vehicle environment," in *2018 IEEE Intelligent Vehicles Symposium (IV)*. IEEE, 2018, pp. 550–555.

[11] A. Alahi, K. Goel, V. Ramanathan, A. Robicquet, L. Fei-Fei, and S. Savarese, "Social LSTM: Human trajectory prediction in crowded spaces," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 961–971.

[12] B. Kim, C. M. Kang, J. Kim, S. H. Lee, C. C. Chung, and J. W. Choi, "Probabilistic vehicle trajectory prediction over occupancy grid map via recurrent neural network," in *2017 IEEE 20th International Conference on Intelligent Transportation Systems (ITSC)*. IEEE, 2017, pp. 399–404.

[13] N. Deo and M. M. Trivedi, "Convolutional social pooling for vehicle trajectory prediction," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, 2018, pp. 1468–1476.

[14] S. H. Park, B. Kim, C. M. Kang, C. C. Chung, and J. W. Choi, "Sequence-to-sequence prediction of vehicle trajectory via LSTM encoder-decoder architecture," in *2018 IEEE Intelligent Vehicles Symposium (IV)*. IEEE, 2018, pp. 1672–1678.

[15] T. Zhao, Y. Xu, M. Monfort, W. Choi, C. Baker, Y. Zhao, Y. Wang, and Y. N. Wu, "Multi-agent tensor fusion for contextual trajectory prediction," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019, pp. 12 126–12 134.

[16] Y. Cai, M. Tian, W. Yang, and Y. Zhang, "Stay point analysis in automatic identification system trajectory data," in *Proc. 2018 Int. Con. Data Science*, 2018, pp. 273–278.

[17] F. H. A. US Department of Transportation, "Traffic signal timing manual," https://ops.fhwa.dot.gov/publications/fhwahop08024/-chapter4.htm, 06 2008.

[18] T. Banerjee, X. Huang, K. Chen, A. Rangarajan, and S. Ranka, "Clustering object trajectories for intersection traffic analysis," in *6th International Conference on Vehicle Technology and Intelligent Transport Systems (VEHITS 2020)*, 2020.

[19] H. Wang, J. Kearney, and K. Atkinson, "Arc-length parameterized spline curves for real-time simulation," in *Proc. 5th International Conference on Curves and Surfaces*, vol. 387396, 2002.

[20] K. Jo, M. Lee, J. Kim, and M. Sunwoo, "Tracking and behavior reasoning of moving vehicles based on roadway geometry constraints," *IEEE Transactions on Intelligent Transportation Systems*, vol. 18, no. 2, pp. 460–476, 2016.

[21] X. Huang, T. Banerjee, K. Chen, N. V. S. Varanasi, A. Rangarajan, and S. Ranka, "Machine learning based video processing for real-time near-miss detection." in *VEHITS*, 2020, pp. 169–179.