# Chapter 5

# Trajectory Prediction and Motion Planning

The description of the proposed models is given in Section 5.2. The next section gives the result generated by the proposed models and their analysis. Section 5.3 concludes the chapter.

## 5.1 Introduction

Autonomous driving is currently an active research area and includes many problems related to TP and understanding the behavior of the surrounding dynamic agents. Safe driving is very challenging for the ADS, which needs to be given more attention to facilitating the extensive use of AVs. AVs must accurately predict the surrounding road agents' trajectories to avoid accidents. The problem of TP is the long-term (3-5 seconds) or short-term (1-3 seconds) spatial coordinates [226] of other road agents such as pedestrians, bicycles, buses, cars and animals etc. Thus, the TP problem is very crucial for the existence of safe travel of the AV.

Recent Work for AVs [227] [228] considers individual motion prediction for the TP. The lack of temporal information of the agents is the cause of inadequate

TP for ADS [229]. There is challenging to provide spatial and temporal information about the road agents to predict the future trajectory. Many researchers early proposed Graph Neural Network (GNN) based deep-learning architecture to address this issue. GNN and its variant [230] are more effective strategies for representing data in the non-euclidean space and are suitable for creating road agents' interactions. A graph-based network aims to learn to embed each node containing information about the neighboring road agents (nodes can connect directly with the target node through edges).

Early work Social-BiGAT utilized graph attention networks [231] to interaction modeling for the agents. Recur-Social [232] [233] introduced a group-based model for social interactions. Model Trajectron [234] proposed a graph structure that simultaneously predicts future trajectories for multiple road agents. These approaches have individually performed better than their Recurrent Neural Network (RNN) and Graph-Convolutional Neural Network (Graph-CNN). However, previous studies were analyzed on hybrid Graph-CNN and RNN architectures implemented, but the road agents have different dynamics, movement patterns and shapes. To ensure better driving decisions, the AV's navigation and perception system should be able to analyze the patterns of motion of the other road agents and accurately predict their trajectory. Not only but also the main challenge for the TP is ensuring more and more accurate long-run prediction (3 to 5 seconds). As increases the prediction horizon, error in the predictions also increases, often due to the diminished influence of the temporal correlation. An attempt was made to resolve this problem in the long-term prediction of trajectories.

Graph structure has been used for modeling the problem, with a unique graph representing the input for each time-step. Each graph node (or vertex) will represent a road agent according to its spatial coordinates. Multi-scale Graph Neural networks were built with the temporal conv architecture, which will directly operate on the Graph, taking it as the input for each time-stamp. The data generated by the previous layers are now fed into a TP Model consisting of a Long-Short Term Memory

(LSTM) based Encoder-Decoder network. It takes the Graph Feature data as its input. Feature vectors of a graph, data are passed through into the corresponding encoder input cell. Inside the encoder and decoder is an inbuilt graph convolution layer to take care of the inter-object interactions from the corresponding adjacency matrix.

Autonomous driving is currently an active area of research and planning the motion of the ego vehicle is one of the most critical problems in this context. Motion planning is a complicated problem as the ego vehicle, while maneuvering, must take care of the pedestrians, crossing cars, interactions at the intersections of roads and the road-signs in urban conditions. Due to such complexity of the problem, it is impossible to plan the motion just based on some classic hand-crafted rules and needs to be handled by some machine learning or deep learning-based model which can learn the driving rules from varied data. In this project, the model was designed and implemented, which takes images as input for planning the motion of the ego vehicle and generates a sequence of coordinates representing the waypoints in the predicted path of the vehicle for the upcoming four time-steps. It must also be noted that the planning task involves not only the forecasting of the movement and path of the vehicle but also the derivation of the driving parameters, like steering angle, throttle and brake value, from the forecasted path. An inverse dynamic algorithm was used to derive the values of the driving parameters from the coordinates of the waypoints, which represent the path predicted for the upcoming instances of time through which the ego vehicle is advised to maneuver.

Lillicrap et al. [235] focused on reaching the final destination point and not caring about the interactions or traffic like a sports car. In [236] [237] used the sensor fusion method and fused LiDAR like bird's eye view with an image front camera that became the model complex and costly. Chen et al. [238] explored spatial and temporal attention to the weight of distinct time-step image inputs for better detection, resulting in smoother lane transition. In the previous image-based proposed models, either a single image from the front camera is used as an input, or

the sensor fusion method is used where LiDAR data in bird's eye view is fused with the front camera image computationally more expensive than using only images. To solve this problem, three images were used to capture the whole environment and context in which the ego vehicle is traveling and solve the problem of lack of information about the surroundings, which arises when a single image is used, and the processing of the input is less expensive than sensor fusion.

## 5.2 Proposed Methods and Models

### 5.2.1 Graph Neural Network-based Trajectory Prediction

A data preprocessing strategy for effectively selecting the semantic road links is more efficient for TP. The proposed method has a backbone network with a spatial-temporal graph convolutional network for the TP. The proposed network overcomes the existing issues of the state-of-the-art techniques to extract the semantic information for the TP accurately. On three large-scale datasets for the TP, the proposed model outperforms existing approaches methods by a significant difference in accuracy. The proposed backbone network extracts multi-scale features using graph and LSTM network for the TP. The proposed architecture surpasses the issues of existing state-of-the-art methods by extracting the semantic features of the TP. The model encoded multi-scale spatial features extracted with GNN to get more semantic features that are useful to capture the neighbor's local geometry which contains a series of the distance between the objects or set of nearest neighbors. To maintain the connection among the dynamic obstacles and occupy an important position in measuring the filter collect and graph dynamicity of the multi-scale features by fusing the short-scale and long-scale parameters. The proposed model outperforms with significant results compared to existing methods with a large margin of accuracy. A depth-analysis of the proposed model with LSTM shows the excellence of the proposed model at different time-instance for these three datasets.

The general problem associated with the long-term prediction of the trajectories in most of the existing models lies in the fact that the relevance of the maneuver pattern of each object while making the future predictions, though captured initially, keeps on diminishing as the data progresses through the sequence-to-sequence model for more and more time-steps. Other possible solutions for this problem were explored. If the pattern can be highlighted before giving the input for the encoder itself, the problem of the diminishing effect of the pattern through time-steps will reduce. Hence, three Graph Temporal Convolution layers were used in the proposed model before giving the input to the encoder. The details of the model's architecture, working, implementation, and training are elaborated in the successive sections.

#### 5.2.1.1 Pre-processing of Datasets

Each dataset is first preprocessed to a text file where each row contains the columns: Frame ID, Object ID, X coordinate, Y Coordinate, and Dataset ID. After preprocessing, the Frame IDs range between 1 to n, and Object IDs range from 1 to N. So, the total number of frames is n, and the total number of road agents and objects is N. Each dataset uses a different convention to represent the Frame IDs (for example, few datasets use Time-stamp as Frame ID), and these initial IDs are mapped to an integer between 1 to n. Similarly, for Object IDs, there are also different initial representations (for example, a few datasets represent Object IDs using a string of characters), and they are mapped to an integer between 1 to N. Even if the Frame IDs and Object IDs of any dataset already range between 1 to n or 1 to N, it is confirmed that they are sequential and there are no missing IDs.

The Dataset IDs are used to differentiate different scenes of the same dataset since the objects or agents are not preserved across scenes. Then the text files are then converted to .npy format. Based on the frame rate at which the trajectories of the vehicles are recorded in the concerned dataset and the training sequence time and prediction sequence time interval, the training sequence length (train_seq_len) and

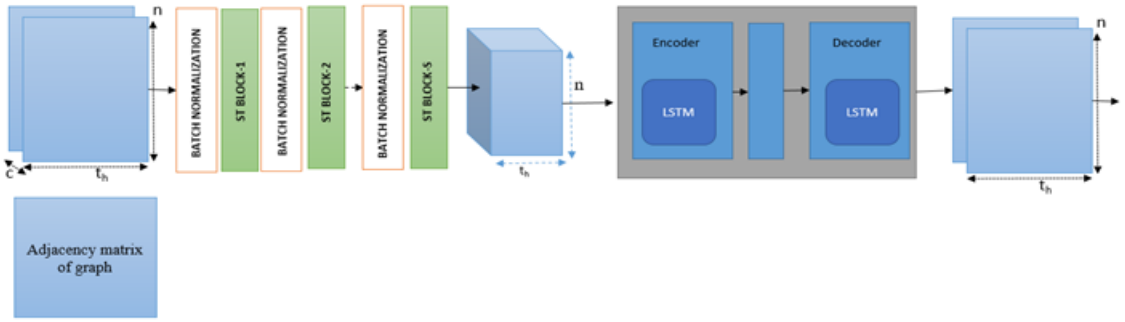



FIGURE 5.1: Architecture of temporal graph model for Trajectory Prediction

prediction sequence length (pred_seq_len) for each dataset are determined. Further, it is ensured that: frame_length_cap (train_seq_len + pred_seq_len). The model is used this frame_length_cap to enforce an Object ID in at least the frame_length_cap number of frames.

If any dataset is too large, only a few scenes from the complete data may be considered and the Dataset IDs list is used to tweak the values and reduce the amount of data. Finally, the pickle files are generated and used for training and evaluating the models. The size of the training, testing and validation samples for each dataset is given by the number of T-second-long observation sequences of the position coordinates of the road agents, where 'T' is the observation time [T=3 is taken here].

#### 5.2.1.2 The architecture of the model and its working

A schematic representation of the basic architecture of the model is shown in Fig- 5.1 based Encoder-Decoder network for the TP. The proposed model does not directly take velocity as an input feature but instead uses temporal layers to extract features from a vehicle's trajectory throughout time. As velocity is a temporal feature, i.e., it is derived by changing position coordinates over time, the temporal layer indirectly considers/extracts it as a feature.

#### 5.2.1.3 Generation of the Graph

Graphs are then generated for each frame from the NumPy arrays obtained after preprocessing from each dataset. The graphs represent the inter-agent interactions, which are essential for accurate prediction of the trajectories of the road agents as the maneuver of the agents are influenced by the surrounding agents, especially in dense traffic conditions. The steps involved in this process of graph generation are:

- The data is first grouped based on the frame_id. Each frame will have a separate graph.
- The object_id of the objects in each frame is then encoded to a label between 0 to $N_{f-1}$ to create the nodes of the Graph so that the corresponding Graph has $N_f$ nodes.
- Each node in a graph has two features: the x-coordinate and y-coordinate of a corresponding object or road agent.
- After this, the edges of the Graph are constructed. The edges are undirected. Out of the possible ${}^{Nf}C_2$ edges, only those are present for which $d(o_i, o_j) < \mu$, where $o_i$ and $o_j$ are the objects corresponding to the nodes connected by that edge, and $d(o_i, o_j)$ is the Euclidean distance of two objects $o_i$ and $o_j$. $\mu$ is a threshold parameter chosen heuristically, and here $\mu = 10$ meters has been taken by considering the rare size of the road agents and the road width [226].
- The graph G = (V, E) thus corresponds to a particular frame, where V is the set of objects in the frame and E is the set of undirected edges generated based on step-4 as described above.
- The adjacency matrix A has been used for the representation of the graphs. The size of the adjacency matrix is made to be N×N to fix the size of the input, where N is the maximum number of objects observed in a single frame

for the corresponding dataset. If the number of objects in the current frame is less than N, all extra rows and columns are filled with zeros.

- The representation of an undirected graph consider as G = V,E,A where number of nodes N = $|V|$. The edges connecting the nodes are $|E|$, and $A = \mathbb{R}^{N \times N}$ is the adjacency matrix encodes the connections. The input representation of graph G is $F_{in} \, \epsilon \, \mathbb{R}^N$. After the Fourier transform of graph G could be transformed, the convolutional filtering of the spatial domain into the spectral domain for inner-product operation.

- Specifically, the Laplacian of graph G is $L_P = I_n D^{-1/2} A D^{-1/2}$ and $D_{ij} = \sum_j \lambda_{ij}$ are used for Fourier transform. Then g operator is used for the graph filter that parameterized by $\theta$, can be represented as

$$F_{out} = g_\theta \left(L_p\right) F_{in} = U g_\theta \left(\lambda\right) U^T F_{in} \tag{5.1}$$

Where $F_{out}$ represents the feature output for graph input, U represents the Fourier basis, $L_p = U^T$, and the $\lambda$ represents the eigenvalue corresponding to $L_p$. Hammond et al. [239] verified that $g_\theta$ filter could approximate the Chebyshev polynomials [240] $R^{th}$ order,

$$F_{out} = \sum_{r=0}^{R} \theta_r^{'} T_r \left(Lp^{'}\right) F_{in} \tag{5.2}$$

Where $\theta_r^{'}$ denotes Chebyshev coefficients $\theta_r^{'}$. The Chebyshev polynomial is recursive and defined as

$$T_r \left(Lp^{'}\right) = 2L^{'} T_{r-1} \left(Lp^{'}\right) - T_{r-2} \left(Lp^{''}\right) \tag{5.3}$$

- With $T_0 = 1$ and $T_1 = Lp'$. $Lp^{'} = \frac{2L}{\lambda_{\max}} - I_n$ is normalized between [-1, 1]. The $g_\theta$ filtered the graph Lp that can be well approximated as illustrated in equation-1 (represented by a linear combination of inputs) transformed by

Chebyshev polynomials. Consequently, it can formulate the linear transformation of Chebyshev polynomial with kth order and the receptive field of k with spatial graph convolution.

#### 5.2.1.4 Spatial Sampling of Adjacency Matrix for Spatial Features

The module of feature sampling is reformulated (like the spectral formulation of graph convolution) in matrix multiplication between agent representation and transformation of Graph explained by Chebyshev polynomials. The connection of hops between the agents is $r^{th}$ order. Thus in this work, R = 3 has been considered to approximate the filtering operation of the multi-scale Graph. The number of increments in order of approximation may cause higher performance with more computational cost, but the focus of this proposed paper is not this. The representation of $0^{th}$ to $3^{rd}$ order Chebyshev polynomials are represented as follows.

$$T_0 = I, \tag{5.4}$$

$$T_1 = Lp^{'} \tag{5.5}$$

$$T_2 = 2Lp^{'2} - I \tag{5.6}$$

$$T_3 = 4Lp^{'3} - 3Lp^{'} \tag{5.7}$$

As shown in Fig-5.2, the spatial path of the backbone network has three branches corresponding with the 1st to the third-order transformation of graph G, respectively. The identical transformation represents $T_0$, similar to a residual path.

#### 5.2.1.5 Convolutional Module

The convolutional module of graph G is generated the graph features for each scale separately. The convolution module has a $1 \times 1$ convolutional layer, batch normalization and leaky Relu layer. The output features are set to one-third of the spatial path width for efficient computations.

#### 5.2.1.6 Temporal Features

In the Graph Temporal Convolution layer, the convolution operation is first applied using a kernel of size $(1 \times 3)$ along with the temporal dimension for data processing, which is the second dimension in this case. After the temporal convolution operation, the stride is set to 1, and appropriate padding is done to maintain the feature matrix shape. The output of the convolution operation is a feature matrix, say $f_{tempconv}$. Once the temporal convolution is done, the interaction among the objects is now considered using the adjacency matrix of the Graph. It uses multiplying normalized form of the adjacency matrix A with $f_{tempconv}$ representing with a formula as given below:

$$f_{final} = \sum_{j=0}^{1} \lambda_j^{-1/2} A_j \lambda_j^{-1/2} f_{temp-conv} \tag{5.8}$$

Where $\lambda_j$ can be computed as $\lambda_j^{ii} = \sum_k (A_j^{ik}) + \alpha$, here, $\lambda^{1/2} A \lambda^{1/2}$ is a normalized form of A to maintain the default range of the values of the coordinates in the feature matrix and is used to avoid empty rows in $A_j$ as in [241], [242].

#### 5.2.1.7 Feature Fusion Module

The module of the feature fusion is used for more discriminative and robust representation by aggregating the output of all order ($1^{st}$ to $3^{rd}$) spatial paths. In this method, the module of feature fusion is generated to concatenate all output layers to apply on $1 \times 1$ convolutional layer, which is set according to the input channels to maintain the cardinality.

#### 5.2.1.8 Path Fusion

The final merging stage has the outputs of the residual path, the Spatio-temporal path, that are aggregated by summation and inputting into the LSTM network for future TP. The torch has been used as it is impossible to perform the conventional matrix multiplication between $f_{temp_conv}$ of each layer due to dimensional

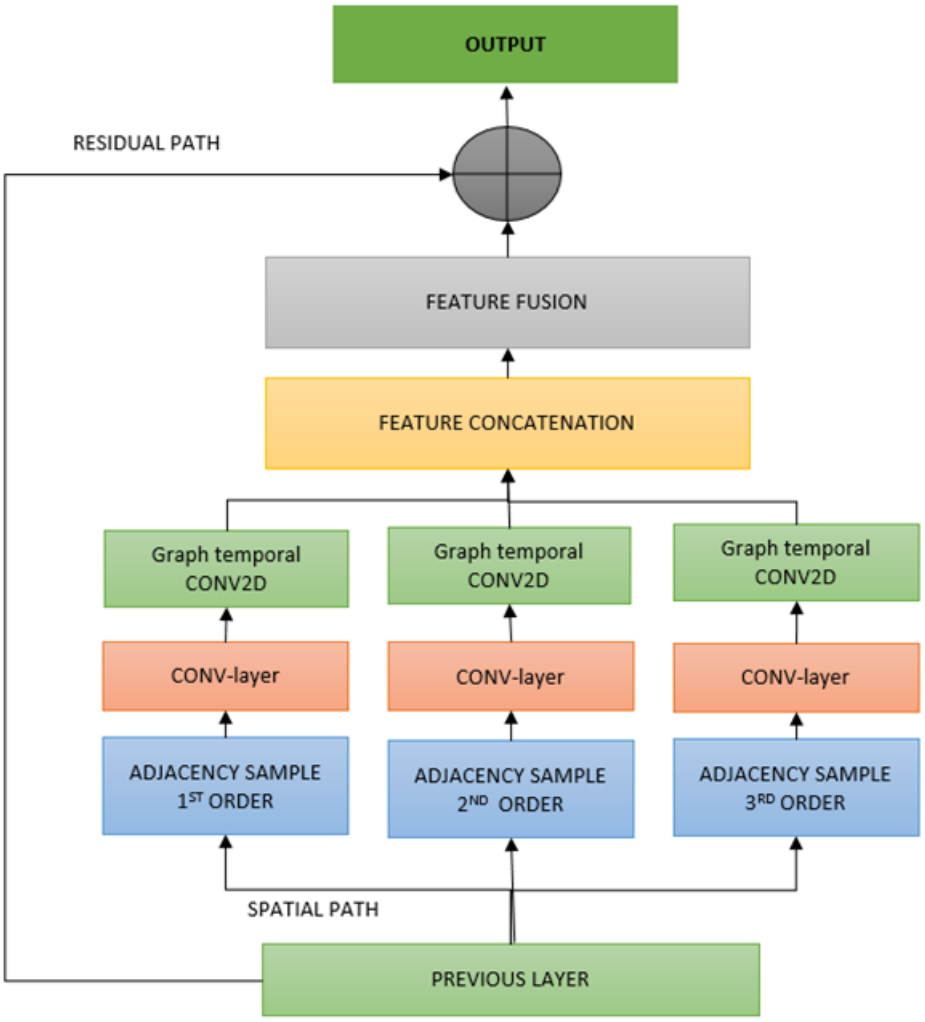



FIGURE 5.2: The architecture of each spatial-temporal block for Graph convolutional network

differences .einsum function here is to operate according to the Einstein summation [243]. Convention.

#### 5.2.1.9 Trajectory Prediction Module

The features generated by the previous layer are now fed into a TP model consisting of LSTM based Encoder-Decoder Network. It takes the graph features as its input. Feature vectors of the graph feature data are fed into the corresponding

input cell of the encoder. Inside the encoder and decoder is an inbuilt graph convolution layer to take care of the inner object interactions from the corresponding adjacency matrix. The output of the last Graph Temporal Convolution layer is fed

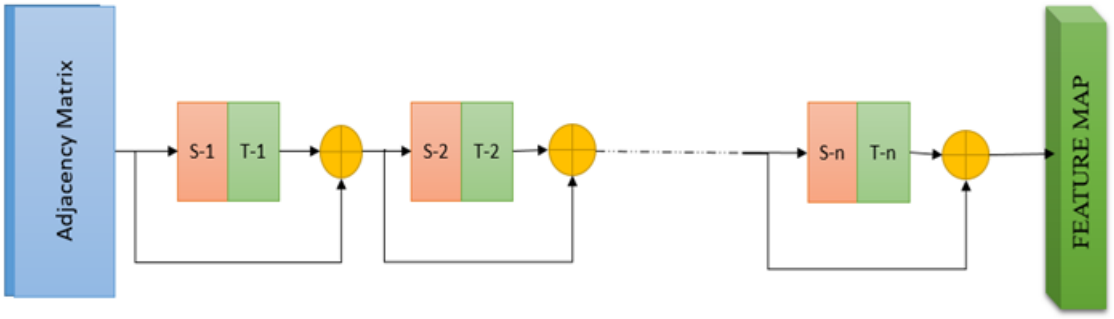



FIGURE 5.3: Basic architecture of backbone network

into a TP Model consisting of LSTM based Encoder-Decoder network. The $f_{final}$ generated by the last graph temporal convolution layer is given as input for the LSTM encoder at every time-stamp. Then, the hidden feature that represents the coordinates of the objects of the LSTM encoder passes through an LSTM decoder to predict the coordinate locations of the objects for the next time step. The decoding process must be repeated several times whenever the model predicts the location for all $t_f$ time-steps. Finally, the future predicted trajectories are determined for n objects and the next $t_f$ time steps.

### 5.2.2 Result Analysis and Discussions for TP

#### 5.2.2.1 Experimental setup

The model was trained with PyTorch (version 1.8.0) for 50 epochs at 128 batch sizes. The Adam optimizer (torch.optim.Adam) has an initial learning rate of 0.002 and other default parameters, as in PyTorch for the convolution layers. RMSProp optimizer (torch.optim.RMSprop), with an initial learning rate of 0.002 and other default parameters as in PyTorch, for the LSTM encoder-decoder network. The initial parameters/weights of the spatial Graph Temporal Convolution layer were initialized by the default initialization method of PyTorch. PyTorch, by default,

uses the Kaiming Uniform Initialization method (reference) for the initialization of weights and biases for the layers used by the proposed model.

The layers are trained by the corresponding optimizers for the below-mentioned loss function, using back-propagation. The training and testing data are separately available for each dataset, so cross-validation is unnecessary. The network was trained separately for each dataset, as the datasets are from different traffic scenarios, some being sparse and others being dense datasets, While in all other approaches, the training is done separately for individual datasets. The Graph-LSTM (Stream 1) and GRIP models were trained and evaluated by us. The other approaches directly reported the results from the other state-of-the-art methods.

#### 5.2.2.2 Loss Function

The Mean Squared Error loss function has been used between the PT and the corresponding GT labels to train the model. The complete loss can be calculated as:

$$Loss = \frac{1}{t_f} \sum_{t=1}^{t_f} loss_t = \frac{1}{t_f} \sum_{t=1}^{t_f} \left\| Y_{pred}^t - Y_{GT}^t \right\|^2 \tag{5.9}$$

Where $t_f$ denotes the number of future time steps, loss t represents loss at t time, $Y_{GT}$ and $Y_{pred}$ are ground truth (GT) and predicted positions, respectively. The model was for loss minimization.

Compared to the other existing models, the percentage of decrease in the errors (compared to the other models) is more in sparse datasets as illustrated in Table-5.1 and Table-5.2 like Lyft and Argoverse datasets. The average density (average no. of objects in each time-frame) is less in these datasets, which implies that owing to less traffic density, the objects will be more likely to follow their movement pattern than being influenced by other road agents. The ADE is less in datasets like Apolloscape due to greater density. The reduction in error is more for FDE than ADE, as in the case of ADE still captured the pattern in the initial time-steps, but for FDE, it was diminished at the final time-steps of prediction. With the

addition of the Temporal Convolution layers, the pattern being highlighted initially, the diminishing effect is undoubtedly reduced. The reduced FDE shows the better long-term prediction ability of the model. Predicted trajectories and the originally provided GT trajectories of some agents (two agents are randomly chosen from each dataset) are shown pictorially using graphs in this section. The prediction time is 0.85 seconds on average for 30 frames for the proposed model, and the runtime of the proposed model is 35.26 fps.

TABLE 5.1: Comparison of the metrics values obtained for Apolloscape and Lyft datasets using for the model with the other existing models

| **Dataset** | **Apolloscape** (Pred. interval=3 sec.) | | **Lyft** (Pred. interval=5 sec.) | |
|---|---|---|---|---|
| ***Models*** | **ADE** | **FDE** | **ADE** | **FDE** |
| **Proposed Model** | **1.49** | **1.89** | **1.57** | **1.25** |
| Graph LSTM (S1) [226] | 3.59 | 3.71 | 5.32 | 5.79 |
| GRIP [241] | 1.25 | 2.34 | 3.04 | 3.21 |
| GRIP++ [244] | 1.25 | 2.36 | - | - |
| CS-LSTM [245] | 2.144 | 11.699 | 4.423 | 8.64 |
| TraPHic [226] | 1.283 | 11.674 | 5.031 | 9.882 |
| Social-GAN [229] | 3.98 | 6.75 | 7.86 | 14.34 |
| SCOUT [246] | 1.26 | 2.35 | - | - |
| UST [247] | 1.24 | 2.25 | - | - |
| F-Net [248] | 1.7 | - | - | - |
| Hybrid Approach for Vehicle TP [249] | - | - | 2.132 | 5.079 |

In the proposed work, the size of the adjacency matrix is considered to be $N$, where N is the maximum number of objects observed in a single frame. The prediction time depends on the value of N, and the relation between the two is analyzed below. By analyzing the traffic densities of the datasets, the average density of the Lyft dataset is found to be 0.82, the average density of the Apolloscape dataset is 3.19, and that of the Argoverse dataset is 1.03. Apolloscape is the most dense dataset. Hence it is safe to use its analysis as an upper bound. The maximum value of N found among the three datasets is 120, and therefore N was set to 120. Moreover, the prediction time for a one-second-ahead prediction is 0.14 seconds. The

TABLE 5.2: Comparison of the metrics values obtained for Argoverse dataset with prediction interval = 5 and the other existing models

| **Methods** | **ADE** | **FDE** |
|---|---|---|
| **Proposed Model** | **0.53** | **0.87** |
| Graph LSTM (S1) [226] | 2.40 | 3.09 |
| CS-LSTM [245] | 1.05 | 3.09 |
| TraPHic [250] | 1.03 | 3.08 |
| Social-GAN [229] | 3.61 | 5.39 |
| VectorNet [118] | 1.66 | 2.21 |
| LaPred [107] | 1.48 | 3.29 |
| DATF [105] | 2.04 | 3.69 |
| CSG [251] | 1.39 | 2.95 |
| Probabilistic Multimodal Trajectory Prediction with Lane Attention [103] | 1.24 | 2.49 |
| UST [247] | 1.47 | 2.94 |

Table-5.3 below shows the change in time required for a one-second-ahead prediction with the 'N' value increase. The real-time prediction is thus not made for about $N \geq 630$. N=630 implies that 630 road agents are observed in a single frame by the camera/LIDAR (as the case may be) mounted on the top of the Ego vehicle, which is only possible in case of really heavy traffic and is not very frequent in the real-world.

TABLE 5.3: Change in time required for one-second-ahead prediction with increasing value of N

| **# of road-agents (N)** | **Time required for one-second-ahead prediction (in seconds)** |
|---|---|
| 120 | 0.14 |
| 240 | 0.32 |
| 480 | 0.73 |
| 500 | 0.77 |
| 550 | 0.86 |
| 600 | 0.95 |
| 625 | 0.99 |
| **630** | **1.01** |

The FNET is the fusion architecture of CNN and RNN with an attention mechanism that extracts the features with adequate progress of both RNN and CNN. Hybrid Approach for VTP using weighted integration of MM [249] combines learning, physics and maneuver-based model according to their uncertainties for the TP. UST [247] used a spatial and temporal context that intended behaviors and interactions from traffic participants. CSG [251] socially acceptable trajectories that allow socially acceptable trajectories and controlled generation of diverse, based on user-controlled speed. CSG predicts the future speed during trajectory forecasting from its generation and latent space.

The CS-LSTM [226] generates the decision tree (T-Pattern Tree) using the trajectory pattern under a particular area and is also used as a predictor for the future location of a future trajectory and getting the best matching path of the tree. The CS-LSTM [226] utilized three different best-matching methods to classify a new moving object trajectory. Social GAN [229] combines the tools of a Generative Adversarial Network based on a seq to seq model that observes the history of motion and future behavior prediction and aggregates the people's information using a novel pooling technique and sequence prediction tools. Traphic [250] proposed a hybrid method using CNN and LSTM for the TP that is useful for modeling the interaction between the dynamic agents.

DATF [105] synthesized a model that multiple inputs signal from the multi-model world the environment unscene context and interaction between the dynamic objects for all admissible and diverse trajectories. VectorNet [118] proposed a hierarchical GNN that first explores the locality of individual road objects represented by a vector and then generates higher-order interactions among the road-components. SCOUT [246] proposed an attention-based graph neural network that uses a flexible and generic representation of the scene as a graph for modeling interactions. Grip [241] and Graph-LSTM [226] used the graph neural network for the feature extraction and LSTM to predict the long-term prediction, while Grip++ [244] is the enhanced of the Grip [241] that aggregate a temporal context to extract the

temporal features with spatial features. The proposed approach deviates from the state-of-the-art in the following manner:

1. The proposed model over the existing model by the spatial correlation leads to a significant improvement in the TP. The Eigenvector matrix needs to compute the eigenvalue decomposition of the graph laplacian matrix and hence suffers from the time complexity $O(n^3)$, which is inappropriate for the large-scale graph. Though the eigenvectors can be precomputed, the time complexity is still $O(n^2)$ and the third can learn the parameters with O(n) for each layer. While the Chebyshev, to be specific, use the first r eigenvectors of V that carry the most smooth geometry of the Graph and the Chebyshev polynomial has only k learnable parameters $k << n$, hence their learning complexity is O(k).

2. The larger graph size leads to high complexity to compute the node output and is not scalable. Most existing models have the computation bottleneck and should hop over a limited number of neighbors (at most k hops from the central node). This leads to applying approximation to the spectral graph convolution using the Chebyshev polynomial. The Chebyshev polynomial approximation is used to compute the spectral graph convolution to reduce the computational complexity.

3. The proposed model encoded the multi-scale spatial-temporal Graph neural network and extracted the more refined features that help to capture the local geometry of neighborhoods defined by a series of the distance between objects or sets of nearest neighbors. Keep the connection between the dynamic agents and occupy an important position in capturing the dynamicity of graphs and filter leverages the multi-scale information by adding long-scale and short-scale parameters.

**Apolloscape Dataset** In Fig-5.4(a), the GT trajectory and the PT of the object or road agent, labeled with agent_ID=337 in the Apolloscape dataset are shown for the prediction time interval. The corresponding ADE and the FDE, in meters, for

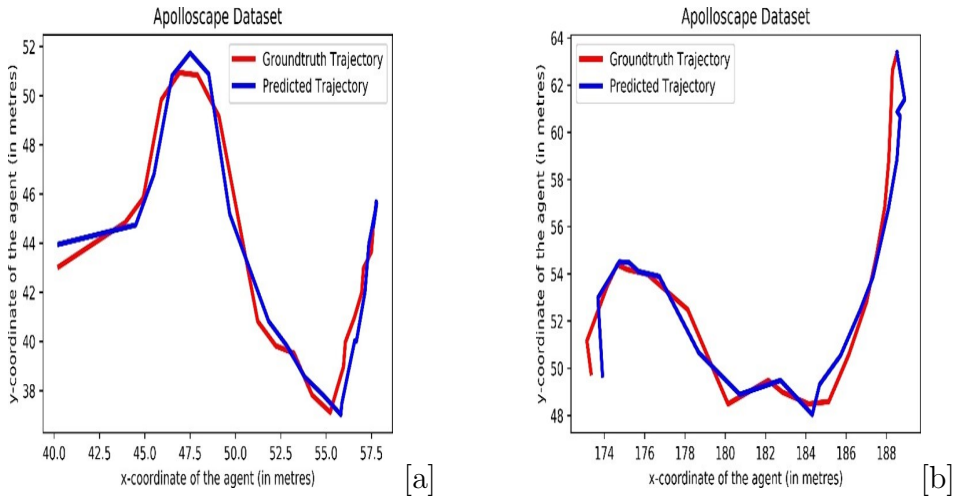



FIGURE 5.4: (a) GT trajectory and PT of the road-agent with agent_ID = 337 of the Apolloscape Dataset (b) GT trajectory and PT of the road-agent with agent_ID = 659 of the Apolloscape Dataset

this object in the considered interval are ADE=1.486 and FDE=1.913. Considering the object's length, width, and height, which are 11.773 meters, 2.716 meters, and 2.86 meters, respectively, the PT error can be considered insignificant.

In Fig-5.4(b), the GT trajectory and the PT of the road-agent with agent_ID = 659 in the Apolloscape dataset are shown. In meters, the corresponding ADE and the FDE for this object in the considered interval are ADE=1.727 and FDE=1.945. The length, width, and height of the object are 9.835 meters,2.764 meters and 2.93 meters, respectively. The PT error can be considered relatively negligible.

**Lyft Dataset** In Fig-5.5(a), the GT trajectory and the PT of the road agent with agent_ID = 2 in the scene labeled with scene_ID=127 of the Lyft (level-5) dataset are shown. The corresponding ADE and the FDE, in meters, for this object in the considered interval are ADE=1.964 and FDE=2.345.

In Fig-5.5(b), the GT trajectory and the PT of the road agent with agent_ID = 16 in the scene labeled with scene_ID=141 of the Lyft (level-5) dataset are shown. In meters, the corresponding ADE and the FDE for this object in the considered interval are ADE = 2.047 and FDE = 2.258.

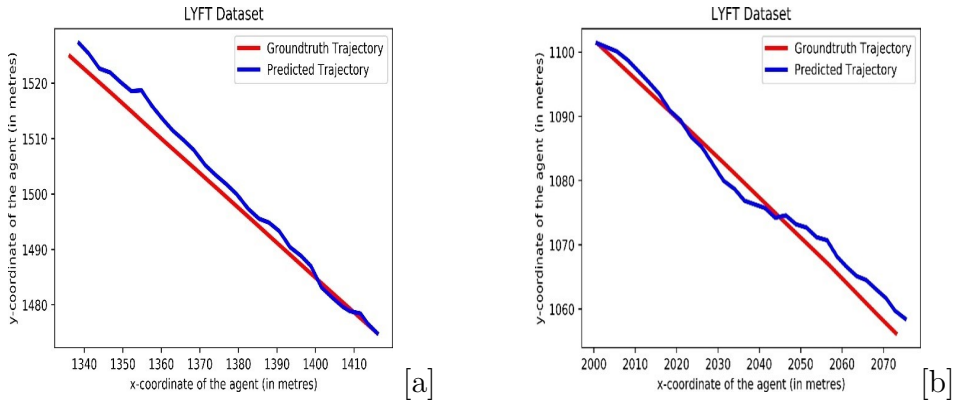



FIGURE 5.5: (a)GT trajectory and PT of the road-agent with agent_ID = 2 of a scene with scene_ID = 127 of the Lyft Dataset (b) GT trajectory and PT of the road-agent with agent_ID = 16 of a scene with scene_ID = 141 of the Lyft Dataset

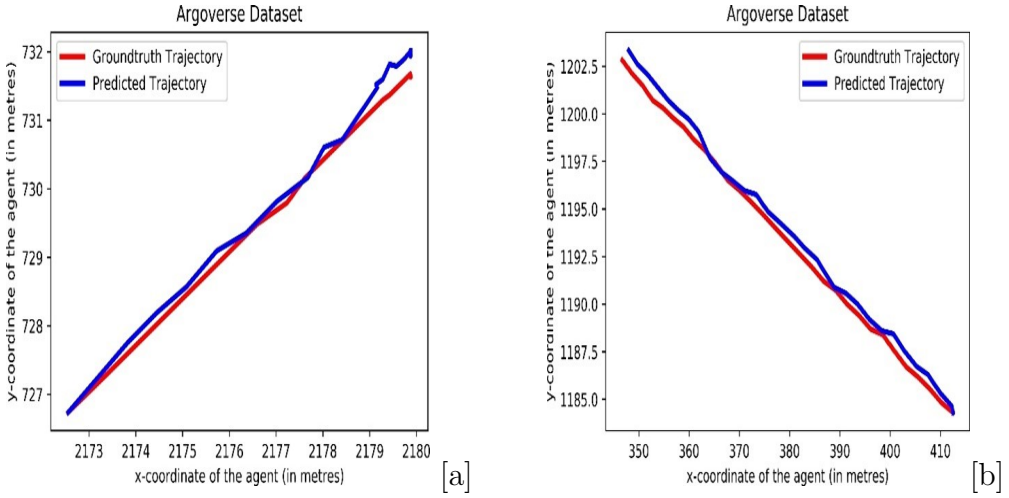



FIGURE 5.6: (a)GT trajectory and PT of the road-agent with agent_ID = 11 of the Argoverse Dataset (b) GT trajectory and PT of the road-agent with agent_ID = 27 of the Argoverse Dataset

**Argoverse Dataset** In Fig-5.6(a),the GT trajectory and the PT of the road_agent with $agent\_ID = 11$ of the Argoverse dataset are shown. In meters for this object in the considered interval, the corresponding ADE and the FDE are 0.893 and 1.434 respectively. In Fig-5.6(b), the GT trajectory and the PT of the road -agent with $agent\_ID = 27$ of the Argoverse dataset is shown. The corresponding ADE and the FDE, in meters, for this object in the considered interval are ADE=0.917and FDE=1.496.

### 5.2.3 Motion Planning Using CARLA Simulator

The proposed model aims to generate a set of waypoints (in the form of coordinates of the points), following which the ego vehicle is expected to steer through by taking the RGB images captured from the ego vehicle as inputs. A PID (proportional integral derivative) controller at the end converts the series of waypoints into a set of control commands, which provides the ego vehicle with the necessary information regarding the control of the steering, throttle and brake required to drive through the specified waypoints, and the simulator simulates this drive. It must also be noted that there are specific routes with varying scenarios in the simulator, and for each route, the destination is pre-defined by the dataset extracted from the simulator. The details of the architecture, working, implementation and training of the model are elaborated in the successive sections.

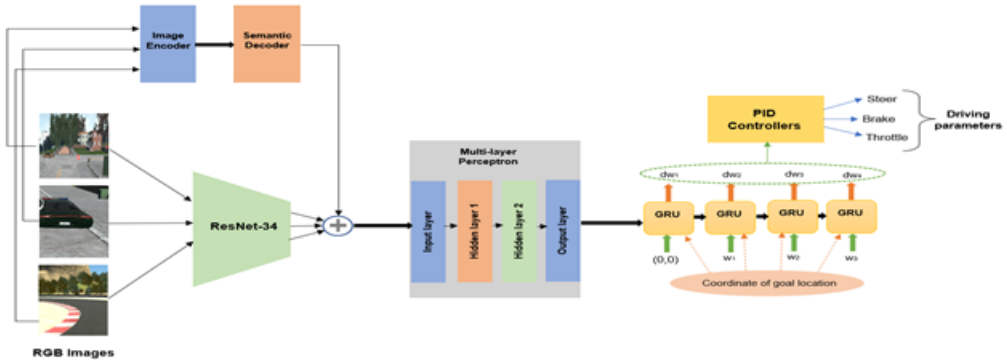



FIGURE 5.7: Architecture of the Motion Planning Model

#### 5.2.3.1 Architecture of the model and its working

A schematic representation of the basic architecture of the model is shown in Fig-5.7. The input to the model is a set of 3 RGB images - one captured by the front camera with a field-of-view of 100°, one captured by a camera aligned at 45° left and the other captured by a camera aligned at 45° right. Each of the images is cropped to a resolution of $256 \times 256$ pixels. Each image tensor thus has a dimension $(3 \times 256 \times 256)$. These images are then processed with the ResNet-34 module. The

image-based ResNet-34 network is initialized with ImageNet pre-trained weights. The output from the ResNet module is a 512-dimensional feature vector for each of the three images. A separate network consisting of an image encoder module and a semantic decoder module is used to extract semantic information from the input images by performing semantic segmentation of the images. The image encoder is used to downsample the spatial resolution of the input, developing lower-resolution feature mappings that are learned to be highly efficient at discriminating between classes. The semantic decoder upsamples the feature representations into a full-resolution segmentation map.

The ResNet-18 module with modified kernel size and stride values is used in the image encoder to downsample the input. When downsampling, the first $1 \times 1$ projection of the convolutional branch is done, i.e., kernel size is (1,1), with a stride of 2 in both dimensions. This effectively discards almost three-fourths of the input. After that, for the successive layers, the kernel size is increased to (2,2) and the stride remains unchanged, which allows considering total input, thereby improving the accuracy and information flow. After passing the input image through the layers of ResNet-18 sequentially, the output is finally passed through another convolution layer with input channels = output channels =512, kernel size = (2,2) and stride = 2 for both dimensions; and then it is finally passed through a Batch Normalization layer which completes the downsampling process and the work of the Image Encoder ends here.

The semantic decoder takes the output of the image encoder as input. The decoder consists of 5 upsampling blocks arranged sequentially. Each upsampling block consists of a sequential arrangement of an Upsample layer that performs upsampling based on the nearest neighbor algorithm, followed by a Convolution layer with kernel size = (3,3) and stride and padding of 1 in both dimensions, followed by a Batch normalization layer, followed by a layer to apply the rectified linear unit (ReLU) function element-wise and in-place, which is then followed by another similar Convolution layer as the previous one, and a Batch normalization layer. The

semantic supervision module consisting of the image encoder and semantic decoder are trained separately with the high-level semantic information available in CARLA (in the form of semantically segmented images) as ground truth and pixel-wise cross-entropy loss as the loss function. The output of the semantic supervision module is combined with that of the ResNet-34 by performing element-wise summation.

The combined 512-dimensional vector generated is now passed onto a multi-layer perceptron to reduce the dimensionality of the feature vector. The multi-layer perceptron consists of two hidden layers besides the input and output layers. The first hidden layer is made up of 256 units and the second one is made up of 128 units. On passing the 512-dimensional feature vector (which was obtained by combining the outputs of the ResNet network) through this multi-layer perceptron, the output generated is a 64-dimensional feature vector. The reduced dimensionality of the feature vector helps increase the computational efficiency in the subsequent GRU network of GRUs. The network consisting of 4 GRUs is used to predict the waypoints. The number of GRUs is 4 as the waypoints are predicted for T=4 future time-steps since the inverse dynamic model is so implemented using the PID controllers that it requires 4 waypoints as input.

The 64-dimensional vector which is the output from the multi-layer perceptron, is used for the initialization of the hidden state of the first GRU unit. The input to the first GRU is (0,0) as the considered coordinate frame is centered at the ego-vehicle. Each GRU outputs the predicted change in coordinates of waypoints $dw_t$. Thus, the actual coordinate of the waypoint $w_t$ can be calculated as $w_t = w_t - 1 + dw_t$ for t = 1 to T. Apart from the first GRU, the input to the $i^{th}$ GRU unit is $w_{i-1}$. Also, the destination/goal location coordinates are input to every GRU unit. The information passed onto the next GRU unit using a hidden state and given as output are controlled by the update gate of each GRU unit. Two PID controllers (lateral and longitudinal) are then used to derive the driving parameters for vehicle control from the sequence of waypoints predicted by the GRU network.

#### 5.2.3.2 Algorithm for PID Controller

The algorithm used for this derivation is illustrated in Algorithm-3. $\gamma$ is the desired velocity of the vehicle calculated from the waypoints by doing a weighted average (where $\lambda_t$ are the weights) of the displacement vector between the consecutive waypoints. The longitudinal PID controller tries to bring the current vehicle velocity v in sync with the desired velocity $\gamma$. The lateral PID controller oversees handling the steering angle $\alpha$, which is calculated based on the orientation of the midpoint of $w_1$ and $w_2$. Also, a brake is applied when the desired velocity is less than the brake threshold speed $B_{min}$, which is calculated based on the current speed of the vehicle v and a threshold ratio $B_{ratio}$.

**Algorithm 3:** Derivation of the driving parameters from waypoints

**Input:** sequence of 2D waypoints $\{W_t = (X_t, Y_t)\}_{t=1}^{T}$

**Output:** $Throttle\epsilon$ (0, 1), steer $\epsilon$ (1, 1), brake $\epsilon$ (0, 1)

$\gamma = 0$

**For** t=1 to T-1 **do** {

$\gamma = \gamma + \lambda_t \|w_{t+1} - w_t\|$

}

$\omega = \frac{(\omega_1 + \omega_2)}{2}$

$\alpha = \tan^{-1}\left(\frac{\omega|1|}{\omega|0|}\right)$

steer=Lateral_PID($\alpha$)

**if** $\gamma \leq vB_{ratio}$**then**{

throttle = 0

brake =1

}

else {

throttle = Longitudinal_PID $(\gamma - v)$

brake = 0

### 5.2.4 Result Analysis and Discussion of Motion Planning

#### 5.2.4.1 Experimental Setup

The model is trained to minimize the Loss. AdamWoptimizer (torch.optim. AdamW), with initial learning rate of 0.001, weight decay coefficient of 0.01 and other default parameters as in PyTorch. The layers are trained by this optimizer with respect to the above-mentioned loss function, using backpropagation. The batch size of the model is 128 and number of epochs are 50.

#### 5.2.4.2 Loss Function

Squared Error loss function or L2 loss were used between the predicted waypoints and the ground-truth waypoints to train the model. The overall loss can be computed as:

$$Loss = \sum_{t=1}^{T} \left\| w_{pred}^{t} - w_{GT}^{t} \right\|^2 \tag{5.10}$$

Where, $w_{pred}^{t}$ and $w_{GT}^{t}$ are predicted and ground truth coordinates of waypoints respectively.

Three images were separately processed using the ResNet network and later combined three and also used a multi-layer perceptron to reduce the dimension of the input vector to the GRU. Otherwise, computations to be done by the GRU network would have been increased to a larger extent. The inverse dynamic algorithm used to generate the values of the driving parameters from the sequence of waypoints predicted by the model is also formulated by the proposed model after examining the general nature of maneuver of vehicles in the real-world. The high-level semantic information available in the dataset can be explicitly used to decrease the occurrence of infractions by the model, especially to reduce the violation of traffic signals. In this semantic supervision phase, traffic light state and other such aspects are mainly considered using semantic segmentation. The following classes are used for the semantic mask: road markers, traffic lights, moving obstacles, background and

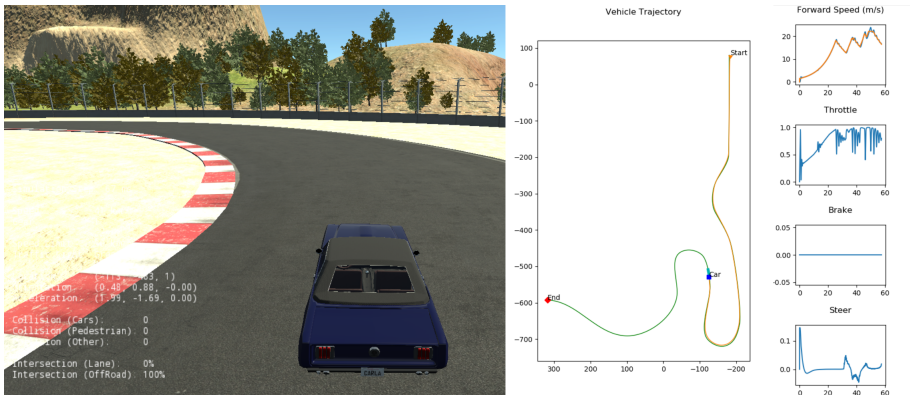
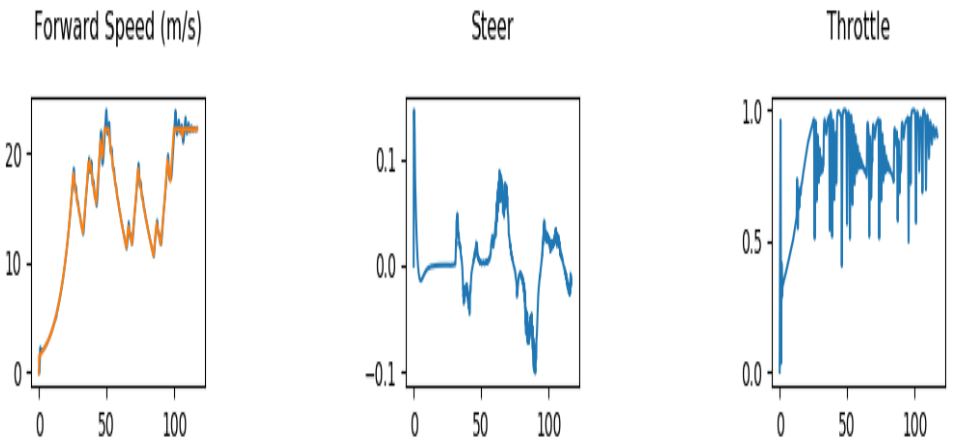



FIGURE 5.8: Ego vehicle turning at road curve

road sidewalk. The proposed model successfully drives from vision to show the capability of the method in an urban environment, including traffic light detection and intersection management by using a camera only.

In Fig-5.8, the screenshot of an instance of the CARLA simulator shows the ego vehicle turning at a road curve. The position of the vehicle and the target destination or goal is also shown in a vehicle trajectory map for the corresponding scene/route in the simulator. Along with it, the graphs showing the variations of the forward speed of the vehicle and the values of throttle, brake, and steer, with time, are also shown up to the considered instance.

In Fig-5.9, the 3 graphs show the variation (with time) of the values of the parameters, viz. forward speed of the vehicle, steer, and throttle, respectively, for the entire route of the scenario in the simulator one of whose instances in shown in Fig-5.9. Fig-5.10 presents the whole vehicle trajectory map from start to destination of the scenario in the simulator, one of whose instances is shown in Figure. In Fig-5.11, the screenshot of an instance of the CARLA simulator shows the ego vehicle overtaking another vehicle on the road. The position of the vehicle and the position of the target destination or goal is also shown in a vehicle trajectory map for the corresponding scene/route in the simulator. Along with it, the graphs showing the variations of the forward speed of the vehicle and the values of throttle, brake and steer with time are also shown up to the considered instance. In Fig- 5.12, the




[a] [b] [c]

FIGURE 5.9: Variation of forward speed, steer and throttle parameters with time for the entire route of the scenario two of whose instances are shown in (a). (b) and (c). The variations of the forward speed of the vehicle and the vehicle and the values of throttle, brake and steer, with time are also shown up to the considered instance

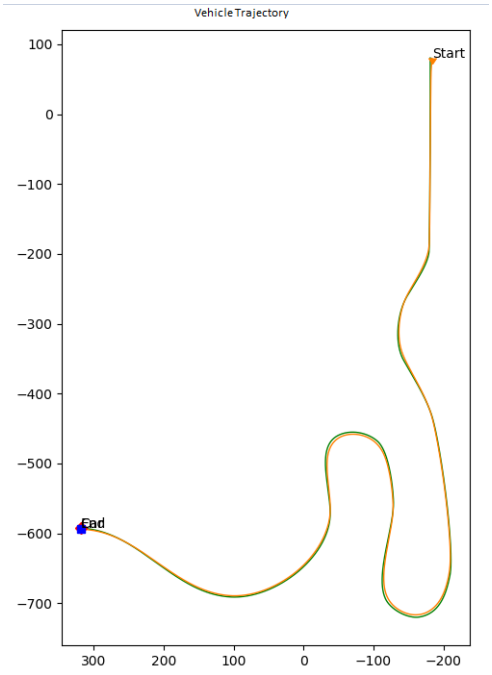



FIGURE 5.10: Vehicle Trajectory Map for the scenario one of whose instances are shown in Figure5.8.

screenshot of an instance of the CARLA simulator shows the ego vehicle stopping at a STOP road-sign. The position of the vehicle and the position of the target destination or goal is also shown in a vehicle trajectory map for the corresponding scene/route in the simulator. Along with it, the graphs showing the variations of the forward speed of the vehicle and the values of throttle, brake and steer, with time, are also shown up to the considered instance. In Fig-5.13, the 3 graphs show the variation (with time) of the values of the parameters, viz. forward speed of the vehicle, steer, and throttle, respectively for the entire route of the scenario in the simulator two of whose instances are shown in Fig-5.13 (a),(b) and (c) presents the whole vehicle trajectory map from start to destination of the scenario in the simulator two of whose instances are shown in Fig-5.11 and Fig-5.12.

TABLE 5.4: Summarizes the mean values and standard deviation of the three metrices, viz. DS, RC and IM, obtained using the model and the other image-based existing models on the CARLA simulator

| **Models ↓** | **Driving Score (DS)** | **Route Completion (RC)** | **Infraction Multiplier (IM)** |
|---|---|---|---|
| **Proposed Model (With semantic supervision)** | **41.95 ±4.17** | **79.12 ±5.14** | **0.68** |
| **Proposed Model (Without semantic supervision)** | **25.13 ±6.83** | **51.28 ±5.39** | **0.49** |
| MaRLn [252] | 24.98 ±5.24 | 46.97±7.40 | 0.52 |
| AIM [237] | 12.88 ±3.82 | 41.52±7.66 | 0.59 |
| LBC [253] | 8.94 ±2.13 | 17.54±4.70 | 0.73 |
| CILRS [254] | 5.37 ±2.16 | 14.40±2.95 | 0.55 |

Table-5.4 below summarizes the mean values and standard deviation of the three metrics, viz. DS, RC and IM are obtained using the model and the other image-based existing models on the CARLA simulator, viz. MaRLn, AIM, LBC, and CILRS. Fig-5.15 shows the average number of different types of infractions, viz. collision with vehicles, collision with pedestrians, collision with static element,

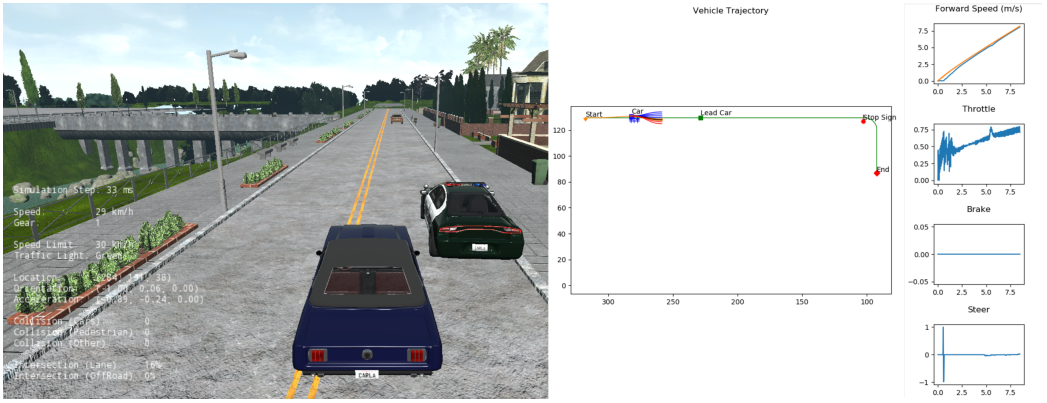



FIGURE 5.11: Ego vehicle overtaking another vehicle

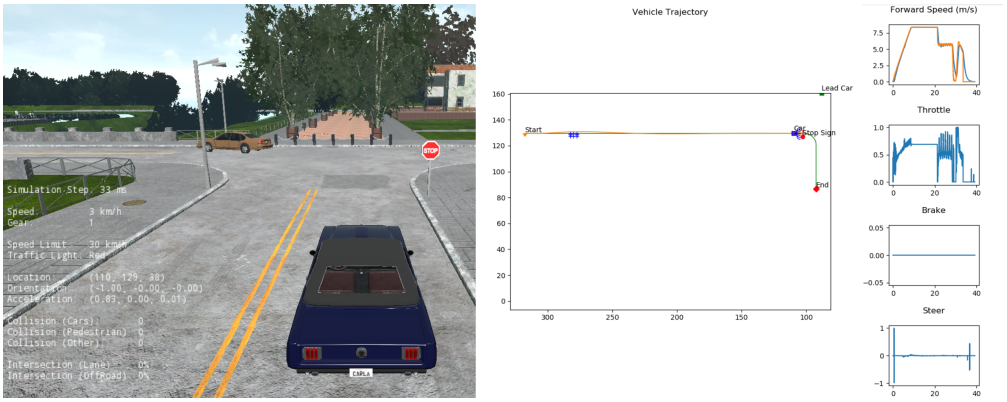



FIGURE 5.12: Ego vehicle stopping at a STOP road-sign

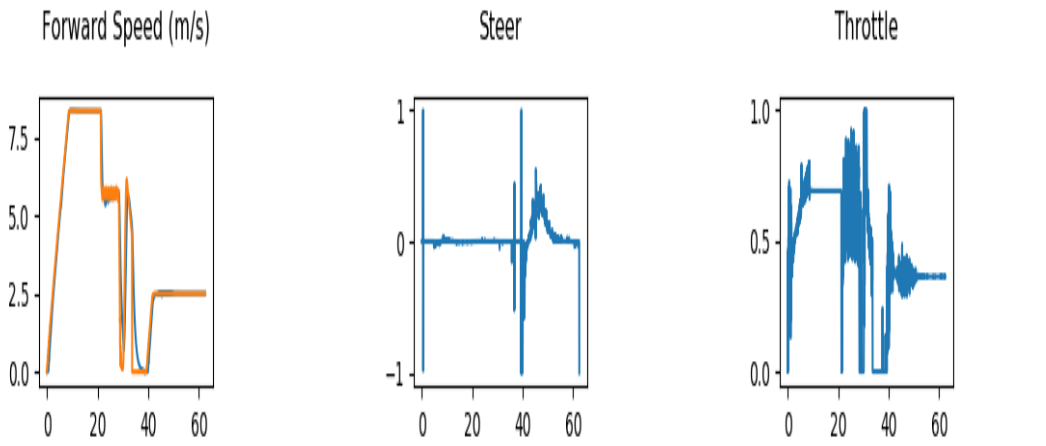



FIGURE 5.13: Variation of forward speed, steer and throttle parameters with time for the entire route of the scenario two of whose instances are shown in (a). (b) and (c). The variations of the forward speed of the vehicle and the vehicle and the values of throttle, brake and steer, with time are also shown up to the considered instance

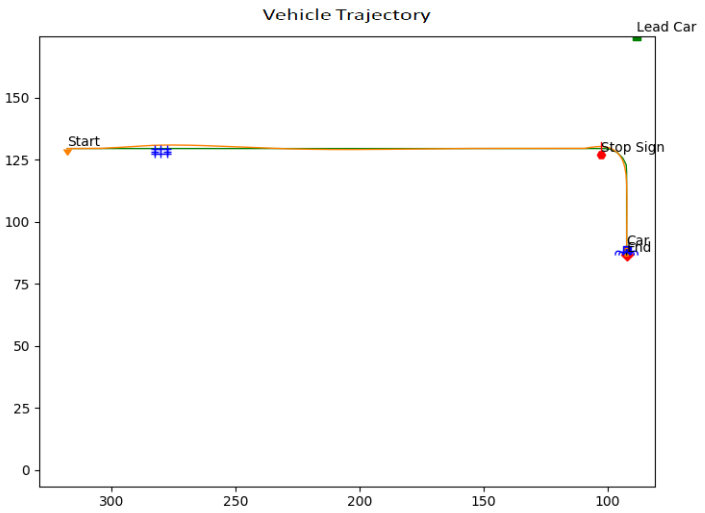



FIGURE 5.14: Vehicle Trajectory Map for the scenario one of whose instances are shown in Figure5.8.

and red-light violation, incurred by the model while testing by simulating different scenarios.

A significant improvement is seen in the Route Completion (RC) metric. This can be attributed to images from three view-angles which give the vehicle a better idea of the route ahead. Also, as the model is trained by supervised learning, the model decides accordingly by analyzing the circumstances (traffic, road-turns etc.) in three angles (from left, right and front view images) and instructs the vehicle to maneuver to reach the target destination. However, this improvement in the RC metric is not well reflected by a similar improvement in the Driving Score (DS) metric. This is because the model does not explicitly have semantic supervision to handle the infractions, and this needs to incorporate some additional semantic supervision.

#### 5.2.4.3 Ablation Study

The model with semantic supervision uses an encoder to train and predict the 2D semantics and specific affordances such as traffic lights. This is computationally

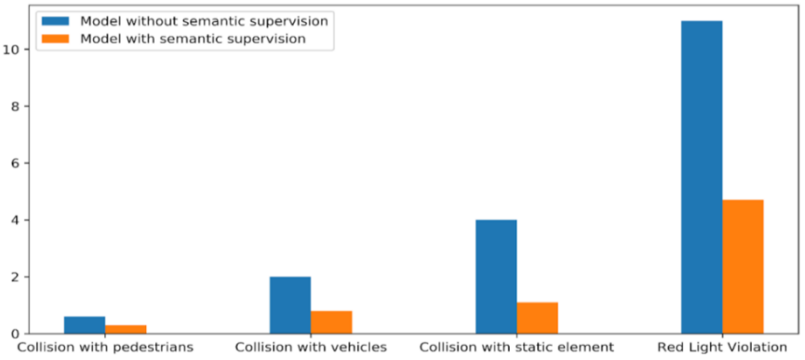



FIGURE 5.15: Comparison of the average number of different types of infractions incurred during testing for the two models

much more expensive than the proposed method but simultaneously reduces the occurrence of infractions, thereby improving the metrics Infraction Multiplier (IM) and the Driving Score (DS). The time required for planning the motion of the ego vehicle for 4 seconds ahead is approximate: 0.48 seconds on average for the model without semantic supervision and 0.79 seconds on average for the model with semantic supervision; thus, there is a trade-off in this case between computational complexity and performance and can be handled on the circumstance for which vehicle is made to travel in, e.g., for vehicles designed for purposes like racing, sporting or for adventurous travels where road-signs or traffic lights are of little importance, the model is better, but for urban scenarios or congested city roads additional semantic supervision for road-signs etc. is incorporated into the proposed model.

Fig- 5.15 compares the average number of different types of infractions, viz. collision with pedestrians, collision with vehicles, collision with static element, and red-light violation, incurred by the two models (with and without semantic supervision) while testing by running the simulation in different scenarios.

## 5.3 Conclusion

This TP model is used a deep-learning-based novel spatial-temporal Graph convolutional network for TP that represents the road agents by several nodes and an interaction network between them by the edges. The proposed model has a feature extraction from different order adjacency matrix, 1×1 convolutional layer temporal Conv-layer which will directly operate on the Graph taking it as the input for each time-step through spatial, temporal convolutional blocks, that passes the features as input for the LSTM. The proposed model surpasses the state-of-the-art on three publically available datasets for the TP, including its great potential for exploring spatial-temporal structure from the input. It also achieves low time complexity and scalability. These features are quite promising, but the parameter settings of the network structure were further optimized, especially for the dense datasets, by enhancing the ability of the proposed model to capture the agent interactions more efficiently and effectively. The Chebyshev polynomial approximation is used for the real-time system to compute the spectral graph convolution to reduce the computational complexity. The low error in sparse datasets such as Lyft and argoverse is due to less traffic density, while the apolloscape has more error with dense traffic. The FDE error is less than ADE because ADE captured the initial time-step pattern, while in the case of FDE, it was diminished at the final time-step. The model has limited flexibility and needs to perform eigenvector decomposition or handle the whole Graph simultaneously and is limited for the undirected Graph. The focus of future Work would incorporate these cues into the upcoming models like graph wavelet neural networks.

Multi-view images and a semantic supervision-based model for planning the motion of the ego vehicle, which have been trained and tested using data from the CARLA simulator, have been proposed. The different view of images helps understand the surroundings and a separate semantic supervision layer is trained with an image encoder and a semantic decoder module is extracted the semantic information that helps to reduce the violation of traffic signals. The performance of

the model analyzed with semantic and without semantic supervision achieved state-of-the-art results. The comparison of the performance of the proposed model with and without semantic supervision in four cases: the collision of pedestrians, collision with static element, red light violation and collision with vehicles.