Towards Multi-Agent Autonomous Racing with the DeepRacing framework

Trent Weiss, John Chrosniak, and Madhur Behl
Department of Computer Science
University of Virginia
Charlottesville, VA, USA
{ttw2xk, jlc9wr, madhur.behl}@virginia.edu

***Abstract*—Multi-agent autonomous racing is a challenging problem for autonomous vehicles due to the split-second, and complex decisions that vehicles must continuously make during a race. The presence of other agents on the track requires continuous monitoring of the ego vehicle's surroundings, and necessitates predicting the behavior of other vehicles so the ego can quickly react to a changing environment with informed decisions. In our previous work we have developed the DeepRacing AI framework for autonomous formula one racing. Our DeepRacing framework was the first implementation to use the highly photorealisitc Formula One game as a simulation testbed for autonomous racing. We have successfully demonstrated single agent high speed autonomous racing using Bezier curve trajectories. In this paper, we extend the capabilities of the DeepRacing framework towards multi-agent autonomous racing. To do so, we first develop and learn a virtual camera model from game data that the user can configure to emulate the presence of a camera sensor on the vehicle. Next we propose and train a deep recurrent neural network that can predict the future poses of opponent agents in the field of view of the virtual camera using vehicles position, velocity, and heading data with respect to the ego vehicle racecar. We demonstrate early promising results for both these contributions in the game. These added features will extend the DeepRacing framework to become more suitable for multi-agent autonomous racing algorithm development.**

I. Introduction

Multi-agent autonomous racing still remains a largely unsolved research challenge. The high-speed and close proximity situations that arise in multi-agent autonomous racing present an ideal condition to design algorithms which trade off aggressive overtaking maneuvers and minimize the risk of collision with the opponent.

Most past research in autonomous racing has focused on a single-agent time-trial style of racing, i.e, a single autonomous racecar completes a lap in the shortest amount of time. Time-trial poses a number of challenges in terms of dynamic modeling, on-board perception, localization and mapping, trajectory generation and optimal control. Much less attention has been devoted to the multi-agent style of racing that we address in this paper. In addition to the aforementioned challenges, multi-agent autonomous racing also requires inferring the states of other agents, and opportunistic passing while avoiding collisions. Multi-agent autonomous racing provides the opportunity for testing ground for developing and testing more widely applicable non-cooperative multi-robot planning strategies.

In our previous work [6, 5] we have developed the DeepRacing AI framework for autonomous formula one racing. Our DeepRacing framework was the first implementation to use the highly photorealisitc Formula One game as a simulation testbed for autonomous racing. We have successfully demonstrated single agent high speed autonomous racing using Bezier curve trajectories. In this paper, we extend the capabilities of the DeepRacing framework towards multi-agent autonomous racing.

Our methods involve using the DeepRacing autonomous formula one framework to collect data on how racing vehicles interact over time. We then use a novel approach to extract the relevant information from the data stream by filtering out the historical data of vehicles that are not in view of the ego vehicle. This is accomplished by creating a "virtual camera" that represents the F1 game's output images as a pinhole camera model that maps the 3D positions of the other agents to a pixel location in the 2D images. Using these data, we then generate a deep learning model that predicts the future waypoints of other vehicles given a history of their positions.

II. Related Work

Several researchers have suggested methods for accurate and reliable trajectory prediction of commercial vehicles. [4] proposed a recurrent neural network encoder-decoder framework that models how vehicles interact in complex and dynamic scenes. [1] introduced a convolutional model that uses raw sensor data to predict the trajectories and high-level action of vehicles. [7] presented an LSTM-driven network for predicting vehicle trajectories in urban environments given sensor data and a detailed map with traffic regulations. [2] and [8] discussed predicting future vehicle trajectories based on instantaneous motion and an understanding of motion patterns of freeway traffic. Our work builds upon these methods by performing trajectory prediction in the domain of multi-agent racing. We present a novel way of collecting vehicle trajectory data from a photo-realistic racing simulator, as well as a deep recurrent neural network for multi-agent racing trajectory prediction.

III. Problem Formulation

Our work aims to address two issues related to predicting the future trajectories of other agents. The first issue involves efficiently collecting data from a reliable source, which has been partially solved by our DeepRacing framework. However, the video game that forms the basis of our framework does not have a sensor model, but has only a fixed view camera with unknown parameters. We thus demonstrate a method on how to construct a sensor model using only the provided fixed-view images to better simulate the vehicle driving autonomously, accounting for vehicles at varying distances and vehicles occluded from view.

The second issue relates to the state estimation of opponents during an autonomous race. Our previous work [6] has solely focused on single-agent racing, in which the ego vehicle is alone on the track. The high-speed and close interactions involved with multi-agent racing necessitate the future prediction of how other vehicles will interact with the ego. We present a method for predicting the future trajectories of other racing agents given historical data of their previous behavior. To simulate the true conditions of a multi-agent race, we must ensure that our vehicle is only able to predict the trajectories of opponents that are within view of the ego. Therefore, the problem of state estimation is inherently related to the problem of constructing a virtual sensor model.

First we present the method used to construct the virtual sensor model, followed by the methods and results for the problem of state estimation.

A. Formula One Simulation Environment

In order to train our algorithm to predict the trajectories of other vehicles, we need a reliable way to collect data that contains information on how racing vehicles interact over time. As described in our previous work, we utilized our DeepRacing framework that has converted the official Formula One video game released by Codemasters© into a simulation environment for the collection of the necessary data. The game is extremely photo-realistic, as shown in Figure 2, and is based on high-fidelity simulated physics.

The game advertises a "fire-and-forget" stream of telemetry data containing a variety of information about the game's state over a User Datagram Protocol (UDP) network socket. Each packet in the stream offers a snapshot of the game's state along with a timestamp for when the state was generated. The variables of interest to our problem include (1) the positions of all vehicles, (2) the velocity vectors of all vehicles, (3) the forward heading vectors of all vehicles, and (4) the rightward heading vectors of all vehicles, where all variables have three components (X, Y, and Z) expressed in the world coordinate system of the track. Although other information such as the steering angle, throttle pressure, and brake pressure may also provide value to predicting trajectories, we limited data usage to variables that an object detection algorithm run through the vehicle's camera could reasonably predict.

In a separate process, we can also sample images from the ego vehicle's point of view over time, similar to what is seen in Figure 2. These images simulate what an autonomous F1 vehicle will see during a race and are useful for constructing a virtual "camera," which is described in more detail in the following section.

IV. Virtual Camera Sensor Implementation

A. Calibrating a Virtual Camera

We now present our method for estimating a projective transformation from 3D points in the ego vehicle's coordinate system to a pixel position in the image plane of our infrastructure's virtual camera. In this work, the "ego coordinate system" refers to the conventional "base link" system. This coordinate system is centered at the ego vehicle's rear axle, with it's X direction vector pointing forward, Y pointing to the left, and Z vector as the cross product of X and Y.

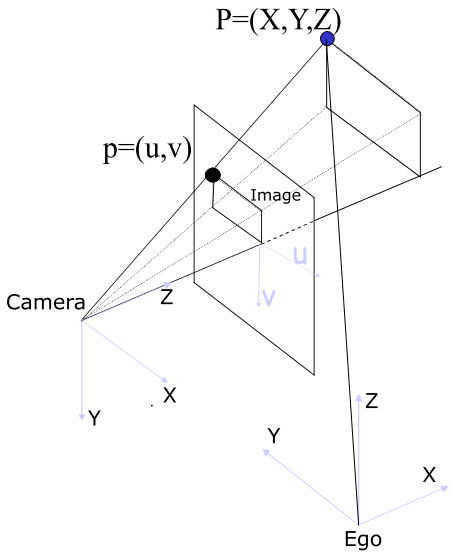

Fig. 1: The geometry of our camera model.

In essence, this problem boils down to estimating a projective transformation that maps a point, $P \in \mathbb{R}^3$, in the ego vehicles coordinate system to a pixel position, $[u, v] \in \mathbb{R}^2$. This projection, expressed graphically in figure 1, has two can be decomposed into two components:

1) A euclidean transformation, $\mathbb{R}^3 \rightarrow \mathbb{R}^3$, that maps points in the ego coordinate system to points in a *camera* coordinate system, with an origin at the camera's optical center, Z axis pointing normal the image plane, and X/Y directions parallel to the rows and columns of the image plane.
2) A projective transformation, $\mathbb{R}^3 \rightarrow \mathbb{R}^2$, that maps points in the camera coordinate system into $[u, v]$ pixels on the image plane.

Component 1 in the above list is expressed mathematically as:

$$\begin{bmatrix} X_C \\ Y_C \\ Z_C \end{bmatrix} = \begin{bmatrix} \mathbf{R} & \mathbf{T} \end{bmatrix} \begin{bmatrix} X_E \\ Y_E \\ Z_E \\ 1 \end{bmatrix} \tag{1}$$

Where $\begin{bmatrix} X_E \\ Y_E \\ Z_E \end{bmatrix}$ is a point in the ego-centric coordinate system and $\begin{bmatrix} X_C \\ Y_C \\ Z_C \end{bmatrix}$ is that same point transformed into the camera-centric coordinate system. Note that the conventional bottom row of $\begin{bmatrix} 0 & 0 & 0 & 1 \end{bmatrix}$ is left out of this euclidean transform. This is because we are not composing this transformation with any others, so the 3×4 matrix without the bottom row is sufficient for this model. $\mathbf{R} \in SO(3)$ and $\mathbf{T} \in \mathbb{R}^3$ are the rotation matrix and translation of the euclidean transformation, respectively.

Component 2, the projective transformation onto the image plane, is expressed as:

$$Z_C \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & 0 \\ 0 & f_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X_C \\ Y_C \\ Z_C \end{bmatrix} \tag{2}$$

Where u, v are the pixel coordinates of the 3D points corresponding projection onto the image and f_x, f_y are the focal lengths in the row and columns directions of the image, respectively. It easy to verify that this formulation corresponds to a typical projective transformation derived from similar triangles:

$$u = \frac{f_x X_C}{Z_C}, \; v = \frac{f_y Y_C}{Z_C} \tag{3}$$

Substituting in equation 1, we have our full camera model:

$$Z_C \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & 0 \\ 0 & f_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \mathbf{R} & \mathbf{T} \end{bmatrix} \begin{bmatrix} X_E \\ Y_E \\ Z_E \\ 1 \end{bmatrix} \tag{4}$$

Note that this model assumes no radial distortion and assumes the optical center of the virtual camera lies exactly at the center of the image. Fitting this model to experimental data is now very easy with our DeepRacing framework. This procedure has the following high-level steps.

1) Record a set of images from the F1 game
2) Label each image with the corresponding 3D positions of all other agents visible in that image, expressed in the ego coordinate system
3) Label each image with the corresponding 2D pixel positions of each visible agent
4) Fit a camera model to these corresponding sets of pixel positions and 3D points

Steps 1 and 2 can be done automatically with our DeepRacing API. Step 3 must be done by manually by labelling a bounding box around each visible agent. Figure 2 shows an example of such a labelled image. Step 4 is carried out

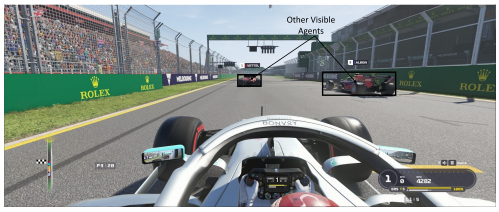

Fig. 2: An example calibration image. The goal is to fit a camera model that can project 3D points in the ego coordinate system to their corresponding pixel location in the F1 game's output image.

with PyTorch's torch.optim library. Note that because we can control both the aspect ratio and the size of the images the game renders, it is readily observed that $f_x = \frac{W}{H} f_y$, where W, H are the width and height of the image (in pixels), respectively. Therefore, we define our camera model's parameters to be f_y (with f_x computed from the image size), the translation between the ego coordinate system and the camera coordinate system, and a unit quaternion representing the rotation between the ego coordinate system and the camera coordinate system. We then "train" this camera model to minimize the reprojection loss from the 3D point in the ego coordinate system onto the image plane, defined as follows:

$$p_{model} = \begin{bmatrix} \frac{f_x X_C}{Z_C} \\ \frac{f_y Y_C}{Z_C} \end{bmatrix} \tag{5}$$

$$l_{reprojection} = || \begin{bmatrix} u_{label} \\ v_{label} \end{bmatrix} - p_{model} || \tag{6}$$

Where $\begin{bmatrix} u_{label} \\ v_{label} \end{bmatrix}$ is the ground-truth labelled pixel location of the other agent visible in the image, and p_{model} is computed with the camera model parameters according to equations 1 and 5. Figure 3 shows the results of this model fit.

Fig. 3: After model fitting, we can map 3D points in the ego vehicle's coordinate system to 2D pixel locations in the game images

V. State Estimation for Multi-Agent Racing

The problem of trajectory prediction can be reduced to a simple mapping problem, where given the context C of a vehicle's previous actions and the future waypoints W of the vehicle, we wish to find a mapping $C \rightarrow W$. Throughout the next sections we will present how we formulate both C and

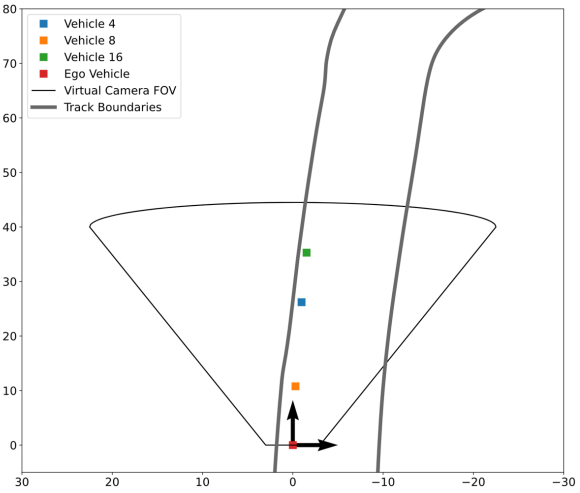

Fig. 4: Results from virtual camera without accounting for occluded vehicles (top) with in-game view for comparison (bottom).

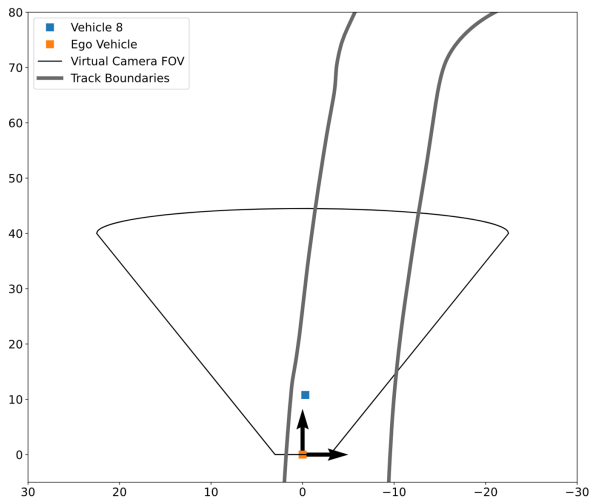

Fig. 5: Results from virtual camera accounting for occluded vehicles.

W, as well as the process of generating the mapping between the two.

A. Data Processing

The data captured from our DeepRacing framework needs slight modifications before being fed to the state estimation neural network. First, we must structure the data such that we have a context matrix C and a waypoint matrix W as previously discussed. The context matrix was constructed by extracting the relevant information of a singular vehicle at a timestamp t, as well as the same data at p previous timestamps. Furthermore, there are twelve variables extracted at each timestamp: being the X, Y, and Z components of the variables discussed previously. Therefore, a single context matrix C will have the dimensions $p \times 12$. A similar procedure is conducted to generate the waypoint matrix W with f future waypoints. The waypoint matrix, however, only includes the future positions of the vehicle, and as such a singular matrix will have the dimensions $f \times 3$. This process was repeated for each of the nineteen vehicles present in a singular packet across all packets collected. Next, we must translate the coordinate system of the collected data from the global coordinate system to the ego coordinate system, as all data captured during a race will come from the perspective of the ego vehicle. The transformation process is conducted using an Affine transformation that maps all values in C and W to values in the ego coordinate system at the timestamp t associated with the packet. The math behind this transformation is beyond the scope of this paper, but [3] provides a comprehensive demonstration of the intuition and procedures behind the transformation. At this point, we have structured all of the data appropriately and translated the data to the coordinate system of the ego vehicle. However, we want to ensure that our neural network only runs predictions using the data associated with vehicle's within the ego's field of view. To filter out the data of vehicles not present in the ego's field of view, we constructed a "virtual camera" represented as a conical shape to detect which vehicles should be visible by the ego at each timestamp t. The parameters of the shape, e.g. base, height, radius, were tuned empirically using the images collected from the DeepRacing API, shown in Figure 4.

As seen in Figure 4, a major issue with the current fake camera is that it captures vehicles that are obstructed from the view of the ego. To filter out obstructed vehicles, we created an algorithm that considers other agents as polar coordinates. The algorithm flags vehicles as obstructed if they lie within a threshold θ of another vehicle and are radially further from the base link of the ego. The results of the filtering algorithm applied to the same packet as Figure 4 can be seen in Figure 5.

Figure 5 demonstrates that the algorithm was successfully able to filter out the obstructed vehicles, and thus the data is now ready for training a deep neural network.

B. State Estimation RNN

Using the data collected from our DeepRacing API and processed using the aforementioned steps, we were able to train a deep recurrent neural network (RNN) that generates a mapping from the scene context C to future waypoints W. The model consists of three stacked Long Short-Term Memory (LSTM) cells, with each cell having thirty-two units in the hidden state. Other model structures were experimented with, e.g. varying the number of LSTM layers and the size of the hidden state, and the three-layer architecture with

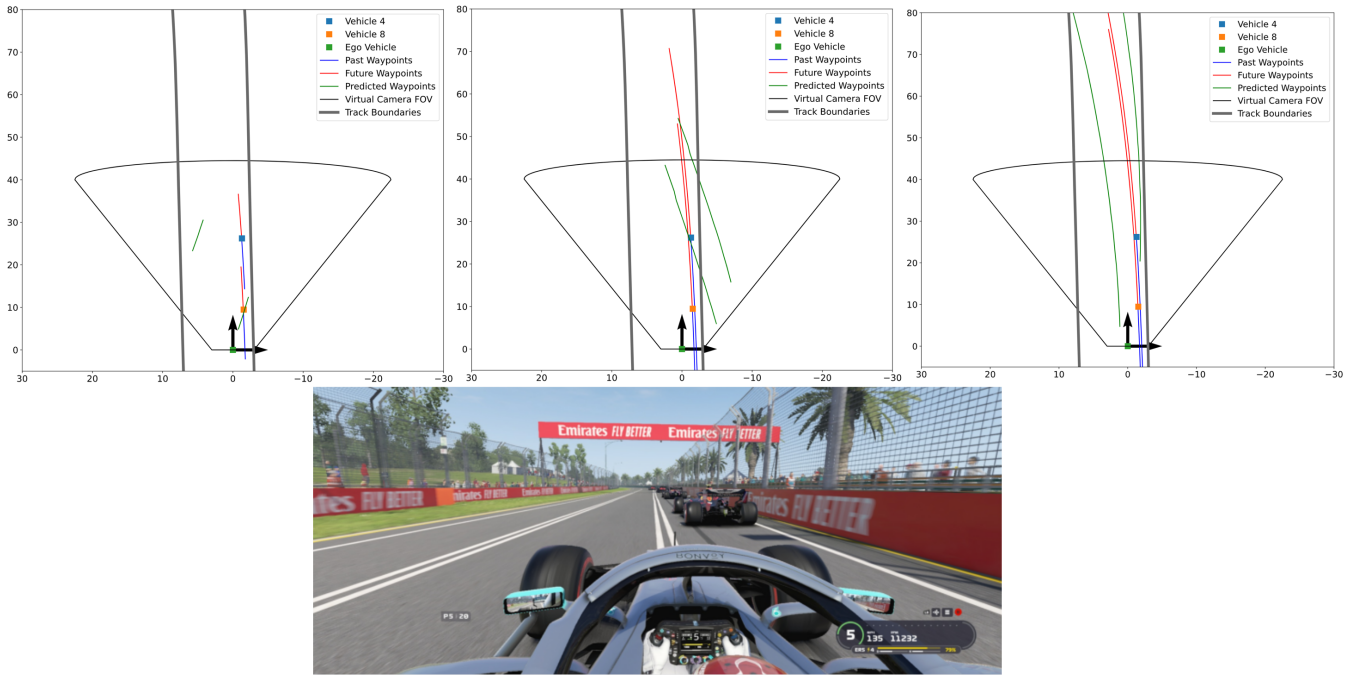

Fig. 6: Trajectory Predictions for 167 ms (left), 667 ms (center), and 1 s (right) for in-game scene (bottom).

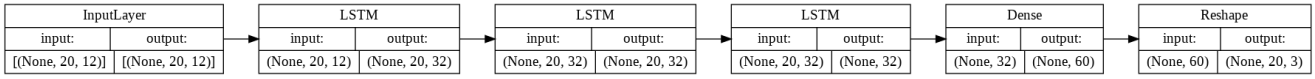

Fig. 7: Architecture of our trajectory prediction network. The network consists of three stacked LSTM layers, with each layer having a thirty-two dimensional hidden-state. Dropout layers (not pictured) with probabilities of 0.0002 were added inbetween the stacked LSTM layers. The stacked LSTM layers feed into a fully-connected layer to format the predictions.

thiry-two hidden state units yielded the best results in terms of training and validation set MSE. Dropout layers with a dropout probability of 0.0002 were also added in between the stacked LSTM cells. The final LSTM cell then feeds into a fully-connected layer with a linear activation function to generate the predicted future waypoints. A visualization of the model structure is displayed in Figure 7.

The model was trained using the Adam optimizer with a learning of 0.003, and mean squared error was used as a loss function.

VI. Results

For our experiments, we tested the performance of our model for various sizes of the context matrix C and the waypoint matrix W. Specifically, we used five, twenty, and thirty previous timesteps for context to predict the same number of timesteps into the future. The packet sampling rate runs at approximately 30 Hz, so these trajectories have a prediction horizon of 167 ms, 667 ms, and 1 s, respectively. The models were evaluated based on their training and validation mean squared error (MSE) between the predicted future waypoints from our model and the ground-truth future waypoints. Each model was trained using 100 epochs and a batch size of 32. The results for each model are presented below in Table I.

Number of Packets (ms)	**Training MSE** (m^2)	**Validation MSE** (m^2)
5 (167)	9.4117	15.977
20 (667)	26.502	21.663
30 (1000)	30.657	37.635

TABLE I: Model Error Evaluation

As expected, the overall error of the model increases as it attempts to predict farther into the future. Figure 6 visualizes the performance of the model when predicting 167 ms, 667 ms, and 1 s into future.

The predicted vehicle trajectories in Figure 6 appear to improve as the neural network is given more context. The predictions generated when given 167 ms of context originate behind and to the side of the ground-truth trajectories, and the predictions made with 667 ms of context originate outside of the track boundaries entirely. Our RNN model clearly formulates better predicted trajectories when given 1 s of context. Although the predicted trajectories do not overlap closely to the ground-truth waypoints, the model is capable of predicting the general direction and shape of the trajectories of other agents. This indicates that our RNN model is correctly learning the other agents internal state information, but requires sufficient measurements to correctly predict future behavior from that state.

VII. Conclusion and Future Work

In conclusion, we present a state estimation framework for autonomous motorsport racing. This RNN-based model is can estimate the future trajectories of other racing agent's in the ego agent's field-of-view. As the need for agile autonomy grows, it will be critical that autonomous agents understand not just the current states of other agents, but also their future intent. These trajectory predictions for the other agents in the ego's field-of-view represent a crucial piece of an agile autonomous system.

Additionally, because the F1 game offers no camera model out-of-the-box, we present an update to our DeepRacing API that allows fitting a pinhole camera model for the F1 game's rendered images as a virtual camera. This will enable robotics developers to pursue exciting lines of research in vision based methods for autonomous racing as well as enable automatic generation of training data for machine-learning models for detection of F1 vehicles in an autonomous motorsport context.

Future work would include extending this model to replace the waypoint predictions with predicting the control points for a Bèzier curve as a canonical representation of another agent's trajectory. Our previous work [6] has indicated that a deep learning model is significantly better at predicting these control points over waypoints due to the dimensionality reduction of using a parameterized curve to represent a trajectory rather than a list of waypoints. Restricting the prediction of the network to be all but the first control point of the Bèzier curve will guarantee that the predicted trajectory for any particular agent start at the known initial position that agent. Furthermore, we plan to explore more complex RNN architectures to create a model that is better able to understand the context of the provided scene.

References

[1] Sergio Casas, Wenjie Luo, and Raquel Urtasun. *IntentNet: Learning to Predict Intention from Raw Sensor Data*. 2021. eprint: arXiv:2101.07907.

[2] Nachiket Deo, Akshay Rangesh, and Mohan M. Trivedi. "How would surround vehicles move? A Unified Framework for Maneuver Classification and Motion Prediction". In: (2018). DOI: 10.1109/TIV.2018.2804159. eprint: arXiv:1801.06523.

[3] Don Fussell. *Affine Transformations*. URL: https://www.cs.utexas.edu/users/fussell/courses/cs384g-fall2011/lectures/lecture07-Affine.pdf.

[4] Namhoon Lee et al. *DESIRE: Distant Future Prediction in Dynamic Scenes with Interacting Agents*. 2017. eprint: arXiv:1704.04394.

[5] Varundev Suresh Babu Trent Weiss and Madhur Behl. "Bezier Curve Based End-to-End Trajectory Synthesis for Agile Autonomous Driving". In: *NeurIPS 2020 Machine Learning for Autonomous Driving Workshop*. 2020.

[6] Varundev Suresh Babu Trent Weiss and Madhur Behl. "DeepRacing AI: Agile Trajectory Synthesis for Autonomous Racing". In: *International Conference on Intelligent Robotis and Systems (IROS): Workshop on Perception, Learning, and Control for Autonomous Agile Vehicles.* IEEE Robotics and Automation Society. 2020.

[7] Shaobo Wang et al. "Vehicle Trajectory Prediction by Knowledge-Driven LSTM Network in Urban Environments". In: *Hindawi* (Nov. 2020). URL: https://downloads.hindawi.com/journals/jat/2020/8894060.pdf.

[8] Wei Xiao, Lijun Zhang, and Dejian Meng. "Vehicle Trajectory Prediction Based on Motion Model and Maneuver Model Fusion with Interactive Multiple Models". In: *SAE Technical Paper Series* (2020). DOI: 10.4271/2020-01-0112.