DeepRacing AI: Agile Trajectory Synthesis for Autonomous Racing

Trent Weiss, Varundev SureshBabu, and Madhur Behl
Department of Computer Science
University of Virginia
Charlottesville, VA, USA
{ttw2xk,vss8sm,madhur.behl}@virginia.edu

***Abstract*—Demonstrating high-speed autonomous racing can be considered as a grand challenge for vision based end-to-end deep learning models. DeepRacing AI is a novel end-to-end framework for trajectory synthesis for autonomous racing. We train and demonstrate the effectiveness of our approach using a high fidelity and photo-realistic Formula One gaming environment - used by real racing drivers. This is the first work that has used the highly realistic F1 game as a simulation environment for deep learning models. We present a novel method for single agent autonomous racing by training a deep neural network to predict a parameterized representation of a trajectory. Our Bezier curve based trajectory synthesis approach outperforms several other end-to-end DNN approaches for autonomous racing. In addition to evaluating our methodology in a closed-loop manner in the game; we also implement the DeepRacing algorithm on a 1/10 scale autonomous racing test-bed and show its ability to handle real-world data at high speeds.**

I. Introduction

What would an autonomous car do if a vehicle swerves across lanes without any indication? How about a high-speed unexpected lane merge ? or when the car in front of you brakes aggressively without warning? The biggest challenge with current deep learning based approaches for autonomous driving is, how to ensure that the car drives safely, and reliably in situations that don't happen very often in day to day driving, and are therefore difficult to gather data on.

In our research we are teaching autonomous cars to learn how to drive at the limits of their agility, and maneuverability. The way we do this is by autonomously racing these cars against each other, both in highly photo-realistic simulation and on real 1/10 scale physical test-beds. The high-speed, close proximity nature of racing makes it an ideal candidate to learn agile autonomous behavior which can eventually inform motion planning for commercial autonomous self-driving.

Demonstrating high-speed autonomous racing can be considered as a grand challenge for vision based end-to-end models, and making progress here has the potential to enable breakthroughs in agile and safe autonomy. To succeed at racing, an autonomous vehicle is required to perform both precise steering and throttle maneuvers in a physically-complex, uncertain environment, and by executing a series of high-frequency decisions. End-to-end models for autonomous driving have attracted much research interest because they eliminate the process of feature engineering.

Autonomous racing is also slowly becoming a motorsport featuring head-to-head battle of algorithms. Roborace [1] is the Formula E's sister series, which will feature fully autonomous race cars in the near future. Autonomous racing competitions, such as F1/10 racing and Autonomous Formula SAE are, both figuratively and literally, getting a lot of traction and becoming proving grounds for testing perception, planing, and control algorithms at high speeds.

We present DeepRacing AI, a novel end-to-end framework for training and evaluating algorithms specifically for autonomous racing. DeepRacing uses the Formula One (F1) Codemasters game as a virtual testbed [2]. Previously [2] we have demonstrated end-to-end autonomous racing in the F1 game. In this paper we extend our approach for predicting parameterized Bézier Curves trajectory for the autonomous agent instead of mapping pixels directly to control. This results in a more stable and faster autonomous racing behavior.

II. Problem Statement

The problem of autonomous driving distills to the task of scene understanding through sensor measurements, e.g. cameras, LIDAR point clouds, ultrasonic sensors, etc., and producing control inputs for the car, typically steering angle and throttle. Expressed mathematically, if the domain of the vehicle's entire sensor suite is $\mathbb{X}$ and the space of the vehicle's control outputs is $\mathbb{U}$, then the general problem of autonomous driving is a mapping from $\mathbb{X} \rightarrow \mathbb{U}$. Autonomous racing requires a great many control inputs: steering, acceleration, clutch, fuel mix selector, clutch bite point, and regenerative braking; just to name a few. For simplicity, we assume that the control domain of the racecar is steering and acceleration (a 2-dimensional euclidean space, $\mathbb{R}^2$). We focus on a vision based approach and assume that the car's input sensor domain is fixed-width images: $\mathbb{R}^{3xHxW}$, i.e. images with 3 channels (3-channel color images are assumed for this work) of height H and width W.

We consider three high-level approaches to this problem. Figure 1 provides a graphical description of these three approaches.

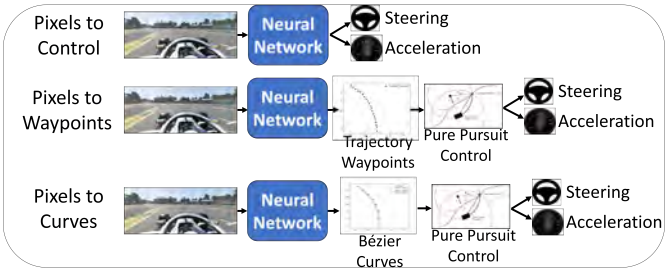

Fig. 1. Compared to pixels to control, a pixels to waypoint approach is developed; where low level steering is handled using a pure-pursuit controller.

A. Approach 1: Pixels to Control

There is a great body of work in this domain centered around the special case where $\mathbb{U}$ consists of only steering angles. NVIDIA's PilotNet architecture [3] is considered a seminal work on this approach. It focuses on mapping what the car's sensor suite is seeing at the present time to a single control command, at the present time. This is done in an end-to-end manner i.e. the DNN is trained to directly map pixels to control outputs: $\mathbb{R}^{3xHxW} \rightarrow \mathbb{R}^2$. However, this model of autonomous driving does a poor job of capturing how expert drivers behave. An expert driver considers a history of previous observations to build up some temporal *context* about the scene. Using only single static images can create an ill-posed problem in which the map from pixels to control is not a function. Consider the images and corresponding ground-truth trajectories in Figure 2. Even though the images are similar, their ground-truth trajectories differ noticeably. Within only 1.4 seconds, the paths diverge by almost 9 meters. In a high-speed scenario like racing, this could easily be the difference between a successful race and a devastating crash.

An alternative approach deepracing-date is to consider a sensor reading as a sequence of images: $\mathbf{I}_{i-N}, \mathbf{I}_{i-N+1}, \mathbf{I}_{i-N+2}, ..., \mathbf{I}_i$, where the subscript i represents the current time and c represents some number of time-steps into the past, as a single measurement that is then used to predict a sequence of control outputs $\mathbf{r}_{i+1}, \mathbf{r}_{i+2}, \mathbf{r}_{i+3}, ...\mathbf{r}_{i+P} \in \mathbb{R}^3$, where $\mathbf{r}_k$ represents a control output at time k. This view of the problem is a mapping from a *context window* of sensor readings to an *intent window* of control outputs for the autonomous vehicle: $\mathbb{R}^{NxCxHxW} \rightarrow \mathbb{R}^{Px3}$ However, this fully end-to-end approach has it's weaknesses. For very high-speed systems such as a race car, small errors in a control input map to very large errors in the actual path the car follows and can create catastrophic results.

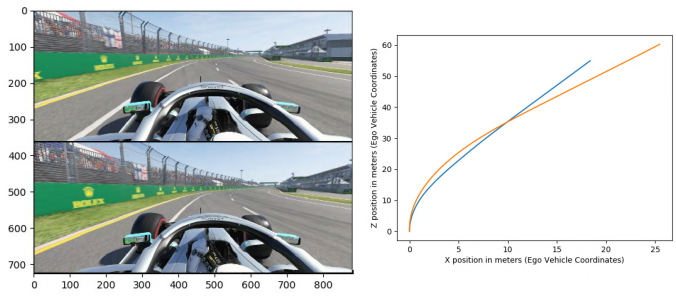

Fig. 2. Purely Markovian methods like CNNs can present an ill-posed problem. For two very similar static images, the trajectories differ significantly.

B. Approach 2: Pixels to Waypoints

To remedy this problem, one could view the problem of autonomous driving not as a static function from pixels to control, but as a temporally varying task that maps sensor inputs to curves in the ambient task space of the ego vehicle. In this approach, the autonomous car needs to predict a sequence of *waypoints* in the ego vehicle's task space. I.e. the problem becomes mapping as sequence of images: $\mathbf{I}_{i-N}, \mathbf{I}_{i-N+1}, \mathbf{I}_{i-N+2}, ..., \mathbf{I}_i$ to a sequence of waypoints: $\vec{r}_{i+1}, \vec{r}_{i+2}, \vec{r}_{i+3}, ...\vec{r}_{i+P} \in \mathbb{R}^D$ where D is the dimensionality of the task space. Expressed mathematically, a function of the form:$\mathbb{R}^{Nx3xHxW} \rightarrow \mathbb{R}^{P\ x\ D}$ This approach falls along a similar lines as end-to-end control by mapping a context window to a long-term intent, but the intent window is a set of points for the car to follow rather than a control schedule. The task of mapping a predicted trajectory for the car to follow down to a specific steering and throttle value for the car's low-level control is left to classical control techniques based on a bicycle model of the car's kinematics. For this work, a Pure Pursuit controller is used.

This approach also has it's weaknesses. It suffers from the so-called "curse of dimensionality" and can produce very non-smooth paths. With so many parameters required to represent a single output ($P\ x\ D$ of them), models that are already prone to over-fitting like deep neural networks can predict very noisy output trajectories that can vary significantly with only minor (possibly imperceptible) changes in the input images. If a model predicts even a single waypoint incorrectly, this can cause a serious error in the car's chosen control action.

C. Approach 3: Pixels to Bezier Curves

Finally, we consider a novel approach to autonomous racing. Rather than training a neural network fully end-to-end, we view the problem of trajectory prediction not as a mapping from images to waypoints, but from images to a parameterized description of a smooth 1-manifold embedded in the car's task space. B-Splines are a very intuitive choice, as they are C^∞ curves and are commonly used for motion planning tasks [4]. However, their recursive representation does not fit well with gradient back-propagation, making it difficult to apply them in the context of trajectory prediction. To remedy this, we use *Bézier Curves* as a canonical form of curves in an autonomous racecar's task space. Bézier Curves are a linear combination of Bernstein Polynomials and are described in more detail in section V-A. Our model uses the images in the context window to predict the control points of a Bézier curve to generate the trajectory for the autonomous racecar.

III. Related Work

We divide the related work into simulation testbeds and autonomous driving methods and provide a brief reprise on both. Simulation capabilities have been primarily used in the planning and control phase of AD [5]–[7]. More recently, simulation has been used in the entire AD pipeline, from perception and planning to control [8]. Researchers have tried to use images from video games to train deep-learning-based perception systems [9], [10]. End-to-end driving was showcased in the car racing game TORCS [11] using Reinforcement Learning but its physics and graphics lack realism.

In one of the earliest work on end-to-end autonomous driving NVIDIA [3] presented the PilotNet CNN architecture. PilotNet is a feed-forward style network that directly regresses to a single steering value for each input image obtained from a front facing dashboard camera. [12] uses an event camera to batch images from an arbitrary number of time-steps as an input to a CNN and achieve a performance increase. [13] present a different approach that uses Long Short-term Memory Cells [14] as a means of capturing a history of the steering trajectory and encode temporal structure of the problem. [15] use a novel combination of CNN and a traditional auto-encoder approach. This network uses a CNN for feature extraction and applies an encoding function to translate the regression problem into a more manageable classification problem. [16] also present a novel approach that blends expert domain knowledge of highway driving by defining a notion of image affordance that is then mapped to a steering command.

[17] use an LSTM cell to predict a series of waypoints for images of highway driving at varying traffic densities. [18] present an approach for generating waypoints with a CNN and then using a Model Predictive Controller (MPC) to control the vehicle. [19] use a similar approach to predict waypoints based on expert demonstration. [20] is a blend of several techniques, but their neural network is penalized (increased loss) for veering too far off of a prescribed list of waypoints. [21] use a similar approach and also intentionally focus on high-speed scenarios, similar to racing.

IV. DeepRacing: F1 Racing Simulation

In order to train an algorithm to race autonomously we need a reliable way to generate annotated training data. Furthermore, it is not enough to only obtain training data, but also important to close-the-loop and autonomously race in the same environment to enable empirical evaluation. Consequentially, in order to generate training data under realistic racing conditions, we have converted [2] the official F1 racing game released by Codemasters©, into a simulation environment. This is the first time, the high-fidelity and photo-realistic F1 game has been used as a platform for training autonomous race cars. The game is extremely photo-realistic, as shown in Figure 3, and is based on high-fidelity simulated physics. Due to its realism the game is also used in the Formula One eSports Series. There is a lot of evidence to suggest that real-life F1 drivers use this game for practicing [22]. During the 2020 Covid-19 lock-down, many real F1 drivers were competing

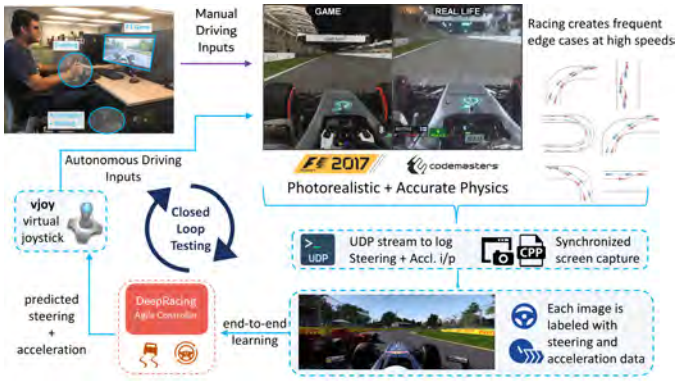

Fig. 3. We use the F1 Codemasters© game as a virtual testbed for training and closed-loop testing for our autonomous racing deep neural network. This is the first time the highly photo-realistic and high fidelity physics engine enabled game has been used for autonomous racing development.

online in virtual races in this game, due to its realism. The game can emulate weather conditions and its effects on tire degrading, braking, and handling. The game advertises a "fire-and-forget" data stream of telemetry data containing a variety of information about the game's current state over a User Datagram Protocol (UDP) network socket. Each packet in the stream is a snapshot of the game's state tagged with a timestamp for when that state was generated (Figure 3). The state variables broadcast by the game include, but are not limited to: (1) Steering angle, throttle and brake of all vehicles (including the ego vehicle), (2) Position and velocity of all vehicles, (3) Various state information about the ego vehicle such as wheel speed, amount of fuel remaining, and tire pressure. We have developed a C++ API for collecting training data, and testing learned models in the F1 game (Figure 3). Users can receive timestamped UDP packets and images as C++ objects. Additionally, we provide bindings to the very popular Robot Operating System 2.0 (ROS2). The testbed also supports the ability to close-the-loop and autonomously drive the F1 car in the game using control inputs predicted by autonomous driving policies.

V. Bezier Curve trajectory synthesis

In our previous work [2], we presented AdmiralNet as an extension of NVIDIA's PilotNet [3] for autonomous racing. It combines the feature extraction capability of a Deep Convolutional Neural Network (CNN) with the temporal memory capabilities of a Long Short-Term Memory (LSTM) Cell. In this work we modify our network to map images to waypoints for a car to follow using a parameterized trajectory. Once the waypoints are obtained in the ego vehicle's frame of reference, they are passed onto a pure-pursuit [23] lateral controller to transform into steering control commands.

A. Brief background on Bézier Curves

A Bézier curve is a parametric curve heavily used in computer graphics and related fields. Interestingly enough, the curve, a linear combination of Bernstein Polynomials, is named after Pierre Bézier, who originally developed them

to model car bodies on Renault racecars. A Bézier curve is formed from a combination of Bernstein polynomials that maps a scalar parameter $t \in [0, 1]$ to a point in a euclidean space of dimension d, $\mathbb{R}^d$. The Bézier Curve is a weighted combination of a set of "control points", with the weights computed from the Bernstein polynomial basis. For a set of control points:

$$\mathbb{P} = \{\mathbf{P}_0, \mathbf{P}_1, \mathbf{P}_2, \mathbf{P}_3, ..\mathbf{P}_n \in \mathbb{R}^d\}$$

The corresponding Bézier curve, $\mathbf{B} : [0, 1] \rightarrow \mathbb{R}^d$, as a function of the parameter, t, is:

$$\mathbf{B}(t) = \sum_{k=0}^{n} \binom{n}{k} (1-t)^{n-k} t^k \mathbf{P}_k$$

In our case, the parameter t will represent time (or step size) normalized to the interval $[0, 1]$ d, is called the "dimension" of the Bézier curve and the integer n, is called the "degree" or "order" of the curve.

We use Bézier curves as a parameterized representation of trajectories for a pure pursuit controller as well as a means of predicting trajectories for a racecar to follow.

B. Supervised Waypoint Prediction

We present a network architecture designed to map sequences of images (a context window) to sequence of waypoints in the car's task space. The input of this model is a sequence of C color (RGB) images, each with height H and width W, such that each input tensor is $Cx3xHxW$. For our experiments, we use $C = 5$, $H = 66$, and $W = 200$. It's output is a sequence of waypoints in the car's task space that are are passed to a pure-pursuit lateral controller to follow. This architecture is shown in Figure 4.

Each image is passed through a CNN that maps it to a vector. These C feature vectors are then used as the inputs to C recurrent calls to an LSTM with a hidden dimension h to build up the LSTM's hidden state with a learned encoding of the context window. We also extend [24]'s method of 3D "spatio-temporal convolution" by passing the same sequence of images through a 3D convolutional network. The output of this 3D convolution is then used as the input for p additional recurrent calls to the LSTM. The resulting p outputs are then passed to a linear layer with input dimension h and output dimension 2. The outputs of this linear layer are taken as the sequence of predicted waypoints, $[\hat{\vec{y}}_1, \hat{\vec{y}}_2, \hat{\vec{y}}_3, ...\hat{\vec{y}}_p \in \mathbb{R}^2]$, each being a point in the car's local coordinate system, consisting of a lateral axis and a forward axis. We train this network to minimize the average euclidean distance between the predicted waypoints and the ground-truth waypoints gathered from expert driving, $[\overset{*}{\vec{y}}_1, \overset{*}{\vec{y}}_2, \overset{*}{\vec{y}}_3, ..., \overset{*}{\vec{y}}_p \in \mathbb{R}^2]$,

$$\mathbb{L}_{waypoint} = \sum_{i=0}^{p} \sqrt{\hat{\vec{y}}_i - \overset{*}{\vec{y}}_i}$$

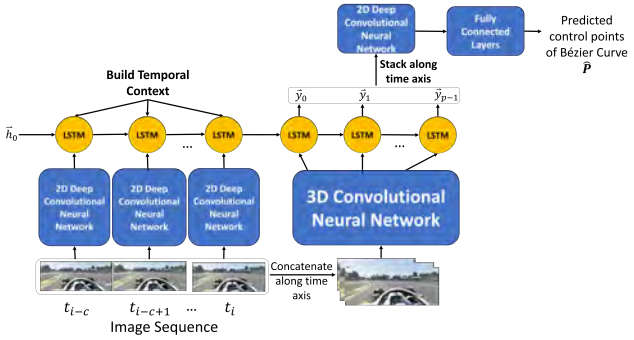

Fig. 4. Architecture For The AdmiralNet Bézier Curve Predictor.

C. AdmiralNet For Bézier Curve Prediction

Waypoint prediction also has it's limits. Derivative information cannot be implicitly encoded in a list of waypoints, they must be inferred numerically and are therefore not well suited to gradient back-propagation. Additionally, this approach suffers from the "curse of dimensionality". Each trajectory required Nxd parameters to represent, forcing the machine learned model to learn significantly more parameters to predict more dense trajectories.

To address these limitations, we present a novel approach for predicting future trajectories by using Bézier Curves as a dimensionality reduction technique. We train our model to predict the control points of a Bézier Curve instead of directly predicting waypoints. I.e., *we train a model to predict a parameterized representation of a curve rather than to learn samples from that curve*. This method addresses both limitations of the waypoint method. Derivatives of a Bézier Curve can be readily computed in closed-form. Additionally, any number of points can be sampled from a Bézier Curve without the need to learn any additional parameters in a machine learned model.

Just as for waypoint prediction, we use a Pure Pursuit controller to map a predicted trajectory to steering and throttle. We evaluate the predicted Bézier Curve on a fixed sample of the interval $[0, 1]$ to produce predicted waypoints rather than predicting the waypoints directly. For each pair of image sequences and ground truth waypoints, the network is trained to minimize a loss function that is a weighted sum of three terms:(1) The average squared norm between the predicted Bézier Curve control points and that of a least squared fit to the ground truth trajectory points, (2) The average euclidean distance between the predicted Bézier Curve evaluated on the interval $[0, 1]$ and the ground truth waypoints: What we call "Position Loss", and (3) The average euclidean distance between the predicted Bézier Curve's derivative (velocity) evaluated on the interval $[0, 1]$, and the ground truth velocities.

$$\mathbb{L}_{position} = \frac{1}{N} \sum_{i=0}^{N-1} \hat{\vec{y}}_i - \overset{*}{\vec{y}}_i; \mathbb{L}_{velocity} = \frac{1}{N} \sum_{i=0}^{N-1} \frac{d\hat{\vec{y}}_i}{dt} - \frac{d\overset{*}{\hat{\vec{y}}_i}}{dt}$$

$L_{control\_point} = \frac{1}{n-1} \sum_{i=0}^{n} \hat{\vec{p}}_i - \vec{p}^*_i$

To generate a lookahead point for the Pure Pursuit controller, the Bèzier Curve predictor takes uniformly spaced samples from the predicted Bèzier Curve. To generate throttle commands, a bang-bang control approach is used. If the car's current velocity is slower than the reference velocity of the predicted Bèzier Curve at the lookahead point, the throttle is set to it's maximum value and brake is set to 0. Otherwise, the brake is set to it's maximum value and throttle is set to 0.

VI. Experimental Results

Using our DeepRacing F1 simulator, we provide case studies which compare the Bezier curve trajectory synthesis approach to the following end-to-end supervised (behavioral cloning) methods (1) PilotNet (pixels to control); (2) CNN + LSTM (pixels to control), and (3) Waypoint trajectory predictor.

Each neural network architecture was trained for 100 epochs under it's corresponding loss function with a mini-batch size of 128. Stochastic Gradient Descent with a step size of 10^{-4} was used as the underlying optimization routine for network weight training. For the Bézier Curve predictor, we use weighting factors of $w_{position} = 1.0, w_{velocity} = 0.1, w_{control\_point} = 0.05$. We trained each model on $\sim$25000 images of training data from the Australia F1 circuit. The data was obtained using our DeepRacing API while an expert was driving the racecar on the track. Each model was then tested in a close-loop manner on the F1 Australia circuit for 5 test laps around the track, the car was reset to the same starting position on each lap. For this work, we only consider the single-agent version of the problem - i.e. only the ego racecar is present on the track at any time. The multi-agent version of the problem is part of ongoing and future work.

1) Closed Loop Autonomous Racing Results: For PilotNet and the CNN-LSTM architectures, each image is labeled with steering and acceleration. For both the Waypoint predictor and the Bézier Curve predictor, each image is labeled with 60 future waypoints from an expert driver. For these experiments, a context length of $C = 5$ is used for both the waypoint predictor and the Bèzier Curve predictor. The waypoint predictor was configured to predict 20 timesteps into the future, corresponding to 1.4 seconds. The same timescale was used for the Bèzier Curve predictor. To measure performance, we define a "boundary failure" (BF) to be when an autonomous agent veers outside the bounds of the track. We ran each model for 5 laps and recorded the following metrics: (1) Whether the model successfully completed a lap, (2) Mean Lap time (if a lap was successfully completed), (3) Mean time between boundary failures (TBF), (4) Mean distance along the track between boundary failures (DBF), and (5) Number of Boundary Failures (NBF).

The results from these experiments are tabulated in Table 6 . All figures are arithmetic means across all 5 laps. DNF indicates "did not finish" a successful lap.

The Bézier Curve predictor outperforms all other models on all metrics. PilotNet was unable to complete a successful lap, as it crashed into a wall almost immediately (within 5 seconds on all 5 runs). The CNN+LSTM architecture was also unable to complete a successful lap, but managed to make it around the first turn and down the initial straightaway. The Bézier Curve predictor also results in a smoother velocity curve than direct waypoint regression (see Figure 5).

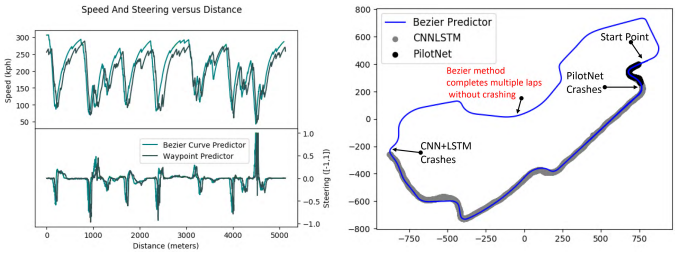

Fig. 5. A F1 style control plot for a test run. The Bézier Curve predictor produces smoother velocity profiles. [Right] A plot of the path followed by our approach versus the others. PilotNet fails almost immediately and the CNN+LSTM only makes it about half-way around the track.

Model Configuration	Lap Time (s)	TBF (s)	DBF (m)	Number of boundary failures	Successful laps
PilotNet	DNF	4.0	181.4	1,800	0
CNN-LSTM	DNF	6.4	304.5	3,600	0
Waypoint Predictor	106.7	16.7	855.8	6	4
Bezier Curve Trajectory Predictor	**101.7**	**33.6**	**1786.36**	**2**	**5**

Fig. 6. Results of our closed-loop testing. The Bézier Curve Predictor outperforms all of the other models on all metrics. "DNF" indicates the model did not finish a lap. TBF: Mean time between boundary failures, DBF: Mean distance between boundary failures.

Longevity test: We also performed a longevity test of each of the trajectory-based methods. In this test, each model was deployed to a test run and allowed to continue until the vehicle either crashed or became inoperable. The direct waypoint predictor + pure pursuit lasted for 11 laps before the car completely crashed into a barrier. The Bezier curve approach lasts for 106 laps before the car crashes (likely due to heavy tire degradation modeled in the game).

Computation Time: The control loop for each approach consists of three steps: Get a snapshot of the circular buffer containing the C most recently sampled images, evaluate the model on this snapshot, and generate control commands from the model's output. We measured the running time of this control loop for each approach. The Bèzier Curve predictor is the slowest model (PilotNet was fastest at $125Hz$). This is not surprising as it involves the most neural network layers. However, at $17Hz$, it is still sufficiently fast for autonomous racing in the game, especially given the stability and autonomous racing performance of this method.

1/10 scale autonomous racing testbed: Additionally, we conduct an offline evaluation of our Bèzier Curve Predictor on a $\frac{1}{10}$-scale autonomous racecar (Figure 7[Top]). The Bèzier

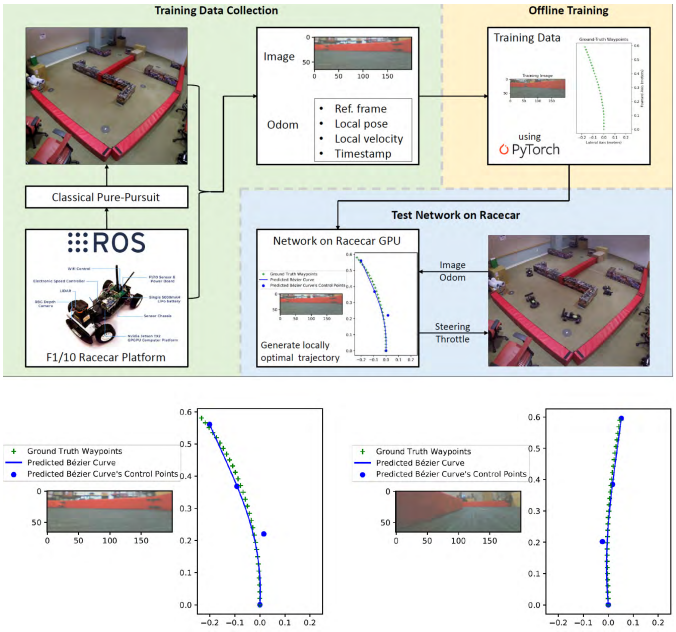

Fig. 7. [Top]DeepRacing AI was implemented on a 1/10 autonomous racing testbed using ROS. [Bottom] The ground-truth waypoints and predicted B'ezier Curve next to sample images from the testbed.

Curve Predictor was trained on a set of 15000 images taken from a webcam attached to the front of the racecar while an expert driver drove the car around a track. We then train the Bezier curve trajectory predictor on a randomly selected 13500 of these images for 200 epochs, with the remaining 1500 images reserved for offline validation. After training was complete, we evaluate the trained model with the RMSE between the predicted Bèzier Curve and the ground-truth waypoints for an image sequence ending in each of the unseen images. the Bèzier Curve Predictor achieved an RMSE value of 0.045. Figure 7[Bottom] shows some example images and the corresponding predicted Bèzier Curves.

VII. Conclusion and Future Work

We present DeepRacing - a novel end-to-end framework, and a virtual testbed for training and evaluating algorithms for the hard challenge of autonomous racing. We also develop and present a new parametrized trajectory-based end-to-end learnable network for autonomous racing which uses Bézier Curves. It outperforms traditional end-to-end networks by a significant margin and outperforms waypoint-based planning by ~ 5 seconds in terms of lap time and by over 100% in terms of time between failures, all while being robust and computationally tractable. The results and the work so far has focused on autonomous racing in a time-trial manner, i.e. only one vehicle on the track at a time. Our ongoing and future work include a more focused study of head-to-head racing (multi-agent setting) with a stronger focus on real-world racing metrics like lap time and overall race position. We also intend to explore and compare our method with reinforcement learning based approaches for this problem.

References

[1] Global championship of driverless cars. url=https://roborace.com/, journal=Roborace.

[2] Trent Weiss and Madhur Behl. Deepracing: A framework for agile autonomy. *Design, Automation and Test in Europe Conference*, 2020.

[3] M. Bojarski, P. Yeres, A. Choromanska, et al. Explaining how a deep neural network trained with end-to-end learning steers a car. *CoRR*, abs/1704.07911, 2017.

[4] Jung Leng Foo, Jared Knutzon, Vijay Kalivarapu, James Oliver, and Eliot Winer. Path planning of unmanned aerial vehicles using b-splines and particle swarm optimization. *Journal of aerospace computing, Information, and communication*, 6(4):271–290, 2009.

[5] Maxim Likhachev and Dave Ferguson. Planning long dynamically feasible maneuvers for autonomous vehicles. *The International Journal of Robotics Research*, 28(8):933–945, 2009.

[6] Martin Buehler, Karl Iagnemma, and Sanjiv Singh. *The DARPA urban challenge: AV in city traffic*, volume 56. springer, 2009.

[7] Christos Katrakazas, Mohammed Quddus, Wen-Hua Chen, and Lipika Deka. Real-time motion planning methods for autonomous on-road driving: State-of-the-art and future research directions. *Transportation Research Part C: Emerging Technologies*, 60:416–442, 2015.

[8] Scott Pendleton and Hans et al. Andersen. Perception, planning, control, and coordination for autonomous vehicles. *Machines*, 5(1):6, 2017.

[9] Matthew Johnson-Roberson, Charles Barto, Rounak Mehta, Sharath Nittur Sridhar, Karl Rosaen, and Ram Vasudevan. Driving in the matrix: Can virtual worlds replace human-generated annotations for real world tasks? *arXiv preprint arXiv:1610.01983*, 2016.

[10] Stephan R Richter, Vibhav Vineet, Stefan Roth, and Vladlen Koltun. Playing for data: Ground truth from computer games. In *European Conference on Computer Vision*, pages 102–118. Springer, 2016.

[11] Bernhard Wymann, Eric Espié, Christophe Guionneau, Christos Dimitrakakis, Rémi Coulom, and Andrew Sumner. Torcs, the open racing car simulator. *http://torcs. sourceforge. net*, 4:6, 2000.

[12] Ana I. Maqueda, Antonio Loquercio, Guillermo Gallego, Narciso N. García, and Davide Scaramuzza. Event-based vision meets deep learning on steering prediction for self-driving cars. *CoRR*, abs/1804.01310, 2018.

[13] Tharindu Fernando, Simon Denman, Sridha Sridharan, and Clinton Fookes. Going deeper: Autonomous steering with neural memory networks. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 214–221, Hawaii Convention Center HI, 2017.

[14] Felix A Gers, Jürgen Schmidhuber, and Fred Cummins. Learning to forget: Continual prediction with lstm. 1999.

[15] Hesham M. Eraqi, Mohamed N. Moustafa, and Jens Honer. End-to-end deep learning for steering autonomous vehicles considering temporal dependencies. *CoRR*, abs/1710.03804, 2017.

[16] Chenyi Chen, Ari Seff, Alain Kornhauser, and Jianxiong Xiao. Deepdriving: Learning affordance for direct perception in autonomous driving. In *ICCV*, December 2015.

[17] F. Altché and A. de La Fortelle. An lstm network for highway trajectory prediction. In *2017 IEEE 20th International Conference on Intelligent Transportation Systems (ITSC)*, pages 353–359, Oct 2017.

[18] Eslam Mohammed, Mohammed Abdou, and Omar Ahmed Nasr. End-to-end deep path planning and automatic emergency braking camera cocoon-based solution. In *Machine Learning for Autonomous Driving, NeurIPS 2019 Workshop*, December 2019.

[19] Dian Chen, Brady Zhou, Vladlen Koltun, and Philipp Krähenbühl. Learning by cheating, 2019.

[20] Mayank Bansal, Alex Krizhevsky, and Abhijit S. Ogale. Chauffeurnet: Learning to drive by imitating the best and synthesizing the worst. *CoRR*, abs/1812.03079, 2018.

[21] Jeff Michels, Ashutosh Saxena, and Andrew Y. Ng. High speed obstacle avoidance using monocular vision and reinforcement learning. ICML '05, page 593–600, New York.

[22] Max verstappen trains using sim racing. *Redline UK*, 2017. http://www.teamredline.co.uk/work/max-verstappen/.

[23] Richard S Wallace, Anthony Stentz, Charles E Thorpe, Hans P Moravec, William Whittaker, and Takeo Kanade. First results in robot road-following. In *IJCAI*, pages 1089–1095. Citeseer, 1985.

[24] Lu Chi and Yadong Mu. Learning end-to-end autonomous steering model from spatial and temporal visual cues. In *Proceedings of the Workshop on Visual Analysis in Smart and Connected Communities*, VSCC '17, pages 9–16, NY, USA, 2017. ACM.