DeepRacing: A Framework for Autonomous Racing

1st Trent Weiss
Department of Computer Science
University of Virginia
Charlottesville, VA, USA
ttw2xk@virginia.edu

2nd Madhur Behl
Department of Computer Science
University of Virginia
Charlottesville, VA, USA
madhur.behl@virginia.edu

***Abstract*—We consider the challenging problem of high speed autonomous racing in realistic dynamic environments. DeepRacing is a novel end-to-end framework, and a virtual testbed for training and evaluating algorithms for autonomous racing. The virtual testbed is implemented using the realistic Formula One (F1) Codemasters game, which is used by many F1 drivers for training. We present AdmiralNet - a Convolution Neural Network (CNN) integrated with Long Short-Term Memory (LSTM) cells that can be tuned for the autonomous racing task in the highly realistic F1 game. We evaluate AdmiralNet's performance on unseen race tracks, and also evaluate the degree of transference between the simulation and the real world by implementing end-to-end racing on a physical 1/10 scale autonomous racecar.**

I. Introduction

Vision-based solutions are believed to be a promising direction for autonomous driving due to their low sensor cost, and recent developments in deep learning. End-to-end models for autonomous driving have attracted much research interest [1, 2, 3], because they eliminate the tedious process of feature engineering. Algorithms for end-to-end driving are being trained and evaluated in both simulation [4, 5], and in some cases on real vehicles [6]. However, there is a lot of progress to be made as these studies primarily use simulators with simplified graphics and physics [TORCS [7], Udacity [8]] and therefore, the obtained driving results lack realism.

Demonstrating high-speed autonomous racing can be considered as a grand challenge for vision based end-to-end models. Autonomous racing can be considered an extreme version of the self-driving car problem, 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. Autonomous racing is also highly likely to become a futuristic motorsport featuring a head-to-head complex battle of algorithms [9]. For instance, Roborace [10] 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, Autonomous Formula SAE [11]) 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, 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. This game is highly realistic - both in physics and graphics - and is used by many real world F1 drivers for training. Our DeepRacing C++ API enables easy generation of training data under a variety of racing environments, without the cost and risk of a physical racecar, and racetrack. This allows anyone to use the high fidelity physics and photo-realistic capabilities of the F1 game as a simulator, and without hacking any game engine code. The DeepRacing framework is open-source *https://github.com/linklab-uva/deepracing*.

In addition we present AdmiralNet - a CNN integrated with a LSTM that can be tuned for the autonomous racing task in the F1 game. We conduct comprehensive case studies using the F1 simulation environment. We also evaluate the degree of transference between solutions in simulation and the real world. Our evaluation demonstrates the ability to train and test end-to-end autonomous racing algorithms using both the F1 game, and the physical testbed.

A. Contributions of this paper

The paper has the following contributions:

1) This is the first paper to demonstrate and enable the use of a highly photo-realistic Formula 1 Codemasters© game, with a high fidelity physics engine as a test-bed for developing autonomous racing algorithms, and testing them in a closed-loop manner. This framework will be made publicly available.
2) We implement and evaluate a deep neural network (DNN) called AdmiralNet. AdmiralNet builds upon the NVIDIA's PilotNet [12] architecture but uses optical flow, and a combination of CNN and an LSTM to learn spatio-temporal aspects of racing.
3) We implement and test AdmiralNet both in the F1 simulation, and on a real 1/10 scale autonomous F1 racecar.

II. Related Work

Autonomous racing can be considered a superset of autonomous driving, and as such, there is a lot of literature on methods for end-to-end autonomous driving. We divide the related work into simulation testbeds and autonomous driving methods and provide a brief reprise on both.

A. Autonomous driving simulators

There are several examples of video games being used as a simulator to aid development of autonomous driving. Several are based on the popular Grand Theft Auto (GTA) game [13], utilizing the high fidelity graphics of Rockstar Games'©

state-of-the-art rendering engine. However, while being photo-realistic, this game is known for flaws in the underlying physics engine, since it was not intended to be used as a simulator. This technique also requires a modification to the underlying game engine code in order to extract data (steering, acceleration, etc.) attached to each game screenshot. This modification ran afoul of Rockstar's copyright protections, and both of these projects infamously received cease and desist letters from Rockstar Games to pull the code from public domain. While creating DeepRacing, we ensured that our F1 2019 simulator requires no such "hacking" of the unaderlying game engine. It uses public APIs for screen capturing as well as a stream of UDP packets that the F1 game broadcasts. End-to-end driving was showcased in the car racing game TORCS [7] using Reinforcement Learning but its physics and graphics lack realism.Microsoft AirSim [14] and CARLA [15] are examples of open-source autonomous driving simulators but they are are largely restricted to urban driving scenarios and are not suited for development and testing of end-to-end autonomous racing.

B. Autonomous Driving Architectures

In one of the earliest work on end-to-end autonomous driving NVIDIA [12] 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. However, PilotNet is limited by it's inability to capture temporal information. Each input image is run through the CNN separately with no time-varying context around that image. This is not just a limitation of PilotNet, but of CNNs in general. Fernando et al [16] present a different approach that uses Long Short-term Memory Cells [17] as a means of capturing a history of the steering trajectory and encode temporal structure of the problem. Xu et al[18] present a similar technique that uses a Fully Convolutional Network (FCN) to extract a feature representation of the input space, but limit their model to classification among a discrete set of actions: go straight, stop, left turn, and right turn.

Eraqi et al [19] 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. Eraqi also presents the novel concept of a "sliding window", a variable length temporal sequence that is input into the LSTM, allowing the model to encode temporal information at arbitrary lengths.

AdmiralNet extends the sliding window of Eraqi into the prediction layer, allowing the LSTM model to learn how the time-varying flow of pixels can predict future control outputs using Optical Flow fields. This is described in detail in section V. Chen et. al [5] 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. However, like PilotNet, their approach only considers current image data and is limited to a fixed set of affordance templates that are purpose-built for only highway driving.

III. Autonomous Racing 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 vehicle's control outputs is $\mathbb{U}$, then the general problem of autonomous driving is a mapping from:

$$\mathbb{X} \to \mathbb{U} \tag{1}$$

There is a great body of work centered around the special case where $\mathbb{U}$ consists of only steering angles [12, 18]. We will call this subset of the broader control domain u. Existing work has focused on mapping what the car's sensor suite is seeing at the present time to a single steering angle, at the present time. This is done in an end-to-end manner i.e. the DNN is trained to directly map pixels to steering angles. Expressed mathematically, a function of the form:

$$\mathbb{X} \to u \tag{2}$$

However, this model of autonomous driving does a poor job of capturing how expert drivers behave. For instance, a Formula One racing driver does not simply analyze the pixels directly in front of him and map those pixels directly to a single steering angle and throttle pressure. An expert driver considers some history of what he/she has previously seen. In our framework, this is equivalent to a list of sensor readings taken in the past: $X_{i-c}, X_{i-c+1}, X_{i-c+2}, ..., X_i \in \mathbb{X}$, where the subscript i represents the current time and c represents some number of time-steps into the past. We call this list of sensor readings a *context window*.

We apply the same thinking to the control outputs of the race car. Expert racing drivers don't just apply single control commands one at a time, they apply smooth transitions to the car's control as a means to an end. An expert driver surmises a "long" term intent for how they want the car to behave some amount time into the future and then apply a continuous control input to accomplish that end. For example, continuously turning the steering wheel to the right as a means of making the car turn right or smoothly pressing the accelerator to get the car up to speed. These transitions need not be "smooth" in the sense of being slow and steady, a driver might slam on the brakes as a means of avoiding a crash, but they still represent a continuous curve of control inputs that are a means of accomplishing the drivers high-level intent. Expressed mathematically, such a set of control is: $\vec{U}_{i+1}, \vec{U}_{i+2}, \vec{U}_{i+3}, ...\vec{U}_{i+p} \in \mathbb{U}$, where $\vec{U}_k$ represents the driver's chosen control output at time k. These commands start at the current timestep, i, and go forward into the future. We call this list of forward-looking control outputs an *intent window*, representing the set of commands the driver uses to accomplish his intended goal.

The problem formulation is to show that this approach of predicting long-term *intent* based on some amount of *context* produces more accurate predictions of expert driver behavior than statically mapping immediate sensor measurements to control output.

More specifically, AdmiralNet provides a mapping:

$$\mathbb{X}^c \to u^p \tag{3}$$

Where c represents the size of the context window and p represents the size of the intent window. We next describe our F1 Codemasters© virtual testbed that is used for approaching the autonomous racing problem.

IV. DeepRacing: F1 Racing Simulation

In order to train an end-to-end neural network to race autonomously using the context and the intent described in the previous section; we need a reliable way to generate annotated training data. Obtaining such annotated data for real motorsport racing drivers is difficult since these data are often trade-secrets and not available in the public domain. 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 evaluation of end-to-end models, and reinforcement learning approaches.

Consequentially, in order to generate training data under realistic racing conditions, we use the F1 2019 racing game released by Codemasters©. This is the first time, the immensely popular and photo-realistic F1 game has been used as a platform for training autonomous race cars. Unlike the ill-fated GTA simulator, the F1 simulation we present does not involve modifying any game code or behavior but instead taps into the UDP data stream broadcast by the game. The game provides access to twenty major Formula One racetracks. The driver can choose to race alone or with multiple opponents, allowing for training data in more relaxed single-car scenarios as well as more hectic multi-agent racing environments.

A. DeepRacing realism

Photo-realism and physics modeling: The game is extremely photo-realistic, as shown in Figure 1, and is based on high-fidelity simulated physics. Due to its realism, the F1 series was the first game to be used in the Formula One eSports Series, which debuted in 2017 [20]. The photo-realism in the driver's point-of-view combined with the physics realism of the game's engine provide a strong opportunity to gather training data as close to real-world racing scenarios as one can get without the cost and risk of a real race-car.

Deep customisation: The game facilitates a high degree of customization including adjustable dead zones, linearity, and saturation for vehicle control. Settings for aerodynamics, traction, tyre choices, etc. are also highly customizable.

The F1 game advertises a telemetry stream of information about the games's current state over a UDP socket in a "fire and forget" type broadcast on the host machine's network interface card. Each packet in the stream, is a snapshot of the game's state at the time that packet was generated.

We developed our own software infrastructure for both grabbing screenshots of the driver's perspective (the "ego" vehicle) in the F1 game and automatically tagging them with ground-truth values of the game's state at the time that image was captured. The state variables 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 (including the ego vehicle)
3) Various state information about the ego vehicle such as wheel speed, amount of fuel remaining, and tire pressure.

A full description of the F1 telemetry stream and all of the information it provides is available on a Codemaster's™ forum [21].

Finally, our test-bed setup also supports the ability to close the loop by autonomously driving the F1 car in the game using control inputs predicted by autonomous driving policies. This is accomplished by pushing the steering and acceleration control inputs back into the game via a virtual joystick API built on top of vJoy [22].

We evaluate this closed-loop capability by utilizing "oracle" data from the simulation framework and a simple pure-pursuit controller to steer the car with ground-truth knowledge of the track's optimal raceline. We then measure the effectiveness of this controller by the distance from the path followed by the pure-pursuit controller to the optimal raceline. Note that this evaluation is **not** intended to evaluate any particular autonomous driving model, but is intended to show the accuracy of our closed-loop framework for testing such models. This closed-loop test shows a mean distance to the optimal raceline of **0.928963 meters**. The bottom-right side of figure 1 shows a plot of the corresponding probability density function. The top-right of figure 1 shows an optimal raceline overlaid with the path followed by our pure pursuit control, the two paths are almost indistinguishable.

This data collection and testing infrastructure, implemented in C++, is called the DeepRacing API, the first closed-loop environment of it's kind for collecting training data, and testing learned models on simulated F1 race-cars in the photo-realistic F1 2019 game. The software itself is architected as an object-oriented library that exposes a simple interface for allowing user-written code to handle data captured by the underlying infrastructure with the "hard work" of actually obtaining that data handled automatically. This software will be released under an Open-Source license.

V. AdmiralNet: Mapping Context to Intent

In this section we describe in detail the second research contribution of this work - AdmiralNet - an end-to-end deep neural network capable of learning to race autonomously both in the F1 simulation and on real datasets. The key building blocks in AdmiralNet's architecture are:

1) Convolution Neural Networks
2) Recurrent Neural Networks
3) Optical Flow Fields

A. Optical Flow

Optical flow [23] is a spatio-temporal representation of a pair of images, defined as a vector field over the image pixels representing which direction and how quickly each pixel is moving. Rather than having only the raw input images be the input to our model, we compute the optical flow vector field between adjacent pairs of images in the input sequence.This transformation serves to represent how the pixels are flowing through the scene, and can be interpreted as a first derivative

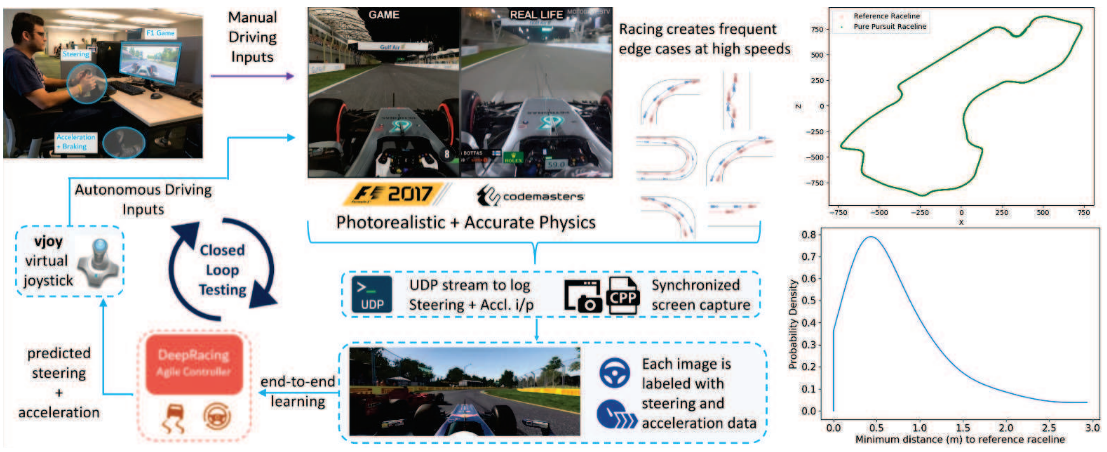

F1 Game
Steering
Acceleration + Braking
Manual Driving Inputs
GAME
REAL LIFE
F1 2017
codemasters
Photorealistic + Accurate Physics
Racing creates frequent edge cases at high speeds
Reference Raceline
Pure Pursuit Raceline
Autonomous Driving Inputs
vjoy virtual joystick
Closed Loop Testing
UDP
UDP stream to log Steering + Accl. i/p
CPP
Synchronized screen capture
predicted steering + acceleration
DeepRacing Agile Controller
end-to-end learning
Each image is labeled with steering and acceleration data
Probability Density
Minimum distance (m) to reference raceline

Fig. 1. DeepRacing uses the Formula One Codemasters game as a virtual testbed for training and closed-loop testing for our autonomous racing deep neural network - AdmiralNet. This is the first time the highly photo-realistic and high fidelity physics engine enabled game has been used as a simulation environment without cracking and tweaking any game engine code.

TABLE I
COMPARISON BETWEEN STEERING ANGLE PREDICTIONS PILOTNET, CNN-LSTM, AND ADMIRALNET. THE MODELS WERE TRAINED ON THE DATA-SET FROM AUSTRALIAN GP, AND TESTED ON UNSEEN DATA-SETS FROM BOTH AUSTRALIA, AND BAHRAIN TRACKS IN THE F1 GAME.

	Australia	Bahrain
	Testing (RMSE)	Testing (RMSE)
PilotNet	0.1699	0.3309
CNN-LSTM	0.1485	0.40255
AdmiralNet	**0.0368**	**0.067**

of the image. The intuition being that expert drivers don't simply map color information into steering commands, even with temporal context, but rather they watch how objects are flowing(moving) through the scene. E.g. seeing the edges of a road flow to the right indicates that a right turn will soon be necessary to stay on the road. Using optical flow vectors as the input for a temporal-context task has been shown to perform well for video classification [24]. Each optical flow vector field is interpreted as a 2-channel image, 1 for the horizontal component of the vector and 1 for the vertical component. This vector-field representation of an image has the added benefit of being invariant to changes in the distribution in the color values of the image pixels themselves. What colors the image contains is no longer the only input to our model, we additionally use how the pixels are moving. We show that this makes AdmiralNet more accurate on input distributions that were not seen during the neural network training process.

B. *AdmiralNet: CNN-LSTM for end-to-end autonomous racing*

Our approach, named AdmiralNet as an improvment over NVIDIA's PilotNet, combines the static analysis capabilities of a Deep Convolutional Neural Network (DCNN), the temporal memory capabilities of a Long Short-Term Memory (LSTM) cell(who's internal state vector is initialized with zero-mean gaussian noise), and the pixel flow encoding of Farneback's optical flow. Given a time sequence of past images, a context window, we predict a time sequence of future steering angles, an intent window. Much of the related work in this domain is centered around combining CNNs and LSTM cells together, with a CNN serving as an image encoder and an LSTM as an image decoder. In particular, Eraqi [19] uses this technique with a sliding window to build up temporal context. AdmiralNet builds on this approach with two additional novel steps.

Firstly, we convert each image in an input sequence of c images to an optical flow vector field with Farneback's algorithm. Each image and it's corresponding flow field is interpreted as a 3-channel input to a standard CNN. The CNN acts as an image encoder that maps optical flow fields (a form of image velocity) to deep feature vectors, enabling the network to learn how the systems 0th order (greyscale images) and 1st order (optical flow) dynamics influence the driving behavior of the human example. In our case, the CNN is NVIDIA's PilotNet architecture, except with batch normalization layers in between each convolutional layer. The feature vector from this CNN is then applied to the input of an LSTM cell. The resulting LSTM state is then projected into a prediction of p control output vectors, each of dimension d, by calling the LSTM p additional times. The outputs from these p calls are taken as predictions of control outputs p timesteps into the future.

Secondly, we extend Chi [25]'s method of 3D "spatio-temporal convolution" by passing this sequence of flow fields through a 3D convolutional network. However, unlike Chi, we use 2D convolution over the optical flow fields to build up the LSTM's state vector, and then connect the output of the 3D convolution only to the input of the LSTM for the p additional calls after the context window, taking each of the

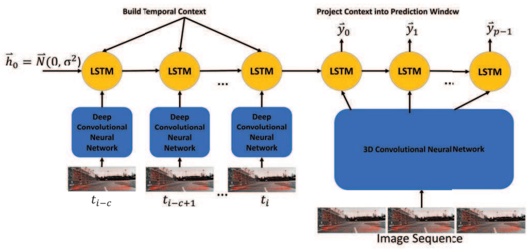

Build Temporal Context
Project Context into Prediction Window
$\vec{h}_0 = \vec{N}(0, \sigma^2)$
LSTM
Deep Convolutional Neural Network
3D Convolutional Neural Network
$\vec{y}_0$
$\vec{y}_1$
$\vec{y}_{p-1}$
t_{i-c}
t_{i-c+1}
t_i
Image Sequence

Fig. 2. AdmiralNet architecture

.

output vectors as the sequence of d predicted control outputs. For our experiments in Section VI, we only consider predicting steering angle, so $d = 1$. However, this model is scalable to predicting any number of control outputs. This combination of an encoder and decoder network allows the model to learn how the temporal dynamics of image flow map to future control outputs. Intuitively, the network is learning how a history of images and image *velocities* map to future control commands, similar to how an expert driver will estimate a scene's spatio-temporal dynamics based on how a scene is flowing.

Figure 2 is a graphical description of our DCNN architecture. This technique utilizes both the static "feature extraction" power of 2D and 3D DCNNs coupled with the temporal context-saving power of an LSTM. We flatten the output of the final convolutional layer into a vector of size 1152 and apply it to the input of an LSTM for c timestamps. We then call the LSTM p times with the output of the 3D convolution as it's input. The corresponding p outputs are taken as predictions of a future steering trajectory.

VI. Experimental Results

We present 3 case studies to evaluate AdmiralNet and demonstrate its ability to correctly predict steering angles for an autonomous race car both in the F1 simulator and a real 1/10 scale autonomous racecar test-bed. The focus on predicting steering is not a limitation of this work - AdmiralNet can be easily extended to predict multivariate outputs e.g. both steering and acceleration.

A. Result I: AdmiralNet in F1 simulator

We evaluate our approach on two data-sets obtained from the Australian GP circuit in the F1 game. The training data-set contains roughly 12000 raw annotated images. Another 12,000 images from a different lap of the same circuit are used as the test-data. We train three different DNNs on the training data-set and evaluate their performance on the test data-set. Specifically, we train PilotNet, a standard CNN-LSTM, and AdmiralNet with optical flow. All three networks were trained for 100 epochs with a learning rate of 0.001 and a batch size of 16. The steering values are normalized between -1 and 1 which correspond to rotating the steering the wheel between $[-180, 180]$ degrees. We train on a Mean Squared Error between the predicted steering and the ground truth steering. We use a context window of $c = 10$, except for PilotNet, which is evaluated image-by-image, and separately test a prediction window $p = 1$. We measure the effectiveness of each of the evaluated models by the Root Mean Square Error on the test-data. Table V-A shows the comparison between the three networks for test data-set obtained from Australia. We can see that AdmiralNet predicts the ground truth normalized steering angles with the highest accuracy compared to the CNN-LSTM, and compared to Nvidia's PilotNet. We compare the prediction performance of the same three networks (trained on the Australia data-set) on an unseen data-set from a different track altogether (the Bahrain F1 circuit). This is the real challenge for the networks since it has never seen any images from the Bahrain track during training. The performance is an indication of whether the network is capturing driver behavior vs. learning artifacts of the tracks. Table 1 summarizes the prediction RMSE for the different networks trained. AdmiralNet outperforms both PilotNet and CNN+LSTM networks by a siginificant margin, even on a track never seen during training. Based on these results, we can conclude that AdmiralNet with optical flow can successfully capture expert driver behavior in the game.

B. Result II: On the physical F1/10 test-bed

We also conduct similar experiments on a 1/10 scale physical autonomous race-car testbed. Our F1/10 testbed is fully capable of implementing the entire end-to-end AdmiralNet driving pipeline, from data gathering and annotation, to running the trained AdmiralNet in real-time for controlling the steering and acceleration of the car fully autonomously. The car can reach speeds of up to 20mph indoors, and is a realistic representation of a racecar. Figure 3 shows our F1/10 autonomous racing testbed. As shown in Figure 3, we integrate a First Person View (FPV) camera and headset with the F1/10 autonomous racecar. We are able to drive the car manually with a USB steering wheel and pedals, just like in the F1 game. The setup enables tele-operation for the purposes of collecting data to train the end-to-end DNNs. AdmiralNet requires that each image frame from the front facing camera be annotated with a steering angle and an acceleration value. We evaluated AdmiralNet on this physical testbed as well and it has a lower RMSE (0.14) than PilotNet (0.18) on the real world data-set.

VII. Conclusion and Discussion

The paper's primary contribution is DeepRacing - a novel end-to-end framework, and a virtual testbed for training and evaluating algorithms for the hard challenge of autonomous racing. The virtual testbed is implemented using a highly realistic and professional Formula One game environemnt. This is the first time that this F1 game has been used as a testbed for end-to-end autonomous racing. Unlike, using the GTA game as a simulator, our framework does not hack any game engine code making it very easy to setup. We also present AdmiralNet, a modified and improved version of Nvidia's PilotNet which uses CNN and LSTM layers with optical flow image inputs. We show that AdmiralNet trained with optical flow data, significantly out performs (by 80%) other DNNs on unseen data-sets from the F1 game, and from our 1/10 scale autonomous racing test-bed. We present results which demonstrate the ability of AdmiralNet to generalize

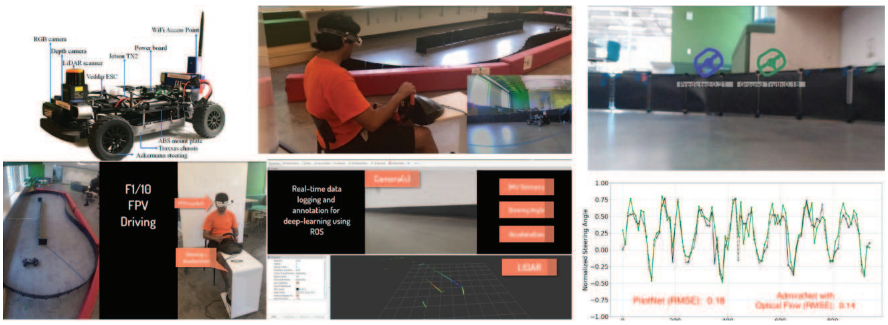
!htb 

Fig. 3. The F1/10 autonomous race car setup. We can generate annotated HD images with steering and acceleration from the front camera automatically using manual driving and a first person view headset. We use ROS to automatically synchronize and label the training images.

to both simulated and real-world image data-sets, including testing on a real video of a F1 racing. Our open-source DeepRacing framework will enable researchers to explore the limits of vision based end-to-end autonomous racing, which is increasingly getting a lot of attention from researchers in the field.

REFERENCES

[1] Eder Santana and George Hotz. Learning a driving simulator. *CoRR*, abs/1608.01230, 2016.

[2] Joel Janai, Fatma Güney, Aseem Behl, and Andreas Geiger. Computer vision for autonomous vehicles: Problems, datasets and state-of-the-art. *CoRR*, abs/1704.05519, 2017.

[3] Huazhe Xu, Yang Gao, Fisher Yu, and Trevor Darrell. End-to-end learning of driving models from large-scale video datasets. *CoRR*, abs/1612.01079, 2016.

[4] Etienne Perot, Maximilian Jaritz, Marin Toromanoff, and Raoul De Charette. End-to-end driving in a realistic racing game with deep reinforcement learning. In *International conference on Computer Vision and Pattern Recognition-Workshop*, 2017.

[5] Chenyi Chen, Ari Seff, Alain Kornhauser, and Jianxiong Xiao. Deepdriving: Learning affordance for direct perception in autonomous driving. In *The IEEE International Conference on Computer Vision (ICCV)*, December 2015.

[6] Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, et al. End to end learning for self-driving cars. *arXiv preprint arXiv:1604.07316*, 2016.

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

[8] A Brown et al. Udacity self-driving car simulator. In *GitHub Repository*. 2018.

[9] Walt Scacchi. Autonomous emotorsports racing games: Emerging practices as speculative fictions. *Journal of Gaming & Virtual Worlds*, 10(3):261–285, 2018.

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

[11] Skanda Koppula. Learning a cnn-based end-to-end controller for a formula sae racecar. *arXiv preprint arXiv:1708.02215*, 2017.

[12] M. Bojarski, P. Yeres, A. Choromanska, K. Choromanski, B. Firner, L. D. Jackel, and U. Muller. Explaining how a deep neural network trained with end-to-end learning steers a car. *CoRR*, abs/1704.07911, 2017.

[13] Stephan R. Richter, Vibhav Vineet, Stefan Roth, and Vladlen Koltun. Playing for data: Ground truth from computer games. *CoRR*, abs/1608.02192, 2016.

[14] Shital Shah, Debadeepta Dey, Chris Lovett, and Ashish Kapoor. Airsim: High-fidelity visual and physical simulation for autonomous vehicles. In *Field and Service Robotics*, 2017.

[15] Alexey Dosovitskiy, German Ros, Felipe Codevilla, Antonio Lopez, and Vladlen Koltun. Carla: An open urban driving simulator. *arXiv preprint arXiv:1711.03938*, 2017.

[16] 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.

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

[18] Huazhe Xu, Yang Gao, Fisher Yu, and Trevor Darrell. End-to-end learning of driving models from large-scale video datasets. *CoRR*, abs/1612.01079, 2016.

[19] 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.

[20] Formula one esports series. *F1 Esports*, Nov. 2017. https://f1esports.com/.

[21] Codemasters. F1 2017 d-box and udp output specification. http://forums.codemasters.com/discussion/53139/f1-2017-d-box-and-udp-output-specification.

[22] Shaul Eizikovich. vjoy. http://vjoystick.sourceforge.net/site/.

[23] Berthold KP Horn and Brian G Schunck. Determining optical flow. *Artificial intelligence*, 17(1-3):185–203, 1981.

[24] Karen Simonyan and Andrew Zisserman. Two-stream convolutional networks for action recognition in videos. In *Advances in neural information processing systems*, pages 568–576, 2014.

[25] 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.