2020-01-1017

# Implementation and Validation of Behavior Cloning using Scaled Vehicles

**Author, co-author (Do NOT enter this information. It will be pulled from participant tab in MyTechZone)**

Affiliation (Do NOT enter this information. It will be pulled from participant tab in MyTechZone)

## Abstract

Recent trends in autonomy have emphasized end-to-end deep-learning-based methods that have shown a lot of promise in overcoming the requirements and limitations of feature-engineering. However, while promising, the black-box nature of deep-learning frameworks now exacerbates the need for testing with end-to-end deployments. Further, as exemplars of systems-of-systems, autonomous vehicles (AVs) engender numerous interconnected component-, subsystem and system-level interactions. The ensuing complexity creates challenges for verification and validation at the various component, subsystem- and system-levels as well as end-to-end testing. While simulation-based testing is one promising avenue, oftentimes the lack of adequate fidelity of AV and environmental modeling limits the generalizability. In contrast, full-scale AV testing presents the usual limitations of time-, space-, and cost. Hence in this paper, we explore the opportunity for using experiential learning possible with a scaled vehicle-based deployment to overcome the limitations(e.g. simulation fidelity or experimentation costs) of scaled vehicles to lower the barriers especially at the early stages of testing of autonomy algorithms.

In recent times, several efforts have emerged for testing deep-learning-based autonomy algorithms on scaled vehicles – the Nvidia Jet racer, Amazon Deep racer, and Donkey car are being widely used. In this paper, we examine a deployment of the Donkey car Behavior Cloning software stack on a $1/10^{th}$ scaled vehicle (F1tenth) and the issues faced while deploying the other software stacks. In particular, we explored the effectiveness of: (i) mixing and matching frameworks; and (ii) use of scaled vehicles in an academic set up to support testing and deployment of supervised learning (behavior cloning) technique to achieve lane-keeping and obstacle-avoidance. We showcase that the use of this scaled-vehicle framework permitted the rapid exploration of many different test tracks (challenging with full-scale vehicle tests) while retaining realistic environmental conditions (challenging with simulation-alone testing).

## Introduction

### Sim2Real inefficiency

SAE classifies Level 3 as conditional automation where a vehicle can autonomously accelerate, brake, steer and switch lanes in a constrained environment In recent times, DL-based approaches take advantage of flexibility created by DbW (drive-by-wire) to create an end to end deployments. However, the DL approach needs good data. A small amount of high-quality data can result in a comparatively well-performing agent [1]. However, variability analysis and verification are needed to explore the limits of any deep-learning-based approach either through simulation or real-world experiments. Driving requires reacting to a wide variety of complex environmental conditions and agent behaviors. Explicitly modeling each possible scenario is unrealistic. This is an advantage of using a simulated environment [2]. However, there exists a Sim2real gap while transitioning into a real-world deployment. Exploring the real-world visual challenges beyond what can be achieved in simulation requires physical experimental testing that can be expensive.

Imitation learning has some advantages over supervised and reinforced learning [3] because we depend on an expert (human) to demonstrate the desired behavior rather than encoding a reward function. To ground our work, we focus our attention in this paper on the implementation, evaluation, and comparative analysis of a specific model i.e. the end-to-end imitation learning (behavioral cloning) model based on the paper by Bojarski et al [4]. We test this implementation with scaled autonomous vehicles, performing relatively complex level 4 tasks such as autonomous lane-keeping and obstacle avoidance. To fully exercise the variability, testing is done end-to-end on the system as a whole and not component-wise to the level of the model architecture.

### Use of scaled vehicles

Gaining experience with real-world deployments is also important for training the next generation of students. The use of scaled vehicles offers a powerful platform for testing and validation of a large class of problems [5] as they sufficiently capture the dynamics, sensing modalities, decision making, and risks of real autonomous vehicles, but are a safe, cost-friendly, and accessible platform to teach the foundations of autonomous systems[6]. Due to these above advantages Scaled vehicles have become a cardinal platform to prove the robustness of the algorithms developed for an Autonomous vehicle [7] [8] [5] principle for the safety of a vehicle.

### Conventional Control vs Deep Learning

In a conventional autonomous driving pipeline, the model must undergo a series of steps before it can output the steering or throttle values. This generally takes the form of a five-step process [9]as shown in Figure 1 which entails: (i) Sensor input processing (from cameras, LIDAR, radars, etc.) as a low-level perception to extract the relevant features (e.g., object/feature detection); (ii) Multimodal sensor-data fusion for localization to create a map and simultaneously localize the system; (iii) High-level motion-plans/behavior generation and down-selection of the best, typically applying some optimization

criteria, to achieve mission-level objectives; (iv) Motion-plan translation into a series of vehicle-maneuvers to be executed to satisfy the chosen behavior and finally (v) Control actions execution via the low-level actuator interface modules.

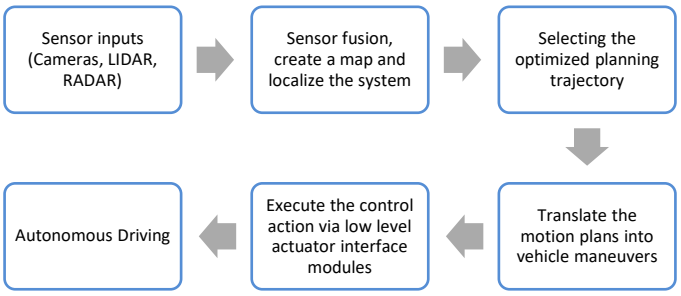


*Figure 1: Conventional autonomous driving model pipeline includes multi-stage feature-engineering as a crucial element.*

In contrast to the above discussed conventional autonomous driving technique, the end-to-end deployment framework maps the input video-stream to the lower-level throttle- and steering-outputs used to control the vehicle through a weighted non-linear model that is defined by deep neural networks. In the implementation by Bojarski [4] this is achieved via an end-to-end supervised learning framework with the video stream as an input and steering angle as a training signal.

With the advancement of machine learning, end-to-end learning methods to process bulk visual data to perform these ADAS tasks [4, 10-13] have begun to gain popularity not only in academic domains but also as a growing niche in autonomous vehicle startups [14, 15]. Deep learning frameworks when applied to ADAS applications can learn directly from large streams of data, with strong predictive models developed offline and updated systematically through online reinforcement. Unsupervised networks can develop these predictions without the need for extensive labeling [16]. Despite the success and popularity of deep learning methodologies in academic domains and research projects, they have not gained broad popularity in commercial ADAS deployment with any of the top automakers. This is largely due to the challenges inherent to any deployment of deep learning – the size of the dataset required to cover a large spectrum of visual cues and driver behaviors can be prohibitively large. Further, overfitting and underfitting of networks can affect the generalization of driving behavior. While helping eliminate/merge multiple stages of explicit intermediate feature detections (e.g. road features, obstacles) such deep-learning frameworks have been unable to account for the rare but safety-critical edge-case scenarios by design and in practice [2]. Statistical methods have determined that an autonomous vehicle must be driven hundreds of millions of miles to demonstrate its reliability in terms of failure modes that lead to fatalities or injuries [17]. Thus, there is a need to properly benchmark these performances to generate reliable criteria that need to be met before a deep learning-based ADAS system can be considered a safe alternative.

**Deployment benchmarking - Requirements & specifications**

The primary step in testing the limitations of any end-to-end learning network is to develop a benchmarking set up under which the algorithm should be tested. Machine learning, and in particular, imitation learning like behavior cloning can be highly susceptible to dataset bias. A given pre-trained network can demonstrate diverse driving behaviors based on time of day (lighting conditions), weather (clarity of vision), environment (colors and textures in data), and camera angle (point of view) due to bias in the training data.

The selection of test criteria follows the five challenges in verifying AI and assumptions to be accounted for, as described by Seshia *et al.* [18]:

1. **Environment modeling:** Accounting for all environmental variables is an impossible task. Often, the non-deterministic nature of the output in training data can be challenging to account for, especially in behaviors like obstacle avoidance where similar scenarios may result in different outcomes with different drivers. Thus any benchmarking experiment must allow for generalization in environment variables. This is done by selecting five test scenarios from diverse environments to collect training data – (i) indoor lab, fluorescent lighting (ii) pedestrian track, bridged by grass, daytime (iii) on-road, night-time, (iv) scaled racetrack, outdoor, daytime and (v) simulation environment (smooth textures, shadow-less lighting).

2. **Modeling the Learning Systems:** The high dimensional and "black box" nature of deep learning models make the model specification for a particular task partly a form of art and intuition. There are many evaluation methods in literature to compare the predicted output behavior with respect to training input to modify the model accordingly. In this paper, we implemented a CNN model without changing its inherent model structure but evaluate it's Model Mean Square Error, Histogram Prediction Error, R Squared value, Scatter Plot Error, and R-squared value. The accuracy of the prediction for training data is alone not a good enough metric for performance as it does not easily account for generalizability under untested conditions (as seen in the paper) – hence there is a need for a more generalizable metric.

3. **Formal Specification:** Specification of desired behavior is often a challenging task to accomplish objectively in the absence of ground truth data during road tests. A common metric used to evaluate ADAS systems in real-world observations [17] counts the number of human interventions per meter. This metric is based on the number of times the human safety driver feels the need to take over control in the event of a possibly dangerous scenario. Analogous to this, the most objective metric to evaluate the real performance of the model is based on the *number of mistakes per meter* during road tests. The mistakes counted during testing include jerky wheel motion, vehicle drift from the centerline, and touching or crashing into obstacles.

4. **Scalability:** Models designed to work in a subset of conditions may not be scalable outside of this subset. The behavior cloning model is applied on a scaled RC vehicle with only throttle and steering commands for vehicle control. The model predictions (or even the model itself) are not linearizable to full-scale vehicles as they do not account for the non-linear vehicle dynamics, wheel slip, and unmodelled vibrations. Generalized training data that applies to diverse conditions may cause an over-approximated model that performs poorly in complex scenarios. Addressing this is beyond the scope of this paper, which seeks to verify the algorithm only in a scaled vehicle context.

5. **Standards for training data:** Given the challenges in scalability and environmental modeling, performance is suitably improved by trimming the training data for a given scenario. For instance, if the vehicle is expected to be driving past 7 pm in the winter months, it may be suitable for the model to draw its weights from

training data accumulated in those months at night-time. This selection is demonstrated in the paper, where the model retrained with training data for that specific environment is chosen for testing in that same environment.

This exploration of behavior cloning was done in an academic context, to promote the application-based understanding of the value and limitations of deep learning frameworks. In some of our group's past efforts, we have examined the use of end-to-end learning for achieving limited sets of autonomy assists (e.g. parallel parking). Based on the input images from a rear camera on the vehicle, a convolutional neural network (CNN) is trained to automatically output the steering and velocity commands for controlling the vehicle. These efforts were also carried out with F1tenth cars[19].

Much of the work in this paper builds on the efforts of the open-source robotics community and popular implementations designed and deployed by NVIDIA (Jetracer), Amazon (Deepracer), and DIY Robotics (Donkeycar). The makers of the Donkeycar, in particular, provide an easy-to-use deep learning software stack that provides GUI based software encapsulation for deploying the behavior cloning CNN. This GUI was used to run most of the experiments in this paper, but the software encapsulation became cumbersome when attempting to perform model evaluations. The same model was later implemented in Keras/TensorFlow and used to complete the experiments, especially in simulation. However, explicit cross-comparisons of the two model implementations were not performed given they possess the same underlying model structure.

## Implementation Framework

Behavior cloning is a form of supervised imitation learning whose main motivation is to build a neural-network model of the behavior of a human when performing a complex skill. The human's actions are recorded along with synchronized video-feeds as the skilled activity is being performed. A log of these records is used as input to a learning program that outputs a set of rules that reproduce the skilled behavior. In particular, the emphasis in the training is on the usage of machine learning which minimizes the dependence on prior knowledge of the environment (e.g. features such as lanes, etc.) relying instead on end-to-end learning.

Figure 2 gives an overview of how the end-to-end learning framework in Bojarski et al [4] collects the data from three different cameras along with steering inputs given by a human (our implementation uses a single camera). The training data maps every image with vehicle steering input during that frame. These training images were also augmented to diversify the data set.

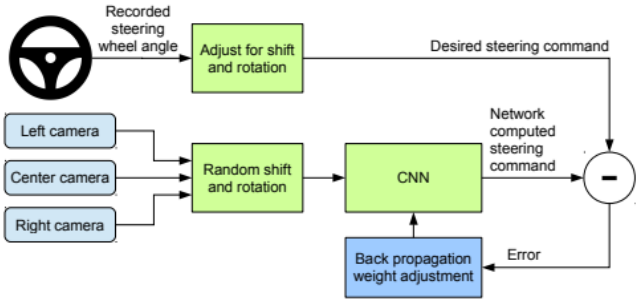


*Figure 2: Nvidia's end-to-end learning framework[4]*

Our work focused on implementing and categorically evaluating behavior cloning for predicting steering angle by giving a single-color camera image input to enable a scaled F1tenth vehicle to perform lane-keeping and obstacle avoidance. The model was tested within the ROS framework with software-in-the-loop (in GazeboSim) and with hardware in the loop on five test scenarios in the real world.

**Platform:**

We initially investigated available software stacks on scaled vehicles and came across the three most popular platforms viz. Amazon Deep Racer, Donkeycar, and Nvidia Jetracer[20]. A comparative analysis was made based on the hardware integration and software compatibility with the F1tenth scaled vehicle. The Donkeycar framework offered many advantages including the ability to implement end-to-end learning on a single board computer (Raspberry Pi), training neural-nets using GPU server, and trading off simulated vs real-data. To this end, we chose the Donkey car framework as a viable platform for our implementation.

| | **Donkey Car** | **Jet racer** | **Deep racer** |
|---|---|---|---|
| Supported by | DIY Robotics | NVIDIA | Amazon |
| Project started | 2016 | 2019 | 2006 |
| SBC hardware | RPi, Nvidia Nano | Nvidia Nano | Intel Atom |
| Deep learning software | TensorFlow | Pytorch | TensorFlow, Pytorch |
| UI | UNIX Shell, python | Jupyter Notebook | AWS CLI |
| Training | GPU server | On car | GPU server |
| Vehicle platform support | Many | Tamiya TT02 and Latrax Rally | AWS deep racer |
| Simulation platform available | Yes | No | Yes |
| Cost | $200 | $600 | $400 |

*Table 1: Platform selection*

The F1/10th platform created by O'Kelly [21] from the University of Pennsylvania serves as our primary experimental platform. The main chassis of the vehicle is a 1/10$^{th}$ scale TRAXXAS Ford Fiesta ST. Digital wheel velocity command inputs are provided to a stock Electronic Speed Controls (ESC) which provides the requisite current to power a high RPM DC motor driving the rear wheels through a differential. The second motor is a servo motor at the front for steering. The main computational board is the Nvidia Jetson TX2 running Ubuntu 18.04 installed with ROS Melodic along with the orbitty carrier board and an active cooling installed.

The Donkey Car software stack deployment serves on either of the two CPUs a) Jetson Nano b) Raspberry Pi. We used Raspberry pi 4b as a CPU to drive the car while the Jetson TX2 was used for CNN training onboard the vehicle. The vehicle is additionally equipped with PCA 9685 Servo driver to control the steering motor. Figure 3 shows the mounting of different hardware components on the F1/10$^{th}$ vehicle.

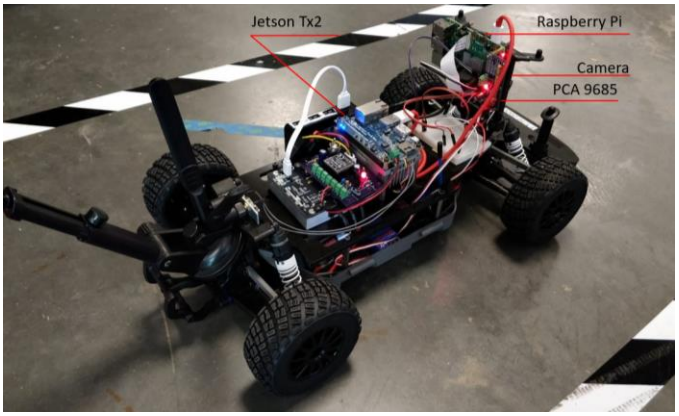


*Figure 3: F1/10th scaled vehicle hardware*

**Model:**

The Donkeycar and Jetracer implement variations of the machine learning model described by Bojarski et al [4]. This model has a track record of proven performance[22, 23]. The network architecture consists of 9 layers, including a normalization layer, 5 convolutional layers, and 3 fully connected layers. The system was trained using real-world data collected in five different test tracks and also on the simulator. Figure 4 shows the CNN architecture. Since the Raspberry Pi is not computationally very powerful, the augmented data was transferred to an Nvidia Jetson TX2 to train the convolutional neural networks. As seen in figure 4, the images are inputted into the CNN to produce a predicted steering output which is compared with the desired steering command, and the difference is minimized by adjusting weights through backpropagation. Training the model encounters the issue of overfitting and underfitting depending on the sample's ratio chosen for testing and validation of our training process. We halted the training process when there is not much significant change in the validation accuracy from epoch to epoch. This way, we made sure our model is not overfitting the samples chosen both in the simulator and also while training on the five different tracks

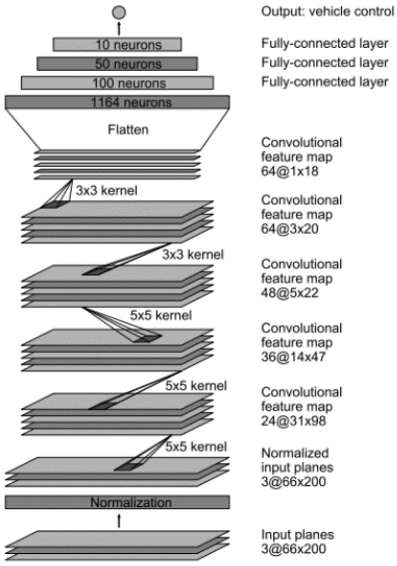


*Figure 4: CNN Architecture[4]*

**Workflow**

In this project, we mapped the single front-facing camera of the F1tenth car to the steering input and throttle values while driving the vehicle. This information was captured using the onboard camera/electronics of the F1tenth car and training was done offline. Finally, a trained network was deployed for mapping inferences throttle and steering values based on camera inputs on the hardware. A YouTube Video of the key steps in this process is uploaded here (https://www.youtube.com/watch?v=4NuUonLs3Bs).

**Training data:**

The Donkey car package has an Interactive UI that made the initial data collection data easy and hassle-free. Upon migration to Keras/TensorFlow implementation (simulator), the training data was collected as rosbags by teleoperating the vehicle around the chosen environment to collect the image, steering, and throttle inputs. CNN maps the input image to output by detecting important features from the image (outline/border of lanes).

The data was gathered by teleoperating the RC car using a Bluetooth joystick around the track for about 10 to 15 laps for different cases. The dataset for training needed at least 10000 images, generated by dividing the training session into four parts:

i) Driving slowly and precisely at the center of the lane. This consists of 30% of the data which gives the neural networks chance to observe the track from different angles.
ii) Driving slowly with small oscillations about the centerline. This consists of 30% of the data. This teaches neural networks how to correct back to the center.
iii) Driving like we normally drive – with a little bit of speeding. This consists of 20% of the data.
iv) Driving with oscillations by bouncing back and forth at the lane extremes to help neural networks to understand the lane extremes. This consists of about 20% of the data.

Figure 5 gives an overview of how the original system collects the training data. Our model slightly modifies the input layer to account for data from only one camera.

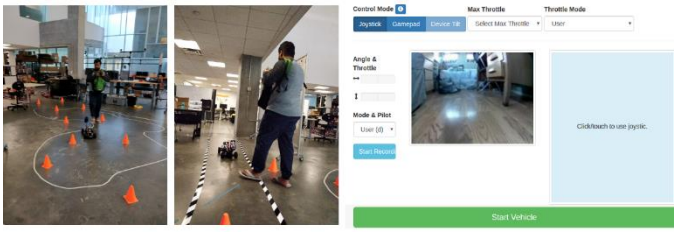


*Figure 5: Training data collection with Donkey car software*

The procedure of training the data even in a simulator is no different to the hardware data collection. The ROS based framework stores and collects the data in rosbags along with images and steering and throttle inputs were given by the human.

Data Preparation: The collected datasets were split into 1GB tubs and each tub contains the timestamped JSON files with the path of the images collected along with the steering and throttle inputs provided. A sample of how the data is stored from our data collected can be seen in Table 2

| Camera | Steering | Throttle |
|---|---|---|
| 2020_4_20_18_24_281 | 0.0 | 12 |
| 2020_4_20_18_24_340 | 0.0 | 12 |
| 2020_4_20_18_24_478 | 0.0 | 12 |
| 2020_4_20_18_24_513 | 0.0 | 12 |
| 2020_4_20_18_24_632 | 0.0 | 12 |

*Table 2: Data Preparation*

Data augmentation techniques (crop, resize, and flip the image) were used to multiply the input in a similar approach to the one used in the Donkey car backbone to desensitize the neural network to lighting and image-formation parameters. Figure 6 shows an original image and augmented image which is achieved by flipping the image horizontally. For such cases, we also flipped the steering input.

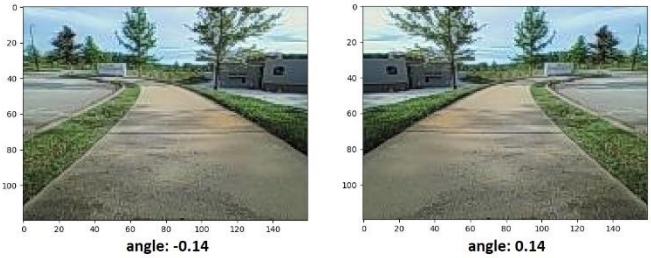


*Figure 6: Data Augmentation*

The collected images and the augmented images were processed from RGB to YUV using the OpenCV library as the YUV color-spaces are more efficient with CNN and reduces the bandwidth of processing. Figure 7 shows an original RGB image and its converted YUV image

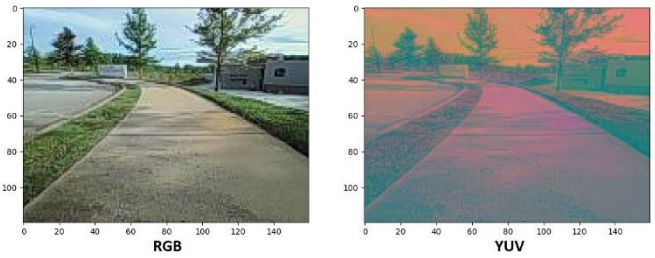


*Figure 7: Data Processing RGB to YUV*

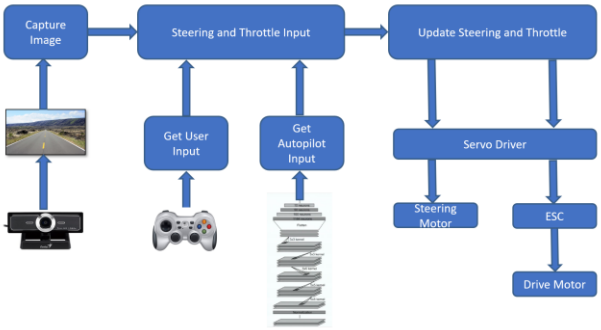


*Figure 8: Inferencing Deployment Architecture*

**Test Scenarios**

We sought to validate the performance and limitation of the Nvidia CNN by testing it on several test scenarios with wide-ranging environmental conditions.

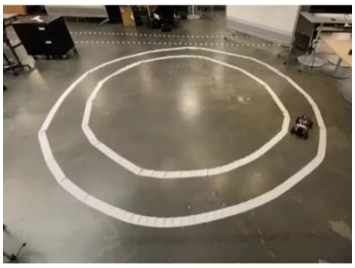

*Figure 9: Indoor, fluorescent lighting*

Indoor, fluorescent lighting: This test scenario consisted of a circular track made of white lanes inside our OpenCAV lab space at the International Center for Automotive Research (ICAR). This setup was to ensure that the model is working as expected with all the hardware and software working in sync. The training consisted of driving it for 10 laps with varying steering input. i.e. not a constant steering input. As this test was our preliminary test to ensure our implementation and deploy it on outdoor environments, we ran this with constant velocity.

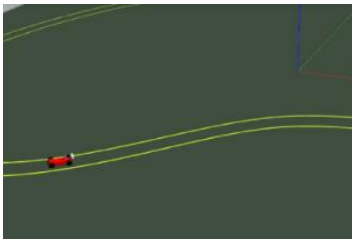

*Figure 10: Simulation track*

Simulation: The simulation worlds were created in Gazebo by modifying an open-source Ackerman-steered scaled vehicle model[24]. The training and testing algorithms were taken from the implementation of [4] by Wil Selby[25].

Outdoor, daytime, pedestrian track: The major intention of this testing scenario was to evaluate our training data standards. This test includes collecting and training our data on a pedestrian track with green grass on both sides in moderate lighting conditions (after sunset) and testing on a different pedestrian track with similar to slightly varying features.

*Figure 11: Pedestrian track*

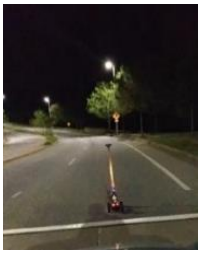

*Figure 12: On-road*

Outdoor, night-time, on-road: To perform this test, the camera height had to be modified to mimic the viewpoint of a full-scale vehicle. The camera was positioned at 1.2m high from the ground to capture full lane width images as the lane markings on the road are wide compared with the scaled vehicle. The vehicle was equipped with the LEDs and for caution. The training was conducted on a 1-mile long road in the night light while going uphill and tested uphill and downhill.

Outdoor, daytime, scaled racetrack: This serves as our final testing scenario to perform lane-keeping and obstacle avoidance. To perform our desired tasks, we created a scaled version of the Melbourne F1 racing track with yellow markings at the center to define the center of the track. We arranged cones on the racing track to serve as obstacles. While training the cones were stationary however while testing the position of the cones was constantly changed

*Figure 13: Racetrack*

**Evaluation Metrics**

Model Mean Squared Error: Model Mean Square Error (MMSE) is the most commonly used regression loss function. MMSE is the sum of squared distances between our target variable and predicted values. The larger the MMSE more the data values are widely spread around its central moment (mean). It is the preferred and/or desired choice for model evaluation as it shows if data values are closely dispersed around its central moment (mean). It reflects on the distribution of the data values - if is centralized, not skewed, and most of all, that it has small errors.

Histogram of Predicted Error: To view the prediction statistics, we have plotted the Histogram of Predicted error. Using this one can load a subset of the dataset and predict steering values for each of the images in the subset of data. By plotting such a graph, we can compare the predicted values to the known true values and visualize the error. If the error seems to be evenly distributed with the majority of the errors near 0 then the model can be presumed to be functioning well.

Predicted vs Actual Errors: This is represented as a scatter plot of the errors. In an ideally trained model, this scatter plot should coincide closely with the line along the diagonal. The quality of this fit is described by the R squared value:

$$R^2 = 1 - \frac{\sum(y - \hat{y})^2}{\sum(y - \bar{y})^2}$$

Where $y$ is the actual value of the steering angle, $\hat{y}$ is the predicted value from the model and $\bar{y}$ is the mean of the $y$ values. An $R^2$ value of 1 signifies a good fit, and 0 shows a poor fit.

Mistakes-per-meter metric: Each of these metrics relies on the availability of some ground truth data with which to compare the training model. They are, at best, preliminary checks on quality. Good performance may only be a signifier of how close the validation data is to training data. This more objective *mistake per meter* metric requires manual testing to measure vehicle performance. This metric was calculated by measuring the number of mistakes made by the vehicle for every kilometer of physical test data available.

## Results:

MMSE loss depicts the training loss and validation accuracy and how the validation data fit our training dataset. Overfitting or underfitting of the model is caused by various reasons such as the increased or decreased amount of data for training, the complexity of the model [26]. Overfitting is observed when a model trained on one track cannot be effectively used on another track as the model learns minute track details of the training track. Whereas underfitting means that the model learns track inadequately which results in low generalization. For our model to reduce overfitting we have added multiple dropout layers [27].

In most test cases except for Figures 17 and 18, the MMSE loss is constantly decreasing for both training and validation showing that the models are training well without overfitting. However, the validation loss seems to trend upwards for the outdoor racetrack and simulation test case indicating that the model might be overfitting.

The model training process provides some insight into the model's performance via some of the machine learning output performance parameters such as R-square and mean square error [28, 29]. However, to better understand how the model will perform, we can use the trained model to make predictions on some of our sample data and view the results. We can compare the predicted values to the known true values and visualize the error. These results are plotted as Histogram of Predicted Error and as scatter plots and are discussed below.

When the Histogram of Predicted Error is evenly distributed with the majority of the errors near to 0 value shows that the model is evenly trained for steering. This is seen in figures 14, 17, and 18, for the indoor, outdoor, daytime, and simulation test cases.

The scatter plots however visually show proper spreads for only the indoor and simulation test cases and this is verified by their $R^2$ values.

Given the slightly conflicting results from the evaluation metrics, it is apparent that only the indoor and simulation environment might be expected to perform well. This is confirmed by the *mistakes-per-meter* metric which was 0 for both test cases.

In the pedestrian track test case although the $R^2$ 0.7499 and the histogram spread was not even, the vehicle did not make mistakes when there was green grass on the sides of the walkway (which were represented in the training data) however on the sections where the grass was brown, or no grass was present, the *mistakes-per-meter* was 2.

In the on-road, night test, the MMSE showed a small degree of overfitting possibly due to the unchanging nature of the road (curves were small compared to the size of the vehicle). The HPS spread was even and $R^2$ 0.6842. The mistakes-per-meter metric was 1 in this test case.

On the outdoor-daytime test case, the MMSE showed some overfitting and HPE has a good spread. On testing the vehicle collided with an obstacle placed at a particular location every time. This was identified due to the sun being at 180º to the obstacle increased the object-reflectivity and difficult to be read in the camera frame. The complete demonstration of our final efforts can be seen in this link: https://youtu.be/KiZOjB10wDU

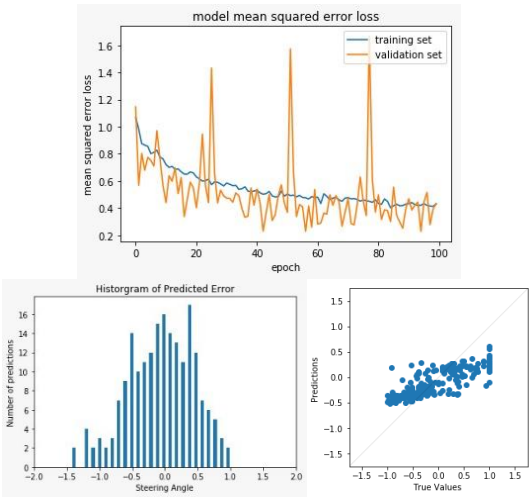


*Figure 14: MMSE, HPE & SPE for test 1: Indoor*

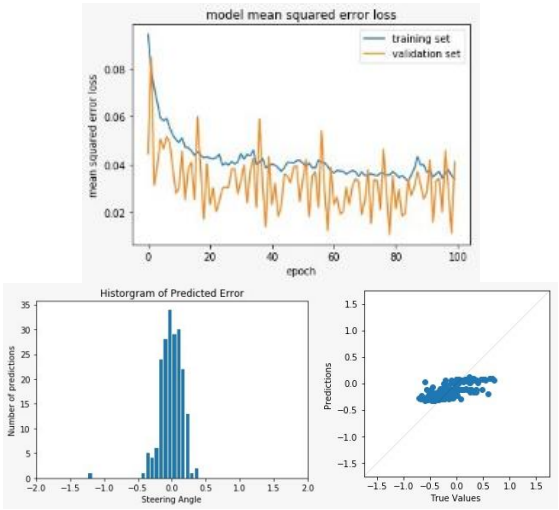


*Figure 15: MMSE, HPE & SPE for test 2: Outdoor, pedestrian walk,*

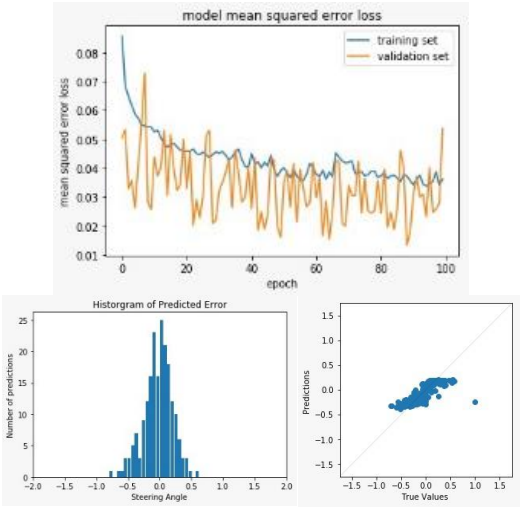


*Figure 16: MMSE, HPE & SPE for test 3: Outdoor, on-road, nighttime*

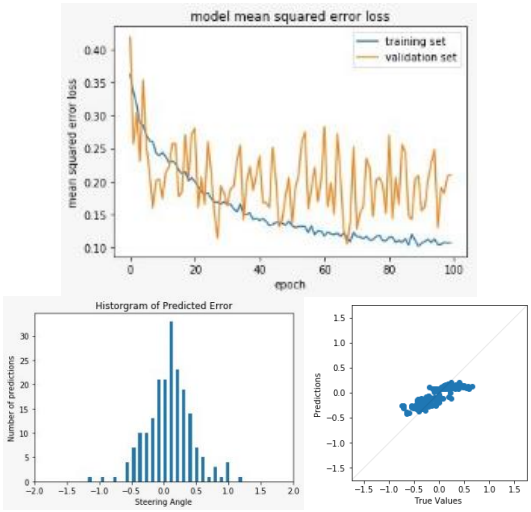


*Figure 17: MMSE, HPE & SPE for test 4: Outdoor, race track, daytime*

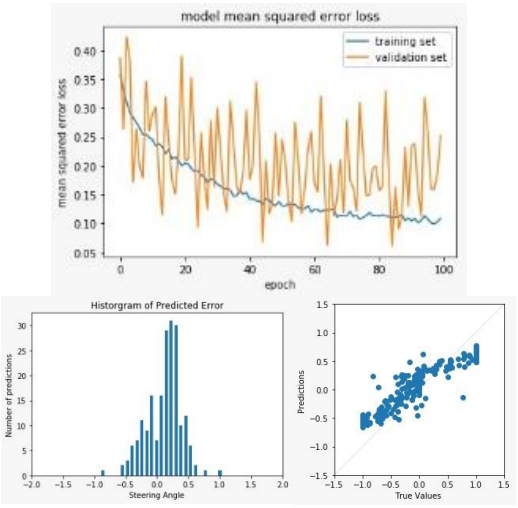


*Figure 18: MMSE, HPE & SPE for test 5: simulation*

| Environment | No of Samples collected | Epoch | Loss | Mistakes per Meter |
|---|---|---|---|---|
| Indoor lab | 12566 | 34 | 0.025 | Nil |
| Pedestrian track | 22324 | 41 | 0.0075 | Nil (Green grass on side)<br>2 (Brown grass on sides) |
| On-road (night) | 24232 | 50 | 0.0042 | 1 |
| Scaled racetrack | 22445 | 35 | 0.0024 | Nil |
| Simulation | 14562 | 42 | 0.072 | Nil |

*Table 3: Evaluating different test tracks*

| Environment | $R^2$ Value |
|---|---|
| Indoor lab | 0.7764 |
| Pedestrian track | 0.7499 |
| On-road (night) | 0.6842 |
| Scaled racetrack | 0.7821 |
| Simulation | 0.8125 |

*Table 4: $R^2$ for different test tracks*

## Discussion

In this paper, we explored the best available metrics to validate a popular algorithm (first proposed by Nvidia) that uses end-to-end learning for lane-keeping and static obstacle avoidance in different settings and conditions on a scaled Ackermann vehicle platform. We explored the Donkey car software platform and found that it provides a robust and versatile implementation that can be easily deployed on any scaled vehicle with small modifications. The deployment on a scaled F1tenth vehicle was intended to serve as an alternative for testing algorithms on full-scale vehicles. In doing so, the scaled vehicle deployment gives us a hardware-in-the-loop experience which is preferable to software-in-the-loop deployments (only simulation) while maintaining a safe and repeatable environment for an academic setting. Future work with considers upgrades by using the Nvidia Jetson Tx2 as the on-board computer for driving as well as training on the real track for faster frame rates. This is so that we can eventually implement an unsupervised (reinforcement) model to achieve the same task. As the application is restricted to lane-keeping and static obstacle avoidance in this project, future work can be training the model to avoid dynamic obstacles and detect the traffic signs. Future work can also include using a more capable scaled vehicle to mimic the dynamics of a full-scale vehicle like using a VESC to obtain motor performance graphs and enhanced mode transition like braking and ABS.

## Contact Information

Ankit Rajendrakumar Verma

3434 Laurens Road, Apt 823, Greenville, SC, 29607

ankit.rverma.mec12@iitbhu.ac.in

+1 956-507-0707

## Acknowledgments

The authors would like to acknowledge support of their colleagues from ARMLab at the Clemson University International Center for Automotive Research. We also acknowledge partial support for this work from the National Science Foundation via the National Robotics Initiative grant (CMMI- 1924721) and the Computing Community Research Infrastructure grant (CNS-1925500).



## References

[1] A. Kanervisto, J. Pussinen, and V. Hautamäki, "Benchmarking End-to-End Behavioural Cloning on Video Games," *arXiv preprint arXiv:2004.00981,* 2020.

[2] F. Codevilla, E. Santana, A. M. López, and A. Gaidon, "Exploring the limitations of behavior cloning for autonomous driving," in *Proceedings of the IEEE International Conference on Computer Vision*, 2019, pp. 9329-9338.

[3] W. Du and Y. Ji, "An Empirical Comparison on Imitation Learning and Reinforcement Learning for Paraphrase Generation," *arXiv preprint arXiv:1908.10835,* 2019.

[4] M. Bojarski *et al.*, "End to end learning for self-driving cars," *arXiv preprint arXiv:1604.07316,* 2016.

[5] A. Bulsara, A. Raman, S. Kamarajugadda, M. Schmid, and V. N. Krovi, "Obstacle Avoidance Using Model Predictive Control: An Implementation and Validation Study Using Scaled Vehicles," SAE Technical Paper, 0148-7191, 2020.

[6] A. Agnihotri, M. O'Kelly, R. Mangharam, and H. Abbas, "Teaching Autonomous Systems at 1/10th-scale," 2020.

[7] R. Ivanov, T. J. Carpenter, J. Weimer, R. Alur, G. J. Pappas, and I. Lee, "Case study: verifying the safety of an autonomous racing car with a neural network controller," in *Proceedings of the 23rd International Conference on Hybrid Systems: Computation and Control*, 2020, pp. 1-7.

[8] M. O'Kelly, H. Zheng, A. Jain, J. Auckley, K. Luong, and R. Mangharam, "TunerCar: A superoptimization toolchain for autonomous racing," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*, 2020: IEEE, pp. 5356-5362.

[9] F. Roza, "End-to-end learning, the (almost) every purpose ML method," March 30, 2019.

[10] Z. Chen and X. Huang, "End-to-end learning for lane keeping of self-driving cars," in *2017 IEEE Intelligent Vehicles Symposium (IV)*, 2017: IEEE, pp. 1856-1860.

[11] W. Schwarting, J. Alonso-Mora, and D. Rus, "Planning and decision-making for autonomous vehicles," *Annual Review of Control, Robotics, and Autonomous Systems,* 2018.

[12] H. J. Vishnukumar, B. Butting, C. Müller, and E. Sax, "Machine learning and deep neural network—Artificial intelligence core for lab and real-world test and validation for ADAS and autonomous vehicles: AI for efficient and quality test and validation," in *2017 Intelligent Systems Conference (IntelliSys)*, 2017: IEEE, pp. 714-721.

[13] S. Mozaffari, O. Y. Al-Jarrah, M. Dianati, P. Jennings, and A. Mouzakitis, "Deep Learning-Based Vehicle Behavior Prediction for Autonomous Driving Applications: A Review," *IEEE Transactions on Intelligent Transportation Systems,* 2020.

[14] D. Shapiro, "Accelerating the race to autonomous cars," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2016, pp. 415-415.

[15] Wayve, "Learning to Drive like a Human," 2019, 3rd April.

[16] A. Moujahid *et al.*, "Machine learning techniques in ADAS: A review," in *2018 International Conference on Advances in Computing and Communication Engineering (ICACCE)*, 2018: IEEE, pp. 235-242.

[17] N. Kalra and S. M. Paddock, "Driving to safety: How many miles of driving would it take to demonstrate autonomous vehicle reliability?," *Transportation Research Part A: Policy and Practice,* vol. 94, pp. 182-193, 2016.

[18] S. A. Seshia, D. Sadigh, and S. S. Sastry, "Towards verified artificial intelligence," *arXiv preprint arXiv:1606.08514,* 2016.
[19] R. Li, W. Wang, Y. Chen, S. Srinivasan, and V. N. Krovi, "An end-to-end fully automatic bay parking approach for autonomous vehicles," in *Dynamic Systems and Control Conference*, 2018, vol. 51906: American Society of Mechanical Engineers, p. V002T15A004.
[20] D. McCreary, "DonkeyCar vs. JetRacer," Jan 2.
[21] M. O'Kelly *et al.*, "F1/10: An open-source autonomous cyber-physical platform," *arXiv preprint arXiv:1901.08567,* 2019.
[22] Q. Zhang, T. Du, and C. Tian, "Self-driving scale car trained by deep reinforcement learning," *arXiv preprint arXiv:1909.03467,* 2019.
[23] A. Intisar, M. K. B. Islam, and J. Rahman, "A Deep Convolutional Neural Network Based Small Scale Test-bed for Autonomous Car," in *Proceedings of the International Conference on Computing Advancements*, 2020, pp. 1-5.
[24] A. CLEMSON, "F10_simulator," 2020.
[25] W. Selby, "RC Car End-to-end ML Model Development," Dec 9,2019.
[26] H. Jabbar and R. Z. Khan, "Methods to avoid over-fitting and under-fitting in supervised machine learning (comparative study)," *Computer Science, Communication and Instrumentation Devices,* pp. 163-172, 2015.
[27] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," *The journal of machine learning research,* vol. 15, no. 1, pp. 1929-1958, 2014.
[28] A. Swalin, "Choosing the right metric for evaluating machine learning models," ed: Retrieved from Medium: https://medium. com/usf-msds/choosing-the-rightmetric …, 2018.
[29] A. Mishra, "Metrics to evaluate your machine learning algorithm," *Towards Data Science,* 2018.