Journal of Korea Robotics Society (2019) 14(2):122-130
https://doi.org/10.7746/jkros.2019.14.2.122 ISSN: 1975-6291 / eISSN: 2287-3961

유니티 실시간 엔진과 End-to-End CNN 접근법을 이용한 자율주행차 학습환경

Autonomous-Driving Vehicle Learning Environments using Unity Real-time Engine and End-to-End CNN Approach

사비르 호사인[1] · 이 덕 진†
Sabir Hossain[1], Deok-Jin Lee†

Abstract: Collecting a rich but meaningful training data plays a key role in machine learning and deep learning researches for a self-driving vehicle. This paper introduces a detailed overview of existing open-source simulators which could be used for training self-driving vehicles. After reviewing the simulators, we propose a new effective approach to make a synthetic autonomous vehicle simulation platform suitable for learning and training artificial intelligence algorithms. Specially, we develop a synthetic simulator with various realistic situations and weather conditions which make the autonomous shuttle to learn more realistic situations and handle some unexpected events. The virtual environment is the mimics of the activity of a genuine shuttle vehicle on a physical world. Instead of doing the whole experiment of training in the real physical world, scenarios in 3D virtual worlds are made to calculate the parameters and training the model. From the simulator, the user can obtain data for the various situation and utilize it for the training purpose. Flexible options are available to choose sensors, monitor the output and implement any autonomous driving algorithm. Finally, we verify the effectiveness of the developed simulator by implementing an end-to-end CNN algorithm for training a self-driving shuttle.

Keywords: Autonomous Shuttle Vehicle, Artificial Intelligence, Virtual Environment, Behavior Learning

1. Introduction

Experimental setup to train an autonomous shuttle vehicle in a real physical world will be extremely troublesome. Since there are a lot of scientific parameters ought to be considered to make a training and test situation[1]. Moreover, the expense and the strategic troubles in case of the real-world experimental training is also a hindrance[2]. To able to learn properly, a machine needs to collect and train a large number of parameters in the model. In that case, the requirements of large data arise. Using a single vehicle, it is impossible to perform all the event needed to train a vehicle perfectly and gather data for those situations. Thus, alternate planning is badly needed where a virtual world can perform the continuous experiment of training a vehicle for autonomous car research. The simulator also gives an inherently safe environment for driving research[3]. There is no endangerment to the driver or other vehicle users or pedestrians in case of the simulator. However, training autonomous driving vehicle with reinforcement learning in the real environment involves non-affordable trial-and-error. It is more desirable to train in a virtual environment initially and then transfers to the real environment[4]. A brief survey about the

Received : Dec. 21. 2018; Revised : May. 13. 2019; Accepted : May. 23. 2019
※ This work was financially supported by the Kunsan National University's Long-term Overseas Research Program for Faculty Member in the year 2018.
1. Graduate Research Student, Center for Artificial Intelligence and Autonomous system, Mechanical Engineering, Kunsan National University, Gunsan, Korea (sabir@kunsan.ac.kr)
† Director, Associate Professor, Corresponding author: Center for Artificial Intelligence & Autonomous Systems, Mechanical Engineering, Kunsan National University, Gunsan, Korea (deokjlee@kunsan.ac.kr)

existing open-source vehicle simulators available online was presented in[5] where an efficient virtual simulating environment, CAIAS simulator for the self-driving car by utilizing the Unity real-engine was introduced, but there is no detailed description about specific deep-learning based training methods.

This paper extends the previous work done in a previous paper[5] by adding more hostile and realistic environments and implementing an end-to-end deep learning method to teach a self-driving car for controlling the motion of the vehicle.

In this paper, three major contributions will be highlighted. First, we discuss open-source simulators and their features. All the features in those simulators are explained in details and key factors that are necessary for training autonomous vehicle with application to deep learning methods are discussed. Second, we propose a new effective approach to make a virtual simulation environment suitable for teaching deep learning based agents which control the motion of a self-driving car. Specially, we developed a synthetic virtual simulator by using the real-engine from Unity which could reflect various realistic situations and weather conditions, making the autonomous vehicles to learn and understand more realistic situations and some unexpected events. Finally, we verify the effectiveness of the developed simulator by implementing an end-to-end CNN algorithm[6] for training a self-driving shuttle.

The remainder of the section is organized as follows. Section 2 provides an overview of the current open-source simulators used for self-driving car researches. In Section 3, an autonomous vehicle simulating environment for training the self-driving car is presented. Section 4 describes the detailed implementation of an end-to-end CNN based decision-making agent. Section 5 shows the verification of the performance of the proposed method.

2. Open-Source Simulators

There are many open-source vehicle simulators available these days, and each was built for a specific purpose and test. An overview of the simulators is represented below.

2.1 GAZEBO

The simulator is basically used for robot simulation. It is a well-designed simulator for rapidly test algorithms, design robots and train AI system using realistic scenarios. The Gazebo simulator offers the ability to accurately and efficiently simulate populations of robots in complex indoor and outdoor environments[7,8]. But, the problem is that the user needs to prepare the whole virtual world before training. Moreover, it is very hard to produce a dynamic 3D world in Gazebo[9].

2.2 TORCS

TORCS (The Open Racing Car Simulator)[10] is an open-source 3D car racing simulator which provides a realistic physical racing environment with a set of highly customizable API[10]. TORCS allows both automatic features for AI drivers and manual driving features using keyboard, mouse and wheel. But it is not so feasible to train an RL model in this environment since it does not provide proper visual API.

2.3 CARLA

An open-source simulator for autonomous driving research, CARLA[11] provides the development, training, and validation of autonomous urban driving systems. CARLA also provides open digital assets (urban layouts, buildings, vehicles) that were created for this purpose of dynamic real-life situations. The simulation platform supports flexible specification of sensor suites and environmental conditions[11].

2.4 AirSim

AirSim[12] simulator is an open-source, cross-platform for drone and car both, and supports physically and visually realistic simulations. It is developed as an Unreal plugin that can simply be dropped into any Unreal environment as well as unity environment which is developed recently. AirSim platform is arranged for AI research to experiment with deep learning, computer vision and reinforcement learning algorithms. It is possible to retrieve data and control vehicles in a platform independent way from the API[12].

3. Autonomous Vehicle Simulator

Unity is used to build a virtual simulating environment for training autonomous vehicles. This vehicle simulator is modified version of Udacity simulator[13]. Since Unity has a good user

interface and powerful interactive design module, this integrated platform is best for creating 3D simulator or other interactive contents such as virtual reconstructions or 3D animations in real time[14]. To design the simulation environment and for training purpose, we used a hardware system which consists of Intel Core i7 CPU, 64 GB RAM and GTX 1080 GPU.

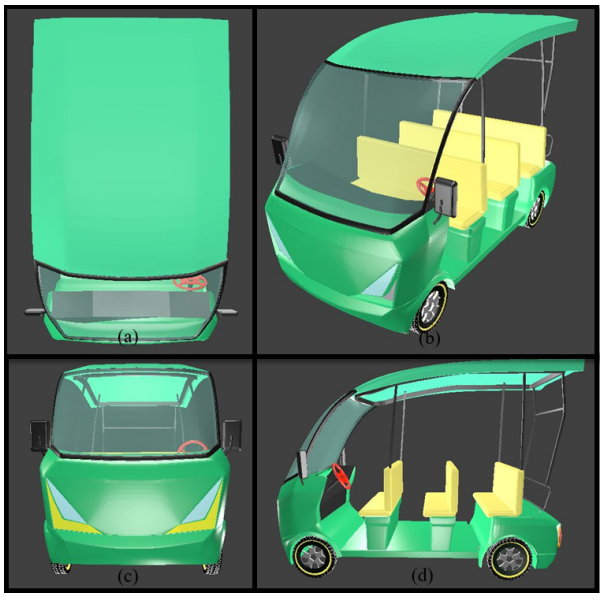
3.1 The process of importing Shuttle cart in Unity

An autonomous vehicle for training purpose is designed using Both SolidWorks and Blender software as a prototype of the real shuttle vehicle shown in [Fig. 1] and then is converted to fbx file and later imported environment. The server-client connection is initiated in the simulation environment using Socket. This SocketIO initiates the communication between the agent and the algorithm. Agent vehicle and environment work as a server and the model algorithm for training which works as a client. The simulator is made compatible for common operating systems like Windows, Linux and Mac OS with both 32-bit and 64-bit.

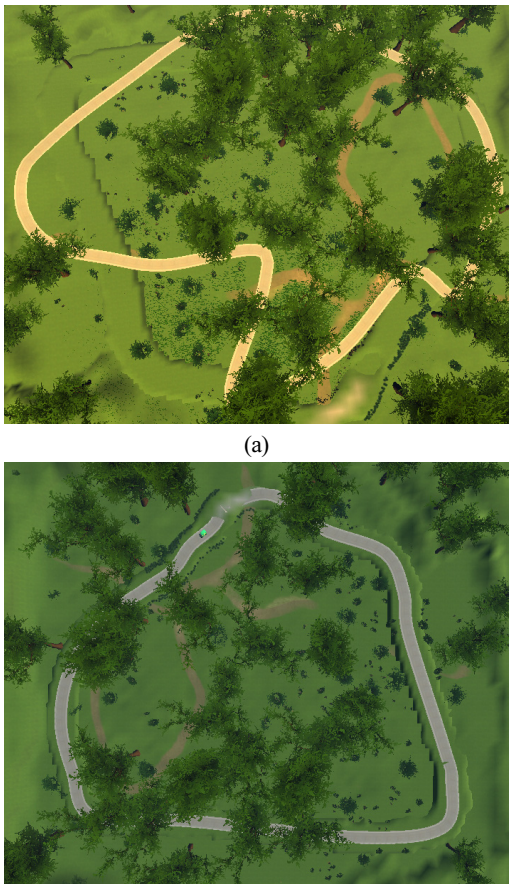
3.2 Environments Numbers

Two distinctive worlds with two different tracks, situations and weather are provided. One of the environment track is a forest environment. Here, the road is used in a loop, because in real life most of the time, a shuttle car is used in campus premises in a loop path. The same environment but different path track is used to verify the performance after training the model in the training path. Different path scenarios is a way to authentic that the simulator result can easily be transferred to the real physical world since the test ground path is totally different for the training path. Forest environment contains critical path turning, step slop, bumpy road condition, broken roads, sunny and gloomy weather. Sunny weather for training shown in [Fig. 2(a)] and gloomy or raining weather for the test shown in [Fig. 2(b)].

The user can choose preferable tracks to train the model. For training, path design, color is different [Fig. 2(a)] and the weather condition is sunny. For the test, road color and weather will be gloomy like [Fig. 2(b)]. The reason for converting one real-life scenario to convert to a virtual environment is to check and test some algorithm which can imitate the behavior from the image. So, having one scenario as like real life will give the advantage to evaluate those algorithms as well. Another track is randomly designed just to check all the steering angles and



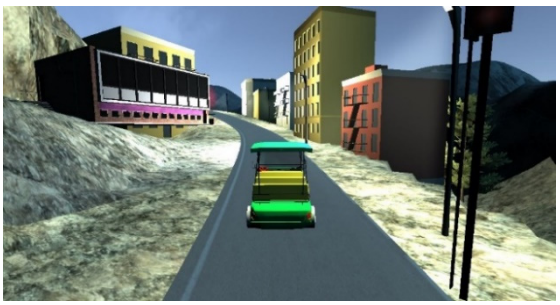
[Fig. 1] Blender and SolidWorks software both are used to design the prototype vehicle. (a) Top View (b) Isometric View (c) Front View (d) Side View



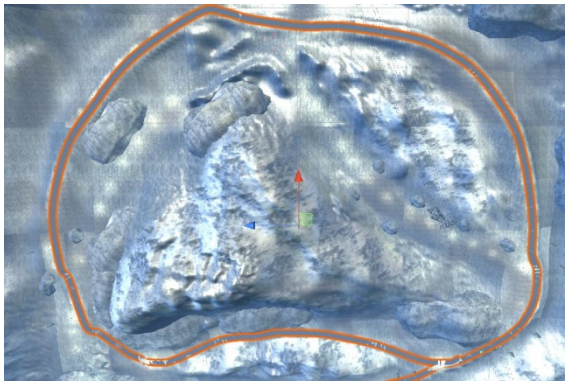
(a)

(b)

[Fig. 2] The forest environment for Train (a) and Test (b)



[Fig. 3] The second virtual environment on the simulator is the depiction of shuttle rail on a campus area



[Fig. 4] The map of the second track of campus rural area

speeds of the agent car produced during the simulation. This environment contains the countryside road, trees, random obstacles, grasses and driving lanes.

The environment is constructed using different critical conditions and scenarios to make the model from training more well-efficient. The simulator contains another campus rural map shown in [Fig. 3] & [Fig. 4] in which we tried to develop one environment mimics with the campus environment. Skybox used to make the environment look more realistic which is a wrapper around your entire scene that shows what the world looks like beyond your geometry[15].

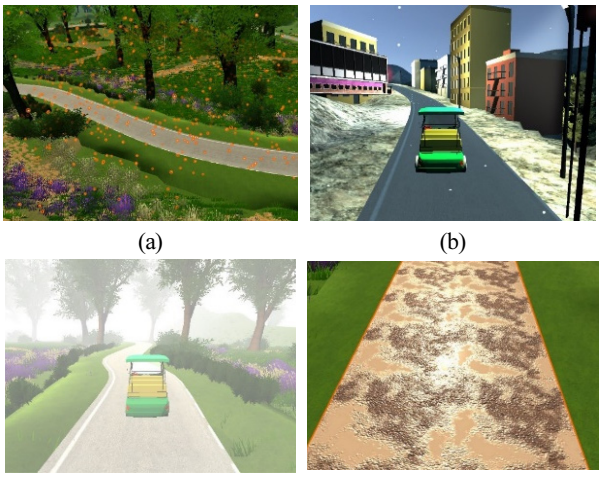
3.3. Weather Conditions and Different Road Layout

The tracks both contain rainy, hailing, fog, snowy, dry weather with both day and night time feature. To check further performance road layout and the color is changed.

To make it more realistic closely related to the physical world, some feature like a road bump, broken road, road slop, sun reflection in the road are included shown in [Fig. 5] and [Fig 6]. [Table 1] show the list of all simulation condition in the simulator.

(a) (b)

[Fig. 5] Rainy Weather in the Environments (a) Wet road Forest Environment, (b) Campus rural area



(a) (b)

(c) (d)

[Fig. 6] Hailing Weather Condition in the Environments (Marked yellow particles are Hail Stone) in (a), Snowy Weather Condition in the Environments (b), Foggy Weather Condition with less visibility in the Environments (c), Sun Reflection in the wet road in (d)

[Table 1] All tracks and Weathers

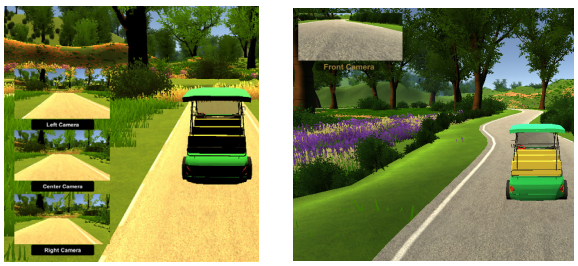
Environment		Weather
Forest Shuttle Environment	Training Track	All
	Testing Track	Rainy, Hailing, Foggy, Slippery Road, Day & Night
Campus Rural Area (Test)		Snow, Foggy, Night

3.4 Types of Sensors

Three types of sensors configuration are provided in the agent vehicle. The user can choose one or multiple sensors to generate results according to their model.

3.4.1 RGB Camera

A built-in script in Unity is used for the camera image. A camera is a device through which the player views the world like a simple camera. Here specifying pixel value is one of the



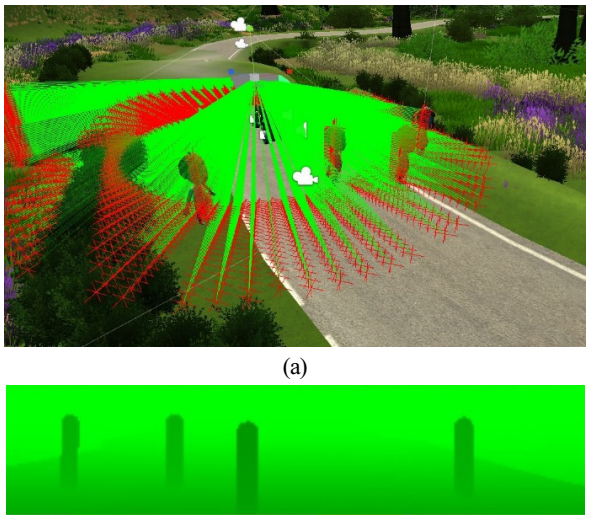
(a) (b)

[Fig. 7] Multiple camera (Left-Center-Right) train and receive data from the environment. (a) And a Single camera to take image input for the test (b)

important things. The single-camera is shown in [Fig. 7(b)] and multiple cameras showed in [Fig. 7(a)] is used as a sensor input. The idea of three multiple cameras is from behavior learning using convolutional network[6].

3.4.2 LiDAR Sensor

LiDAR sensor imitation and obtain the LiDAR data as a form of an image by casting each individual laser in the simulator. This implementation is intuitive and accurate. For each laser in the simulated LiDAR, a raycast is used to detect the distance. In the update loop, if the timer exceeds the limit, a list of raycast will be a trigger to gather distance information[5,16]. The result of all raycast is stored in a depth map image which is shown in [Fig. 8(b)]. The green image below the picture of the environment is the distance matrix correspond to the positions of all the obstacle in the environment, depicted in [Fig. 8(a)].



(a)

(b)

[Fig. 8] LiDAR sensor casting ray over different obstacles (a) and a depth map of the distance matrix corresponding to the positions of all the obstacle in the environment (b)

3.4.3 Depth Camera Sensor

A depth or motion vector texture can be generated from camera function in unity. This is a minimalistic G-buffer Texture that can be used for post-processing effects or to implement custom lighting models. Hence, it is usable as an output image like [Fig. 9] from the depth camera which produces the layer of the depth sensor completely based on the experiment layout[5,16].

3.5 User Interface of Simulator and Operation

The physics of shuttle vehicle is used for the driving simulation system is similar to shuttle vehicle dynamics. The functions and particle effects in the imported Unity were used to create trees, grass, object, and climate shown in [Fig. 5] and [Fig. 6] and lighting effects to simulate the real environment. The user can choose the favorable track and environment with respect to the sensor options. From the car user interference display, car speed and the angle are visible. The user can obtain the steering angle and speed during the training mode by using socket communication.

4. End-To-End CNN based Self-Driving

In this section, we will discuss the algorithms we used to verify the performance of the simulator. Also, the procedures for training a vehicle will be thoroughly described for algorithms like behavior learning[6,17] in this section. The methodology displays an end-to-end convolutional neural network (CNN)[6], trained with labeled data (steering angle, obstacles, roads, lanes) on different street conditions to avoid obstacles and autonomous navigation in an obscure condition. It includes image input from

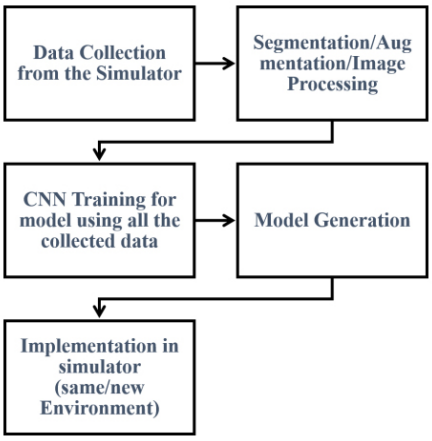
[Fig. 9] These figures show the output of the RGB camera and depth camera simultaneously

the simulation environment, image processing and transformation, color modification for lanes and array representation of images (NumPy array[18]). The labeled data and images are fed into the convolutional neural network, which generates a calculated steering angle and speed values as output. [Fig. 10] presents a flowchart for the steps related to autonomous driving in the methodology.

4.1 Data Collection and Preprocessing

The car is driven manually in simulated roads to gather the data. Three cameras (front left, front middle and front right) are connected to the car which allows recording the environment simultaneously record three other parameters: steering angle, speed and throttle synchronized with the recorded images.

Image cropping, Pre-processing[19], Random shadow and brightness algorithms are applied. The image segmentation[20] and augmentation are applied to increase performance output in the CNN model and reduce noise and unimportant features from the image.



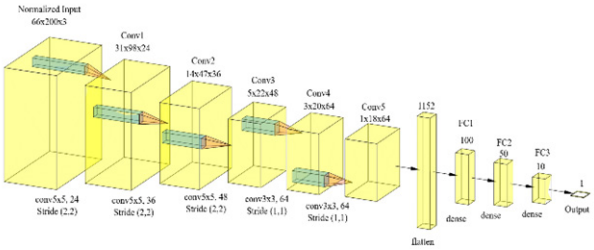
[Fig. 10] Step by Step Process of training and testing of Behavioral Cloning

4.2 Architecture of CNN Training

The depicted technique structure is a fully-connected convolutional neural network which can learn representations from input camera images and use it for autonomous driving. The shown [Fig. 11] display the end-to-end convolutional neural network used for autonomous driving in this work. This architecture is similar to the architecture of Nvidia CNN architecture model[6]. The network consists of 5 convolutional layers and 3 fully-connected layers. The convolutional layers are designed for features extraction. First 3 convolutional layers use 2x2 strides and next 2 convolutional layers use 1x1 strides. All the convolutional layers use 5x5 kernels. Fully-connected layers are designed to function as a steering controller which generate the steering control value output. [Table 2] presents a summary for the network with layers, kernel size, strides, number of filters (convolutional layers) and the number of neurons (fully-connected layers).

The CNN is trained with the weights to minimize the mean square error between predicted steering command and either steering commands from the human driver or the adjusted steering for off-center and rotated images.

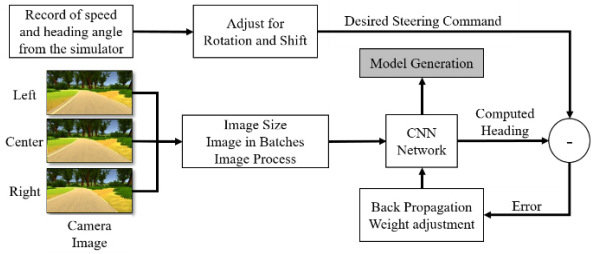
The reordered data from the simulation environment (record image) is labeled with road-type and driver's behavior (turning, lane switching, lane following, brake, etc.). Data is preprocessed



[Fig. 11] Fully Connected CNN Architecture

[Table 2] Behavioral learning configurations for Layers, Filters, Kernels, Strides and Neurons

Convolution Layer 1 : 5x5	Filter : 24	Stride : 2x2	Activation: ELU (Exponential Linear Unit)
Convolution Layer 2 : 5x5	Filter : 36	Stride : 2x2	Activation: ELU (Exponential Linear Unit)
Convolution Layer 3: 5x5	Filter : 48	Stride : 2x2	Activation: ELU (Exponential Linear Unit)
Convolution Layer 4: 3x3	Filter : 64	Stride : 1x1	Activation: ELU (Exponential Linear Unit)
Convolution Layer 5: 3x3	Filter : 64	Stride : 1x1	Activation: ELU (Exponential Linear Unit)
Fully Connected 1	Neuron : 100		Activation: ELU (Exponential Linear Unit)
Fully Connected 2	Neuron : 50		Activation: ELU (Exponential Linear Unit)
Fully Connected 3	Neuron : 10		Activation: ELU (Exponential Linear Unit)
Fully Connected	Neuron : 1 (Output)		Activation: ELU (Exponential Linear Unit)



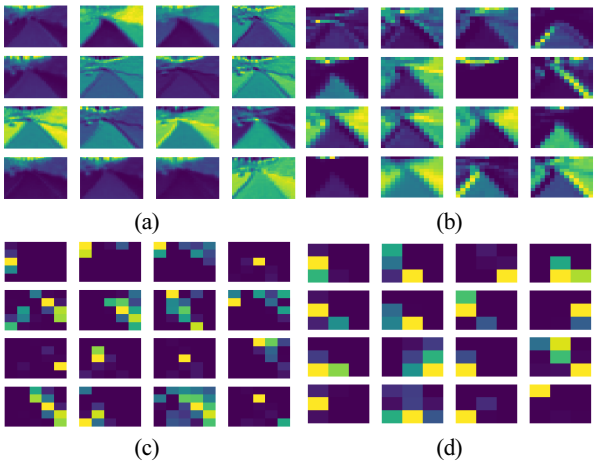
[Fig. 12] Training Strategy using behavior learning

and fed into the network for training with their corresponding labels. [Fig.12] illustrates the data collection and training for autonomous driving using end-to-end CNN.

5. Results

Convolutional Neural Network training is dependent on how it sees the images. Keras helps to visualize the CNN layers output[21] and to understand how an input image is processed inside the network. Following output in [Fig. 13] is generated on an image input by the CNN network. Keras feature map generates all the plots for the convolutional layer. From convolutional layer 1 & 2, the model can get features like edges, curves and lines. The features in convolutional layer 3 & 4 are a higher level of complex feature which is learned from previous output.

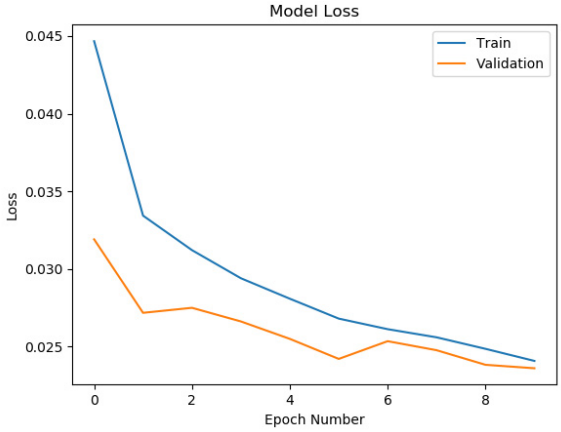
The network is trained with 10 epochs and 20000 samples per epochs. The learning rate for the network is in exponential form, α = 0.0001 and, a batch size of 40 is applied. The training loss and validation loss is calculated and visualized in run-time using



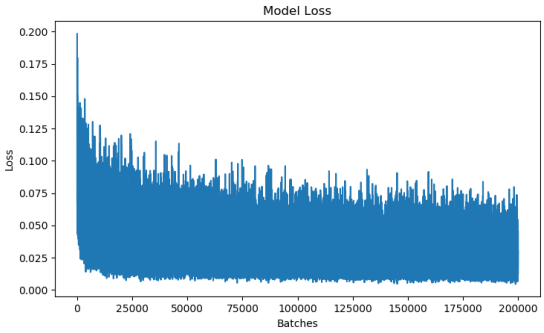
[Fig. 13] CNN Layer Output Visualization (a) Conv. 1 (b) Conv.2 (c) Conv.3 (d) Conv.4

Keras. History can be tracked in Keras while training is going on. Later, it is possible to use those data to plot the graph using matplotlib python library. [Fig. 14] presents the training accuracy and training loss and validation loss respectively. Since the loss of both training and validation reduced systematically, it can be said that the model is not over-fitted. [Fig. 15] plot shows the model loss after 200000 batches. As the training proceed, the model loss reduced. It indicates the model got better with more training data.

[Table 3] reports the number of times vehicle effectively finished test condition under two distinct scene and two different



[Fig. 14] The training loss after 10th Epoch



[Fig. 15] The training loss after 200000 batches

[Table 3] Qualitative Evaluation of performance

Scenarios	Behavior Learning	
	Forest Shuttle Track (Rain)	Campus Track (Night & Snow)
Straight	20	20
Left Turn	20	16
Right Turn	20	16
Hard Left	18	17
Hard Right	17	17
Bumpy Road	17	18
Obstacle Avoid	16	12

weather using the same model generated after training. The quantitative evaluation is performed in both tracks. One of them is Forest shuttle track in rainy weather another of them is campus track during the night and snowy weather. We have executed each of the conditions mentioned in [Table 3] 20 times using our trained model on the mentioned scenarios and collected the number of times it succeeded. Though the result may vary with a different model of the dataset. Since a good amount of dataset will help to generate a perfect model. In general, the execution of all strategies is working properly and the achievement rate is good. The test results from the performance verify the efficacy of the simulator. From the result, it was evident that end-to-end CNN works efficiently in a similar environment it was trained. Validating the model in a different environment gives less success rate in case of behavioral cloning method. So, it is worthwhile to implement behavioral cloning in shuttle car since there is less change in a feature of a shuttle car and its environment.

6. Conclusion

In this study, we discussed about various simulators and their supporting API. One of the main reason to choose this simulator and train especially for behavioral cloning is the data generation for various environment and weather is quite easier in this simulator. Recording data for training using three camera sensor is a key feature to train using Nvidia end-to-end CNN behavioral cloning model. Also, the socket interface made the communication easier between the code and simulator. We discussed the development process of a dynamic simulator which has two different worlds consisting of different weather condition and situations. The dataset from the synthetic environment made the model more robust for shuttle car environment. Since it is kind of supervised learning, the dataset from the diverse environment and weather can boost up the accuracy of the steering angle of the model. We learned about the training procedures and methods in the platform. We implemented behavioral learning algorithms and checked quantitative performance in the road. We found that in an unknown environment the prediction accuracy gets reduced. In case of shuttle car, this algorithm is beneficial, since the shuttle car has less change in feature in front of the camera sensor in its environment and very specific road lane for shuttle car.

Acknowledgment

This work was financially supported by the Kunsan National University's long-term Overseas Research Program for Faculty Member in the year 2018, and I would like to express special thanks.

References

[1] L. Meeden, G. McGraw, and D. Blank, "Emergent control and planning in an autonomous vehicle," *15th Annual Conference of the Cognitive Science Society*, 1993.

[2] P. Koopman and M. Wagner, "Challenges in autonomous vehicle testing and validation," *SAE International Journal of Transportation Safety*, vol. 4, no. 1, pp. 15-24, 2016, doi: 10.4271/2016-01-0128.

[3] E. Blana, "A survey of driving research simulators Around the World," *Institute of Transport Studies, University of Leeds*, Dec. 1996.

[4] X. Pan, Y. You, Z. Wang, and C. Lu, "Virtual to real reinforcement learning for autonomous driving," *British Machine Version Conference (BMVC)*, pp. 11.1-11.13, Sept., 2017, doi: 10.5244/C.31.11.

[5] S. Hossain, A. R. Fayjie, O. Doukhi, and D.-J. Lee, "CAIAS simulator: self-driving vehicle simulator for AI research," *International Conference on Intelligent Computing & Optimization*, pp. 187-195, 2018, doi: 10.1007/978-3-030-00979-3_19.

[6] M. Bojarski, D. Del Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. D. Jackel, M. Monfort, U. Muller, J. Zhang, Xin Zhang, Jake Zhao, and Karol Zieba, "End to end learning for self-driving cars," *arXiv:1604.07316 [cs.CV]*, 2016.

[7] Gazebo, [Online], http://gazebosim.org, Accessed: May 26, 2019.

[8] ROS.org, [Online], http://wiki.ros.org/gazebo, Accessed: May 26, 2019.

[9] T. Linner, A. Shrikathiresan, M. Vetrenko, B. Ellmann, and T. Bock, "Modeling and operating robotic environent using Gazebo/ROS," *28th international symposium on automation and robotics in construction (ISARC2011)*, pp. 957-962, Seoul, Korea, 2011, doi: 10.22260/ISARC2011/0177

[10] B. Wymann, C. Dimitrakakis, A. Sumner, E. Espié, and C. Guionneau, "Torcs, the open racing car simulator," *TORCS: The open racing car simulator*, Software available at http://torcs.sourceforge.net, Mar., 2015.

[11] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun, "CARLA: An open urban driving simulator," *arXiv:1711.03938 [cs.LG]*, 2017.

[12] S. Shah, D. Dey, C. Lovett, and A. Kapoor, "Airsim: High-fidelity visual and physical simulation for autonomous vehicles," *Field and service robotics*, pp. 621-635, 2018, doi: 10.1007/978-3-319-67361-5_40.

[13] Udacity, *A self-driving car simulator built with Unity*, [Online], https://github.com/udacity/self-driving-car-sim, Accessed: Dec. 14, 2018.

[14] V. De Luca, A. Meo, A. Mongelli, P. Vecchio, and L. T. De Paolis, "Development of a virtual simulator for microanastomosis: new opportunities and challenges," *International Conference on Augmented Reality, Virtual Reality and Computer Graphics*, pp. 65-81, 2016, doi: 10.1007/978-3-319-40651-0_6 .

[15] L. Ni, Q. Peng, L. Yu, and J. Wang, "The research and application of products virtual exhibition technology base on unity 3D," *Digital Technology and Application*, vol. 9, p. 36, 2010.

[16] Y. Wang, *Self-driving car simulation*, [Online], http://wangyangevan.weebly.com/lidar-simulation.html, Accessed: Feb. 16, 2018.

[17] F. Codevilla, M. Miiller, A. López, V. Koltun, and A. Dosovitskiy, "End-to-end driving via conditional imitation learning," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*, Brisbane, QLD, Australia, 2018, doi: 10.1109/icra.2018.8460487.

[18] S. v. d. Walt, S. C. Colbert, and G. Varoquaux, "The NumPy array: a structure for efficient numerical computation," *Computing in Science & Engineering*, vol. 13, no. 2, pp. 22-30, Mar.-Apr., 2011, doi: 10.1109/mcse.2011.37.

[19] A. Lorsakul and J. Suthakorn, "Traffic sign recognition using neural network on OpenCV: Toward intelligent vehicle/driver assistance system," *Department of Biomedical Engineering, Faculty of Engineering, Mahidol University*, Thailand, 2008.

[20] N. R. Pal and S. K. Pal, "A review on image segmentation techniques," *Pattern recognition*, vol. 26, no. 9, pp. 1277-1294, 1993, doi: 10.1016/0031-3203(93)90135-J.

[21] M. Hänggi, S. Moser, E. Pfaffhauser, and G. S. Moschytz, "Simulation and visualization of CNN dynamics," *International Journal of Bifurcation and Chaos*, vol. 9, no. 7, pp. 1237-1261, 1999, doi: 10.1142/s0218127499000882 .

Sabir Hossain

2015 Chittagong University of Engineering and Technology, Bangladesh. (B.Sc.)

2017~ Kunsan National University, Korea (Master of Sciences)

Interests: Autonomous Vehicles, Image Processing, Deep Learning.

Deok-Jin Lee

1996 Chonbuk National University , Korea (B.S.)

1999 Aerospace Engineering, Texas A&M University, USA. (M.S.)

2005 Aerospace Engineering, Texas A&M University, USA (Ph.D.)

2006-2007 Agency for Defense Development (ADD), Korea

2007-2010 Naval Postgraduate School (NPS), USA

Current: Associate Professor at School of Mechanical Convergence Systems Engineering, and director, Center for Artificial Intelligence & Autonomous Systems (CAIAS) in Kunsan National University, Korea.

Interests: Intelligent Autonomous Systems, Machine & Robotics Learning, Deep Reinforcement Learning, Sensor Fusion and Sensor Networks, Adaptive Estimation and Control, and Integrated Navigation and Localization