Self-Driving a Car in simulation through a CNN

Javier del Egido Sierra ✉
Javier.egido@edu.uah.es
Luis Miguel Bergasa Pascual ✉
luism.bergasa@uah.es
Eduardo Romera Carmena ✉
eduardo.romera@edu.uah.es
Carlos Gómez Huélamo ✉
carlos.gomezh@edu.uah.com
Javier Araluce Ruiz ✉
javier.araluce@edu.uah.es
Rafael Barea Navarro ✉
rafael.barea@uah.es

Universidad de Alcalá, Madrid, Spain

Abstract. This work presents a comparison between different Convolutional Neural Network models, testing its performance when it leads a self-driving car in a simulated environment. To do so, driving data has been obtained manually driving the simulator as ground truth and different network models with diverse complexity levels has been created and trained with the data previously obtained using end-to-end deep learning techniques. Once this CNNs are trained, they are tested in the driving simulator, checking their ability of minimizing the car distance to the center of the lane, its heading error and its RMSE. The neural networks will be evaluated according to these parameters. Finally, conclusions will be drawn about the performance of the different models according to the parameters mentioned before in order to find the optimum CNN for the developed application.

Keywords: Convolutional Neural Network (CNN), self-driving.

1 Introduction. State of the Art

Transport has been a fundamental pillar of the economical evolution history since the wheel invention was made five thousand years ago. Many improvements has been made through all these years, reaching the current vehicle which allows people and goods to cover long distances without effort.

Nevertheless, many lives are lost every year due to car accidents, which are mainly produced by human factors such as fatigue, distractions or imprudences.

To solve these problems, driving aids have been implemented in new cars. These systems alerts the driver when the vehicle crosses road lines or automatically brakes the vehicle when an obstacle is on the road.

Depending on technologies implemented in the vehicle, there are different self-driving levels according to Society of Automotive Engineers (SAE) [1]:

- Level 1. Without assistance. Traditional vehicle controlled by a human driver.
- Level 2. The vehicle can control longitudinal or lateral movements under certain conditions.
- Level 3. The vehicle can control longitudinal and lateral movements but cannot detect eventualities, so the human driver must stay in alert .
- Level 4. The vehicle does not need a human driver. It can move autonomously and acts when system failures are detected, but it only works under certain conditions.
- Level 5. The vehicle does not need a human driver. It can move autonomously under all conditions.

Nowadays the car is quickly evolving into autonomous machines with the ability of interact with the environment and solve driving situations without any problem, being able to drive following the rules as a human but avoiding the problems mentioned above.

1.1 Autonomous Systems

There are two main approaches to control an autonomous car, which will be introduced next.

Traditional perception and navigation techniques [2] The traditional way to control a self-driving vehicle is to implement a lot of code for perception techniques distinguishing between different classes of objects seen by diverse technologies such as RGB cameras, LIDAR, radar. This information feeds traditional navigation algorithms based on mapping, planning, low-level control and high-level control in charge with the decision making maneuvers. The methods need a big process capacity and expensive technologies that increase the costs.

End-to-End learning by using Neural Networks [3] The incipient way studied in this project, much more simpler, feeds a deep convolutional neural network which controls the car movements with just simple images and applying an end-to-end strategy. The code needed to implement this technology is easier to implement. The used neural network learns how to drive by taking information about manual driving by a human.

This paper researches into Artificial Neural Networks, and more specifically, into Convolutional Neural Networks and its architecture, to develop a new CNN with the capacity to control lateral and longitudinal movements of a vehicle in an open-source driving simulator replicating the human driving behavior.

1.2 Previous Developments

As mentioned before, End-to-End learning is the new way to control autonomous cars. Due to this, many recent tools have been developed to be able to train and test the CNNs ability to control an autonomous car. Some of them are mentioned next.

NVIDIA Self-Driving Car [3] This project trains a Convolutional Neural Network to analyze a single front-facing camera to control steering commands of a real car, monitoring how many times the driver has to take control of the car.

Europilot [4] Europilot is a platform that allows to control Euro Truck Simulator 2 (a driving simulator game) with an Artificial Neural Network programmed in Keras or Tensorflow.

Gazebo [5] Gazebo is a realistic city simulator created to train neural networks to control autonomous cars.

Udacity Self-Driving Car Simulator[6] This project, produced by Udacity, provides a racing driving simulator and tools to communicate the car and the Convolutional Neural Network.

1.3 Training the CNNs

Training data are collected by driving a car in an open-source driving simulator by a human, obtaining images from a front-facing camera and synchronizing steering angle and throttle values performed by the driver. Steering angle values are relative to the maximum (r/r_{max}), where r_{max} means the maximum steering value (25), oscillating between -1 when turning left and 1 when turning right, and throttle values oscillates between 1 (maximum acceleration) and -1 (maximum braking).

The dataset is augmented by horizontal flipping, changing steering angles sign and taking information from left and right cameras from the car, using them as a center image by modifying the steering value with a deviation factor, established as 0.2. Finally, training data is used in the same order as it was taken to correctly train the LSTM layers.

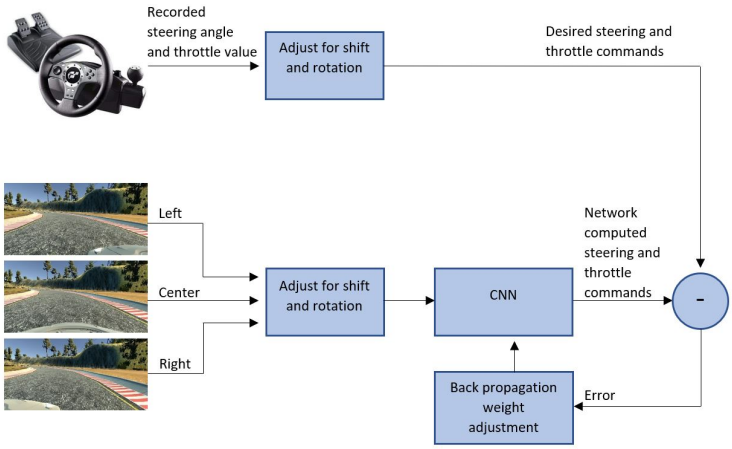

Fig. 1. Left, center and right images from driving simulator captured to train CNNs

When the network is being trained, single images are provided to the CNN as input, obtaining output values which are minimized regarding to output values from dataset created by a human driver through multiple epochs.

2 Network Architecture

The CNNs tested in this project are described, trained and tested using Keras, a high-level neural network API [7], developed as part of the research effort of project ONEIROS (Open-ended Neuro-Electronic Intelligent Robot Operating System) by Franois Chollet, a Google engineer.

2.1 TinyPilotNet

As baseline in our study we use the TinyPilotNet [8] network architecture, developed as a reduction from NVIDIA PilotNet CNN [3] used for self-driving a Car. TinyPilotNet network is composed by a 16x32x1 pixels image input, followed by two convolutional layers, a dropout layer and a flatten layer. The output of this architecture is formed by two fully connected layers that leads into a couple of neurons, each one of them dedicated to predict steering and throttle values respectively. The input image has a single channel formed by saturation channel from HSV color space.

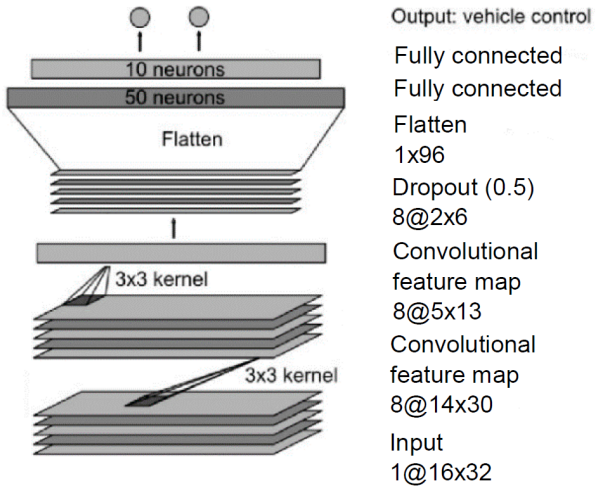

Fig. 2. TinyPilotNet architecture

2.2 Long Short-Term Memory layers

Long Short-Term Memory (LSTM) layers are included to TinyPilotNet architecture with the aim of improving the CNN driving performance and predict new values influenced by previous ones, and not just from the current input image [9], see Figure 3.

These layers are located at the end of the network, previous to the fully-connected layers. During the training, the dataset is used sequentially, not shuffled, in order to allow the network to relate the current output value with the previous ones. This feature makes that current predicted values are influenced by the previous ones, and they do not depend just on the current input image.

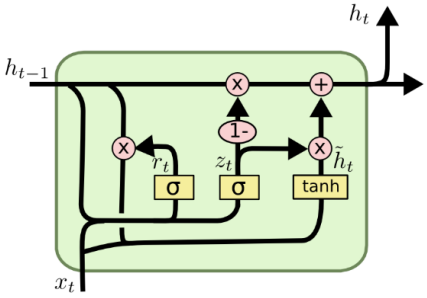

Fig. 3. LSTM cell

2.3 DeepestLSTM-TinyPilotNet

In order to improve the baseline architecture we propose a new network architecture, mainly formed by three 3x3 kernel convolution layers, combined with maxpooling layers, followed by three 5x5 convolutional LSTM layers and two fully connected layers (see Figure 4).

The LSTM layers produce memory effect, so steering angles and throttle values given by the CNN are influenced by the previous ones.

Convolutional layers extract information from the input image, and pooling layers reduce network to prevent overfitting.

The output is formed by two fully-connected neurons. Each one of them gives the information needed to control the car (steering angle and throttle value).

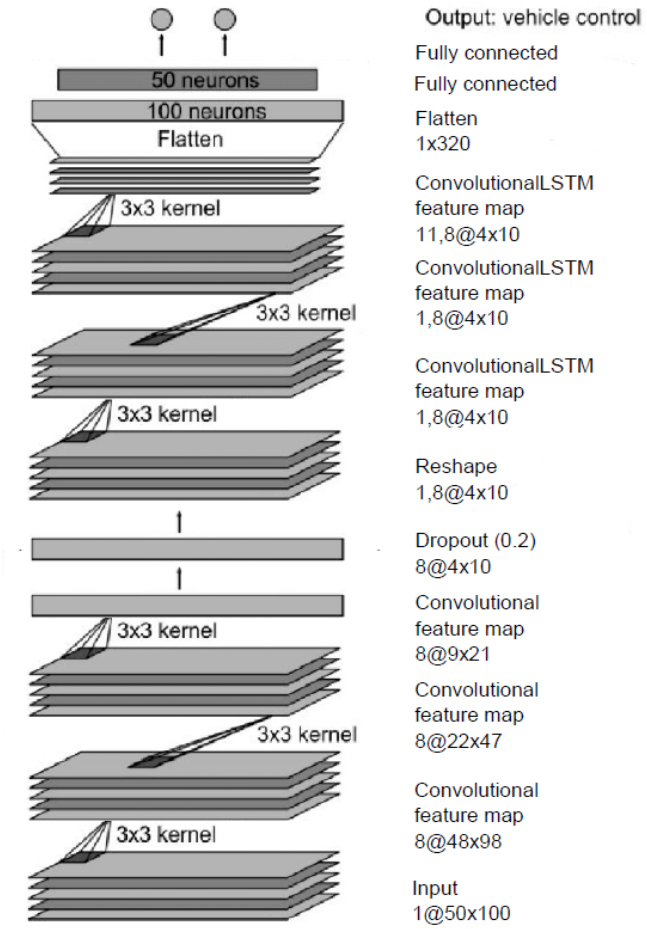

Fig. 4. DeepestLSTM-TinyPilotNet architecture

3 Open-source Driving Simulator

A dataset with a big amount of images and synchronized output values is needed to train the CNN. Due to the technical complication of doing this in a real environment, an open-source driving simulator is chosen to collect this data. The simulator used to obtain training data is Udacity Driving Simulator [6]. Figure 5 depicts a frame of the simulator.

Fig. 5. Udacity's Driving simulator

Some modifications are made in order to collect data from simulator in manual mode. The training data is registered in a Comma-Separated Value file (.csv). Figure 6 shows a fragment of the training dataset used to train the CNN and the types of data registered.

center	left	right	steering	throttle	brake	speed
C:\Users\jds96\OneDriv	C:\Users\jds96\OneDriv	C:\Users\jds96\OneDriv	-0.2122216	0.8741225	0	21.06124
C:\Users\jds96\OneDriv	C:\Users\jds96\OneDriv	C:\Users\jds96\OneDriv	-0.2122216	0.8741225	0	21.72626
C:\Users\jds96\OneDriv	C:\Users\jds96\OneDriv	C:\Users\jds96\OneDriv	-0.2122216	0.8741225	0	22.38842
C:\Users\jds96\OneDriv	C:\Users\jds96\OneDriv	C:\Users\jds96\OneDriv	-0.2122216	0.8741225	0	22.83872
C:\Users\jds96\OneDriv	C:\Users\jds96\OneDriv	C:\Users\jds96\OneDriv	-0.2122216	0.8741225	0	23.50869
C:\Users\jds96\OneDriv	C:\Users\jds96\OneDriv	C:\Users\jds96\OneDriv	-0.2122216	0.8741225	0	24.1702
C:\Users\jds96\OneDriv	C:\Users\jds96\OneDriv	C:\Users\jds96\OneDriv	-0.2122216	0.8741225	0	24.82263

Fig. 6. Training data captured from simulator

Once the CNN is trained, it is tested driving the car in the simulator, obtaining the new test metrics defined in chapter 4.2 by collecting driving data. This method allows to analyze the real driving ability instead of a frame-to-frame comparison with human data as ground-truth.

4 Testing Performance

In order to compare the performance of the CNN with other networks, a frame-to-frame comparison is made between CNN steering angle and throttle values and human values as ground truth using only the center camera images.

4.1 Frame-to-Frame Comparison. RMSE Metric

One of the most common test metrics is the Root-Mean Square Error (RMSE) [10] obtained with the difference between CNN steering and throttle predicted values and human given ones. Evaluating the CNN using this method allows to compare the performance with other published networks of the state of the art.

The equation followed is the following:

$$RMSE = \sqrt[2]{\frac{\sum_{t=1}^{T}(\hat{y} - y)^2}{T}}. \tag{1}$$

Driving data collected (y) by a human driver are needed to compare the CNN given values ($\hat{y}$) for each frame with the values used by the human driver. This parameter does not evaluate the ability of the network to use previous steering and throttle values to predict the new ones, so the LSTM layer does not make effect and appears underrated in comparison with other CNNs that do not use these kind of layers.

4.2 New Test Metrics

To solve the problem previously mentioned, new quality parameters have been proposed to quantify the performance of the network driving the vehicle. These parameters are measured with the information extracted from the simulator when the network is being tested.

To calculate these parameters, center points of the road -named as waypoints- are needed, separated 2 meters as we show in Figure 7.

Fig. 7. Waypoints of the road in simulator

Center of Road Deviation One of these parameters is the shifting from the center of the road, or center of road deviation. The lower this parameter is, the better the performance will be, because the car will drive in the center of the road instead of driving in the limits.

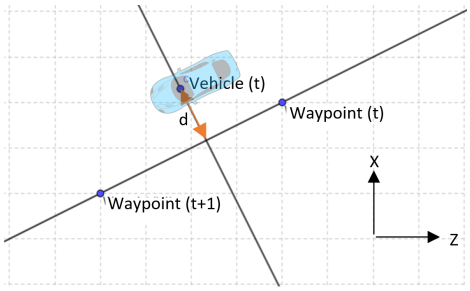

Fig. 8. Distance to the center of the road calculation

To calculate deviation, nearest waypoint is needed to calculate distance between vehicle and segment bounded by that waypoint and previous or next. This waypoint is obtained by using the equations (2) and (3).

Definition 1. *Assume A is the vehicle position and C is Waypoint position. Set:*

$$Waypoint = min\left\{norm(A, C)\right\}. \tag{2}$$

Once the nearest waypoint is known, center of road deviation is calculated by using the next equation:

Definition 2. *Assume A is*

$$A = \begin{bmatrix} Waypoint_x(t+1) & Waypoint_z(t+1) & 1 \\ Waypoint_x(t+1) & Waypoint_z(t+1) & 1 \\ Vehicle_x(t) & Vehicle_z(t) & 1 \end{bmatrix}$$

Set:

$$Distance = \frac{abs[norm(A, C)]}{norm(Waypoint(t+1) - Waypoint(t))} \tag{3}$$

Heading Angle The other parameter is the heading angle. Also, the lower this parameter is, the better the performance will be, because lower heading angle means softer driving, knowing better the direction to follow. To calculate heading angle equation (4) is applied.

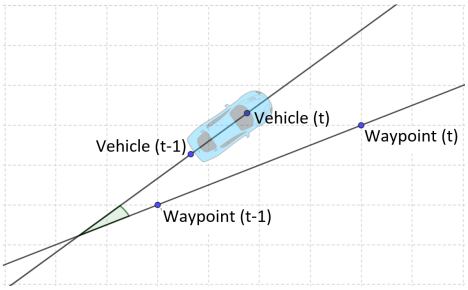

Fig. 9. Heading angle calculation

Definition 3. *Assume veh_vector = Vehicle(t) - Vehicle(t-1), and road_vector = Waypoint(t) - Waypoint(t-1). Set:*

$$heading\ angle = \text{acos}\left(min\left\{1, max\left\{-1, \frac{veh\_vector * road\_vector}{norm(veh\_vector) * norm(road\_vector)}\right\}\right\}\right) * \frac{180}{\pi} \quad (4)$$

5 Experimental Results

In order to determine the performance of the CNNs using the metrics previously established, different experiments are made.

To obtain RMSE parameter, a frame-to-frame comparison between CNN predited values and human given values driving in the simulator is made. RMSE results are displayed in Table 1. The dataset, composed by 17943 frames from center, left and right cameras, used for RMSE calculation is the same that was used for training the CNN.

Table 1. Obtained RMSE metric values.

CNN	RMSE
TinyPilotNet	0.0912475
DeepestLSTM_TinyPilotNet	0.116321

According to Table 1, TinyPilotNet should drive more efficiently than DeepestLSTM_TinyPilotNet due to TinyPilotNet RMSE values are lower than DeepestLSTM_TinyPilotNet ones, so predicted values are more accurate. But this comparison is made frame-to-frame, not feeding the CNN with a driving sequence, so LSTM layers cannot predict values according to the previous ones, so DeepestLSTM-TinyPilotNet hability is underrated in RMSE calculation.

Frame-to-frame comparison is shown graphically in Figures 10 and 11.

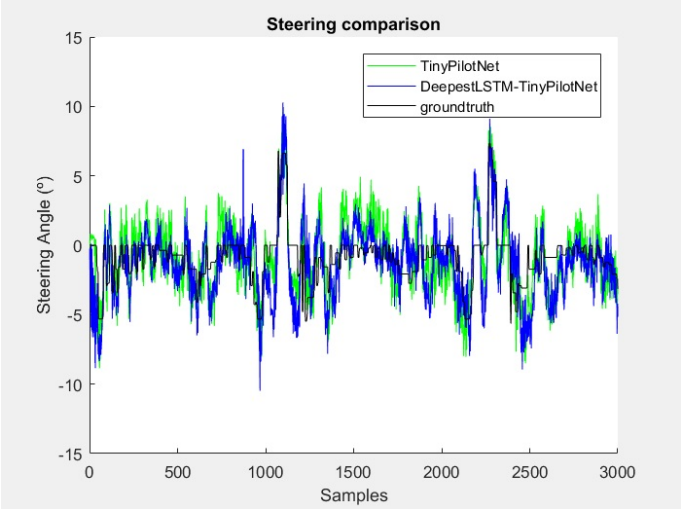

Fig. 10. Steering angles frame-to-frame comparison between DeepestLSTM_TinyPilotNet, TinyPilotNet and human ground-truth

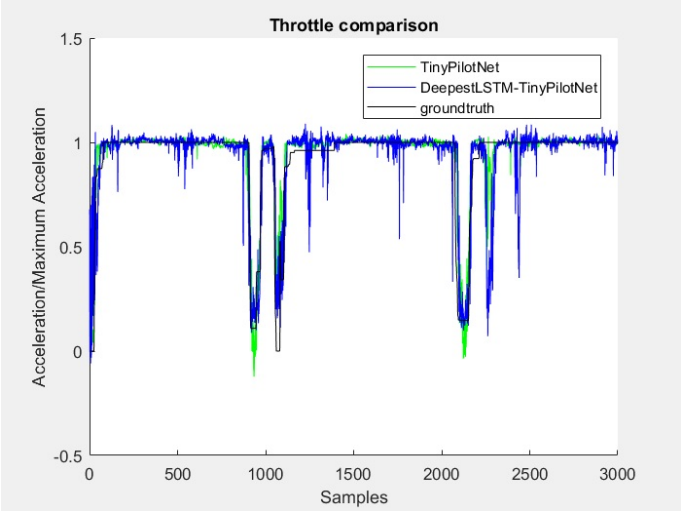

Fig. 11. Throttle frame-to-frame comparison between DeepestLSTM_TinyPilotNet, TinyPilotNet and human ground-truth

RMSE values obtained for the different CNNs can be compared to other networks from the state of the art, taking into account that TinyPilotNet and DeepestLSTM_TinyPilotNet were tested with the same dataset used to train them.

Table 2. RMSE obtained values compared with state of the art CNNs.

CNN	RMSE
TinyPilotNet[8]	0.0912475
DeepestLSTM_TinyPilotNet	0.116321
AlexNet[11]	0.1299
PilotNet[3]	0.1604
VGG-16[12]	0.0948

For the CNNs proposed in this paper RMSE results are very close to other state of the art networks such as PilotNet[3], AlexNet[11] and VGG-16[12], as can be seen in Table 2.

New test metrics values are obtained from driving data extracted from the simulator when the car is controlled by the CNN, using the equations (2), (3) and (4). Obtained metric values are displayed in Table 2.

Table 3. Obtained new test metrics values.

CNN	Mean distance	Max. distance	Mean heading angle	Max. heading angle
TinyPilotNet	1.07	7.17	3.38	44.91
DeepestLSTM_TinyPilotNet	0.72	5.00	2.15	26.35

As can be seen in Table 3, DeepestLSTM_TinyPilotNet is able to reduce the mean and maximum distance to the center of the road, which leads to safer driving, and also reduces mean and maximum heading angle values, which means the car follows the direction of the lane smoothly, without big changes.

Figure 12 shows the trajectories followed by the car in Udacity's driving simulator when the vehicle is leaded by TinyPilotNet and DeepestLSTM_TinyPilotNet

Figure 12 reaffirms what Table 3 quantifies, that DeepestLSTM_TinyPilotNet drives more directly and centered than TinyPilotNet.

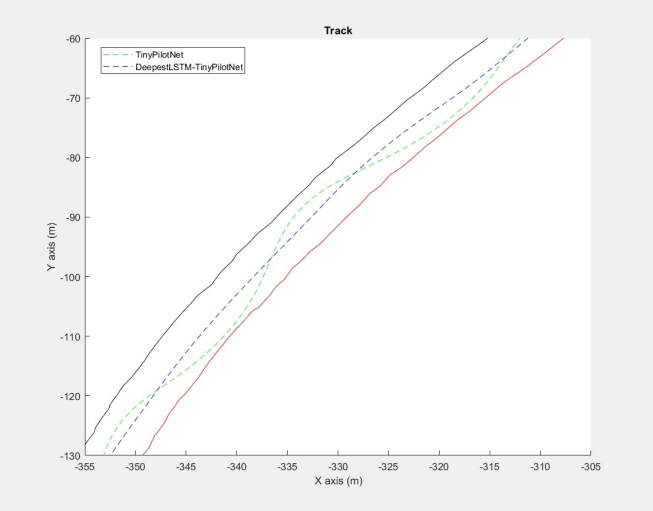

Fig. 12. Fragment of track showing TinyPilotNet and DeepestLSTM_TinyPilotNet car trajectories.

6 Conclusions

This paper describes a new testing metrics system that allows for a better knowledge of the CNN driving ability because it tests the network by doing the real task it was trained for.

The inclusion of Long-Short Term Memory (LSTM) layers in the network architecture produce a smoother driving by taking into account the previous steering and throttle values given by the CNN (a sequence), and not just a single moment. DeepestLSTM_TinyPilotNet drives nearest to the center of the road and straighter than the original TinyPilotNet, as can be seen in Figure 12.

7 Acknowledgment

This work has been partially funded by the Spanish MINECO/FEDER through the SmartElderlyCar project (TRA2015-70501-C2-1-R), the DGT through the SERMON project (SPIP2017-02305), and from the RoboCity2030-III-CM project (Robótica aplicada a la mejora de la calidad de vida de los ciudadanos, fase III; S2013/MIT-2748), funded by Programas de actividades I+D (CAM) and co-funded by EU Structural Funds.

References

1. Society of Automotive Engineers self-driving vehicles classification. https://www.cnet.com/roadshow/news/self-driving-car-guide-autonomous-explanation/
2. Vehicle Detection Using Machine Learning And Computer Vision. urlhttps://towardsdatascience.com/teaching-cars-to-see-vehicle-detection-using-machine-learning-and-computer-vision-54628888079a
3. Mariusz Bojarski et al.: End to End Learning for Self-Driving Cars. NVIDIA (2017). https://arxiv.org/pdf/1604.07316.pdf
4. Europilot. Marsauto. https://github.com/marsauto/europilot
5. Gazebo http://gazebosim.org/blog/car_sim
6. Udacity Self-Driving Car Simulator project https://github.com/udacity/self-driving-car-sim
7. Keras documentation. https://keras.io/
8. Yunming Shao (2017). End-to-End Learning for Self-Driving Cars. https://github.com/ymshao/End-to-End-Learning-for-Self-Driving-Cars
9. Convolutional, Long Short-Term Memory, fully connected deep neural networks. https://static.googleusercontent.com/media/research.google.com/es//pubs/archive/43455.pdf
10. Lu Chi and Yadong Mu, (2017). Deep Steering: Learning End-to-End Driving Model from Spatial and Temporal Visual Cues. https://arxiv.org/pdf/1708.03798.pdf
11. A. Krizhevsky, I. Sutskever, and G. E. Hinton (2012), Imagenet classification with deep convolutional neural networks, in Advances in neural information processing systems, pp. 10971105.
12. K. Simonyan and A. Zisserman,(Sep. 2014), Very Deep Convolutional Networks for Large-Scale Image Recognition, arXiv:1409.1556 [cs], arXiv: 1409.1556.