Real-Time Self-Driving Car Navigation Using Deep Neural Network

Truong-Dong Do, Minh-Thien Duong, Quoc-Vu Dang and My-Ha Le*

Abstract - ***In this paper, a monocular vision-based self-driving car prototype using Deep Neural Network on Raspberry Pi is proposed. Self-driving cars are one of the most increasing interests in recent years as the definitely developing relevant hardware and software technologies toward fully autonomous driving capability with no human intervention. Level-3/4 autonomous vehicles are potentially turning into a reality in near future. Convolutional Neural Networks (CNNs) have been shown to achieve significant performance in various perception and control tasks in comparison to other techniques in the latest years. The key factors behind these impressive results are their ability to learn millions of parameters using a large amount of labeled data. In this work, we concentrate on finding a model that directly maps raw input images to a predicted steering angle as output using a deep neural network. The technical contributions of this work are two-fold. First, the CNN model parameters were trained by using data collected from vehicle platform built with a 1/10 scale RC car, Raspberry Pi 3 Model B computer and front-facing camera. The training data were road images paired with the time-synchronized steering angle generated by manually driving. Second, road tests the model on Raspberry to drive itself in the outdoor environment around oval-shaped and 8-shaped with traffic sign lined track. The experimental results demonstrate the effectiveness and robustness of autopilot model in lane keeping task. Vehicle's top speed is about 5-6km/h in a wide variety of driving conditions, regardless of whether lane markings are present or not.***

Keywords: ***real-time navigation, self-driving car, deep neural networks, convolutional neural networks, embedded systems.***

I. INTRODUCTION

Self-driving cars have been one of the most promising prospects for Artificial Intelligence research, it would be the greatest technological revolution of the near future. There are several technologies making up an autonomous vehicle - laser, radar, GPS, image processing, computer vision, machine vision, etc. Compared with other sensors like LIDAR or ultrasonic, cameras are lower-cost and can provide more information on the road (traffic signs, traffic light, pedestrian, obstacles, etc.). However, the main drawback of the vision-based method is that it is not robust to illumination changes. Histogram equalization was utilized to overcome this issue but this was not sufficient and computationally costly.

Since their introduction in the early 1990s,Convolutional Neural Networks (CNNs) [1] have been the most popular deep learning architecture due to the effectiveness of conv-nets on image related problems. It is also successfully applied to recommender systems, natural language processing and more. The breakthrough of CNNs is that features are learned automatically from training examples. Although their primary disadvantage is the requirements of very large amounts of training data, recent studies have shown that excellent performance can be achieved with networks trained using "generic" data. For the last few years, CNNs have achieved state-of-the-art performance in most of important classification tasks [2-3], on various visual tasks, such as object recognition [4]; subcategory recognition [5]; scene recognition [6] and detection [7].

Besides, with the increase in computational capacities, we are presently able to train complex neural networks to understand the vehicle's environment and decide its behavior. For example, Tesla Model S was known to use a specialized chip (MobileEye EyeQ), which used a deep neural network for vision-based real-time obstacle detection and avoidance. More recently, researchers are investigating DNN based end-to-end control of cars [8] and other robots. Executing CNN on an embedded computing platform has several challenges. First, despite many calculations, strict real-time requirements are demanded. For instance, latency in a vision-based object detection task may be directly linked to the safety of the vehicle. On the other hand, the computing hardware platform must also satisfy cost, size, weight, and power constraints. These two conflicting requirements complicate the platform selection process as observed in [9]. There are already several relatively low-cost RC-car based prototypes, such as MIT's RaceCar [10] and UPenn's F1/10 racecar.

Encouraged by these positive results, in this paper we develop a real-time end-to-end deep learning based RC-car platform. In term of hardware, it is included a Raspberry Pi 3 Model B quad-core computer, a Pi camera and a 1/10 scale RC car, Arduino Uno R3. This research target is training a vision-oriented model to autonomously navigate the car. Data were collected by manually controlling the RC car and recording the images paired with the time-synchronized steering angle as illustrated in Fig. 1.

Then, training the network offline on a desktop computer, which is equipped with a CPU Core-i5, 2.3GHz, RAM 4GB is shown in Fig. 2.

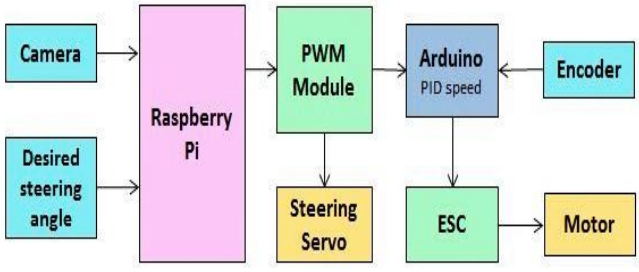

Figure 1. *Block diagram of RC self-driving car platform*

Truong-Dong Do, Minh-Thien Duong and Quoc-Vu Dang are students at Ho Chi Minh City University of Technology and Education, Vietnam.

My-Ha Le is with Faculty of Electrical and Electronics Engineering, Ho Chi Minh City University of Technology and Education, Vietnam. (E-mail : halm@hcmute.edu.vn).

* Corresponding author.

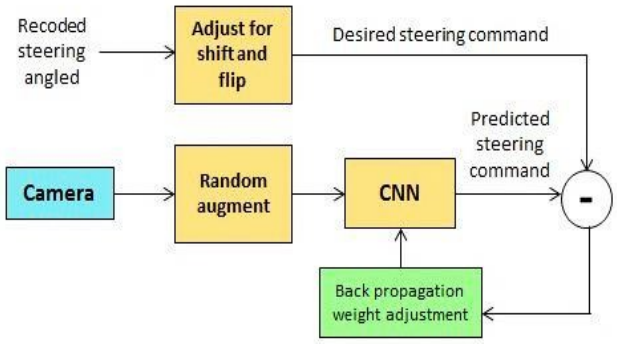

Figure 2. *The process of training neural network*

Finally, the trained network is copied back to the Raspberry Pi, which receives an image frame from a single forward-looking camera as input and generates a predicted steering angle value as output at each control period as shown in Fig. 3.

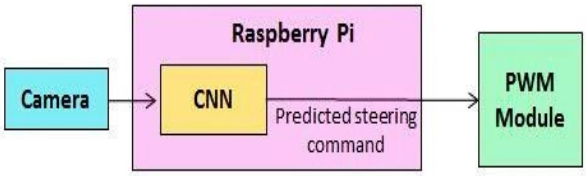

Figure 3. *The trained network is used to generate steering commands*

The paper proceeds as follows. Section II describes the design of hardware platform. The proposed network architecture is presented in Section III. The datasets are described in Section IV, with experimental results presented in Section V. Finally, Section VI concludes the paper, then discusses ongoing and future work.

II. HARDWARE PLATFORM

In this section, we provide an overview of hardware platform, which is involved a Raspberry Pi 3 Single Board Computer (SBC), a Raspberry Pi camera, an Adafruit PWM/Servo Driver PCA9685, an Arduino Uno R3 board, a Li-Po battery, and a 1/10 scale RC car. Fig. 4 shows the hardware platform.

A. Raspberry Pi 3 Model B.

The Raspberry Pi is a small single-board computer developed in the United Kingdom by the Raspberry Pi Foundation. Raspberry Pi 3 Model B was released in February 2016 with a Broadcom BCM2837 SoC, which has a 64 bit quad-core ARM Cortex-A53 processor, running at up to 1.2GHz, 1 GB of RAM, onboard WiFi, Bluetooth and USB boot capabilities [11].

While using Raspberry Pi 3 platform, there are a few system-level factors that need to be considered to achieve steady and high real-time performance. Deep neural network model operations are computationally intensive, thus it is possible for the temperature of the CPU to rise quickly. If the temperature reaches the threshold, Pi 3 decreases the clock speed down to 600MHz - half of the maximum. So that, appropriate cooling solutions (heat sink or fan) are important to sustain CPU frequency. Beside, the power supply used for

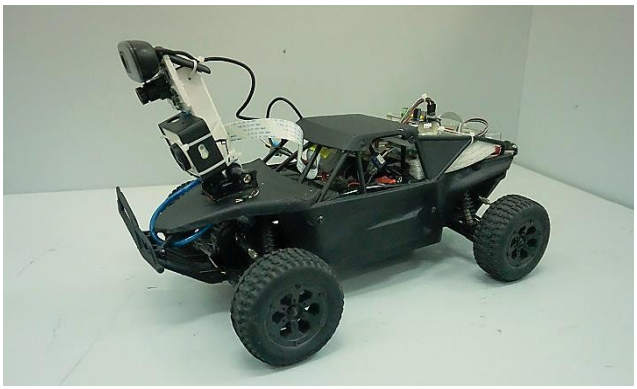

Figure 4. *The hardware platform*

the Raspberry Pi 3 is expected minimum of 2 Amps current, otherwise optimal performance isn't guaranteed.

B. Raspberry Camera Module

It is an adjustable-focus 5megapixel camera using an Omnivision OV5647 sensor. From its first launch, the Raspberry Pi has had a connector on it to attach a camera to the GPU. This connection uses the CSI-2 electrical protocol and is a standard used in most of mobile phones. Moreover, it is an extremely fast connection, which on the Raspberry Pi is capable of sending 1080p-sized images (1920x1080 x10bpp) at 30 frames per second or lower resolution at even higher frame rates [12].

C. Adafruit PWM/Servo Driver PCA9685.

It is an I2C-controlled 16-Channel PWM driver with a built-in clock, PWM frequency can be adjusted up to about 1.6 KHz. PCA9685 has a 12-bit resolution for each output - for servos, that means about 4us resolution at 60Hz update rate [13]. We connect it to Raspberry Pi via two I2C pins (SDA and SCL), 3.3V and ground. The throttle cable runs to Channel 0 and steering is Channel 1.

D. Arduino Uno R3.

The Arduino Uno is a single-board microcontroller based on the ATmega328. It has 14 digital input/output pins (6 of those can be used as PWM outputs), 6 analog inputs, a 16 MHz crystal oscillator. It contains everything needed to support the microcontroller, which allows attaching various extension boards. In this work, Arduino board was used to read the desired throttle from PWM module PCA 9685, the actual speed from the encoder then maintain car's velocity with PID controller algorithm.

III. NETWORK ARCHITECTURE AND PARAMETERS OPTIMIZATION

The network architecture is shown in Fig. 5 comprised of 9 layers, including 5 convolutional layers and 4 fully connected ones. The input image is 120x160x3 (height x width x depth format). Convolutional layers were designed to perform feature extraction and were chosen empirically through a series of experiments with varied layer configurations. The first three convolutional layers use the 5x5 kernel, a stride of 2x2. The respective depth of each layer is 24, 32 and 64 to push network going deeper. The local

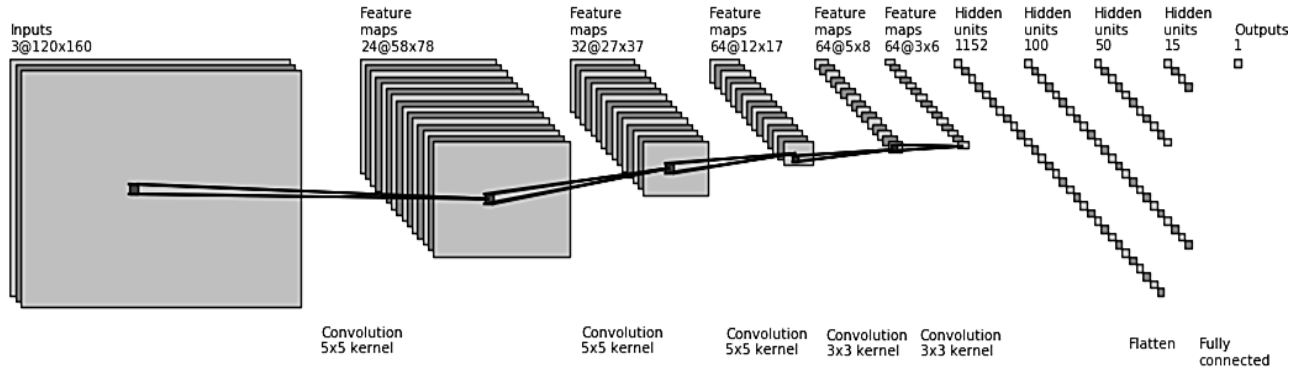

Figure 5. Network architecture

TABLE 1. NETWORK PARAMETERS

Layer (Type)	***Output Shape***	***Parameters***
Input	(120, 160, 3)	0
Conv1 (f=5,s=2)	(58, 78, 24)	1,824
Conv2 (f=5,s=2)	(27, 37, 32)	19,232
Conv3 (f=5,s=2)	(12, 17, 64)	51,264
Conv4 (f=3,s=2)	(5, 8, 64)	36,928
Conv5 (f=3,s=1)	(3, 6, 64)	36,928
FC6	1152	0
FC7	100	115,300
FC8	50	5,050
Softmax	15	765
Total		267,291

feature is continue to be processed in last two convolutional layers with kernel size 3 × 3, depth of 64. After the convolutional layers, the output gets flatten and then followed by a series of fully connected layers, of gradually decreasing sizes: 1152, 100, 50 and 15. All hidden layers are equipped with the rectified linear unit (ReLU) to improve convergence. From feature vectors, we apply softmax to calculate steering wheel angle probability. The whole network will have roughly 267,291 parameters as illustrated in Table. 1 and will offer great training performance on modest hardware. Additionally, to make the architecture more robust and avoid over-fitting, we use dropout [14] at rate 0.1 for two last fully connected layers gathering most of parameters.

A. *Softmax*

The softmax function squashes the outputs of each unit to be between 0 and 1, just like a sigmoid function. However, it also divides each output such that the total sum of the outputs is equal to 1. The output of the softmax function is equivalent to a categorical probability distribution; it indicates the probability that any of the classes are true.

The softmax function is defined as:

$$\sigma(z)_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}} \quad (1)$$

where z is a vector of the inputs to the output layer, j indexes the output units (j = 1, 2, ..., K).

B. *Loss Function and optimization*

To rate our architecture, we use *Cross-Entropy Cost Function* as an objective function. Cross-entropy formula gives two distributions over discrete variable x, where $q(x)$ is the estimation for true distribution $p(x)$ is given by

$$H(p,q) = -\sum_{\forall x} p(x)\log\big(q(x)\big) \quad (2)$$

To optimize this loss, *Adam* was used as the optimizer [15]. This optimization method usually substantially outperforms more generic stochastic gradient descent methods. Initial testing of these models indicates that their loss level change slowly after a few epochs. Although *Adam* computes an adaptive learning rate through its formula, the decay of the learning rate was used. The decay rate of the optimizer was updated from 0 to the learning rate divided by the number of epochs.

IV. DATASET

In this section, the data used to train the network architecture in Section 3 is presented. This includes (1) the experimental environments to collect data, (2) datasets description, and (3) augmentations method to generate various data.

A. *Experimental Environments*

To collect the training data, we manually drive the RC car platform to record timestamped images and control commands in two outdoor tracks created on the asphalted plane.

a) *b)* *c)*

Figure 6. *Experimental track 1: (a) is the overall oval-shaped lined track. (b) is a curve of the track.(c) is the car drive on the track*

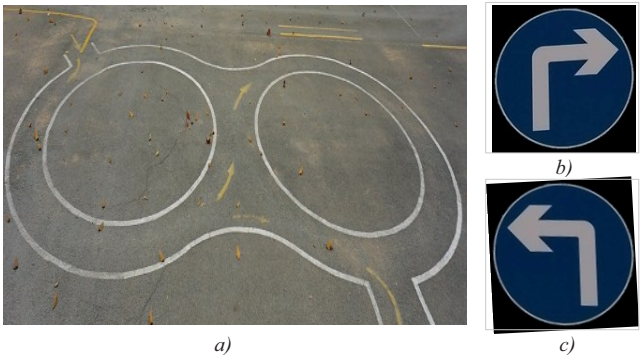

Figure 7. *Experimental track 2: (a) is the overall 8-shaped lined track. (b) and (c) are the traffic signs erected on 8-shaped lined track.*

The oval-shaped lined track is a 50-meter long and 70-centimeter wide course with borders in 10-centimeter wide white tape as shown in Fig. 6a. The car can be driven to turn left or turn right in 3-meter radius curves at a time.

The 8-shaped lined track comprised of two circular loops that form a tangential crossing junction is shown in Fig. 7a. This 40-meter long course has both right and left-hand curves. The curve radius is 3 meters. There are four 15-centimeter diameter traffic signs erected on outside edges of the track to guide direction for the car.

B. Data Description

Data preparation is required when working with deep learning networks. Raspberry Pi records the images and driving information from the user manually driving the car around the track with the speed at 4-5km/h. Collected data contains over 30,000 images couple with steering angles. The original resolution of the image is 160x120. Pi Camera is configured to capture at a rate of 10 frames per second with exposure time 5000us to prevent the blur caused by elastic vibration when the car drive on the track. Sample images of this dataset are shown in Fig. 8.

C. Data Augmentations

Deep learning model tends to over fit the small dataset because of having too few examples to train on, resulting in a model that has poor generalization performance. Data augmentation is a technique of manipulating the incoming training data to generate more instances of training data by

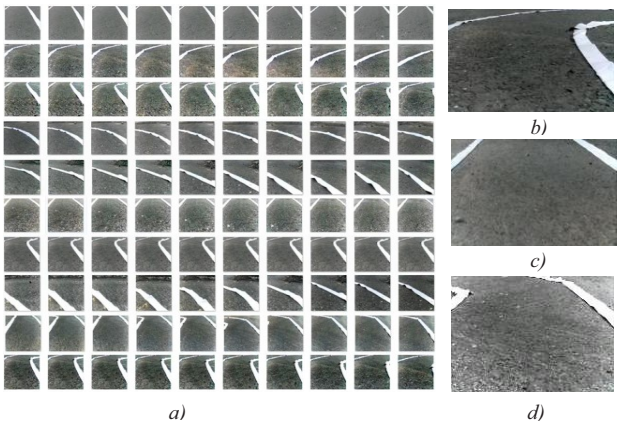

Figure 8. *Collected Dataset: (a) are samples from the driving dataset. (b),(c) and (d) are typical images in dataset*

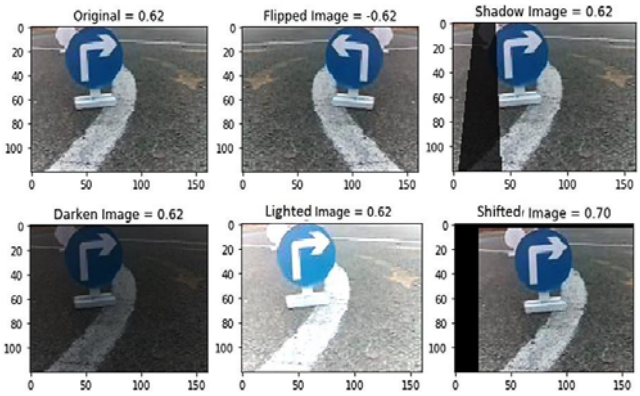

Figure 9. *Data augmentation examples*

creating new examples via random transformation of existing ones. This method boosts the size of the training set, reducing overfitting. Common transformations are horizontal flip, brightness adjusts, random shadow, height and width shift are illustrated in Fig. 9. In addition, data augmentation is only performed on the training data, not validation or test set.

Horizontal Flip

The model needs to learn to steer correctly whether the car is on the left or right side of the road. Therefore, we apply a horizontal flip to a proportion of images and naturally invert the original steering angle.

Brightness Augmentation

Brightness is randomly changed to simulate different light conditions since some parts of the tracks are much darker or lighter, due to shadows and sun light. We generate augmented images with different brightness by first converting images to HSV, scaling up or down the V channel and converting back to the RGB channel.

Shadow Augmentation

The tracks sometimes have patches covered by a shadow, which could throw the model into confusion. The shadow augmentation adds random shadows cast across the image. This is implemented by choosing random points and shading all points on one side of the image.

Horizontal and Vertical Shifts

We will shift the camera images horizontally to simulate the effect of the car being at different positions on the road, and add an offset corresponding to the steering angle for every pixel shifted laterally. In our case, we empirically assign 0.0035 for each pixel. We will also shift the images vertically by a random number to simulate the effect of driving up or down the slope.

V. EXPERIMENTAL RESULTS

A. Training Process

The stored data is then copied to a desktop computer, which is equipped with a Core-i5, 2.13GHz, Ram 4GB, where we train the network to accelerate training speed. We split the dataset into three subsets: a training set (60%), a validation set (20%) and a test set (20%). Training dataset contains 36,000 augmented frames, validation set contains

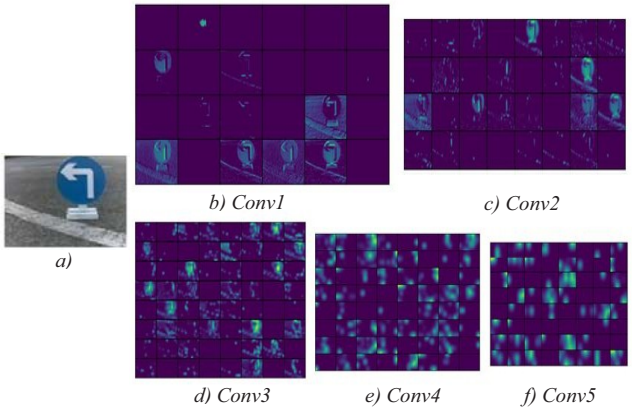

Figure 10. *The visualization output of convolutional layers. (a) is an originally selected frame.(b),(c),(d),(e) and (f) are the feature maps at first five convolutional layers.*

6000 frames.

The code is written in Google's Keras and TensorFlow framework. The following is some crucial parameters for re-implementing our method: dropout with a ratio of 0.1 (in the terminology of TensorFlow, this implies only 10% neurons are active at specific layer) is used in fully connected layers. The learning rate is initialized to 0.001. For the stochastic gradient solver, *Adam* was adopted as an optimizer. The training process takes about 8 hours to train 50 epochs. When a model was trained with the aforementioned data, the training accuracy was 92,38% and the validation accuracy was 89,04%.

B. *Outdoor experiments.*

Once trained on the desktop computer, the model was copied back to the Raspberry Pi. The network is then used by the car's main controller, which feeds an image frame from the Pi Camera as input. In each control period, 10 images per second can be processed and top speeds of the car around curves are about 5 - 6 km/h. The representative testing images and steering wheel angles of model's prediction can be seen in Fig. 11.

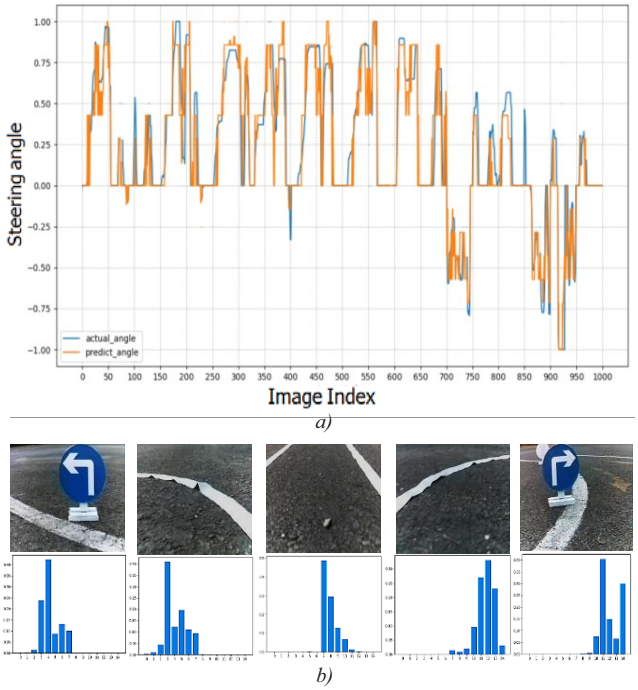

Figure 11. *Experimental Results: (a) is the actual and predicted steering wheel angles of the models. (b) are the representative images and outputs after softmax of the model.*

The trained model was able to achieve a reasonable degree of accuracy. The car successfully navigates itself in both two tracks with diverse driving conditions, regardless of whether lane markings are present or not.

VI. CONCLUSION

In this paper, we presented an autonomous car platform based on state-of-the-art AI technology: End-to-end deep learning based real-time control. This work addresses a novel problem in computer vision, which aims to autonomously drive a car solely from its camera's visual observation.

After we ended up with one that is able to power our car to drive itself on both tracks. The encouraging result showed that it is possible to use a deep convolutional neural network to predict steering angles of the vehicle directly from camera input data in real-time. Despite the complexity of the neural network, the embedded computing platforms like Raspberry Pi 3 are powerful enough to support the vision and end-to-end deep learning based real-time control applications.

In addition, one of the most important learnings from this project is that data**,** without all those images and steering angles, along with their potentially infinite augmentations, we would not have been able to build a robust model.

The issue we observed in training/testing the network is camera latency. It is defined as the period from the time the camera sensor observes the scene to the time the computer actually reads the digitized image data. Unfortunately, this time can be significantly long depending on the camera and the performance of the Pi, which is about 300-350 milliseconds. This is remarkably higher than the latency of human perception, which is known to be as fast as 13 milliseconds [16]. Higher camera latency could negatively affect control performance, especially for safety-critical applications, because the deep neural network would analyze stale scenes.

As can be seen, there are many areas we could explore to push this project further and obtain even more convincing results. In the future, we continue to investigate ways to achieve better prediction accuracy in training the network, identify and use low-latency cameras, as well as improving the performance of the RC car platform, especially related to precise steering angle control. Furthermore, we are going to take throttle into the model with the ambition of achieving higher levels of autonomous vehicles.

REFERENCES

[1] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel, "Backpropagation applied to handwritten zip code recognition," *Neural computation,* vol. 1, no. 4, pp. 541-551, 1989.

[2] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097-1105.

[3] J. Donahue, Y. Jia, O. Vinyals, J. Hoffman, N. Zhang, E. Tzeng, and T. Darrell, "Decaf: A deep convolutional activation feature for generic visual recognition," in *International conference on machine learning*, 2014, pp. 647-655.

[4] L. Fei-Fei, R. Fergus, and P. Perona, "Learning generative visual models from few training examples: An incremental bayesian approach tested on 101 object categories," *Computer vision Image understanding,* vol. 106, no. 1, pp. 59-70, 2007.

[5] P. Welinder, S. Branson, T. Mita, C. Wah, F. Schroff, S. Belongie, and P. Perona, "Caltech-UCSD birds 200," 2010.

[6] J. Xiao, J. Hays, K. A. Ehinger, A. Oliva, and A. Torralba, "Sun database: Large-scale scene recognition from abbey to zoo," in *Computer vision and pattern recognition (CVPR), 2010 IEEE conference on*, 2010, pp. 3485-3492.

[7] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2014, pp. 580-587.

[8] M. Bojarski, D. Del Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. D. Jackel, M. Monfort, U. Muller, and J. Zhang, "End to end learning for self-driving cars," 2016.

[9] N. Otterness, M. Yang, S. Rust, E. Park, J. H. Anderson, F. D. Smith, A. Berg, and S. Wang, "An evaluation of the NVIDIA TX1 for supporting real-time computer-vision workloads," in *Real-Time and Embedded Technology and Applications Symposium (RTAS), 2017 IEEE*, 2017, pp. 353-364.

[10] S. Karaman, A. Anders, M. Boulet, J. Connor, K. Gregson, W. Guerra, O. Guldner, M. Mohamoud, B. Plancher, and R. Shin, "Project-based, collaborative, algorithmic robotics for high school students: Programming self-driving race cars at MIT," in *Integrated STEM Education Conference (ISEC), 2017 IEEE*, 2017, pp. 195-203.

[11] M. Richardson, and S. Wallace, *Getting started with raspberry PI*: " O'Reilly Media, Inc.", 2012.

[12] A. Stone, "MagPi14," *Get to grips with the Raspberry Pi camera module*, Jul-2013.

[13] Adafruit Industries, "Adafruit 16-Channel 12-bit PWM/Servo Driver - I2C interface." [Online]. Available: https://www.adafruit.com/product/815. [Accessed: Jun-2018].

[14] 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.

[15] D. P. Kingma, and J. Ba, "Adam: A method for stochastic optimization," 2014.

[16] T. Burger, "How fast is realtime? human perception and technology," *PubNub*, February, 2015.