International Journal of
INTELLIGENT SYSTEMS AND APPLICATIONS IN ENGINEERING
ISSN:2147-6799 www.ijisae.org Original Research Paper

Real-Time End-to-End Self-Driving Car Navigation

Yahya Ghufran Khidhir*[1], Ameer Hussein Morad [2]

Submitted: 25/10/2022 **Revised:** 24/12/2022 **Accepted**: 23/01/2023

Abstract*:* In this study, a deep neural network (DNN)-based vision-based navigation for autonomous vehicles is proposed. This novel DNN-based system obtains the data from a single camera to provide vehicle control outputs that modify both the steering wheel angle and the vehicle's velocity. In addition, it plays a major role in safely navigating the vehicle in a road traffic environment. Numerous autonomous driving algorithms use end-to-end DNN, where camera data is fed into complex machine learning algorithms exclusively to estimate the steering angle value, but this research proposes a light-novel network model that controls both steering and speed values with much more simplicity. Various neural blocks are organized with the ultimate objective of producing control actions to achieve the aim of the research. Experimental modifications are made to parameters such as the number of convolutional layers, the model size, padding, stride, and the number of neurons in fully-connected layers to make the model simpler and lighter to execute during inference. Using an embedded system called Jetson Nano 2GB, the designed model was trained and tested using the images collected along two different paths. Our DNN-based autonomous driving system successfully predicts speed and steering values with a mean error of 1.58% and maintains performance, allowing for highly efficient autonomous driving. Furthermore, the suggested DNN network maintains performance, attaining autonomous driving success with comparable efficacy to the other autonomous driving models. The lightweight end-to-end architecture with superb performance is especially suited for autonomous driving.

Keywords: *Self-driving car, deep neural network, embedded systems, end-to-end learning.*

1. Introduction

The popularity of machine learning applications in embedded systems, mobile phones, and other Internet of Things (IoT) devices is on the rise. This encourages the development of new hardware platforms capable of processing the data required for machine learning, lightweight machine learning architectures and model designs, and implementations optimized for low-performance hardware. Machine learning also has a significant effect on the manufacturing industry, especially in the development of self-driving cars [1].

The Sequential autonomous driving system generally consists of four stages: sensors, perception, path planning, and vehicle control. The data from the sensors is processed by the perception stage, which combines the sensor data into meaningful information. The path planning stage then uses the output of the perception stage to plan the behavior. Lastly, vehicle control guarantees that the vehicle follows the path determined by the planning stage [2].

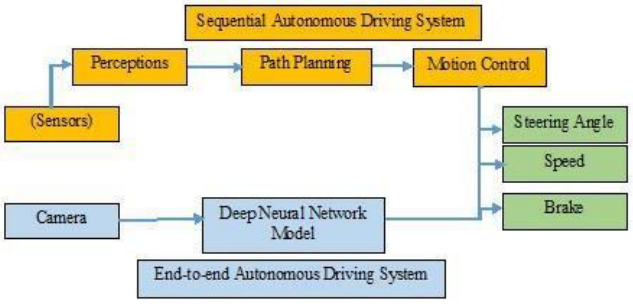

Fig. 1. Sequential and End-to-end Autonomous Driving Systems

An end-to-end deep neural network was designed for autonomous driving, which uses camera images as input and steering angle and speed prediction as outputs. End-to-end learning describes the training of neural networks without human intervention from beginning to end, as shown in figure 1. The primary objective of end-to-end learning is to train the system's internal representation of the essential processing stages, such as the recognition of valuable road characteristics based on input signals exclusively [3].

The objective of this study is to design a lightweight deep neural network (DNN), which is an end-to-end neural network that can carry out the task of autonomous driving on two different tracks. The proposed model is anticipated to be implemented on an embedded device with low-performance hardware. The findings of the suggested

[1] *Department of Mechatronics Engineering, Al-Khwarizmi College of Engineering, University of Baghdad, IRAQ*
ORCID ID : 0000-0003-1850-3856
[2] *Department of Information and Communication Engineering, Al-Khwarizmi College of Engineering, University of Baghdad, IRAQ*
ORCID ID : 0000-0003-1700-8546
** Corresponding Author Email: yahia.ghofran1202a@kecbu.uobaghdad.edu.iq*

model indicate that its size and complexity are smaller in comparison with well-known models such as ReNet18 or AlexNet while achieving a higher level of autonomous driving efficiency; moreover, the new proposed model not only controls the steering angle but also estimates the values of the vehicle's speed and enables the vehicle to stop in the presence of an obstacle, pedestrians, or stop traffic signs. In addition, the importance of a computationally light solution for DNN used for autonomous driving is to execute the model on an embedded system, such as the one used in this research (Jetson Nano 2GB). This is because state-of-the-art deep learning models may not be able to run on embedded systems that might have hardware limitations.

The relevant literature is presented in the following section. In Section 3, we discuss the methodology, tools, data sources, model architecture, and training procedure. Results and discussion of using the novel model and inference while driving autonomously are presented in Section 4. The final part of the paper includes the conclusion.

2. Related work

With advancements in technology, such as the embedded system Jetson Nano, and an increase in available datasets, deep learning techniques for autonomous driving have gained popularity [4], where they are used for both end-to-end driving approaches and the application of deep learning in individual subsystems[5].

[6] Proposed using a convolutional neural network (CNN) (PilotNet) to immediately steer a car by tracking raw pixels from a single camera. The CNN was able to learn important road features with only a few samples of training data. However, it was difficult to see how the network's internal processing steps worked and the network was not sufficiently robust.

In the same manner, Multichannel Convolutional Neural Networks (M-CNNs) were used to determine the vehicle's steering angle and speed by the authors of [7]. The inputs to the model are (227x227x3) front-view camera images and feedback speed sequences. To test the effectiveness of the proposed method, researchers used both the publicly available Udacity dataset and the collected SAIC dataset. The proposed idea was compared to the Cg Network and PilotNet. With a mean absolute error (MAE) of 1.26 degrees for steering angle and 0.19 meters per second for speed values, the M-CNNs did better than the implemented PilotNet and the Cg Network.

The use of convolutional neural networks (CNN) to predict the steering angle was proposed in reference [8]. The CNN was developed with the goal of reducing the number of necessary training parameters. The final layer's output was used as the steering angle prediction value, and the network was constructed from a series of convolutional and dense layers. CNN was able to predict steering angles with a 78.5% accuracy while simultaneously reducing a large number of parameters, avoiding overfitting, and improving model generalization. There was a lot of variation, though, because the dataset was so small.

CNN-based closed-loop feedback control to regulate the steering angle of the vehicle was examined using a driving simulation environment[9]. Input images to the first convolutional layer of the proposed models have a size of (90x160x3).Using the Caffe framework for deep learning, the authors of [9] designed and tested their CNN model (DAVE-2SKY) in a lane-keeping simulation. Although the results were to some extent promising, technical issues occurred as the distance between was smaller than 9 meters.

Reference [10] proposed a CNN-LSTM network that would improve vehicle control by predicting steering angle and speed end-to-end. Networks were tested in both real-world and simulated environments to ensure their efficacy. A visualization analysis is also used to demonstrate the impact of multi-auxiliary tasks on network performance, with an average assistance rate of two times per 10 km.

In another paper [11], researchers compared Nvidia's steering angle control model to the Visual Geometry Group (VGG16), ResNet-152, and Densely Connected Convolutional Networks (DenseNet-201). All models used an input image size of (66x200x3). Twenty-five minutes of driving time was used to train the models with a Raspberry Pi camera at 30 frames per second. With an MSE of 0.3521, Nvidia's model significantly outperformed the competition. To demonstrate Nvidia's model's efficacy, the authors selected videos at random from YouTube.

In this paper, a single light-weight CNN model with an input image size of (66x200x3) is proposed to achieve the task of end-to-end self-driving in terms of steering angle, speed, and automatic braking. The model is employed on low-cost and low-performance hardware, the Jetson Nano, and evaluated online on a dataset collected from two outdoor tracks. Visual analysis has been implemented to show the performance and repeatability of the model output compared to the MPU sensor during inference.

3. Method

This section provides a brief description of the hardware and software setup as well as a thorough explanation of the proposed approach, data collection, model architecture, and training operation for this work.

3.1. Hardware and Software Setup

The main components of the JetRacer 2GB include an

8MP HD resolution 160° FOV wide-angle camera, powered by the Jetson Nano 2GB, two DC gear motors with an idle speed of 740RPM, one MG996R servo motor, three 18650 lithium-ion batteries, and the JetRacer Expansion Board. An Arduino is connected to Jason Nano through the I2C Bus and is powered by the Jetson Nano. A MPU sensor is connected to the Arduino board as shown in figure 2. After the assembly of the robot hardware is accomplished, the software setup takes place. The image of JetPack version 4.5.1 is downloaded from the NVIDIA-AI-IOT website, which includes the bootloader Linux kernel, cuDNN 8.0, TensorRT 7.1.3, and OpenCV 4.1.1. The next step is to install Python packages like torch2trt, JetCam, and JetRacer on Jetson Nano, with the aim of preparing the environment for executing the model.

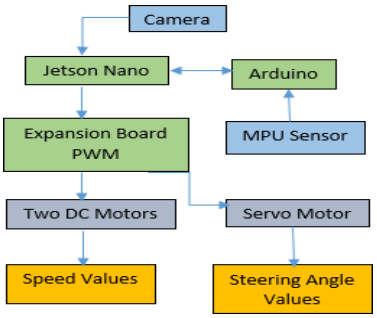

Fig. 2. Jetracer Self driving car Platform

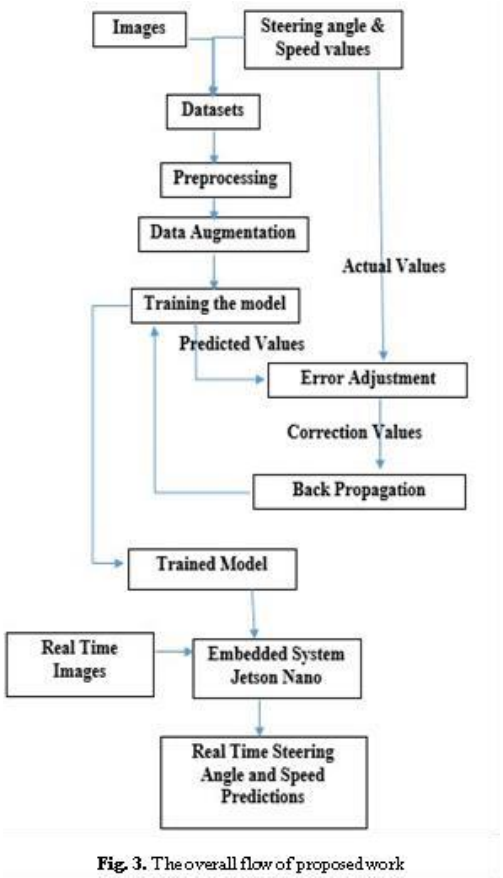

Fig. 3. The overall flow of proposed work

3.2. The overall flow of proposed work

The block diagram of the overall flow of the proposed work is shown in figure 3. As a first step, we drove the car around the representative tracks, recording the steering angle and the speed value manually to get the data we needed. Second, the images were fed into the CNN models for training after undergoing some data preprocessing and data augmentation. After estimating the mean square error between the expected and predicted measurements, the network makes adjustments to the error via backpropagation and the modification of weights. As a final step, we transferred the trained model to the Jetson Nano embedded system so that it could analyze unlabeled images and make predictions about the commands to be given to the Jetracer robot during inference. The model was deemed successful and self-sufficient when it could drive on the centerline and both tracks while avoiding the orange bolder lines.

3.3. Dataset

Data are an essential part of an end-to-end DNN-based self-driving system, in this research dataset was obtained from two separate tracks. The initial 2*3 meters path was utilized for data collecting, whereas the second 4*2 meters path with curvatures in the middle was employed for testing and evaluation. The track edges are defined with orange lines, while dashed white lines outline its center. Usually, the inference path is more shaded than the data collection track. The dataset is an organized collection of images with steering angles and speed values as labels. The Jetracer mobile robot was placed in this research on the typical route to acquire the necessary data for a training procedure. Rather than employing three cameras, we used a single camera attached to the robot's chassis to gather images from three distinct angles (center, left, and right). The objective was to keep the mobile robot in the center of the route, thus images from the camera positioned in the middle and on the left and the right were captured with identical speed and steering readings. During capturing images procedure, the measurements (steering angles and speed values) are entered directly from a remote computer by selecting the appropriate position XY coordinates on each frame. The servo motor attached to the robot's frontal wheels controls the X-axis, latitude, which indicates the degree of angle from left to right and is in the range of (-1.0 to +1.0). The required speed for such an image is defined along the Y-axis, the longitude, and the entire range of the throttle from backward to forward, which is controlled by two DC motors, is from (-1.0 to +1.0), producing a unique training dataset. To acquire high-quality findings, images and measurements should be captured during manual driving in the way expected during the inference phase. To enhance training effectiveness and allow for clockwise and counter-clockwise robot

movement, we utilized data augmentation. “Augmentation” describes the method of improving training data without changing its fundamental features. In our study, the augmentation process consists of flipping the images horizontally. The steering angle must be inverted for each frame to be flipped, resulting in a new value of:

$$Steering\ angle\ after\ flipping = (-1) * original\ steering\ angle \quad (1)$$

681 image samples were collected from the first track as shown in figure (4a), while 83 image samples were taken from the second track as shown in figure (4b). It is evident from the figures that the first track doesn’t contain inside turns, while on the contrary, the second track has. The size of captured images was 224x224x3 (224 widths, 224 heights, 3 number channels). Once data augmentation was performed, the number of the dataset samples increased to 1528. 80% of the dataset was used for training purposes, while 18% and 2% of the dataset were used for validation and testing respectively.

3.4. Network architecture

The main idea of the research was to develop a lightweight model capable of performing autonomy while simultaneously achieving the best feasible performance. The lightweight model is often the one with the fewest parameters affecting its memory footprint and calculations. Consequently, the computational performance is affected by the number of extracted features, the size of kernels, and the depth of layers. Two separate tracks were used to evaluate the capabilities of the self-driving model.

In the beginning step of the design process for this new model, the input image was resized from 224*224*3 to 200*66*3 to reduce the amount of input image parameters. Generally, the model is composed of a single batch normalization layer, five convolutional layers, four max-pooling layers between each convolutional layer, and five fully-connected layers.

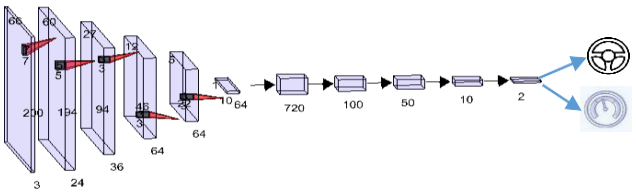

Fig. 5. Network architecture

In more detail, Ensuring that each input to the model has the same range of values can be performed through data normalization; this can provide consistent convergence of weights (w) and biases. As a means to enhance feature extraction, we have settled on five convolutional layers to even further develop the features that have already been extracted; Equation (2).

$$C(i,j) = (I * K)(i,j) = \sum_m \sum_n I(i-m, j-n) K(m,n) \quad (2)$$

The outcome of the convolution stage is the feature maps, denoted by $C(i,j)$, that focus on a specific feature in the input image irrespective of where those features are located. The number of extracted features is proportional to their locations in the networks. Since the input layers acquire raw data, which is always noisy, the layers closest to the input images use fewer features, whereas the layers closest to the outputs use a large number of features for training.

(a) **(b)**

Fig. 4. a. First track b. Second track

Table 1. Network parameters

Layer (type)	Output Shape	Parameters
BatchNorm2d-1	[-1, 3, 66, 200]	6
Conv2d-2	[-1, 24, 60, 194]	3,552
MaxPool2d-3	[-1, 24, 31, 98]	0
Conv2d-4	[-1, 36, 27, 94]	21,636
MaxPool2d-5	[-1, 36, 14, 48]	0
Conv2d-6	[-1, 64, 12, 46]	20,800
MaxPool2d-7	[-1, 64, 7, 24]	0
Conv2d-8	[-1, 64, 5, 22]	36,928
MaxPool2d-9	[-1, 64, 3, 12]	0
Conv2d-10	[-1, 64, 1, 10]	36,928
Linear-11	[-1, 720]	461,520
Linear-12	[-1, 100]	72,100
Linear-13	[-1, 50]	5,050
Linear-14	[-1, 10]	510
Linear-15	[-1, 2]	22
	Steering angle and Speed	

As depicted in figure 5, the first convolutional layer's filter size was 7 × 7 with 24 feature maps, making the number of

parameters at this stage 3552 as shown in table 1. The second stage of the model was the max-pooling layer which featured the stride of two, which plays a significant role in down sampling the parameters of the model. Implementing the max-pooling procedure to reduce the size of the subsequent layer's network of nodes and, as a result, the number of trainable parameters. A pooling layer is typically used to reduce the output size and prevent overfitting. The Max Pooling layer is a layer inserted after the activation function that minimizes spatial variance in pixel values by choosing the maximum value. The most common benefit of using the max-pooling approach is that this operation doesn't require the addition of new parameters, hence it is parameter-free.

To obtain a two-dimensional vector of parameters, the flattened layers were added. Adding the flattened layers did not introduce any new parameters; it only rearranged the pre-existing parameters in one dimension layer. The five flattened layers reorganized the parameters from 650 to two, which were the predicted values for steering angle and speed.

3.5. Training the model

The designed model was trained in python language on the Google Colab platform. In addition to data normalization, color jitter data pre-processing was applied to both different tracks due to the disparity in color brightness. Adam optimizer [12]was employed with a learning rate of $5 * e\hat{} - 4$; this learning rate should be maintained to a minimum value since it is more stable, which will help to prevent missing local optima. The model was trained for 70 epochs, where it was stopped if no changes occurred in validation loss within 10 epochs. Mean Square Error (MSE)[13] was chosen as appropriate for the regression network to reduce the discrepancy between the estimate and the measurements of both speed and steering angle values.

$$MeanSquaredError = \frac{1}{N}\sum_{j=1}^{N}(y_j - \hat{y}_j)^2 \qquad (3)$$

The vector $\hat{y}_i$ represents the values of N forecasts and y_i is a vector that contains N true values.

The performance of the model was accessed based on its capability to guide the mobile robot along the two specified tracks. It was considered effective when the model was capable to drive the robot autonomously on both specified tracks.

4. Results and discussion

The Performance of the proposed architectures on the validation dataset was measured using MSE in this research. The validation loss of a model trained with the above-mentioned data was 0.0158, and the model's estimated size was 6.77 MB, with a total of 607182 parameters.

Showing the efficiency measure of the proposed model, Figure 6 displays the results of recoding the predicted steering angle and speed values with frames for the proposed model along the first track. The figure also illustrates the partitions of the first driving path which are: (A) long end turns. (B) straight-forward paths. It is evident that the normalized steering angle value peaked (0.5) halfway through the long end turns (a); where the normalized speed values started rapidly decreasing to prevent the mobile robot from deviating away from the track center, as shown in figure 9 by the red circles. As the mobile reached the straightforward path (b), the response of the normalized steering angle then gradually decreased. additionally, the values of the normalized steering angle and speed along the straightforward path were within the ranges of (-0.4 to -0.6) and (0.4 to 0.6) respectively, implying that the deviation of the mobile robot from the center of the first track was smaller than what observed during long end turns in section (A), meanwhile, the velocity of the robot was moderate.

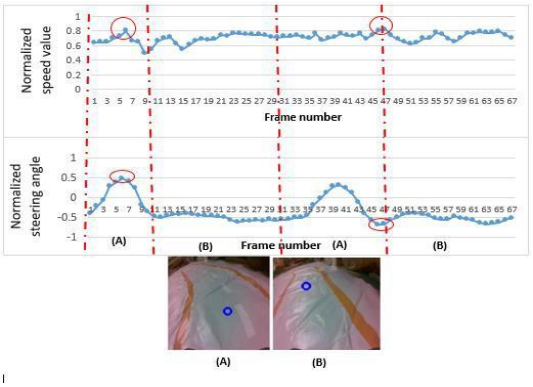

Fig. 6. Normalized Speed and Steering angle values per one full lap of autonomous driving on first track
(A) Long-end turns
(B) Straightforward path

A second track with various surface textures was implemented to evaluate the model performance. Findings from the experiment were satisfactory, which showed that the designed model was able to maintain the mobile robot's movement within a small range of departure from its intended path. Specifically, both short inside and long end turns of the second track were completed by modifying the robot's speed and steering angle accordingly to avoid the mobile robot from deviating out of the track's center. The red circles shown in figure 7 represent the behavior of the proposed model whenever the robot approaches the sharp turns; as can be seen, the robot's speed dropped rapidly in an attempt to avoid drifting off the track.

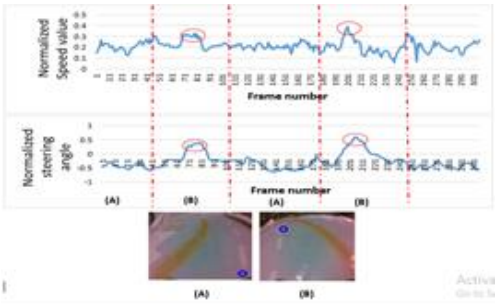

Fig. 7. Normalized Speed and Steering angle values per one full lap of autonomous driving on Second track

In addition, histograms are used to display statistical analysis of autonomous driving. This kind of descriptive statistic has implications for testing the performance of the autonomous vehicle within a longer period. Figure 8 shows histograms of the relative deviations from the center of the trajectory obtained from the designed model and the MPU sensor alike during one complete lap of autonomous driving for both tracks. The difference in the sampling rate between the model and the MPU sensor resulted in the frequency value discrepancy of the dominating steering value shown in Figure 8. The MPU sensor recorded measurements five times greater than the model per second for both tracks. Negative readings of the steering angle were caused by the mobile robot's movement in a counter-clockwise direction.

According to Figures 8 (a) and (b), the mobile robot was typically maintained in the middle of the first track, as indicated by the dominant steering angle (-0.4) computed by the designed model and MPU sensor. On the other hand, in the second track histograms shown in Figures 8 (c) and (d), the dominant steering angles were most frequently (-0.2 and -0.4) when the mobile robot was performing the short inside turns denoted by figure 7(A). Similarly, the MPU sensor also showed that these angle values were dominant. Hence, the actual steering angle values observed by the MPU sensor were approximately equal to the estimated steering angles that were calculated by the developed model for both tracks.

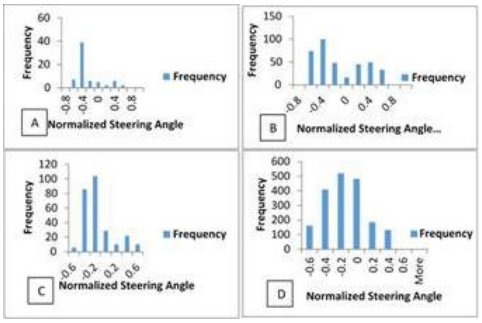

Fig. 8. Histogram of deviation from the center of track (A) Designed model, First Track (B) MPU Sensor, First Track (C) Designed Model, Second Track (D) MPU Sensor, Second Track

Five full laps of the second track were used to verify the model's repeatability. As shown in figure 9, the readings of the steering angle calculated by both the model and the map sensor for the experimental laps were nearly identical. This indicates that the mobile robot completed all required laps of driving on the designated track without any departure from the track.

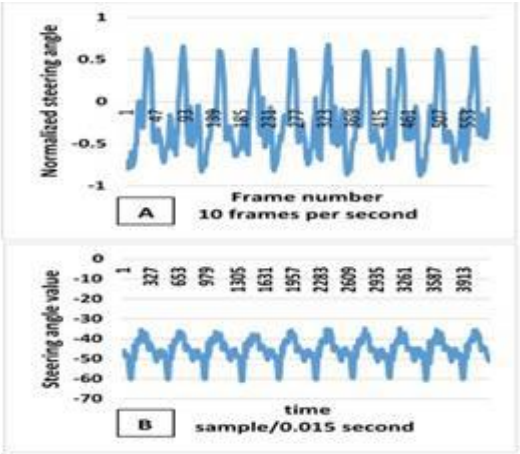

Fig. 9. A. Normalized steering angle values of designed model (s econd track, five full laps) B. Steering angle values recorded by MPU s ensor (s econd track, five full laps)

For this research, one objective was to train the novel model to completely stop the vehicle when either traffic stop signs or pedestrians come into view in front of the mobile robot's track. The robot's throttle was lowered to approximately zero to avoid a collision once a pedestrian came into sight directly ahead of the robot's pathway as depicted in figure 10.

[14] Contains a video that demonstrates our test vehicle's performance on both of the tracks.

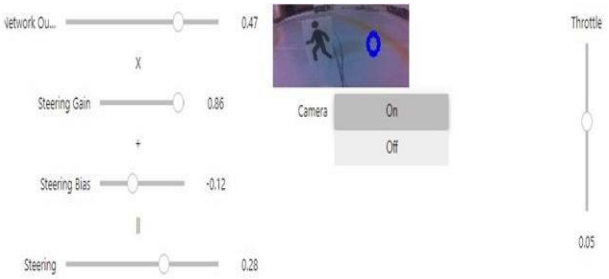

Fig. 10. Full stop at pedestrian detection by front camera

5. Conclusion

Using a lightweight model is critical to success in industrial applications that frequently demand machine learning solutions that can be implemented on computationally inexpensive and memory-demanding embedded platforms with low costs and sizes like the one

utilized in this research. In this paper, the challenging task of developing a lightweight deep neural network capable of handling all vehicle control actions, including steering wheel angle, speed, and braking, is addressed. Furthermore, the article presents an unexplored issue in computer vision: estimating steering angle and speed values solely from visual input from the camera. The stages and detailed design of the developed end-to-end model have been presented. The proposed model has been tested on two different tracks and with an MPU sensor. The presence of lightweight DNN is critical for controlling the speed of a self-driving car. Despite the fact that the camera used has a frame rate of 65 frames per second. We reduced the frame rate to 10 frames per second in order to synchronize the capture of quick movements with vehicle control commands. As a result, the lightweight DNN network reduces synchronization time. The results show that the model can predict the steering angle and speed with a high degree of accuracy and consistency. Applications such as warehouse and delivery vehicle robot cars could benefit from the end-to-end learning network's output.

References

[1] Y. Tian *et al.*, "Lane marking detection via deep convolutional neural network," *Neurocomputing*, vol. 280, pp. 46–55, Mar. 2018, doi: 10.1016/j.neucom.2017.09.098.

[2] S. Z. Yong, D. Yershov, E. Frazzoli, B. Paden, and M. Cáp, "A Survey of Motion Planning and Control Techniques for Self-driving Urban Vehicles," *IEEE Transactions on Intelligent Vehicles*, vol. 1, no. 1. pp. 1–27, 2016.

[3] S. Grigorescu, B. Trasnea, T. Cocias, and G. Macesanu, "A survey of deep learning techniques for autonomous driving," *J. F. Robot.*, vol. 37, no. 3, pp. 362–386, 2020, doi: 10.1002/rob.21918.

[4] L. Li, K. Ota, and M. Dong, "Humanlike Driving: Empirical Decision-Making System for Autonomous Vehicles," *IEEE Transactions on Vehicular Technology*, vol. 67, no. 8. pp. 6814–6823, 2018, doi: 10.1109/TVT.2018.2822762.

[5] C. Chen, A. Seff, A. Kornhauser, and J. Xiao, "DeepDriving: Learning affordance for direct perception in autonomous driving," *Proceedings of the IEEE International Conference on Computer Vision*, vol. 2015 Inter. pp. 2722–2730, 2015, doi: 10.1109/ICCV.2015.312.

[6] M. Bojarski *et al.*, "End to End Learning for Self-Driving Cars," pp. 1–9, 2016, [Online]. Available: http://arxiv.org/abs/1604.07316.

[7] Z. Yang, Y. Zhang, J. Yu, J. Cai, and J. Luo, "End-to-end Multi-Modal Multi-Task Vehicle Control for Self-Driving Cars with Visual Perceptions," in *2018 24th International Conference on Pattern Recognition (ICPR)*, Aug. 2018, pp. 2289–2294, doi: 10.1109/ICPR.2018.8546189.

[8] M. V. Smolyakov, A. I. Frolov, V. N. Volkov, and I. V. Stelmashchuk, "Self-Driving Car Steering Angle Prediction Based On Deep Neural Network An Example Of CarND Udacity Simulator," in *2018 IEEE 12th International Conference on Application of Information and Communication Technologies (AICT)*, Oct. 2018, pp. 1–5, doi: 10.1109/ICAICT.2018.8747006.

[9] J. Jhung, I. Bae, J. Moon, T. Kim, J. Kim, and S. Kim, "End-to-End Steering Controller with CNN-based Closed-loop Feedback for Autonomous Vehicles," *IEEE Intelligent Vehicles Symposium, Proceedings*, vol. 2018-June. pp. 617–622, 2018, doi: 10.1109/IVS.2018.8500440.

[10] D. Wang, J. Wen, Y. Wang, X. Huang, and F. Pei, "End-to-End Self-Driving Using Deep Neural Networks with Multi-auxiliary Tasks," *Automot. Innov.*, vol. 2, no. 2, pp. 127–136, Jun. 2019, doi: 10.1007/s42154-019-00057-1.

[11] M. K. Islam, M. N. Yeasmin, C. Kaushal, M. Al Amin, M. R. Islam, and M. I. Hossain Showrov, "Comparative Analysis of Steering Angle Prediction for Automated Object using Deep Neural Network," in *2021 9th International Conference on Reliability, Infocom Technologies and Optimization (Trends and Future Directions) (ICRITO)*, Sep. 2021, pp. 1–7, doi: 10.1109/ICRITO51393.2021.9596499.

[12] D. P. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization," *2017 2nd IEEE Int. Conf. Cloud Comput. Big Data Anal. ICCCBDA 2017*, pp. 156–160, Dec. 2014, doi: 10.1109/ICCCBDA.2017.7951902.

[13] C. Sammut and G. I. Webb, Eds., "Mean Squared Error," in *Encyclopedia of Machine Learning*, Boston, MA: Springer US, 2010, p. 653.

[14] Y. Ghufran. "End-to-end self-driving car" YouTube, Sep 4, 2022 [Video file]. Available: https://www.youtube.com/watch?v=7HXkBrsrvOI. [Accessed: Sep 4, 2022]