Current Applied Science and Technology Vol. 22 No. 6 (November-December 2022)

Research article

Intelligent Road Tracking and Real-time Acceleration-deceleration for Autonomous Driving Using Modified Convolutional Neural Networks

Youwei Li* and Jian Qu

Faculty of Engineering and Technology, Panyapiwat Institute of Management, Nonthaburi, Thailand

Received: 1 October 2021, Revised: 25 January 2022, Accepted: 16 March 2022

DOI: 10.55003/cast.2022.06.22.013

Keywords

autonomous driving;

raspberry pi;

deep learning;

road tracking;

convolutional neural networks;

PBLM-CNN21

Abstract

Neural network is one of the most widely used method in autonomous driving. Current researchers use only steering angle to train artificial neural networks, ignoring the importance of acceleration and deceleration for autonomous vehicles. We used an intelligent driving platform built with the Raspberry Pi 4 Model B, a front wide-angle camera, and a 1:16 scale model car to achieve real-time acceleration and deceleration while performing road tracking. Existing models cannot learn steering angle and throttle values well. This research proposed a novel architecture CNN model (PBLM-CNN21) to achieve real-time acceleration and deceleration while achieving road tracking. The PBLM-CNN21 model can learn steering angle and throttle value. The training loss value of our proposed PBLM-CNN21 model was 35% lower than the current TDD model, and the stability of our proposed road tracking model was 82% greater than that of the current TDD model. Furthermore, we tested the impact of different hyper-parameters on training model loss and road tracking performance. In addition, we also tested the effectiveness of varying lighting conditions and speed ratios on road tracking performance. The PBLM-CNN21 model proved more robust than the existing TDD models. Moreover, the PBLM-CNN21 model achieved road tracking under different lighting conditions and was more suitable for high-speed ratios.

1. Introduction

Autonomous driving is a critical task in automotive innovation technology and has become a research hotspot. Road tracking is the primary task of autonomous driving, and we implemented the autonomous driving task in a small-scale simulated car driving environment. The first step in the

*Corresponding author: Tel.: (+66) 0966033973
E-mail: jianqu@pim.ac.th

road-tracking mission was to build an intelligent driving platform. There are three main components to be achieved in existing studies on autonomous driving cars using toy cars, such as Karni *et al*. [1] and Bae *et al*. [2]. Although studies based on toy cars are highly cost-effective, the differences between toy cars and real cars are too significant. Thus, toy cars can hardly replicate the actual driving situation. We proposed the use of a 1:16 model car to solve this problem. The steering and driving of the scale model car can replicate the driving system of an actual car to a certain extent. Then there is the section about sensors. Existing autonomous driving researches used multiple sensors. In the studies of Karni *et al*. [1] and Lee and Lam [3], the researchers used a combination of cameras and ultrasonic sensors. In the studies of Du *et al*. [4] and Zang *et al*. [5], a variety of cameras and lidar were employed. Although current works use multiple sensors, humans drive cars utilizing only sight and hearing. Therefore, it is essential to train an agent to achieve autonomous driving using fewer sensors. We proposed to use only one camera as the environment perception sensor to achieve road tracking. The computing platform is another vital component. The Arduino, Raspberry Pi, NVIDIA Jetson TX2 platforms can commonly be seen in the existing research. For example, in the study of Yuenyong and Jian [6], they used the Arduino computing platform. However, the computing power of Arduino is insufficient, and an additional computer is needed for calculation, so the self-driving car is not a standalone agent. In the studies of Do *et al*. [7] and Lee and Lam [3], they used an intelligent driving platform that combined the Raspberry Pi and Arduino platforms. However, the computation and execution of these two methods are separate and are not standalone agents. With the development of an embedded computing platform, Raspberry Pi is now fully capable of loading CNN models for control applications. We proposed using the latest version of the Raspberry Pi 4 Model B as a computing platform.

When implementing road tracking, most researchers used a CNN model. In the study of Rausch *et al*. [8], they proposed a CNN model consisting of three convolutional layers, two pooling layers, and a fully-connected layer. Lin *et al*. [9] proposed a more complex CNN model consisting of five convolutional layers and four fully-connected layers. They experimented with simulated scenarios in a virtual environment of the computer, proving that their model could achieve road tracking. However, the computer-simulated virtual environment is different from the real environment, so more studies use a small-scale intelligent driving platform and a custom track to test road tracking. In the studies of Bechte *et al*. [10] and Do *et al*. [7], they used the same CNN model which consisted of five convolutional layers and four fully-connected layers. They successfully used this neural network to enable their small-scale intelligent driving platform to implement road tracking on their custom track. In all the above studies, research output was only concerned with steering angle. However, the steering angle and the speed significantly affect road tracking performance. Therefore, we add the throttle value (proportional value of speed) to the CNN model for learning to achieve real-time acceleration and deceleration while performing road tracking.

Furthermore, we proposed a CNN model with a novel architecture based on a parameter-based layer modification method to achieve road tracking and real-time acceleration and deceleration. After adjusting the network architecture of the model and the parameters of each layer, we proposed a novel architecture CNN model (PBLM-CNN21) to achieve road tracking. We employed Keras Linear and Keras Categorical as the base deep learning models in our proposed PBLM-CNN21 architecture.

One of the most commonly used neural networks in autonomous driving is the convolutional neural network. This research presents the critical layers of Convolutional Neural Networks. At the same time, we also analyzed and summarized the neural network models and experimental results in existing autonomous driving research. Finally, we introduce an end-to-end approach to autonomous driving and propose a novel method to achieve and improve road tracking performance.

Since the development of deep learning, many neural networks have been proposed. We can regard the neural network as a black box in which we can fit arbitrary functions. We can get the desired Y when a specific X is given as long as we have enough training data. The architecture diagram is shown in Figure 1.

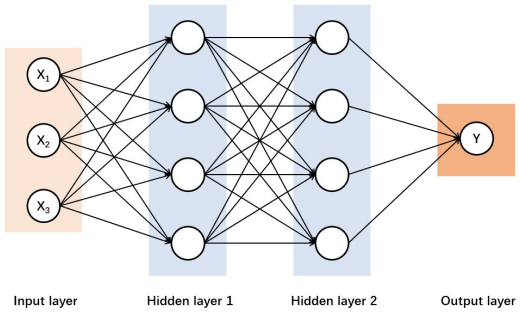


Figure 1. Neural network architecture diagram

A neural network commonly used in autonomous driving is the Convolutional Neural Network [11]. Convolutional Neural Networks (ConvNets or CNNs) are derivatives of the Multilayer Perception (MLP). They were developed from the research of the physicists Hubert and Wiesel on the visual cortex of cats. The architecture of the visible cortex cells is very complicated, and these cells can keenly perceive the input picture's sub-regions. The CNN model can be widely used in its local connection and weight sharing method. First, CNN reduces the number of weights to make the network easy to optimize. Second, it reduces the complexity of the model and reduces the risk of over-fitting. Finally, when the input of the network is an image, the effect is more pronounced.

CNN can directly use images as input, avoiding the feature extraction and data reconstruction needed in traditional recognition algorithms. In the process of two-dimensional image processing, a neural network can extract the color, texture, shape, and other image features by itself. Automatic feature extraction gives CNNs significant advantages in processing two-dimensional images and improving computational efficiency. CNNs are deep neural networks with convolutional structures and multi-layer supervised learning neural networks. The central network architecture of CNN includes convolutional layers, pooling layer, and fully-connected layer.

The convolutional layer [12] mainly comprises filters and convolved features. The term ‘convolution’ comes from the matrix convolution operation it performs. We can see the specific execution process of the convolution operation in Figure 2. For example, suppose we need to process a gray-scale image size 6 × 6 pixels, and a 4 × 4 matrix is also generated. Then, we only need to multiply and sum the elements corresponding to the two orange regions in Figure 2 to get the output result. According to this method, let the 4 × 4 matrix sweep the entire gray-scale image, and finally, a 3 × 3 matrix can be obtained. The calculation results are shown in Figure 2. In CNN, we use the 4 × 4 matrix as a filter, and the resulting 3 × 3 matrix is called the Convolved Feature, and the size of each pixel moved is called the stride. In actual training, CNN will automatically adjust the filter matrix through learning. Thus, the extracted features will increase as the number of filters increases.

Pooling is also called Spatial Pooling [13]. Its primary function is to condense the convolution feature matrix to reduce the dimensionality of the feature space while retaining critical information. The most common pooling methods are maximum pooling and average pooling [14]. The method uses the maximum value of the fundamental input part for maximum pooling, and it produces a calculation example, as seen in Figure 3. The method then uses the average value of the

fundamental input part for average pooling, and generates a calculation example, as seen in Figure 3. The convolutional features obtained by using the maximum pooling method are relatively better.

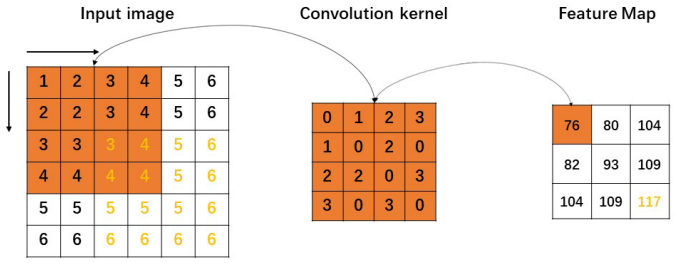


Figure 2. The result of the convolution operation

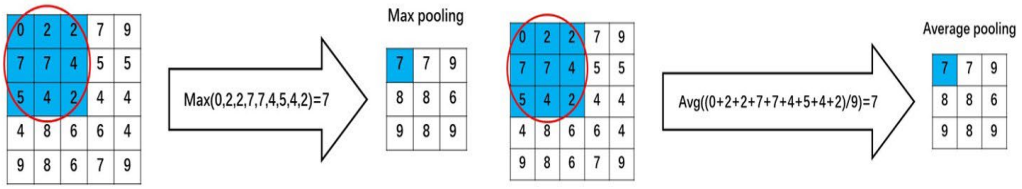


Figure 3. The calculation example of max-pooling and average pooling

The fully-connected (FC) layer [15] usually uses a multi-layer perceptual neural network, and its primary function is to classify the input image using the convolutional features extracted by the convolutional layer and the pooling layer. In an FC layer, the output of the last convolutional layer is usually flat and connects each node of the current layer with the node of the next layer. Thus, all neurons between the two layers have weights to reconnect, and the FC layer is at the tail of the convolutional neural network.

CNN is a deep learning algorithm. First, compared with other algorithms, CNN requires much less preprocessing. Second, it shares the convolution filter and has powerful processing capabilities for high-dimensional data. Third, CNN does not need to manually extract features, as good classification results can only be achieved with training weights.

In the early days of research, it was not possible to rely on a single camera for environment perception. Zang *et al*. [5] used a car with a camera, a global positioning system (GPS), and an intelligent RP lidar (RP lidar is a lidar that can scan 360 degrees). The driving platform actualizes the task of autonomous driving. In the studies of Karni *et al*. [1], Lee and Lam [3], Du *et al*. [4], and Zang *et al*. [5], sensors with more than one camera were used. A multi-sensor driving platform is not a standalone agent, and it makes sense to use fewer sensors for road tracking. The initial study used Arduino as the computing platform, but its computing power was limited. In the study of Yuenyong and Jian [6], Arduino was used as the computing platform, but its computing power was limited. In their study, it was necessary to connect Arduino and computer using Bluetooth. The computer calculated the results and sent instructions to Arduino, and then Arduino controlled the car. Thus, the whole intelligent platform was not a standalone agent. The above problems were solved by Do *et al*. [7] and Lee and Lam [3]. They used two computing platforms, Raspberry Pi and Arduino. Raspberry Pi was used to calculate and issue control commands, while Arduino received commands to control the car. Although such an intelligent driving platform appeared to be

independent, its computing and control sections were still separated, and the system was still not standalone. They use multiple computing platforms because of the lack of computing power. However, Bechtel *et al*. [10] compared the performance of Raspberry Pi 3 B, Intel UP, and NVIDIA Jetson TX2 on deep learning tasks. The results show that if deep learning was required on a computing platform, only the NVIDIA Jetson TX2 could be implemented, and the NVIDIA Jetson TX2 was not recommended for deep learning because the training data they could afford was too small and the training efficiency was low. The current autonomous driving research mainly adopts an end-to-end control method, and the entire training process is completed on other computers with more powerful computing power. Even though they cost about the same as Raspberry Pi 3 B, existing embedded computing platforms are sufficient to support real-time control applications based on CNN models. Furthermore, the more powerful Raspberry Pi 4 Model B has also been released, and as the price is not high, the Raspberry Pi 4 Model B is the better choice for a computing platform.

Behavioral learning is now most commonly used in self-driving cars. Its working principle is to learn the mapping relationship between the first-view image of the car and the steering angle through a neural network. Existing CNN models can already implement behavior learning well. Rausch *et al*. [8] and Lin *et al*. [9] proposed two CNN models, which could achieve the road tracking task in the simulation environment, but the simulation environment ignored the noise of the real environment, so the testing in the real environment was necessary. In the study of Bechte *et al*. [10] and Do *et al*. [7], they used a CNN model of the same network architecture and intelligent driving platform to achieve road tracking on their custom track. We summarized the models from the above studies in Table 1. We found that various CNN models could be trained. The input of the neural network model is all pictures, and the difference is the size of the input picture. The output section is the steering angle. However, genuinely autonomous driving is not a car driving at a constant speed. It makes sense to add throttle value to the model training section. We recorded the corresponding throttle values while collecting the data and training the proposed novel architecture CNN model (PBLM-CNN21) together with the steering angle. Then, the intelligent driving platform was able to achieve real-time acceleration and deceleration while performing road tracking.

The end-to-end control method [8, 16] is a typical vision-based automatic driving method. In the entire field of autonomous driving, the end-to-end approach means that cars collect signals directly through sensors. These signals include pedestrians, obstacles, signal lights, which are then uniformly input into the neural network system. Output instructions are closely related to control so that the car can proceed to the next step. As early as 1989, Pomerleau used neural networks to predict steering commands based on input simulated road images. In 2016, Nvidia disclosed its end-to-end deep learning technology for self-driving cars. It trained a CNN to directly map the raw pixels of a single front camera to steering commands. The end-to-end control system can optimize all processing steps, so it has a wide range of applications, covering ordinary lanes, highways, and forks.

After the CNN processes the visual information through the convolutional layer, it will give a steering angle based on the previous learning experience through the FC layer. Under the framework of end-to-end autonomous driving, sufficient training data needs to be provided for the CNN to be fully trained. The way to collect the data is to allow the car with a camera to drive in the environment where it needs to be driven and get the first-view pictures taken during the driving process with the corresponding throttle value. The end-to-end controlled automatic driving system is shown in Figure 4.

The end-to-end driving method based on deep learning operates in a way that is similar to human thinking. The process of humans driving a car is also a process of accumulation of experience. Vision and hearing are equivalent to sensors, the brain is equivalent to the decision-making unit, and the hands and feet are equivalent to the execution unit. If the data obtained by these three units is correct, the driving process will become smooth and safe. If the sensor has the correct

input in automatic driving, the computer can train a good decision-making model, and the actuator will have the corresponding corrective action.

Table 1. Existing research related to road tracking

Network	Reference	Network architecture		Parameters	Experiment
6-layer CNN	[8]	Input	(100 × 190 × 3)	1183K	Virtual simulation (no intelligent driving platform)
		Conv1	(20, 2 × 2)		
		Pooling	(5 × 5)		
		Conv2	(48, 2 × 2)		
		Pooling	(2 × 2)		
		Conv3	(64, 3 × 3)		
		FC	(500)		
		Output	steering angle		
9-layer CNN	[9]	Input	(66 × 208 × 3)	1060K	Virtual simulation (no intelligent driving platform)
		Conv1	(3, 5 × 5)		
		Conv2	(24, 5 × 5)		
		Conv3	(36, 5 × 5)		
		Conv4	(48, 3 × 3)		
		Conv5	(64, 3 × 3)		
		FC1	(100)		
		FC2	(50)		
		FC3	(10)		
		FC4	(1)		
		Output	steering angle		
	[10]	Input	(66 × 200 × 3)	250K	Road tracking on custom runways using a small-scale intelligent driving platform
		Conv1	(24, 5 × 5)		
		Conv2	(36, 5 × 5)		
		Conv3	(48, 5 × 5)		
		Conv4	(64, 3 × 3)		
		Conv5	(64, 3 × 3)		
		FC1	(1152)		
		FC2	(100)		
		FC3	(50)		
		FC4	(10)		
		Output	steering angle		
	[7]	Input	(120 × 160 × 3)	260K	Road tracking on custom runways using a small-scale intelligent driving platform
		Conv1	(24, 5 × 5)		
		Conv2	(36, 5 × 5)		
		Conv3	(48, 5 × 5)		
		Conv4	(64, 3 × 3)		
		Conv5	(64, 3 × 3)		
		FC1	(1152)		
		FC2	(100)		
		FC3	(50)		
		FC4	(15)		
		Output	steering angle		

Description: input (height × width × depth format), taking Conv1 (24, 5 × 5) as an example, Conv means convolutional layer, 24 is the number of filters, 5 × 5 is the size of the filter, taking FC1 (500) as an example, FC means fully-connected layer, 500 is the number of neurons in the fully-connected layer.

Figure 4. End-to-end controlled automatic driving system

A car with multiple sensors is not a standalone agent because humans drive primarily through sight and hearing. Thus, implementing an agent with fewer sensors will significantly improve the intelligence of the trained model. Therefore, we proposed to build an intelligent driving platform on a scale model car, using only one camera and the Raspberry Pi 4 Model B computing platform. We also needed to set up a simulated tracking environment where data collection and model testing could occur. Existing models could not handle steering angle and throttle values well, so we proposed a novel architecture CNN model (PBLM-CNN21) with six convolutional layers, four FC layers, one max-pooling layer, and ten dropout layers to learn steering angle and throttle values. The PBLM-CNN21 model achieved real-time acceleration and deceleration while performing road tracking. In addition, we conducted noise tests on the lighting source of the experimental environment and tested the road tracking performance of the model under different speed ratios.

2. Methodology

This research proposed a CNN model with a novel architecture based on a parameter-based layer modification method CNN model (PBLM-CNN21). We also added the throttle value to the input of the model and make the model train both the steering angle and the throttle value to achieve real-time acceleration and deceleration while performing road tracking. A detailed flow chart of the proposed model is shown in Figure 5.

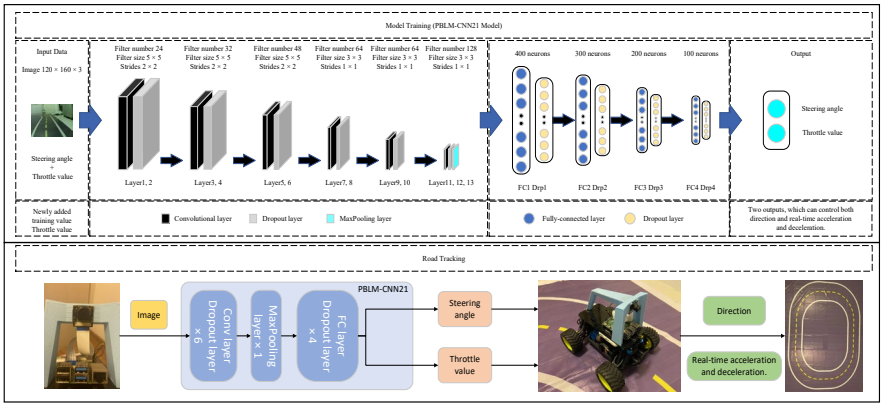


Figure 5. An overview of the proposed novel architecture PBLM-CNN21 model for road tracking and the road tracking process where the PBLM-CNN21 model implements dual-parameter control of steering angle and throttle value

As shown in Figure 5, we added the throttle value to the training input for training in the input and output module. In the neural network mechanism module, our proposed model has twenty-one layers. Compared with the base model, one convolutional layer, two FC layers, one max-pooling layer, and ten dropout layers were added.

2.1 Hardware for constructing the intelligent driving platform

The hardware connection diagram of the intelligent driving platform is shown in Figure 6. As shown in Figure 6, the wiring distribution of each critical hardware can be seen.

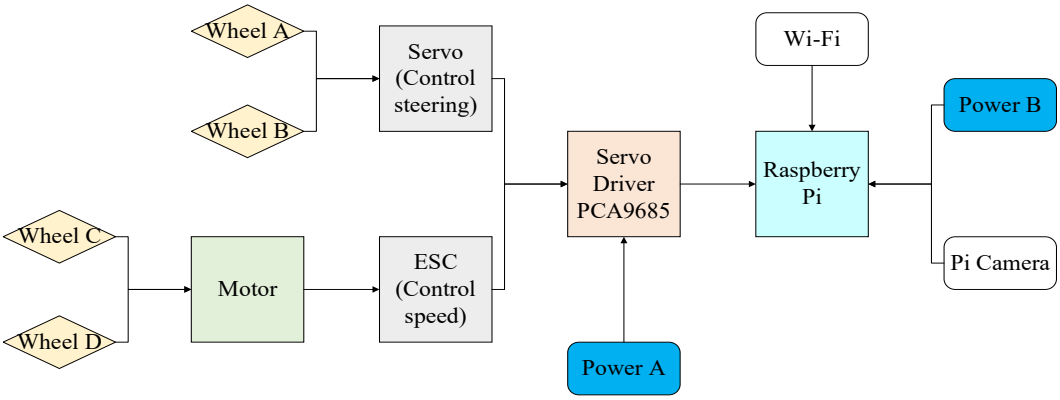
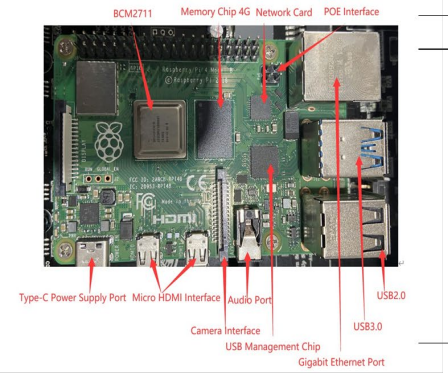


Figure 6. The hardware architecture diagram of the intelligent driving platform

2.1.1 Raspberry Pi 4 model B

Raspberry Pi [8] has always been very popular in the embedded development world. It is a cost-effective and small programmable microcomputer. In June 2019, Raspberry Pi officially launched the Raspberry Pi 4 Model B [3, 17]. It was a comprehensive upgrade of the Raspberry Pi, with soaring performance and rich accessories, supporting 4K dual screens and more extensive storage. The picture of Raspberry Pi 4 Model B is shown in Figure 7.



Item	Explanation
SOC	Broadcom BCM2711
CPU	64bit 1.5GHz Quad-core
GPU	Broadcom VideoCore VI@500MHz
RAM	4G
Network access	Wi-Fi, Bluetooth5.0
USB	USB2.0×2 USB3.0×2
Size	235×235×150mm
USB Type-C	Power supply interface
Ethernet interface	On-board full-duplex Gigabit Ethernet interface

Figure 7. Raspberry Pi 4 Model B [3, 17] and detailed parameter list

The Raspberry Pi 4 Model B uses a quad-core 64-bit ARM CortexA72 CPU, and the model is Broadcom BCM2711 SoC. The primary frequency is up to 1.5GHz, contains two USB 2 ports, two USB 3 ports, and is powered by a USB-C port. In addition, it has a Gigabit Ethernet interface and a headphone jack, two micro-HDMI ports, and supports two 4K displays. The detailed parameters of Raspberry Pi 4 Model B are shown in Figure 7.

Raspbian is an open-source operating system that can run on the Raspberry Pi. This system is an operating system based on the Linux version of Debian so that Linux-based programming can run on the Raspberry Pi platform. The computational power of Raspberry Pi 4 Model B for processing machine learning tasks is more than four times that of Raspberry Pi 3B+ [18, 19].

2.1.2 Wild-angel camera

As the only environmental perception sensor, the choice of camera is essential. First, we needed to determine the size of the input image. In the existing research, the image of 120 × 160 pixels is the most considerable input, which means that more information could be obtained. Considering that the intelligent driving platform could obtain more information, we needed a camera with a broader viewing angle. We chose to use a wide-angle camera with 5M pixels. The diagonal field of view of this camera is 160°, the maximum static picture resolution that can be obtained is 2592 × 1944 pixels, the size of the entire camera is 25 mm × 24 mm × 17mm (length × width × height).

When designing the position of the camera on the intelligent driving platform, considering that the human driving perspective was forward-looking, we put the camera in the front. At the same time, considering that the change of camera position might cause unnecessary noise to the CNN model, we decided to fix the camera position. After experimentation, as shown in Figure 8 (a), we ensured that the map occupied 70% of the entire image. We also measured the angle α between the camera and the floor to be 35°, as shown in Figure 8 (b).

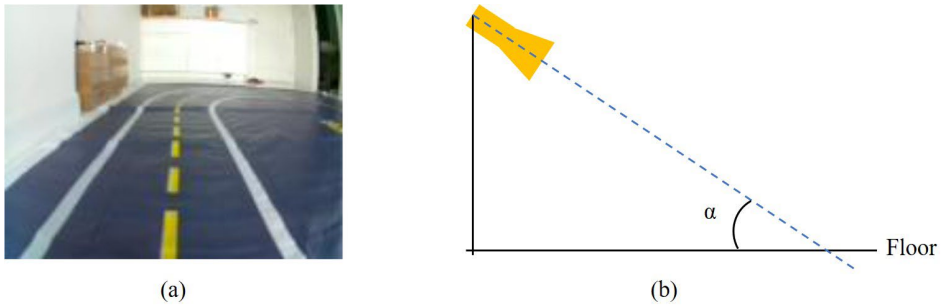


Figure 8. (a) Live view of the front camera, (b) Schematic diagram of the angle between the camera and the floor

2.1.3 Building an intelligent driving platform

Using a scale model car with independent computational power is a good choice. Donkey Car [20, 21] is an accessible open-source autonomous remote control model car project for learning the practical application of deep learning and computer vision in robotic vehicles. It does not take long to set up and get started. We used its existing neural network computer vision library to achieve road tracking. The hardware needed to assemble the RC car is shown in Figure 9.

Item	Quantity
Raspberry Pi 4B	1
RC 380 brushed motor	1
E6001 Servo	1
500W HD wide-angle Camera	1
Ni-MH SC 7.2V 1100mAH battery	2
USB Battery with micro-USB cable (5V 2A)	1
Brushed ESC	1
Servo Driver PCA 9685	1
3D Printed roll cage and top plate	1

Figure 9. Assembled intelligent driving platform and required hardware

After preparing the accessories in Figure 9, we assembled the intelligent driving platform. The assembled intelligent driving platform size was 251 mm × 216 mm × 174 mm (length × width × height). The assembled intelligent driving platform is shown in Figure 9. The Ni-MH SC battery is used as Power A and the USB battery is used as Power B, as shown in Figure 6.

2.2 Scenarios for simulation tracking

2.2.1 Custom track

The establishment of a custom track is mainly used for data collection and model testing of the road tracking experiment. Therefore, the custom track uses white as the inner and outer boundaries, and we added a yellow line in the middle of the track. The size of the entire track was 3 m × 6 m, and the lane width was 25 cm. The design drawing is shown in Figure 10.

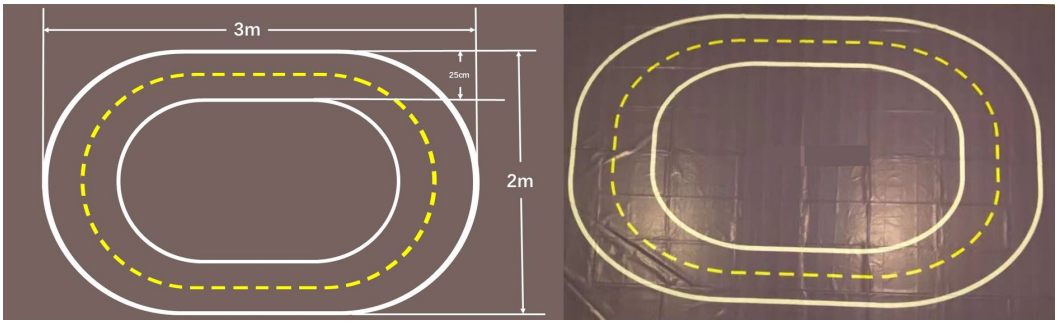


Figure 10. Track design drawing and the completed custom track

Because of COVID-19, we could only conduct experiments at home. To reduce the noise of the experiment, we built a custom track on a tarpaulin. One problem with tarpaulins is that they reflect light. Under the lighting conditions of a window-opened room, we took an overview of the entire map from two different locations and recorded camera images of the intelligent driving platform at the corresponding locations. As shown in Figure 11, a section of the road could not be seen clearly. We set the lighting situation and room conditions to solve the above problems.

(a) (b) (c) (d)

Figure 11. An overview of the actual map and camera images on the driving platform at the corresponding location in the lighting conditions of a window-opened room. The positions of (a) and (b) are the same, and the positions of (c) and (d) are the same.

2.2.2 Lighting and room condition

We solved the reflection problem of the tarpaulin by the following methods. We drew the curtains in the room, closed the door, and used the two lamps on the ceiling as lighting sources. This ensured that the environment used throughout the experiment was consistent so that if problems arose during the study, the problems brought about by the experimental environment could be minimized. After the setup was complete, we also captured the map overview and images from the intelligent driving platform camera at the corresponding location from two angles. As shown in Figure 12, we can see that the problem of reflection was solved very well.

(a) (b)

Figure 12. (a) An overview of the actual captured custom track and camera images on the driving platform at the corresponding location under the lighting conditions of a window-closed room, (b) A simulated tracking scene with lighting and room setup complete

2.3 Parameter based layer modified CNN model

When designing the new model architecture, we modified the number of layers of the model and the parameters of each layer. The experiment was found that reducing the number of model layers reduced the loss value, but road tracking did not improve. It was necessary to deepen the depth of the model. We chose to add one convolutional layer and two FC layers. As the depth of the model increased, so did the number of parameters that were trained. We added a dropout layer after each layer to prevent overfitting during training. At the same time, although the position of the camera was fixed, the car may be unstable due to acceleration and deceleration during driving, so we added

a max-pooling layer between the convolutional layers and the FC layers. After choosing an appropriate network architecture, we optimized the model by modifying the number of filters, filter size, and stride. Finally, we proposed a novel parameter-based layer-modified CNN model with six convolutional layers, four FC layers, ten dropout layers, and one max-pooling layer. This novel architecture parameter-based layer modification model had 21 layers, naming the model PBLM-CNN21. The PBLM-CNN21 model is able to learn the steering angle and throttle value well, thus realizing real-time acceleration and deceleration during road tracking.

2.4 Methods for evaluating the proposed model

The quality of a model was assessed in two parts. Firstly, the loss value from model training. The training loss value can reflect the quality of the model architecture to a certain extent. Secondly, the actual road tracking performance. We found that existing models performed poorly in handling steering angle and throttle values in road tracking, so we proposed a novel architecture, the PBLM-CNN21 model. The PBLM-CNN21 model can better achieve real-time acceleration and deceleration while performing road tracking by learning the steering angle and throttle values.

2.5 Experimental setup

The primary goal of the experiment was to train a model that could conduct automatic driving using an intelligent driving platform so that the car can achieve road tracking. Thus, we divided the process into data collection, model training, and model testing. The practical steps diagram is shown in Figure 13.

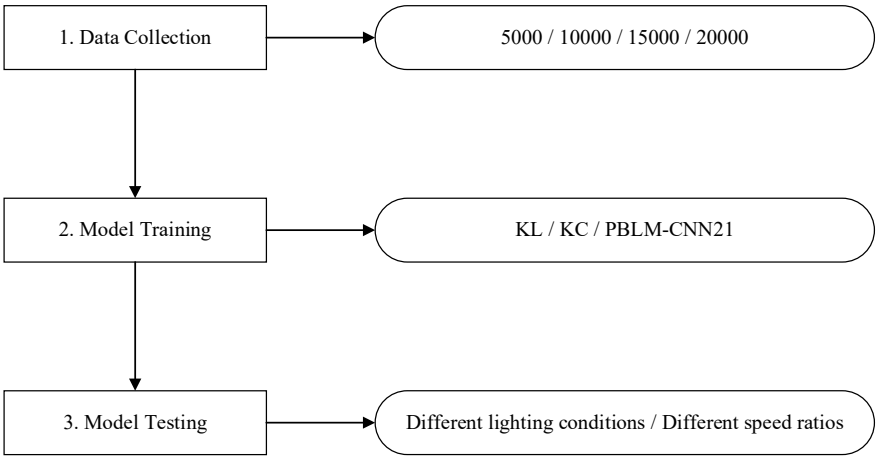


Figure 13. The experimental steps diagram

2.5.1 Data collection

The first step in this experiment is data collection. The quality of the training dataset will directly affect the quality of deep learning. Therefore, we needed to collect data by driving an intelligent driving platform on a custom track in the lighting conditions of a window-closed room. The steps for data collection are shown in Figure 14.

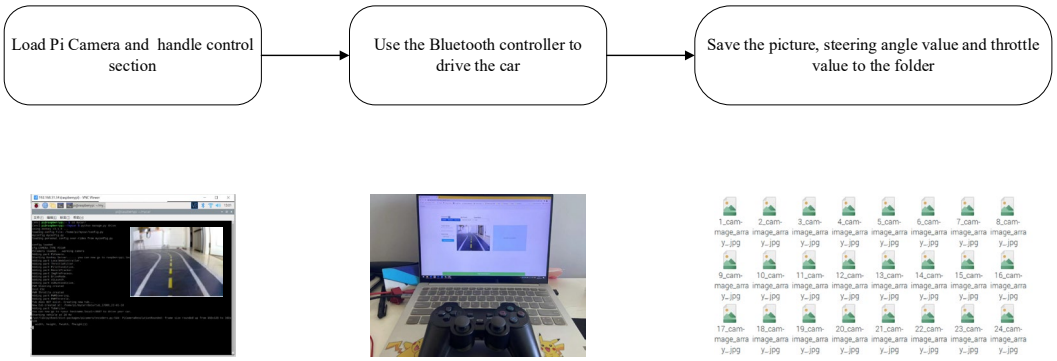


Figure 14. Data collection steps

After setting the Raspberry Pi system, we set the camera resolution to 120×160 pixels and the frame rate to 30fps through Open-Source Computer Vision Library (OpenCV, a cross-platform computer vision library). We could see the live preview of the camera in the web server, as shown in Figure 15. The method used for data collection was to drive the intelligent driving platform on a custom track with a bluetooth gamepad. When collecting data, if the output of the throttle value signal was detected, it would automatically start recording data, and if the throttle signal was lost, it would automatically stop recording data. After collecting the corresponding data, the collected data set was automatically saved to the corresponding new folder.

The collected data was saved in the corresponding folder. This folder mainly contained three types of files. Figure 16 shows an example of the collected data set. Taking the picture framed in red in Figure 16 as an example, an explanation of the files follows:

"1_cam-image_array_.jpg": This is the picture information file. It is the first-view picture of the custom track taken while driving, and the picture size is 160 × 120 pixels.

"meta.json": This is the information record file, a necessary file for deep learning.

"record_1.json": This is the information record file. The record_1.json file corresponds to the 1_cam-image_array_.jpg file with the same serial number. This file records the throttle value and steering angle information under the picture 1_cam-image_array_.jpg.

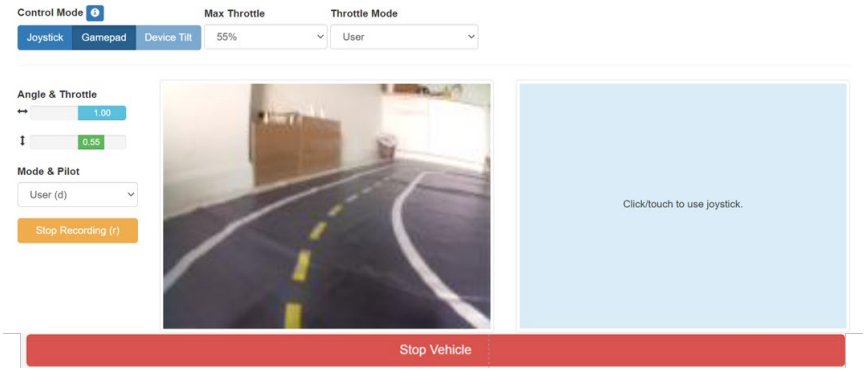


Figure 15. A web server can control the intelligent driving platform, and we can see the real-time web preview of the camera (We can now control your car from a web browser at the URL: < hostname.local of your car>:8887)

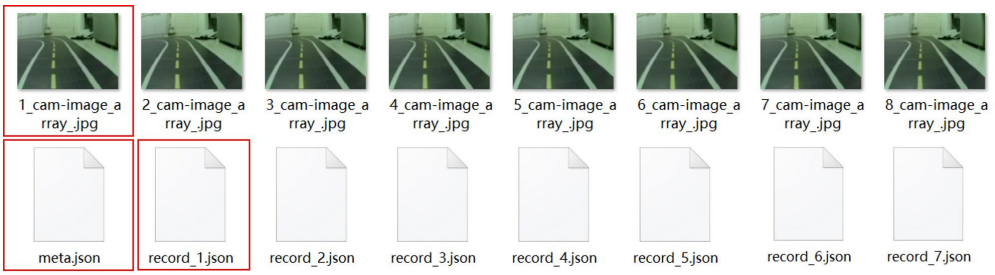


Figure 16. Example of the contents of the data set folder

In this experiment, we chose four different numbers of datasets for training. They were 5,000 pictures, 10,000 pictures, 15,000 pictures, and 20,000 pictures. To ensure the uniformity of the data, we collected 20,000 photos at a time, and then we took as many datasets as we needed from these 20,000 photos.

2.5.2 Model training

The second step was model training, which was very important. For this part, we used the model defined by the Keras high-level API. Keras and TensorFlow backend can be used together for deep learning. The model training part aimed to use a trained artificial neural network to reproduce the steering and throttle of the image seen by a camera. Considering the limited computational power of the Raspberry Pi, the Keras Linear and Keras Categorical models that performed well in a limited computational environment performed deep learning. The Keras Linear, Keras Categorical, and PBLM-CNN21 models are described in Table 2.

Table 2. Description of neural network models

Neural Network Model	Network architecture
Keras Linear	5 Conv2D layers 2 FC layers
Keras Categorical	5 Conv2D layers 2 FC layers
PBLM-CNN21	6 Conv2D layers 1 Max-Pooling layer 4 FC layers 10 Dropout layer

Keras Linear (KL) uses a neuron to output a constant value through the Keras Dense layer with linear activation. Thus, the steering and throttle have a value, and there is no limit to the output. We defined this model as a Keras Linear model containing five convolutional layers and two FC layers.

Keras Categorical (KC) decomposes steering and throttle decisions into cautious angles and then uses classification cross-entropy to train the network to activate a single neuron for each steering and throttle selection. Therefore, there are boundaries between input and output. We defined this model as a Keras Categorical model containing five convolutional layers and two FC layers.

PBLM-CNN21 was our proposed novel architecture CNN model. The PBLM-CNN21 model is a 21-layer neural network. To the basic CNN models, one convolutional layer and two FC layers, one max-pooling layer, and ten dropout layers were added, and the parameters in the model

were also adjusted. The PBLM-CNN21 model thus became more suitable for processing data with steering angle and throttle value.

For deep learning, we needed to configure the training environment on the computer. After completing the environment configuration, we were able to start model training. The detailed steps of model training are shown in Figure 17.

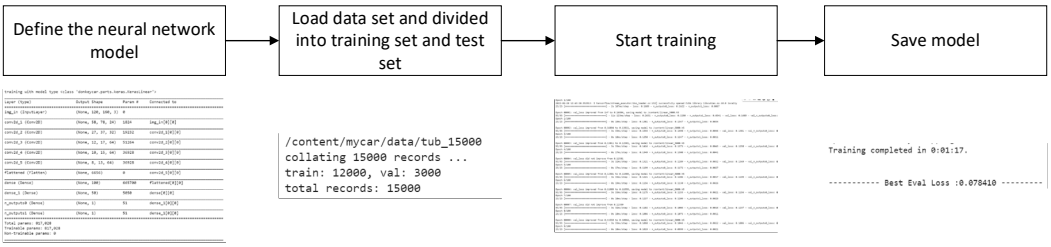


Figure 17. Model training steps

During model training, we divided the dataset into two parts, one was the training set, and the other was the test set. The split ratio depended on the amount of data. If the amount of dataset was large, we used a ratio of 9: 1 to split the dataset, and when the amount of data was small, we used a ratio of 8: 2 to split the dataset.

We selected two models, four datasets, and three different batch sizes for training in the model training section. The model training setup is shown in Table 3. During the model training process, we recorded the loss value and epoch.

Table 3. Model training setup

Neural Network	Datasets	Batch size
Keras Linear Keras Categorical PBLM-CNN21	5,000 10,000 15,000 20,000	64 128 256

2.5.3 Model testing

The model files were trained with different neural networks under different datasets, batch sizes were generated in the model training part, and the training loss value was also obtained. However, we could not judge the quality of the model based on the training loss value alone. So next, we started the last step of the experiment, loaded the trained models onto the intelligent driving platform, and tested models on the custom track. The entire test environment was the same as when the data was collected. The testing was carried out in an enclosed room, conducted both at day and night, and done with two lamps as lighting sources. This setup minimized possible noise in the environment.

We used VNC to remotely connect to the Raspberry Pi and load our trained model and camera in the model testing. The model calculated the picture obtained from the camera output, the steering angle, and the throttle value to achieve real-time acceleration and deceleration while implementing road tracking. Firstly, we tested whether the different models could achieve real-time acceleration and deceleration while performing road tracking. At the same time, to test the influence of lighting conditions on the road tracking performance of the model, we chose to test under the

lighting conditions of a window-opened room and a window-closed room. Although the model could achieve real-time acceleration and deceleration, the speed used to collect data was relatively slow. It was necessary to test the performance of road tracking under different speed ratios. The difference between the speed ratio and the acceleration and deceleration was that the speed ratio limited the maximum driving speed of the intelligent driving platform. In contrast, acceleration and deceleration adjusted the throttle ratio during the driving process. We changed the maximum speed of the intelligent driving platform to test the effectiveness of road tracking under different speed ratios. We found that when the speed ratio is lower than 80%, the intelligent driving platform could not move, so we chose different speed ratios of 80%, 90%, and 100%. We called these three ratios of the vehicle speed: slow, normal, and fast, respectively. In order to analyze the impact of the speed ratio on road tracking, we recorded the number of times the intelligent driving platform touched the white line during the road tracking process of each model in the experiment. An example of the partial model road tracking process is shown in Figure 18.

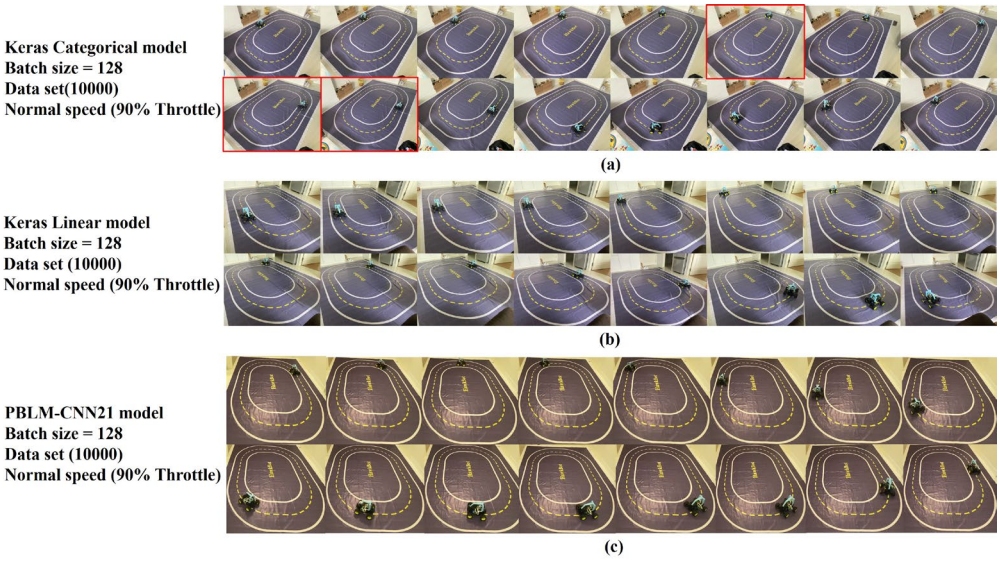


Figure 18. (a) Keras Categorical model road tracking (The red boxes in the diagram show that the car is touching the white line in road tracking It means that the road tracking of the Keras Categorical model was not good), (b) Keras Linear model road tracking, (c) PBLM-CNN21 model road tracking

3. Results and Discussion

In this chapter, we selected one of the state-of-the-art automatic driving models for comparison, and this model was proposed by Truong-Dong Do *et al*. [7]. This model is hereby after called as TDD model. We compared the performance of four models, the KL, KC, PBLM-CNN21, and TDD models in the version of model training and actual road tracking. We divided the experimental results into two sections for discussion. Firstly, the training loss values of different models and the impact of different hyper-parameters on model training loss were discussed. Secondly, the actual road tracking performance after loading the trained model onto the intelligent driving platform and

the effects of different hyper-parameters, lighting conditions, and speed ratios on actual road tracking performance were discussed.

3.1 Analysis of loss values and related hyper-parameters obtained from model training

3.1.1 The influence of different neural network models on training loss

In the initial stage of the experiment, we used 15,000 photos to train the model. The model training loss is shown in Table 4.

As shown in Table 4, we found that the loss of the PBLM-CNN21 model was smaller than other models. The PBLM-CNN21 model was trained with more parameters, but the generated model file was smaller than the TDD model. Furthermore, the model architecture was better. The training loss value of our proposed PBLM-CNN21 model was 75% lower than that of existing TDD models.

Table 4. Model training loss and model size

Model	Training Loss	Trained Parameters	Size
KL	0.084306	817,028	3.15M
KC	0.186804	268,311	1.06M
PBLM-CNN21	0.054834	857,954	3.33M
TDD	0.222084	267,323	6.13M

3.1.2 The influence of the different number of data sets on training loss

We analyzed the effect of four datasets on training loss when the batch size was 128. The training loss comparison chart obtained by training under different datasets is shown in Figure 19.

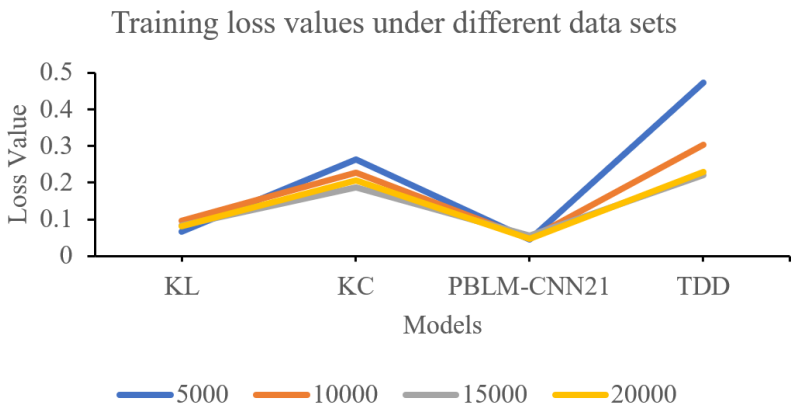


Figure 19. Training loss values under different data sets

As shown in Figure 19, the training loss value did not change linearly with increase in the number of data sets. Proportionally, when the data set was 5,000, the training loss value was relatively high, and when the data set was 15,000, the training loss value was relatively low. More sensitive to the data set was the TDD model. The data set has minimal impact on the PBLM-CNN21 model.

3.1.3 The influence of different batch sizes on training loss

Batch size is an essential parameter in machine learning. It is the number of samples selected by the computer each training time. In the experiment, we chose to train the model under three batch sizes of 64, 128, and 256, and then analyzed the influence of batch size on the training loss value. The comparison chart of the training loss value under different batch sizes is shown in Figure 20. As shown in Figure 20, batch size had little effect on the training loss value of a single model.

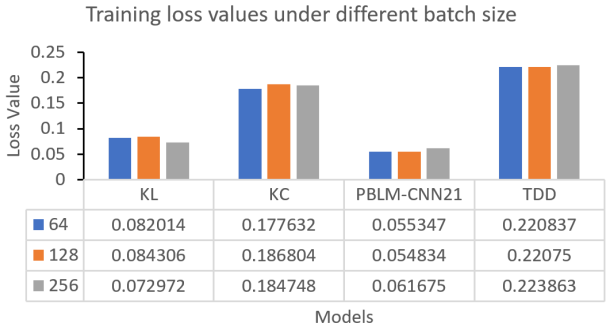


Figure 20. Training loss values under different batch sizes

3.1.4 The influence of epoch

Epoch refers to the number of times to train all data. We dedicated this research to find the minimum loss value and most efficient epoch. We used early stopping in model training. When the training loss was no longer decreasing, we performed five more training sessions. If the training loss did not drop after five training sessions, the training automatically stopped. We compared the epochs at the end of the different models, as shown in Figure 21.

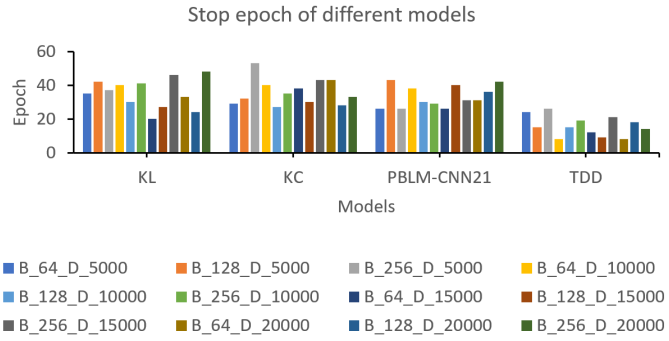


Figure 21. The early stop epoch of different neural network models (Example: B_64_D_5000 format, where B is batch size and D is Data sets.)

As shown in Figure 21, we found that the TDD model was trained with fewer epochs than the other models. Under different hyper-parameters, the epochs performed were relatively less when the batch size was 64 and the data set was 15,000. The highest number of epochs was 53, and the upper training limit was 53, ensuring the highest efficiency and negligible epoch training loss.

3.1.5 Summarizing the training loss of the model trained under different hyperparameters

The loss of the model trained under different batch sizes and different data sets is shown in Figure 22. We found that the training loss of the KL model and the PBLM-CNN21 model did not change much under different hyper-parameters. The loss value of the KC model and TDD model changed significantly after the number of data increased. It showed that the KL model and the PBLM-CNN21 model were more suitable neural network models for calculating the two parameters of steering angle and throttle value.

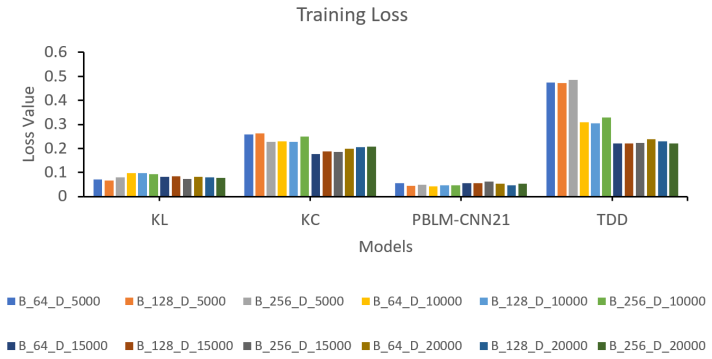


Figure 22. Comparison of loss values for training models under different hyper-parameters

3.2 Test models with the intelligent driving platform

It is not accurate to judge the quality of a model based on the training loss value. After loading the neural network models into the intelligent driving platform, we needed to further evaluate the quality of the neural network model according to the actual road tracking performance. We loaded the model trained under different hyperparameters onto the intelligent driving platform to see how its road tracking worked. In the experiment, we loaded different neural network models onto the intelligent driving platform, and each neural network model was tested on the custom track for ten laps. We record the number of times the intelligent driving platform touches the white line (failures of road tracking). Next, we evaluated the quality of the model by taking the average of the failures of road tracking (AF= average failures). Furthermore, we test the effectiveness of lighting source and speed ratios when conducting road tracking.

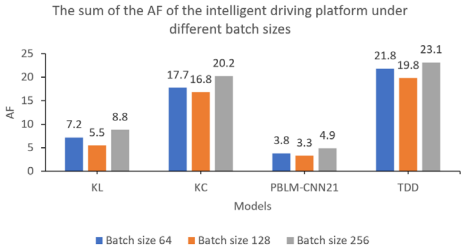
3.2.1 Road tracking performance of different models trained with different hyper-parameters

Experiments show that the KL, KC, and PBLM-CNN21 models achieved real-time acceleration and deceleration while achieving road tracking. The TDD model could only achieve road tracking and could not simultaneously accelerate and decelerate. We recorded the AF of the intelligent driving platform under different parameters in Table 5. The AF of the model under different batch sizes and data sets is shown in Figure 23.

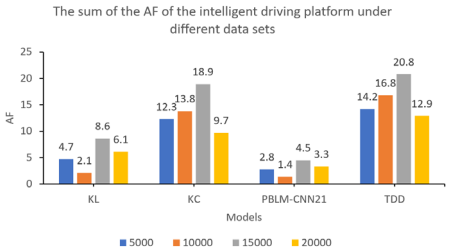
As shown in Figure 23 (a), batch size had little influence on road tracking. Figure 23 (b) shows that the number of datasets had a more significant impact on road tracking performance. As shown in Table 5, when the batch size was 128, and the data set was 10,000, the best road tracking performance was provided by the PBLM-CNN21 model. The AF of the PBLM-CNN21 model was 82% less than that of the TDD model.

Table 5. The AF of models trained with different hyper-parameters

Datasets	Batch size	AF-KL	AF-KC	AF-PBLM-CNN21	AF- TDD
5000	64	1.5	4.3	0.8	5.4
	128	1.3	3.3	0.9	3.8
	256	1.9	4.7	1.1	5
10000	64	0.9	3.7	0.8	4.6
	128	0.5	4.9	0.6	5.8
	256	0.7	5.2	0.7	6.4
15000	64	2.9	6.3	1.2	7.3
	128	2.3	5.7	1	6.3
	256	3.4	6.9	1.1	7.2
20000	64	1.9	3.4	1	4.5
	128	1.4	2.9	1	3.9
	256	2.8	3.4	1.3	4.5



(a)



(b)

Figure 23. (a) The sum of the AF of the intelligent driving platform under different batch sizes, (b) The sum of the AF of the intelligent driving platform under different data sets

We used the Raspberry Pi 4 Model B as the computing platform for the intelligent driving platform in order to implement road tracking on a custom track. There was no work delay during the road tracking process after loading the KL, KC, TDD, and PBLM-CNN21 models. The prediction during the experiment was that the average speed of the update would be 0.04ms, and the memory usage after loading the model was accounted for 68%. We used VNC to connect the computer to the Raspberry Pi. When the Raspberry Pi screen was transmitted to the computer, there may have been a screen delay, but the transfer to the screen was only to detect the motion state of the car and had no effect on the performance of road tracking. In order to better monitor the real-time picture of the car, we used the Xiaomi AC2100 router as the medium in the information transmission part. This router supports a maximum bandwidth of 1000Mbps, which was enough to transmit information. The battery life of the intelligent driving platform was three hours.

3.2.2 Testing the effect of different lighting sources on road tracking

We tested the trained model under the lighting conditions of a window-opened room and a window-closed room. The experiments show that all the KL, PBLM-CNN21, KC, and TDD models achieved road tracking under the lighting conditions of a window-closed room. Under the lighting conditions of a window-opened room, the KL and PBLM-CNN21 models achieved road tracking, but the KC

and TDD models ran out of control when performing turning actions. Road tracking was more stable for both lighting conditions under the lighting conditions of a window-closed room.

3.2.3 The influence of different speed ratios on road tracking

We recorded the AF of each model at different speed ratios. The details are recorded in Table 6.

Table 6. The average number of touches on the white line at different speed ratios

Datasets	Batch size	Speed ratios	AF-KL	AF-KC	AF-PBLM-CNN21	AF-TDD
5,000	64	Slow	0.4	2.3	0.2	3.3
		Normal	1.5	4.3	0.8	5.4
		Fast	3.3	5.3	1.2	5.3
	128	Slow	0.1	1.7	0.1	2.9
		Normal	1.3	3.3	0.9	3.8
		Fast	3.3	4.1	1.5	5.2
	256	Slow	0.7	3.2	0.3	2.8
		Normal	1.9	4.7	1.1	5
		Fast	4.3	5.9	2	6.3
10,000	64	Slow	0.5	2.9	0.2	3.8
		Normal	0.9	3.7	0.8	4.6
		Fast	1	4.8	0.5	5.3
	128	Slow	0.2	2.1	0.1	3.1
		Normal	0.5	4.9	0.6	5.8
		Fast	0.5	7.1	0.4	8.2
	256	Slow	0.6	3	0.5	4.9
		Normal	0.7	5.2	0.7	6.4
		Fast	0.9	8.3	0.7	8.9
15,000	64	Slow	0.6	3.7	0.3	4.5
		Normal	2.9	6.3	1.2	7.3
		Fast	5	10.2	2.3	11.2
	128	Slow	0.5	2.9	0.3	3.7
		Normal	2.3	5.7	1	6.3
		Fast	4.7	9.5	2.6	10.7
	256	Slow	0.8	3.9	0.2	4.8
		Normal	3.4	6.9	1.1	7.2
		Fast	5.1	10.9	2.8	11.1

Table 6. The average number of touches on the white line at different speed ratios (continued)

Datasets	Batch size	Speed ratios	AF-KL	AF-KC	AF-PBLM-CNN21	AF-TDD
20,000	64	Slow	0.8	2.7	0.4	3.2
		Normal	1.9	3.4	1	4.5
	128	Slow	0.6	2.1	0.9	3.6
		Normal	1.4	2.9	1	3.9
		Fast	6.8	6.2	5.6	7.3
	256	Slow	0.9	2.6	0.4	3.7
		Normal	2.8	3.4	1.3	4.5
		Fast	8.5	7.2	3.5	8.4

1) The road tracking effect of model types under different speed ratios

The AF of each model under different speed ratios is shown in Figure 24. As shown in Figure 24, we found that the AF of the four neural network models also increased as the speed increased. The model with the least number of AF was the PBLM-CNN21 model.

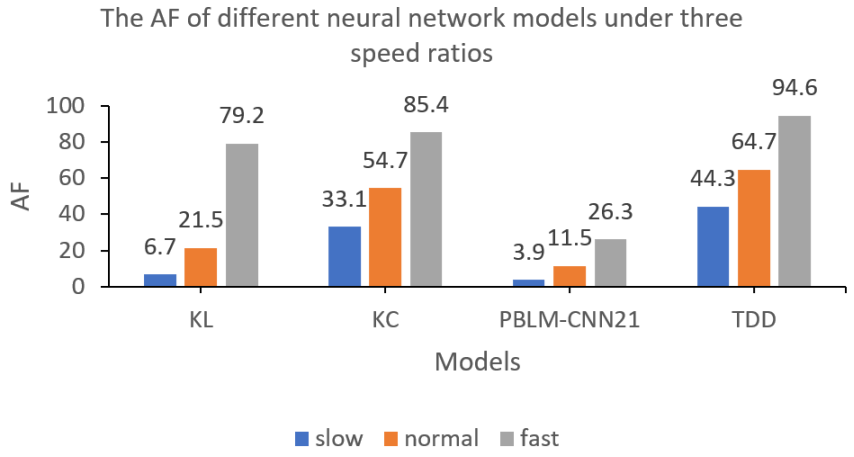


Figure 24. The AF of different neural network models under different speed ratios

2) The road tracking effect of hyper-parameters under different speed ratios

The sum AF of the neural network models trained at different speed ratios with different hyper-parameters is shown in Figure 25. As shown in Figure 25, we found that batch size and dataset had less impact on road tracking performance at the same speed ratio and under different speed ratios; the larger the speed ratio, the worse the road tracking performance.

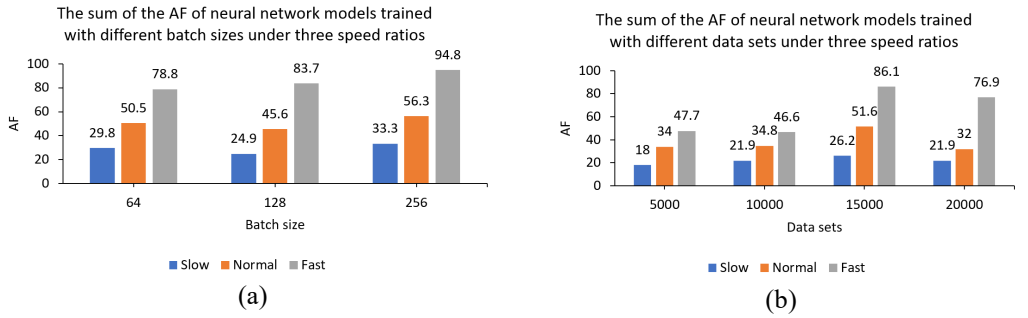


Figure 25. (a) The sum of the AF of neural network models trained with different batch sizes under different speed ratios, (b) The sum of the AF of neural network models trained with different data sets under different speed ratios

3) Comparison of all trained models

The AF of the neural network models trained with different hyperparameters at different speed ratios is shown in Figure 26. As shown in Figure 26, we found that the PBLM-CNN21 model performed better than other road tracking models and was more stable under different speed ratios.

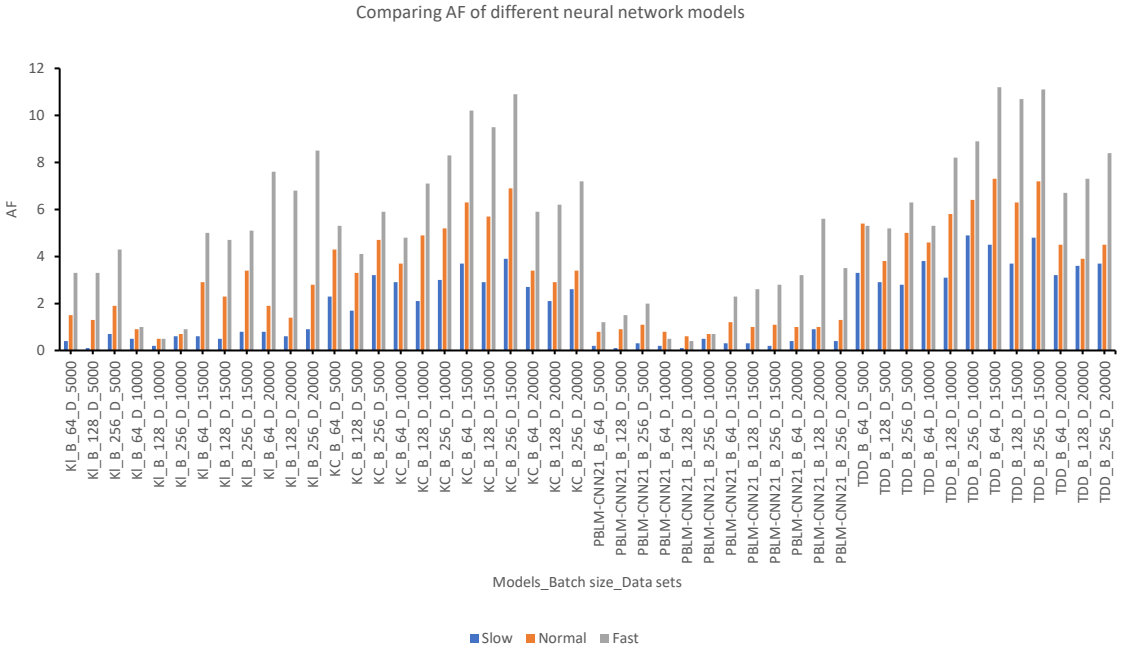


Figure 26. Comparing AF of different neural network models

3.3 Comprehensive training loss and real road tracking performance to evaluate the model

A comparison of training loss values and AF of different neural network models under different hyper-parameters is shown in Figure 27. As shown in Figure 27, we found that the model for road tracking with the most negligible training loss and the lowest number of AF was the PBLM-CNN21 model. The graph also shows that the PBLM-CNN21 model had a better model architecture and better real-world road tracking performance.

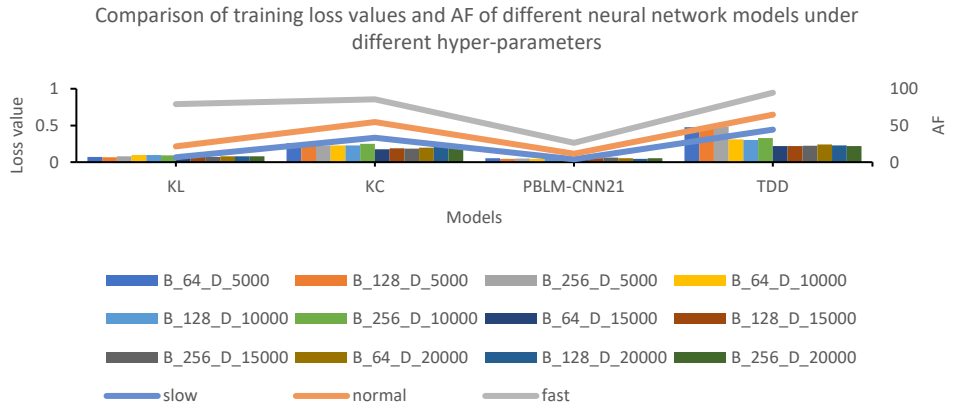


Figure 27. Comparison of training loss values and AF of different neural network models under different hyper-parameters

4. Conclusions

We proposed a novel architecture CNN model to achieve real-time acceleration and deceleration while performing road tracking. We built an intelligent driving platform with only one camera in our research. The research adopted an end-to-end control method, using the raw sensor data of a single camera as the input to control the intelligent driving platform in order to achieve real-time acceleration and deceleration while achieving road tracking. We showed that existing models failed to learn two inputs, steering angle, and throttle value. We proposed a novel PBLM-CNN21 model, which proved more suitable for computing steering angle and throttle values.

During the model training section, we tested the effectiveness of different hyper-parameters. We found that the batch size had less impact on training loss, and the number of data sets had a more significant impact on training loss. We compared the KL, KC, TDD, and PBLM-CNN21 models. The PBLM-CNN21 model had a minor training loss. When the batch size was 64, and the data set was 10,000, the model had the most negligible training loss of 0.043028. The training loss value of our proposed PBLM-CNN21 model was 75% lower than that of existing TDD models. In order to ensure the efficient training of the model and obtain the most negligible training loss, the epoch value should be set at 53.

In the actual road tracking performance, the KL, KC, and PRBLM-CNN21 models achieved real-time acceleration and deceleration while achieving road tracking. On the other hand, the TDD model was only able to achieve road tracking and could not achieve real-time acceleration and deceleration. The KL, KC, TDD, and PBLM-CNN21 models were able to perform road tracking

under the lighting conditions of a window-closed room. Under the lighting conditions of a window-opened room, the KL and PBLM-CNN21 models still achieved road tracking, while the KC and TDD models had problems with running off the track when turning. The road tracking performance of the neural network model deteriorated as the speed ratio increased. The performance of our proposed PBLM-CNN21 model at the different speed ratios does not deteriorate as much as the other models. The PBLM-CNN21 model was more robust than the other models, making it more suitable for high-speed scenarios. The PBLM-CNN21 model displayed an 82% improvement in road tracking performance when compared to the TDD model.

In future work, we will continue to study the relevant neural network architecture and parameters in order to optimize the PBLM-CNN21 model. At the same time, considering that the actual environment will not be as perfect as the simulated scene, we will conduct training and testing under more complex road conditions, which will include rough ground, bumps, and obstacles. It is worth investigating how we can ensure that the model still has good road tracking performance in the face of more ambient noise. Furthermore, many computing platforms can implement the road tracking task, and we will compare the road tracking performance of different computing platforms, such as the STM computing platform.

5. Acknowledgements

The first author conducted the experiment and drafted the manuscript. The second author guided and advised the experiment and co-drafted the manuscript. The first and second authors each contributed 50% equally to this work.

The first author received scholarship support from CPALL for conducting this research in PIM.

References

[1] Karni, U., Ramachandran, S.S., Sivaraman, K. and Veeraraghavan, A.K., 2019. Development of autonomous downscaled model car using neural networks and machine learning. *2019 3rd International Conference on Computing Methodologies and Communication (ICCMC)*, Erode, India, March 27-29, 2019, pp. 1089-1094.

[2] Bae, Y., Gomez, E., Haywood, A., Lazo, J., Whitson, P. and Wang, Y., 2021. Prototyping a system of cost-effective autonomous guided vehicles. *Proceedings of the 2021 Annual General Donald R. Keith Memorial Capstone Conference*, New York, USA., April 28, 2021, pp. 138-143.

[3] Lee, K.L. and Lam, H.Y., 2021. Development of deep learning autonomous car using raspberry Pi. *Progress in Engineering Application and Technology*, 2(1), 534-548.

[4] Du, X., Ang, M.H. and Rus, D., 2017. Car detection for autonomous vehicle: LIDAR and vision fusion approach through deep learning framework. *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Vancouver, Canada, September 24-28, 2017, pp. 749-754.

[5] Zang, S., Ding, M., Smith, D., Tyler, P., Rakotoarivelo, T. and Kaafar, M.A., 2019. The impact of adverse weather conditions on autonomous vehicles: how rain, snow, fog, and hail affect the performance of a self-driving car. *IEEE Vehicular Technology Magazine*, 14(2), 103-111.

[6] Yuenyong, S. and Jian, Q., 2017. Generating synthetic training images for deep reinforcement learning of a mobile robot. *Journal of Intelligent Informatics and Smart Technology*, 2, 16-20.

[7] Do, T.-D., Duong, M.-T., Dang, Q.-V. and Le, M.-H., 2018. Real-time self-driving car

navigation using deep neural network. *The 4th International Conference on Green Technology and Sustainable Development (GTSD)*, Ho Chi Min City, Vietnam, November 23-24, 2018, pp. 7-12.

[8] Rausch, V., Hansen, A., Solowjow, E., Liu, C., Kreuzer, E. and Hedrick, J.K., 2017. Learning a deep neural net policy for end-to-end control of autonomous vehicles. *2017 American Control Conference (ACC)*, Seattle, USA., May 24-26, 2017, pp. 4914-4919.

[9] Lin, W.-Y., Hsu, W.-H. and Chiang, Y.-Y., 2018. A combination of feedback control and vision-based deep learning mechanism for guiding self-driving cars. *2018 IEEE International Conference on Artificial Intelligence and Virtual Reality (AIVR)*, Taichung, Taiwan, December 10-12, 2018, pp. 262-266.

[10] Bechtel, M.G., McEllhiney, E., Kim, M. and Yun, H., 2018. Deeppicar: A low-cost deep neural network-based autonomous car. *2018 IEEE 24th International Conference on Embedded and Real-time Computing Systems and Applications (RTCSA)*, Hakodate, Japan, August 28-31, 2018, pp. 11-21.

[11] Goldfain, B., Drews, P., You, C., Barulic, M., Velev, O., Tsiotras, P. and Rehg, J.M., 2019. Autorally: An open platform for aggressive autonomous driving. *IEEE Control Systems Magazine*, 39(1), 26-55.

[12] Bayar, B. and Stamm, M.C., 2016. A deep learning approach to universal image manipulation detection using a new convolutional layer. *Proceedings of the 4th ACM Workshop on Information Hiding and Multimedia Security,* Vigo Galicia, Spain, June 20-22, 2016, pp. 5-10.

[13] Nagi, J., Ducatelle, F., Di Caro, G.A., Cireşan, D., Meier, U., Giusti, A., Nagi, F., Schmidhuber, J. and Gambardella, L.M., 2011. Max-pooling convolutional neural networks for vision-based hand gesture recognition. *2011 IEEE International Conference on Signal and Image Processing Applications (ICSIPA),* Kuala Lumpur, Malaysia, November 16-18, 2011, pp. 342-347.

[14] Wang, S., Jiang, Y., Hou, X., Cheng, H. and Du, S., 2017. Cerebral micro-bleed detection based on the convolution neural network with rank based average pooling. *IEEE Access*, 5, 16576-16583.

[15] Liu, K., Kang, G., Zhang, N. and Hou, B., 2018. Breast cancer classification based on fully-connected layer first convolutional neural networks. *IEEE Access*, 6, 23722-23732.

[16] Chen, Z. and Huang, X., 2017. End-to-end learning for lane keeping of self-driving cars. *2017 IEEE Intelligent Vehicles Symposium (IV)*, Los Angeles, USA., June 11-14, 2017, pp. 1856-1860.

[17] Setiawan, F.B., Siva, P.M., Pratomo, L.H. and Riyadi, S., 2021. Design and implementation of smart forklift for automatic guided vehicle using raspberry Pi 4. *Journal of Robotics and Control*, 2(6), 508-514.

[18] Maizar, G.A., Hidayat, B. and Kharisma, O.B., 2021. Home electrical intelligent control using node-red based on raspberry Pi-3 B+. *Indonesian Journal of Electrical Engineering and Renewable Energy*, 1(1), 1-7.

[19] Dewangan, D.K. and Sahu, S.P., 2020. Deep learning-based speed bump detection model for intelligent vehicle system using raspberry Pi. *IEEE Sensors Journal*, 21(3), 3570-3578.

[20] Seth, A., James, A. and Mukhopadhyay, S.C., 2020. 1/10th scale autonomous vehicle based on convolutional neural network. *International Journal on Smart Sensing and Intelligent Systems*, 13(1), 1-17.

[21] Hong, P.-H., Qiu, M. and Wang, Y., 2020. Autonomous driving and control: Case studies with self-driving platforms. *IEEE 2020 7th IEEE International Conference on Cyber Security and Cloud Computing (CSCloud)/2020 6th IEEE International Conference on Edge Computing and Scalable Cloud (EdgeCom)*, New York, USA, August 1-3, 2020, pp. 17-22.