ECTI Transactions on Computer and Information Technology
Journal homepage: https://ph01.tci-thaijo.org/index.php/ecticit/
Published by the ECTI Association, Thailand, ISSN: 2286-9131

MFPE : A Loss Function based on Multi-task Autonomous Driving

Youwei Li[1] and Jian Qu[2]

ABSTRACT

Road tracking, traffic sign recognition, obstacle avoidance, and real-time acceleration and deceleration are some critical sub-tasks in autonomous driving. This research proposed to use a single-sensor (camera) based intelligent driving platform to achieve multi-task (four subtasks) autonomous driving. We adjusted the function combinations and hyperparameters of the model to improve the model training and model testing performance. The experiments showed that the existing function combinations could not significantly improve the autonomous driving performance, and the loss function had a significant impact on the autonomous driving performance of the model. Therefore, we designed a novel loss function (MFPE) based on multi-task autonomous driving. The models with the MFPE loss function outperformed the original and existing models in model training and actual multi-task autonomous driving performance. Meanwhile, the model with the MFPE loss function achieved multi-task autonomous driving under different lighting conditions, untrained routes, and different static obstacles, which indicates that the MFPE loss function enhances the robustness of the model. In addition, the speed of the intelligent driving platform can reach up to 5.4 Km/h.

Article information:

Keywords: Convolutional Neural Network, Recurrent Neural Network, Deep learning, Long Short-Term Memory, MFPE Loss Function, Multi-task Autonomous Driving

Article history:
Received: April 26, 2022
Revised: July 8, 2022
Accepted: September 3, 2022
Published: September 30, 2022 (Online)

DOI: 10.37936/ecti-cit.2022164.248304

1. INTRODUCTION

Autonomous driving research focuses on the decision-making and execution of driving behaviors. We will describe the strengths and weaknesses of existing research, as well as the decision algorithms, execution platforms, and experimental environments used in our research.

The selection of hardware and environment for autonomous driving experiments is the primary issue that needs to be addressed for autonomous driving research. The main options for autonomous driving experiments are real cars, toy cars, and scale model cars. The control part of the car consists of steering control and speed control. In terms of steering, the real car relies on turning the steering wheel for steering, the toy car is steered by forward and reverse rotation of the DC motor (DC, Direct Current), and the PWM controlled servo steers the scale model car. Compared to real cars and toy cars, scale model cars use a servo that provides twenty precise angles to control the direction, which is similar to a real car. In addition, real cars have different rotation speeds for the inner and outer wheels when steering. It solves this problem via differential gears in drive shifts. Toy cars ignore this by using a simple drive shift, thus making the cars a bit twitchy all the time. However, steering appears to be less affected because of the light weight of the toy car. Therefore, toy cars that rely on the forward and reverse rotation of the motor to control steering are not smooth. The scale model car uses scaled down differential gears on each driving shift for steering, which maximally simulates the mechanical structure of a real car. In terms of speed control, real cars use a combination of engine and gearbox to control the speed, toy cars control speed by power on/off, and scale model cars control speed by adjusting PWM value. Toy cars and scale model cars are very different from real cars. However, the speed control method of the scale model car is similar to that of the Tesla electric car. Compared with real cars and toy cars, scale model cars control the speed by controlling the PWM value through ESC (Electronic stability control). We can accurately collect the PWM values (throttle values) for deep learning.

[1,2]The authors are with Faculty of Engineering and Technology, Panyapiwat Institute of Management, Nonthaburi, Thailand 11120, Email: youweili19971110@outlook.com and jianqu@pim.ac.th

In addition, the scale model car has roughly fifty internally sealed steel ball bearings to smooth the rotation of each part similar to a real car. At the same time, the scale model car uses oil pressure shock absorbers, which enable smooth driving at high speed.

In addition, we analyzed the cost and security issues of different hardware. A real car and a chip with sufficient computational power are necessary to conduct experiments with real cars. However, building a real autonomous car is expensive ($74,031) [1]. At the same time, safety is also an important issue that must be considered. The environment for conducting experiments with real cars needs to be enormous, sparsely populated, and it must be easy to control noise. The speed of a real car is fast, which increases the risk of a traffic incident. In addition, most of the experiments with real cars for autonomous driving are conducted by automotive manufacturers. Automotive companies have test sites built explicitly for conducting autonomous driving tests. Ultimately, it is almost impossible for a student to experiment with a real car with his/her budget. Therefore, more studies are being conducted to solve these problems by using model cars to simulate real cars for experiments. First, the model car only costs about $100. Secondly, the model car requires only a small experimental space. We can do such an experiment in a room or a few dozen square meters of space. In addition, the model car is relatively less dangerous than a real car. During the experiment, if the model car crashes, it will only cause damage to the model car and the objects in the room, and will not cause human casualties. Therefore, after considering the cost, the safety of the test site, and the related technology, we propose to use model cars to build an intelligent driving platform for autonomous driving experiments.

There are several kinds of intelligent driving platforms built using model cars. The components of the intelligent driving platform are a control chip, sensors, and chassis. First, we must select the control chip. In the early days of autonomous driving research, researchers used Arduino as a control chip. However, the computational power of Arduino is limited, and the computational process requires the help of a computer, in which case such an intelligent driving platform is not a standalone agent. AI algorithms cannot achieve independent reasoning and analysis due to the lack of computational power of the control chip.

Existing studies have addressed this problem by using a combination of vehicle control chips. In the study by Lee and Lam [2], they used a combination of Raspberry Pi 3 and Arduino to distribute the workload. Therefore, we used the Raspberry Pi as the control chip for the intelligent driving platform. Next, we must select sensors. Existing studies used multiple sensors to achieve multi-task autonomous driving. For example, in the autonomous driving study by Lee and Lam [2], they used an ultrasonic sensor and a camera. In the autonomous driving study by Du et al. [3], they used LIDAR and a camera. They used multiple sensors for road tracking and obstacle detection. In their study, ultrasound and LIDAR were used to detect obstacles only, which would increase the computation and workload of the control platform. Fortunately, Shukai and Jian [4] verified in their study that it is possible to achieve double-task autonomous driving with a single camera. Therefore, we propose to use one camera for four-task autonomous driving.

Finally, we must select a chassis. The main options for building autonomous driving platforms in the existing studies are toy cars and scale model cars. Toy cars are mainly chosen because of their cost-effectiveness. The speed control of the toy car only has power-on and power-off states and cannot achieve real-time acceleration and deceleration. The scale model cars can overcome these problems. Scale model cars use ESC to control speed, and servos to control steering. The scale model car can achieve precise real-time speed adjustment and steering control. In addition, the mechanical structure of the scale model car is designed according to the driving system of the real car. Therefore, we propose to use a scale model car to build an intelligent driving platform to achieve autonomous driving. Our research achieved multiple autonomous driving subtasks using a low-cost intelligent driving platform and had good autonomous driving performance.

The core of perception and decision-making technology for autonomous driving is artificial intelligence (AI) algorithms. For example, Lin et al. [5] proposed a ten-layer convolutional neural network (CNN, Convolutional Neural Network) model to achieve autonomous driving. In the study of Valiente et al. [6], they proposed a twelve-layer CNN-LSTM model to achieve autonomous driving. However, their studies were only conducted in Udacity (a simulator for virtual environments) for data collection and model testing. With sufficient hardware computing power, the virtual simulation environment can generate unlimited data for training and testing. Experimenting in a virtual simulation environment undoubtedly reduces the time and difficulty of deep learning. However, there is a big difference between virtual environments and actual environments. Virtual environments cannot simulate realistic lighting changes, shadow shifts, and environmental noise. Therefore, it is necessary to validate neural network algorithms in the actual world.

Fortunately, some AI algorithms have already been proven in the real world. For example, in the study of Bechtel et al. [7], they achieved road tracking on a custom track by an AI algorithm. In the study of Truong-Dong Do et al. [8], they achieved road tracking and traffic sign recognition on a custom track with an AI algorithm. In the study of Karni et al. [9],

they achieved road tracking and obstacle recognition on a custom track with an AI algorithm.

Speed is another important factor that cannot be ignored for autonomous driving. In a study by Youwei and Jian [10], they trained a 21-layer CNN model to achieve road tracking and real-time acceleration and deceleration. The existing studies show that few subtasks are achieved in the multi-task autonomous driving studies. Therefore, we propose to train a deep neural network model capable of simultaneously achieving road tracking, traffic sign recognition, obstacle avoidance, and real-time acceleration and deceleration.

In this study, we trained lightweight CNN14 and RNNL19 models, and the models achieved four-task autonomous driving. However, the model performed poorly in autonomous driving and was prone to go out of control. Therefore, we replaced the function algorithm of the model to improve the performance of the model. The experiment shows that the loss function has a relatively large impact on the performance of the multi-task autonomous driving model. However, the experimental results show that the existing loss functions do not perform well for multi-task autonomous driving.

Therefore, we propose a novel loss function based on multi-task autonomous driving to improve the performance of the model. Experiments show that our proposed loss function reduces the model training loss and improves the stability of the model. In addition, the actual autonomous driving performance of the model was also improved. Furthermore, we fine-tuned the hyperparameters to further improve the autonomous driving performance of the model. At the same time, we analyzed the effects of lighting, speed, untrained routes, and different obstacles on the model performance.

Our study was inspired by Mnih et al. [11], who conducted experimental research on autonomous driving in a virtual environment. It is easy to construct the reward function in a virtual environment. Meanwhile, the training process we used no longer requires manual labeling and calculation, and the computer can calculate the final reward value directly. However, the reward function cannot be applied in a realistic environment. Meanwhile, existing studies using small-scale intelligent driving platforms can only achieve one or two subtasks. Therefore, we propose to validate multi-task (four subtasks) autonomous driving in a realistic environment. In addition, our experiments are conducted using a Raspberry Pi embedded system. Meanwhile, we have achieved four autonomous driving subtasks. Our intelligent driving platform cannot perform more complex tasks, such as avoiding moving obstacles, because of the lack of computational power.

2. MATERIALS AND METHODS

In this chapter, we analyze the studies relevant to our research and propose our research materials, and methods.

2.1 Research Related to Autonomous Driving

2.1.1 Hardware

The components of the small-scale intelligent driving platform are chassis, sensors and chips.

Regarding the selection of the chassis, in the researches of Boloor et al. [12] and Bae et al. [13], they used toy cars to build the intelligent driving platform. Although the toy car is very cost-effective, the "switch-type" control method used by the toy car has only two states of "0%" and "100%" for steering and speed. This control method does not allow for precise control of direction and speed. Meanwhile, the toy car relies on the speed difference of the four wheels for steering. Therefore, there is a large error when collecting data with the toy car. The collected sample data directly affects the performance of the model. Therefore, driving systems that can precisely adjust direction and speed are needed.

A scale model car is designed according to the mechanical structure of a real car, and the servo and ESC can precisely control the direction and speed. Youwei and Jian [10] used a 1/16 scale model car to achieve road tracking and real-time acceleration and deceleration in their study. Therefore, we propose to use a scale model car to build an intelligent driving platform.

Regarding the selection of sensors, in the study by Karni et al. [9], Lee and Lam [2], they used a camera and ultrasonic. In their studies, ultrasonic only enabled obstacle detection, and the intelligent driving platform can only stop and cannot avoid obstacles. In the study by Du et al. [3], they used a camera and lidar. They use lidar primarily to measure distances to aid decision-making. Furthermore, humans can rely on their vision to drive a vehicle. Therefore, we propose to use a single camera for multi-task autonomous driving.

Regarding the selection of chip, in the research of Yuenyong and Jian [14], the chip they used is Arduino. The computing power of Arduino is minimal. The calculation process is done on another computer in their research, and the calculation results are sent back to the Arduino to control the intelligent driving platform. Such an intelligent driving platform is not a stand-alone agent. For this problem, in the research of Truong-Dong Do et al. [8], they used a combination of Raspberry Pi and Arduino to build an intelligent driving platform. The Raspberry Pi does the computing in their intelligent driving platform, and the Arduino receives the computing results to control the agent. The lack of computing power of the chips makes them use two ships to distribute the workload.

In the study of Bechtel et al. [7], they tested the performance of autonomous driving on Raspberry Pi 3 B, Intel UP, and NVIDIA Jetson TX2 chips. Experiments show that the Raspberry Pi 3 B has enough computing power to achieve autonomous driving. We found the Raspberry Pi 4 Model B to be more powerful and more affordable. To address the workload of the control chip, Youwei and Jian [10] in their study proposed using a Raspberry Pi 4 Model B loaded with a lightweight end-to-end neural network model to reduce the workload of the chip. Therefore, we propose using Raspberry Pi 4 Model B with 4Gb memory as the chip of the intelligent driving platform.

2.1.2 Neural Networks for Autonomous Driving

CNNs and Recurrent Neural Networks (RNNs) can be used to train autonomous driving. In the study of Lin et al. [5], they proposed a CNN model to achieve road tracking. In the study of Viktor Rausch et al. [15], they proposed a neural network model combining CNN and LSTM to achieve road tracking. However, these models are only tested in a virtual simulation environment. For this problem, in the research of Bechtel et al. [7], they built a small-scale intelligent driving platform and custom tracks to achieve road tracking. However, single-task autonomous driving is not truly autonomous driving. In the study of Karni et al. [9], they trained a CNN model for road tracking and obstacle detection. In the study of Truong-Dong Do et al. [8], they trained a CNN model that achieved road tracking and traffic sign recognition. In the study of Youwei and Jian [10], they added throttle values while training the autonomous driving neural network model and successfully achieved the ability to accelerate and decelerate during road tracking. The existing multi-task autonomous driving combines at most two sub-tasks.

We propose to train a neural network to achieve multi-task autonomous driving for road tracking, traffic sign recognition, obstacle avoidance, and real-time acceleration and deceleration. The road tracking task is when the trained model recognizes the edge lines and can stay between the two edge lines all the time. We trained only left-turn and right-turn signs in the traffic sign recognition task. Traffic sign recognition is when the trained model recognizes a left-turn or right-turn sign and makes a corresponding turn according to the signs. The obstacle avoidance task is when the trained model recognizes an obstacle and can avoid the obstacle. The real-time acceleration and deceleration task is to adjust the real-time speed of the intelligent driving platform according to different scenarios during driving.

2.1.3 An end-to-end approach for autonomous driving

The non-end-to-end learning method consists of multiple independent modules. Each step is an independent task, and the quality of the result will affect the next step, which affects the training results. Non-end-to-end learning requires data annotation before each learning task is performed. Non-end-to-end driving methods require hundreds of thousands of labelled data samples, the entire data labelling process is expensive, and human labelling cannot be error-free.

End-to-end is a concept in deep learning [16]. A prediction result is obtained from the input data to the output data during the end-to-end model training process. If there is an error between the predicted and actual results during the model training process, that error will be propagated backward through each model layer. Each layer will be adjusted according to the error until the model converges to achieve the expected results [17]. Furthermore, the end-to-end autonomous driving approach allows the model to be lightweight and easy to run on embedded development platforms. End-to-end learning delegates the learning and judgment process to a neural network, improving efficiency and accuracy.

2.2 Construction of Intelligent Driving Platform

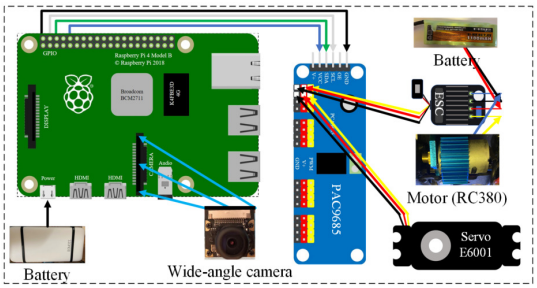


Fig.1: *The schematic diagram of the circuit wiring of the intelligent driving platform.*

We built an intelligent driving platform to conduct experiments. The schematic diagram of the circuit wiring of the intelligent driving platform is shown in Figure 1, and the completed intelligent driving platform is shown in Figure 2.

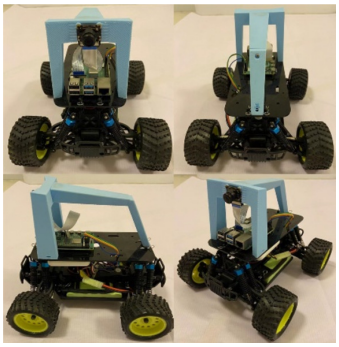


Fig.2: *Intelligent driving platform.*

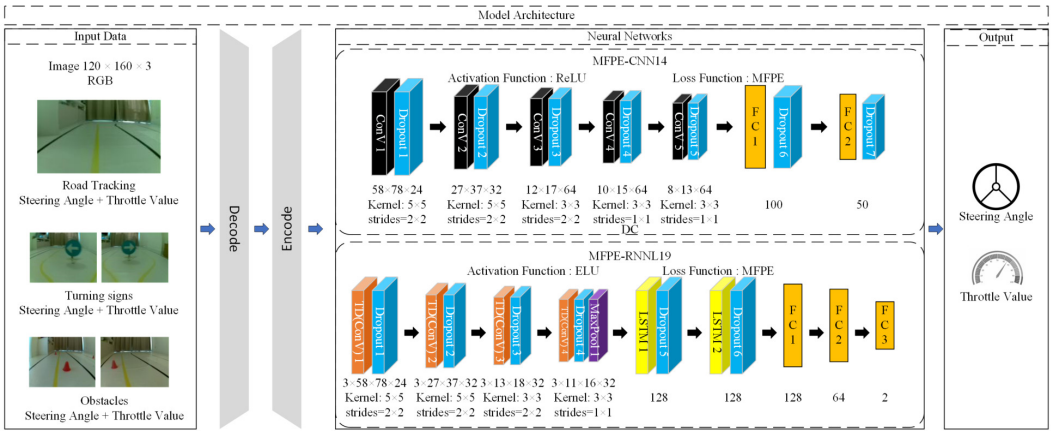


Fig.3: *Model architecture overview.*

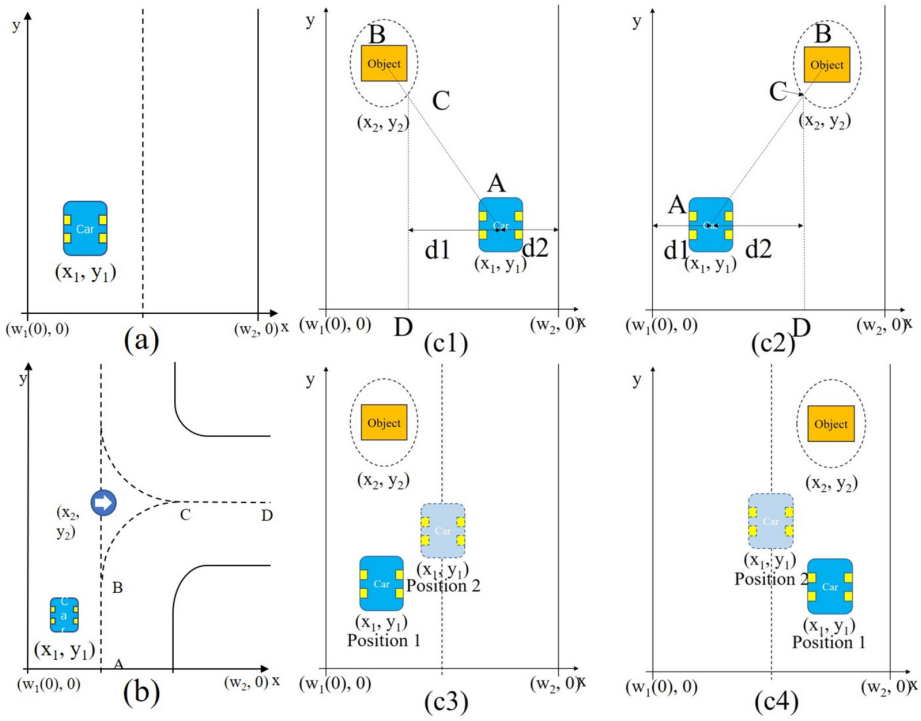


Fig.4: *The coordinate system constructed for the analysis of the loss function.*

The intelligent driving platform is designed and built similarly to the structure of the Donkey Car [18]. We choose a wide-angle camera to get more information about the environment to achieve multi-task autonomous driving. The intelligent driving platform dimensions are 25.1 CM × 21.6 CM × 17.4 CM (Length × Width × Height). The weight of the intelligent driving platform is 1.35 Kg.

2.3 Achieving Multi-Task Autonomous Driving

Our research achieved four sub-tasks in a neural network. A schematic diagram of the model architecture for multi-task autonomous driving is shown in Figure 3. As shown in Figure 3, two neural network models were trained in this work. We fine-tuned the architecture of the two models and applied our proposed novel MFPE loss function. In the MFPE-CNN14 model, we added dropout layers after each layer of the model to improve the generalization ability of the model and prevent overfitting [19]. The MFPE-RNNL19 model introduced LSTM [20] layers to consider the effect of time on the model. In deep learning, the model mainly learns two mapping relationships. The first mapping relationship is between images and steering angles. We mark the steering angle for each image and train the model with the tagged steering angle. The trained model can make steering angle predictions based on the images acquired by the camera.

The second mapping relationship is between images and throttle values. We mark the throttle values for each image and train the model with the tagged throttle values. The trained model can predict the throttle values based on the images acquired by the camera. We adjust the PWM values to control the speed. The range of PWM value is 0-500. We experimentally found that fast speeds can easily cause the intelligent driving platform to run off the track. Therefore, we set 370 as the PWM value when the speed is 0 and 400 as the PWM value when the speed is maximum. Our throttle values range from 0-1. In the speed adjustment, when the throttle value increases by 0.1, the PWM value increases by 3, the output current increases, and the speed increases. Road tracking, traffic sign recognition, and obstacle avoidance learning are the same mapping relationship, and the mutual influence of data makes the performance of multi-task autonomous driving worse than that of single-task autonomous driving. Multi-task autonomous driving focuses on balancing the relationship between these three sub-tasks.

relationships of multiple datasets, we address the problem of data interactions by replacing the activation [21] and loss functions [22] of the model. We found the best combination of existing activation functions and loss functions. However, the performance of multi-task autonomous driving is still poor. We found from the experimental results that the loss function significantly impacts model training and actual multi-task autonomous driving performance. Therefore, we propose to design a novel loss function based on multi-task autonomous driving.

2.4 Loss function based on multi-task autonomous driving

The loss function [22] is used to evaluate the inconsistency between the predicted and actual labels of the model. The key to the optimization model is the loss function. The loss value calculated by the loss function is the error between the predicted value (V_{pre}) and the actual value (V_{act}), allowing us to understand the gap between the predicted and actual values intuitively. We separately analyze suitable loss functions for road tracking, traffic sign recognition, and obstacle avoidance for the three sub-tasks.

2.4.1 Loss function suitable for road tracking

We created a coordinate system for the road tracking custom track, and the completed coordinate system is shown in Figure 4 (a).

The true and predicted values in Figure 4 (a) are set by decoding the images into vectors and calculating them in combination with the tagged data. As shown in Figure 4 (a), the position of the intelligent driving platform $V_{act} = x_1$ the actual value, and it is the ideal state for the intelligent driving platform to drive in the middle of the custom track. The width of the custom track is $W = |w_2 - w_1|$. The ideal predicted value is when $V_{pre} = \frac{w}{2}$. According to the definition of the loss function, the minimum loss calculation equation is shown in Equation (1).

$$L = Min(V_{pre} - V_{act}) \tag{1}$$

According to Figure 4 (a), we calculated the loss function $L_{Road\ tracking}$ for road tracking. The loss function is a non-negative number, and we cannot determine the size of V_{pre} and V_{act} . Generally, the absolute value or the square method is used to calculate the values. The loss in road tracking can be expressed as $L_{Road\ tracking\ 1} = |\frac{W}{2} - x_1|$ and $L_{Roadtracking2}=\left(\frac{W}{2} - x_1\right)^2$. As shown in the equations, we found that the predicted and actual values were suitable for road tracking when they matched the absolute or squared loss relationship. The corresponding loss functions in the existing loss functions are Mean Absolute Error (MAE, Mean Absolute Error) [23] and Mean Squared Error (MSE, Mean Squared Error) [24,25]. Therefore, the MSE and MAE loss functions are applicable to road tracking.

2.4.2 Loss function suitable for traffic sign recognition

We created a coordinate system for the traffic sign recognition custom track, and the completed coordinate system is shown in Figure 4 (b). As shown in Figure 4 (b), the AD segment is the ideal driving route for the intelligent driving platform. The key to traffic sign recognition is to deal with the BC segment. The width of the custom track is $W = |w_2 - w_1|$. We take the position $V_{act2} = (x_1, y_1)$ of the intelligent driving platform as the real value and the function expression of the $V_{pre} = O_{BC}$: $(x - W)^2 + \left[y - \left(y_2 - \frac{W}{2}\right)\right]^2 = \left(\frac{W}{2}\right)^2$ segment as the ideal predicted value. The loss in traffic sign recognition can be expressed as $L_{traffic\ sign\ recognition} = \sqrt{(x_1 - W)^2 + \left[y_1 - \left(y_2 - \frac{W}{2}\right)\right]^2} - \frac{W}{2}, x \in \left[\frac{W}{2}, W\right]$. As shown in the equation, we found that the size of $\sqrt{(x_1 - W)^2 + \left[y_1 - \left(y_2 - \frac{W}{2}\right)\right]^2}$ and $\frac{W}{2}$ cannot be determined. In addition, there is a root operator. We square the traffic sign recognition loss $L_{Truning\ signs\ recognition}$, and the loss in traffic sign recognition can be expressed as $L_{\text{traffic sign recognition}} = \left(\sqrt{(x_1 - W)^2 + [y_1 - \left(y_2 - \frac{W}{2}\right)]^2} - \frac{W}{2}\right)^2$. As shown in the equation, the relationship between the predicted and actual values is the squared error relationship. Although $E_{\text{traffic sign recognition}}$ adds the calculation of the position information of the steering, the relationship between the predicted and actual values is still essentially a squared error relationship. Therefore, the MSE loss function is applicable to road tracking and traffic sign recognition.

2.4.3 Loss function suitable for obstacle avoidance

In the multi-task autonomous driving experiment, we need to pay attention to the relative positions of the intelligent driving platform and the obstacles. Therefore, we established a coordinate system of relative positions to analyze, as shown in Figure 4 (c1-c4).

As shown in Figure 4 (c1-c4), there are four relative positions between the intelligent driving platform and the obstacle. As shown in Figure 4 (c1) and (c2), we assume that the length of the obstacle is m and the width is n. We set the obstacle to be a 1.5 times ellipse. The long semi-axis of the ellipse is set to $a = 1.5 \times \frac{m}{2}$, and the short semi-axis is set to $b = 1.5 \times \frac{n}{2}$. The mathematical expression for an ellipse is $O_e : \frac{x^2}{a^2} + = 1$. The expression for line AB is: $l_A B : (x_2 - x_1)y = (y_2 - y_1)x$. The intersection point C of O_e and l_{AB} is: $\left(\frac{ab}{\sqrt{b^2+a^2\left(\frac{y_2-y_1}{x_2-x_1}\right)}}, \frac{(y_2-y_1)ab}{\sqrt{b^2+a^2\left(\frac{y_2-y_1}{x_2-x_1}\right)^2}(x_2-x_1)} \right)$. The new boundary is the line CD. For the cases in Figure 5 (c) and (d), the intelligent driving platform performs road tracking before obstacle avoidance. When the intelligent driving platform goes from Position 1 to Position 2, it is the same as that shown in Figure 4 (c1) and (c2). In addition, the obstacles we designed are not only the rectangular objects shown in Figure 5. We can modify the coefficients of the loss function according to the actual obstacles. We can determine the values of a and b based on the length and width of a known obstacle (whether it is a regular shape or not).

The obstacle avoidance loss can be expressed as $L_{Obstacle\ avoidance} = d_1 - d_2$. It can be seen from Figure 4 (c1) and (c2) that the sizes of d_1 and d_2 are uncertain in the case of different positions. We squared the obstacle avoidance loss $L_{Obstacle\ avoidance}$ and the loss of the predicted and actual values of obstacle avoidance can be expressed as $L_{Obstacle\ avoidance} = \frac{ab(2x_1-w)^2 \cdot [(x_2-x_1)^2+(y_2-y_1)^2]}{(x_2-x_1)^2} - a^2b^2$. Based on this equation we propose a loss function for the obstacle avoidance task, as shown in Equation (2).

$$L_{Obstacle\ avidance}(V_{act}, V_{pre}) = (V_{act} - V_{pre})^4 \quad (2)$$

In Equation (2), the loss relationship between the predicted and actual values in the obstacle avoidance conforms to the fourth power error relationship. Obstacle avoidance needs to consider the location of obstacles, boundary selection in different situations, and the size of obstacles. Among the existing loss functions, there is no loss function applicable to autonomous driving obstacle avoidance. Based on our loss function analysis on different sub-tasks, we propose the use of a fourth power error loss function for achieving multi-task autonomous driving.

2.4.4 Loss Function for Multi-Task Autonomous Driving

According to the analysis of loss functions for different autonomous driving subtasks in sections 2.4.1-2.4.3. We found that the MSE loss function is suitable for road tracking and traffic sign recognition, and the fourth power error relation is suitable for obstacle avoidance. The fourth power error relation has the advantages of MSE and is also suitable for obstacle avoidance. Therefore, we use the fourth power error relationship as the loss function relationship between the predicted and actual values. At the same time, we found that the existing loss functions used the mean error, and the mean error reflects the central tendency of the error, which can effectively avoid the situation where individual values are too high or too low and have a relatively significant impact on the whole. Therefore, we propose a novel multi-task-based Mean Fourth Power Error (MFPE, Mean Fourth Power Error) loss function for autonomous driving. In addition, we calculated the coefficients of the fourth power error function as 4ab. Combining the above factors, the expression of MFPE is shown in Equation (3).

$$L_{MFPE}(V_{act}, V_{pre}) = \frac{4ab}{n} \cdot \sum_{n=1}^{n} (V_{act} - V_{pre})^4 \quad (3)$$

In this study, we used cone-shaped obstacles. Based on the dimensions of the conical obstacles we calculated $a = b = 0.08$. Substituting a and b into equation (3) we can obtain the practical application loss function as $L_{MFPE} = \frac{0.0225}{n} x^4$. In addition, we plotted the function graphs of MSE, MAE and MFPE, as shown in Figure 5. We found that the function graph of MFPE is smoother than that of MSE and overcomes the problem of MAE being non-differentiable at the origin.

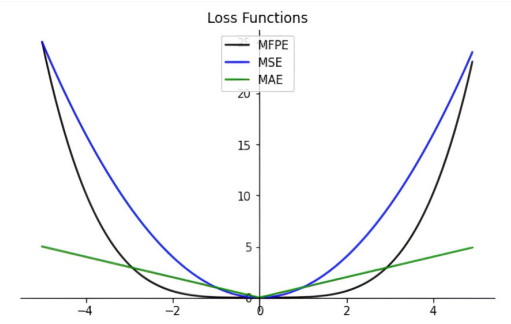


Fig.5: *The mathematical functions graph of three loss functions.*

2.5 Method for Evaluating the MFPE Loss Function in Multi-Task Autonomous Driving

We divided the evaluation of the loss function into whether the loss value of model training is reduced and whether the performance of multi-task autonomous driving is improved. We will synthesize the training loss value and the actual multi-task autonomous driving performance to analyze the quality of different multi-task autonomous driving models.

2.6 Experiment Setup

2.6.1 Custom tracks and simulation environment module

We built different custom tracks and experimental environments for multi-task autonomous driving, as shown in Figure 6.

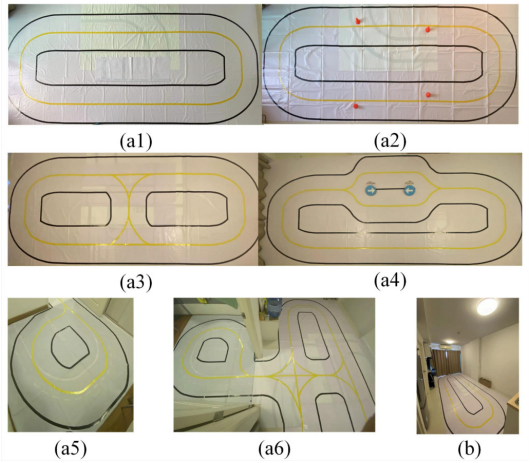


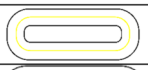
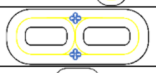
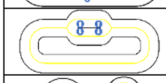
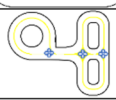
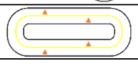
Fig.6: *(a1-a6) Custom tracks overview, (b) Schematic diagram of lighting and room conditions.*

Because of the COVID-19, we had to conduct the experiment in our home. After we moved all the furniture in our home except the bed, this was the largest available area in my house. Therefore, we designed different custom tracks for each task. The completed custom tracks are shown in Figure 6 (a1-a5). In addition, we considered that the obstacle avoidance task needs to be achieved. We designed the width of the custom track to be 2.5 times the width of the intelligent driving platform. Meanwhile, we stitched the built custom tracks into a complex scenario, as shown in Figure 6 (a6). Autonomous vehicles cannot always be driven in a trained scenario. In the custom track in Figure 6 (a6), we can test complex and untrained routes. Our experiments were conducted in a closed room to reduce environmental noise. We closed the doors and windows during the experiment, and used the lamps on the ceiling as the lighting source, as shown in Figure 6 (b).

2.6.2 Data collection module

We collected datasets on custom tracks and the simulated multi-task autonomous driving environment by driving the intelligent driving platform. The collected datasets consist of pictures, steering angles, and throttle values. During the data collection process, we programmed automatic data annotation of the images, recording the throttle value and steering angle corresponding to each image when it was taken. Each image has a corresponding steering angle and throttle value. The collected data was directly used to train the model. Multi-task autonomous driving needs to achieve four sub-tasks, of which real-time acceleration and deceleration are included in the other three tasks. We divided the collected datasets into a road tracking dataset (Dataset I), a traffic signs dataset (Dataset II), and an obstacle dataset (Dataset III). When collecting Dataset III, we used red conical obstacles and the obstacles were not moving. The details of the collected datasets are shown in Table 1.

Table 1: *Overview of the datasets.*

Datasets	Custom tracks	The number of images in the datasets (Clockwise/Counter clockwise)
Dataset I		(1,500/1,500)
		(1,500/1,500)
		(3,000/3,000)
Dataset II		(2,070/2,070)
		(1,035/1,035)
		(4,145/4,145)
Dataset III		(3,600/3,600)

2.6.3 Model Training Module

We trained a 14-layer CNN model and a 19-layer RNNL model to achieve multi-task autonomous driving, and we named them CNN14 and RNNL19. We replaced the loss function, activation function and loss-activation function combination of the two models and trained the models. The improvement of the performance of the models after the function replacement and combination is not significant. Next, we applied the proposed MFPE loss function to the two models. The two models MFPE-RNNL19 and MFPE-CNN14 with the MFPE loss function performed better than the original model and the models

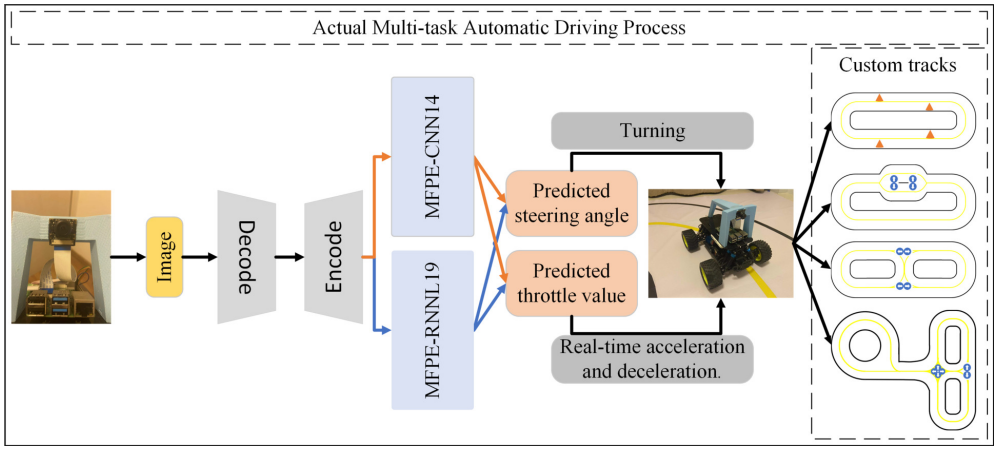


Fig.8: *Actual autonomous driving process.*

in the existing studies. Then, we fine-tuned the hyperparameters of the model to further optimize the model performance. To find a more suitable hyperparameter setting, we trained the model with different batch sizes, epochs, self-drop ratios, learning rates and optimizers. In addition, we selected existing multi-task autonomous driving models for comparison. In the study of Truong-Dong Do et al. [8], they achieved the combined autonomous driving of road tracking and traffic sign recognition, and we called their CNN model TDD-CNN. In the study of Karni et al. [9], they achieved a combination of road tracking and obstacle avoidance for autonomous driving, and we called their CNN model UK-CNN. We trained TDD-CNN and UK-CNN models with our datasets.

The model training process is carried out on Google Colab Pro. Google Colab Pro provides a 16GB memory GPU for training, significantly reducing training time. We trained different routes in different custom tracks, as shown in Figure 7. In particular, in the custom track of Figure 7 (d), we selected three routes for training and did not train all possible routes.

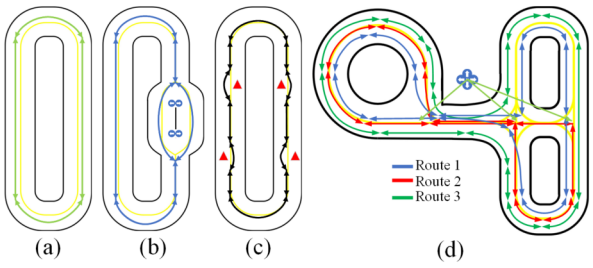


Fig.7: *Overview of model training routes.*

2.6.4 Model Testing module

The model test was conducted by performing actual autonomous driving in custom tracks. The actual autonomous driving process is shown in Figure 8. Furthermore, actual multi-task autonomous driving videos for different models are available at https://github.com/NICESTUDYPAPER/STUDY. The training process of the model is not done on the Raspberry Pi 4 Model B. We only test the models on the intelligent driving platform. Therefore, the computational power of Raspberry Pi 4 Mode B is sufficient. However, the models we tested were lightweight end-to-end neural network models for autonomous driving, and the Raspberry Pi 4 Model B has enough computational power to load the training models for predicting throttle values and steering angles. Meanwhile, our intelligent driving platform used only a camera and the memory of the chip could be used mainly for processing neural network models. The largest model we tested took 74.3% of the CPU when loaded on the Raspberry Pi 4 Model B.

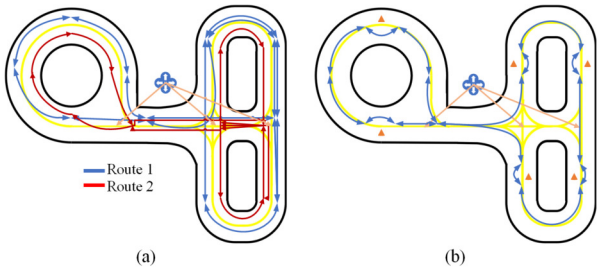


Fig.9: *Overview of untrained routes.*

We tested both trained and untrained routes. The trained route is shown in Figure 8 and the untrained route is shown in Figure 9.

In addition, we tested the autonomous driving performance under different lighting conditions and at different speeds. We tested the effects of trained obstacles, obstacles of the same shape in different colors, obstacles of different shapes, different obstacle positions, and moving obstacles on obstacle avoidance performance. The different obstacles are shown in Figure 10.

Fig.10: *Obstacles.*

3. RESULTS AND DISCUSSION

In this section, we analyze and discuss the experimental results of model training and model testing.

3.1 Model Training Performance

3.1.1 The effect of functions and their combinations on model training

We experimentally found that the existing activation functions ReLU, ELU, SeLU, Softmax, Softplus, Softsign, Tanh, Sigmoid, and Hard_Sigmoid are suitable for training our model [26-28], and the existing loss functions [29-31] MSE, MAE, MSLE, Logcosh Hinge, Squared_Hinge, and Categorical_Hinge are applicable to train our model.

We used the existing activation functions for model training. The loss values of the model training under different activation functions are shown in Figure 11.

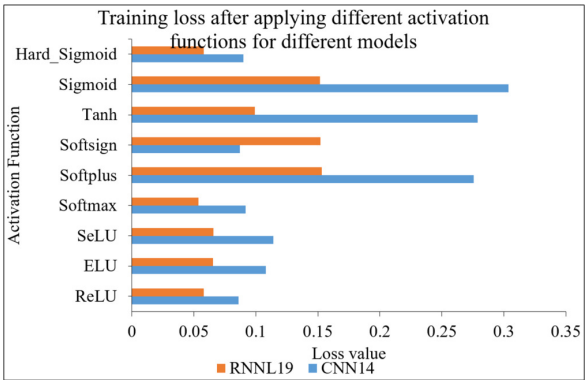


Fig.11: *Plot of the loss values of the models trained with different activation functions.*

As shown in Figure 11, for the RNNL19 model, different activation functions have rarying impact on the model training, and the Softmax activation function is more suitable for the RNNL19 model. For the CNN14 model, the activation function has a greater impact on the model training. The ReLU activation function is more suitable for the CNN14 model.

We used the existing loss functions for model training, and the loss values for model training with different loss functions are shown in Figure 12.

As shown in Figure 12, for the RNNL19 model, the loss activation function has less influence on the model training, and the MSE loss function is more suitable for the RNNL19 model. For the CNN14 model, the loss value of the model trained under the partial loss function is larger, and the MSLE loss function is more suitable for the CNN14 model.

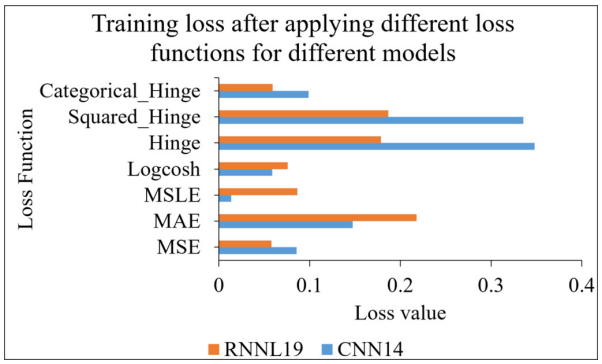


Fig.12: *Plot of the loss values of the models trained with different loss functions.*

However, the control variables approach cannot cover all possibilities. Therefore, we replaced different combinations of activation-loss functions and trained the models again. There are nine existing activation functions and seven loss functions available. Each combination of activation-loss functions consists of a loss function and an activation function. Therefore, there are sixty-three possible combinations of all the possible activation-loss function combinations. All are shown in Table 2. Table 2 shows that among the available combinations of activation functions and loss functions, the combination of ReLU for the activation function and MSLE for the loss function is better for the CNN14 model, and the combination of ELU for the activation function and MSLE for the loss function is better for the RNNL19 model. In addition, the loss function has a large impact on the performance of the model when analyzed in terms of individual functions.

3.1.2 Effect of MFPE loss function on model training

We improved the performance of the multi-task autonomous driving model by combining activation functions and loss functions, but the effect did not reach our expectations. Therefore, we proposed the MFPE loss function to enhance the performance of the multi-task autonomous driving model. Table 3 shows the training losses of different models. Compared with the original model, the model with the MFPE loss function applied shows a significant reduction in the training loss values compared to the model with the other loss functions applied. The model with the best training performance in Table 3 is MFPE-RNNL19, which has 47.14% less training loss than the original model and 38.77% less training loss than the model with the combination of functions.

Table 2: *Loss values of the model trained with different activation-loss function combinations.*

Models	Functions	Categorical_hinge	Hinge	Logcosh	MAE	MSE	MSLE	Square_hinge
RNNL	Elu	0.072927	0.264778	0.026171	0.064497	0.0415	0.007213	0.239841
	Hard_Sigmoid	0.070247	0.476006	0.110291	0.280842	0.250061	0.044825	0.504289
	Relu	0.066714	0.248949	0.014348	0.296403	0.039006	0.056223	0.241759
	Selu	0.571082	0.243625	0.017635	0.0758	0.038907	0.008524	0.246747
	Sigmoid	0.072372	0.468255	0.112267	0.279649	0.25408	0.044539	0.500049
	Softmax	0.064961	0.482878	0.109122	0.279747	0.249597	0.240206	0.504164
	Softplus	0.070469	0.471137	0.110316	0.274023	0.24757	0.044757	0.500106
	Softsign	0.078009	0.473277	0.019898	0.35306	0.25499	0.010227	0.499392
	Tanh	0.07371	0.470465	0.114359	0.090722	0.031641	0.044756	0.502011
CNN14	Elu	0.135902	0.481975	0.024275	0.142888	0.051007	0.477291	0.530116
	Hard_Sigmoid	0.512476	0.944434	0.221624	0.528951	0.503145	0.089327	1.001105
	Relu	0.12283	0.47783	0.024696	0.150699	0.048308	0.014916	0.48125
	Selu	0.141517	0.503492	0.037265	0.143722	0.087184	0.475441	0.505704
	Sigmoid	0.518661	0.955428	0.220491	0.517172	0.517403	0.093382	1.002374
	Softmax	0.519817	0.976039	0.218504	0.525001	0.511803	0.089453	1.011449
	Softplus	0.146964	0.945204	0.219674	0.123859	0.510567	0.476813	0.997498
	Softsign	0.522116	0.956425	0.025348	0.533844	0.510269	0.111712	0.99775
	Tanh	0.526385	0.93198	0.22437	0.532559	0.517649	0.477551	1.004421

Table 3: *Training loss of the model with the combined function.*

Models	Activation Function	Loss Function	Loss
CNN14	Linear	MAE	0.084306
RNNL19	Softmax	MSE	0.058372
Function Combinations-CNN14	ReLU	MSLE	0.072837
MFPE-Combinations-RNNL19	ReLU	MSLE	0.050384
MFPE-CNN14	ReLU	MSLE	0.058706
MFPE-RNNL19	ReLU	MSLE	0.03085

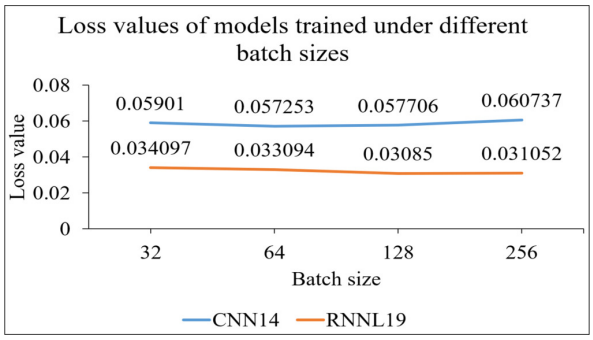


Fig.13: *Loss values of the models at different batch sizes.*

3.1.3 Effect of hyperparameters on model training

Fine-tuning hyperparameters is an important tool to improve the performance of the model. We adjusted the hyperparameters of the model to further improve the model performance. In this work, we discuss the effects of batch size, epoch, self-drop ratio, learning rate, and optimizers on model training performance during model training.

We recorded the loss values of the trained CNN14 and RNNL19 models with different batch sizes, as shown in Figure 13. As shown in Figure 13, we found that different models have different adaptability to the batch size. The MFPE-CNN14 model had the minimum training loss at a batch size of 64. The MFPE-RNNL19 model had the minimum training loss at a batch size of 128. The experimental results indicated that the more suitable batch size for the MFPE-CNN14 model is 64, and the more suitable batch size for the MFPE-RNNL19 model is 128.

We used the early stop method when training the model. The early stop method is used to stop if the training loss value no longer decreases during the training process. The model training will continue five times, and if the training loss value does not decrease, the model training will be stopped. The epoch at which training stops is the epoch required for model training, and the loss value obtained from the training is the minimum loss value. We recorded the epoch of training loss for the MFPE-CNN14 and MFPE-RNNL19 models and plotted the change of loss values with the epoch, as shown in Figure 14.

We found that the training loss of the models decreases as the epoch increases. As shown in Figure 14 (a), the training epoch of the MFPE-CNN14 model is around 30 times. As shown in Figure 14 (b), the training epoch of the MFPE- RNNL19 model is around 20 times. The experimental results indicated that the suitable epoch for the MFPE-CNN14 model is 40, and the suitable epoch for the MFPE-RNNL19 model is 30, under the condition that the minimum training loss can be obtained.

We added dropout layers to the two models in order to prevent overfitting, and the dropout layer needs to set the self-drop ratio value. We trained the models with different self-drop ratio values and recorded the loss values. The loss values of different models trained with different values of self-drop ratio are shown in Figure 15.

As shown in Figure 15, we found that different self-drop ratio values affect the loss value of model training. The MFPE-CNN14 model has the least

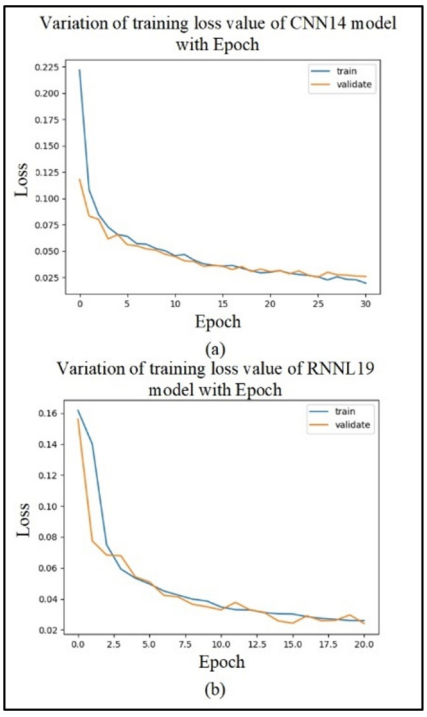


Fig.14: *Plots of training loss value with epoch.*

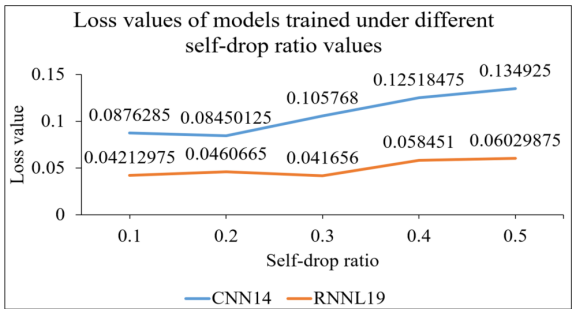


Fig.15: *Plot of the loss values of the models trained at different self-drop ratios.*

training loss with a self-drop ratio value of 0.2, and the MFPE-RNNL19 model has the least training loss with a self-drop ratio value of 0.3. The experimental results indicated that the more suitable self-drop ratio value for the MFPE-CNN14 model is 0.2, and the more suitable self-drop ratio value for the MFPE-RNNL19 model is 0.3.

The learning rate is another crucial hyperparameter, and we trained the two models at different learning rates. The loss values of different models trained with different learning rates are shown in Figure 16.

Both models have the minimum training loss at a learning rate of 0.005. The experimental results indicated that the appropriate learning rate for MFPE-CNN14 and MFPE-RNNL19 models should be 0.005.

The existing optimizers Adam [32], SGD [33], and RMSProp [34] are available to train our model. We used the existing optimizer for model training, and

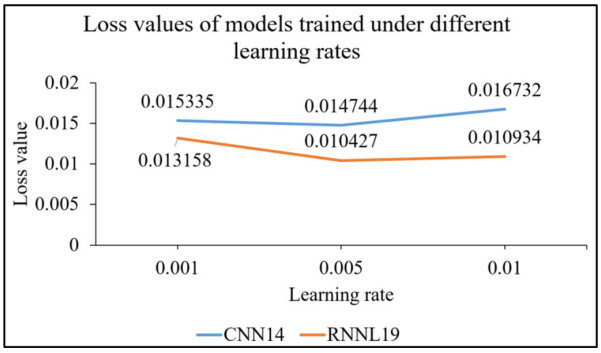


Fig.16: *Plot of the loss values of the models trained at different learning rate.*

the loss values of the models trained under different optimizers are shown in Figure 17.

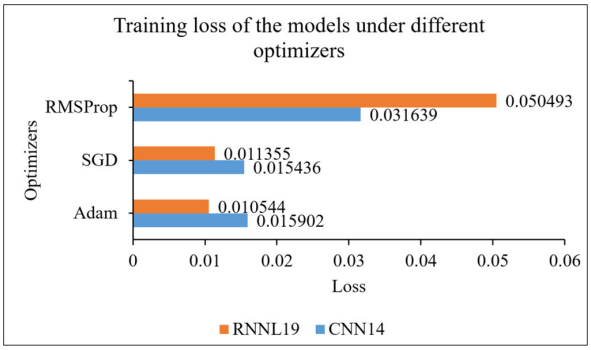


Fig.17: *Plot of the loss values of the models trained at different optimizers.*

Figure 17 indicates that the MFPE-RNNL19 model is suitable for using Adam as an optimizer, and the MFPE-CNN14 model is suitable for using SGD as an optimizer. The training results of the fine-tuned hyperparameter-optimized models and the models in the existing studies are shown in Table 4. Table 4 shows that fine-tuning the hyperparameters can effectively improve the performance of the model. The MFPE-RNNL19 model in Table 4 has the smallest value of loss. The loss of the MFPE-RNNL19 model was reduced by 33.83% compared to the value before adjusting the hyperparameters. In addition, we calculated the variance of the model training loss to evaluate the stability of each model. The MFPE-RNNL19 model had the best stability, which indicated that MFPE can improve the stability of the model as well.

3.2 Model testing performance

3.2.1 Actual performance of multi-task autonomous driving

We divided multi-task autonomous driving into four sub-tasks: road tracking (Task I), traffic sign recognition Task II), obstacle avoidance (Task III),

Table 4: *Loss values of the model trained with different activation-loss function combinations.*

Multi-task autonomous driving NN models	Batch Size	Epoch	Dropout	Learning Rate	Optimizer	Mean Loss (5 times)	Variance
CNN14 (Original)	128	33	0.2	0.001	RMSProp	0.084744	3.9E-05
RNNL19 (Original)	128	25	0.2	0.001	SGD	0.058093	9.23333E-05
MFPE-CNN14 (Combination Functions)	128	33	0.2	0.001	RMSProp	0.01541	9.33333E-06
MFPE-RNNL19 (Combination Functions)	128	25	0.2	0.001	SGD	0.01576	1.63333E-05
MFPE-CNN14 (Optimized)	64	40	0.2	0.005	SGD	0.014744	2.33333E-08
MFPE-RNNL19 (Optimized)	128	30	0.3	0.005	Adam	0.010427	1.30363E-08
TDD-CNN (Existing)	64	47	0.1	0.01	SGD	0.239057	0.0004
UK-CNN (Existing)	64	38	0.1	0.01	SGD	0.262084	0.0013

and real-time acceleration and deceleration (Task IV). We assumed that the total score of multi-task autonomous driving (MTADS, score of multi-task autonomous driving) is 100 points. The total score of each sub-task is 25 points. In the process of multi-task autonomous driving, if the sub-task can be achieved, the original score of the corresponding sub-task is 25 points. In the process of achieving different sub-tasks, if the intelligent driving platform touches the black line, one point will be deducted. If the intelligent driving platform does not recognize the traffic signs or avoid obstacles, two points will be deducted. If the intelligent driving platform runs off the custom tracks, four points will be deducted. We accomplish five laps of multi-task autonomous driving for each sub-task on custom tracks. The MTADS of different multi-task autonomous driving neural network models are shown in Table 5.

Table 5 showed that the MFPE-CNN14 model obtained the highest MTADS of 81. Compared with the original model, the MFPE-CNN14 model improved autonomous driving performance by 13%. The MFPE-CNN14 model improved autonomous driving performance by 41% over the TDD-CNN model compared to the models in the existing studies. In addition, we found that optimizing the hyperparameters of the models improved the multi-task autonomous driving performance of the models. Hyperparameter fine-tuning resulted in an 8% improvement in autonomous driving performance of the MFPE-CNN14 model.

3.2.2 Multi-task autonomous driving in different lighting conditions

We tested the autonomous driving performance of the model under two different lighting conditions. Condition 1 used an indoor light source. Condition 2 used a natural light source. Figure 18 shows the MTADS of the autonomous driving model under different light sources.

Figure 18 shows that the autonomous driving performance of the model under indoor lighting is a little better than under natural lighting. The autonomous driving performance of our models are better under indoor lighting conditions.

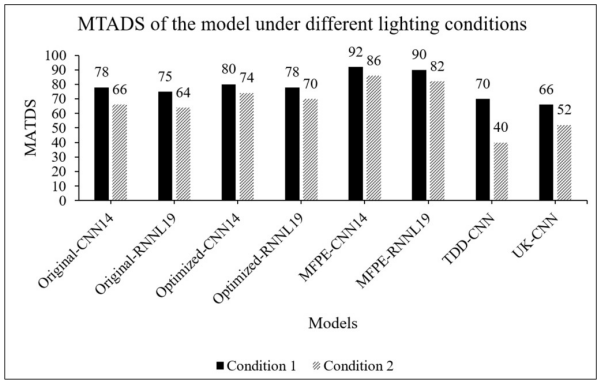


Fig.18: *Magnitude and Phase Response of the Dolph-Chebyshev Filter.*

3.2.3 Multi-task autonomous driving performance at different speeds

We tested the effect of different speeds on the performance of the model. The intelligent driving platform cannot drive when the throttle ratio is below 80%. Therefore, we divided the speeds into three classes: slow speed (80% throttle ratio, 4.8 Km/h), normal speed (90% throttle ratio, 5.4 Km/h) and fast speed (100% throttle ratio, 6 Km/h). The MTADS of the model at different speeds are shown in the Table 6.

We found that the faster the speed, the worse the performance of the model. This is because the increase in speed will increase the number of images acquired in the same time. A large number of images will be transferred to the Raspberry Pi for processing, and the increase in processing volume will lead to inference errors in the model. We must control the speed to be between 4.8-5.4 Km/h to ensure a good autonomous driving performance.

Table 5: *MTADS of different neural network models for multi-task autonomous driving.*

Multi-task autonomous driving NN models	Task	Single task score	MTADS	Multi-task autonomous driving NN models	Task	Single task score	MTADS
Original-CNN14	I	17	68	MFPE-CNN14	I	20	81
	II	15			II	19	
	III	11			III	17	
	IV	25			IV	25	
Original-RNNL19	I	13	60	MFPE-RNNL19	I	18	77
	II	11			II	17	
	III	11			III	17	
	IV	25			IV	25	
Function_Combinations-CNN14	I	19	73	TDD-CNN	I	18	40
	II	17			II	19	
	III	12			III	3	
	IV	25			IV	0	
Function_Combinations-RNNL19	I	14	63	UK-CNN	I	17	37
	II	13			II	2	
	III	11			III	18	
	IV	25			IV	0	

Table 6: *MTADS of the model at different speeds .*

Models	Speed	MTADS
MFPE-CNN14	Slow (80%)	94
	Normal (90%)	92
	Fast(100%)	84
MFPE-RNNL19	Slow (80%)	92
	Normal (90%)	92
	Fast(100%)	82

3.2.4 Multi-task autonomous driving on untrained routes

We tested the performance of the models for multi-task autonomous driving on untrained routes by randomly placing traffic signs. We observed whether different models could achieve autonomous driving in untrained scenarios without going off track. During autonomous driving, if the model achieved autonomous driving and did not lose control (run off track) during the autonomous driving process, then we calculated the MTADS of the models based on their performance on different untrained routes. The details are shown in Table 7.

As shown in Table 7, we found that the Optimized-CNN14, Optimized-RNNL19, MFPE-CNN14, and MFPE-RNNL19 models achieved multi-task autonomous driving for unseen scenarios. From the route analysis, Route 2 is more complex than Route 1, so the MTADS of Route 2 is lower than the MTADS of Route 1. The MTADS of the untrained scenario decreased compared to the MTADS of the trained route. Although the performance of the intelligent driving platform in the untrained scenario is not as good as that in the trained scenario, the intelligent driving platform did achieve multi-task autonomous driving.

3.2.5 Multi-task Autonomous driving obstacle avoidance performance

We tested the performance of the intelligent driving platform for obstacle avoidance using different obstacles. The results showed that the color of the obstacles did not affect the obstacle avoidance performance of the model. The shape of the obstacle had little effect on the obstacle avoidance performance of the model. Even with the changes in color and shape, the model could still achieve the obstacle avoidance task. The position of the obstacle has little effect on obstacle avoidance. Obstacles can be placed in any position, including untrained positions, without impacting obstacle avoidance. The model achieved poor performance in the obstacle avoidance task of moving obstacles. Based on the system memory footprint, the computational power of the Raspberry Pi is not sufficient to support the model's inferential computing process if the obstacles are moved in real time, based on the four complex subtasks already achieved. The disadvantage of the embedded development platform is that the lack of computational power makes certain tasks impossible.

Youwei Li conducted the experiment and drafted the manuscript. Jian Qu guided and advised the experiment and co-drafted the manuscript. Both authors each contributed equally to this work. Jian Qu is the corresponding author.

4. CONCLUSIONS

We integrated four sub-tasks into one neural network to achieve multi-task autonomous driving. We built a single-camera intelligent driving platform for experimentation.

In this work, we trained two types of deep neural networks: CNN14 and RNNL19. We improved the performance of the model by using different activation functions, loss functions, and activation-loss function combinations. However, the existing loss

Table 7: *MTADS of different models on untrained routes.*

Multi-task autonomous driving NN models	Untrained Route	Out of control?	MTADS	Multi-task autonomous driving NN models	Untrained Route	Out of control?	MTADS
Original-CNN14	Route 1	YES	0	MFPE-CNN14	Route 1	NO	75
	Route 2	YES	0		Route 2	NO	77
Original-RNNL19	Route 1	YES	0	MFPE-RNNL19	Route 1	NO	69
	Route 2	YES	0		Route 2	NO	73
Optimized-CNN14	Route 1	NO	55	TDD-CNN	Route 1	YES	0
	Route 2	NO	62		Route 2	YES	0
Optimized-RNNL19	Route 1	NO	46	UK-CNN	Route 1	YES	0
	Route 2	NO	51		Route 2	YES	0

function cannot handle the relationship between different subtask datasets. Therefore, we designed a new loss function (MFPE) based on multi-task autonomous driving. The MFPE loss function has the advantages of MSE and overcomes the disadvantages of MAE.

The MFPE loss function effectively improved the training and testing performance of the model. The MFPE-RNNL19 model performs best in model training, and its loss value is 47.14% less than the loss value of the original model. In addition, we optimized the model by fine-tuning the hyperparameters, which further reduced the loss of the MFPE-RNN19 model by 33.83%. The MFPE-CNN14 model performed the best in actual multi-task autonomous driving, and it improved its performance by 41% over the existing studies. In addition, we tested the effects of different lighting conditions, different speeds, untrained routes, and different obstacles on the actual autonomous driving performance of the model. The model using the MFPE loss function achieved multi-task autonomous driving under different lighting conditions. The intelligent driving platform performs better at speeds between 4.8 Km/h and 5.4 Km/h. The model using MFPE loss function achieved multi-task autonomous driving in different untrained routes. In the autonomous driving obstacle avoidance process, the model using the MFPE loss function can achieve multi-task autonomous driving using different colored obstacles, different shaped obstacles, and even with different positions of obstacles.

In addition, we tested avoidance of moving obstacles. The results show that the limitation of the computational power of the embedded platform makes the failure frequency of avoiding moving obstacles very high. In our future work, we will try more powerful chips to achieve moving obstacle avoidance.

ACKNOWLEDGMENT

Youwei Li received scholarship support from CPALL for conducting this research in PIM.

References

[1] J. Stilgoe, "Machine learning, social learning and the governance of self-driving cars," *Social studies of science*, vol. 48, no. 1, pp. 25-56, 2018.

[2] K. L. Lee and H. Y. Lam, "Development of Deep Learning Autonomous Car Using Raspberry Pi," *Progress in Engineering Application and Technology*, vol. 2, no. 1, pp. 534-548, 2021.

[3] X. Du, M. H. Ang, and D. Rus, "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)*, pp. 749-754, 2017.

[4] S. Ding, and J. Qu, "Smart car with road tracking and obstacle avoidance based on Resnet18-CBAM," *2022 7th International Conference on Business and Industrial Research (ICBIR)*, pp. 582-585, 2022.

[5] W. Y. Lin, W. H. Hsu, and Y. Y. Chiang, "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)*, pp. 262-266, 2018.

[6] R. Valiente, M. Zaman, S. Ozer, and Y. P. Fallah, "Controlling steering angle for cooperative self-driving vehicles utilizing cnn and lstm-based deep networks," *2019 IEEE intelligent vehicles symposium (IV)*, pp.2423-2428, 2019.

[7] M. G. Bechtel, E. McEllhiney, M. Kim, and H. Yun, "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)*, pp. 11-21, 2018.

[8] T. D. Do, M. T. Duong, Q. V. Dang, and M. H. Le, "Real-time self-driving car navigation using deep neural network," *2018 4th International Conference on Green Technology and Sustainable Development (GTSD)*, pp. 7-12, 2018.

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

[10] Y. Li and J. Qu, "Intelligent Road Tracking and Real-time Acceleration-deceleration for Au-

tonomous Driving Using Modified Convolutional Neural Networks," *Current Applied Science and Technology*, vol. 22, no. 6, pp.1-26, 2022.

[11] V. Mnih, et al., "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529-533, 2015.

[12] A. Boloor, K. Garimella, X. He, C. Gill, Y. Vorobeychik, and X. Zhang, "Attacking vision-based perception in end-to-end autonomous driving models," *Journal of Systems Architecture*, vol. 110, no. 101766, 2020.

[13] Y. Bae, E. Gomez, A. Haywood, J. Lazo, P. Whitson, and Y. Wang, "Prototyping a System of Cost-Effective Autonomous Guided Vehicles," *Proceedings of the Annual General Donald R. Keith Memorial Conference*, pp. 138-143, 2021.

[14] S. Yuenyong and J. Qu, "Generating synthetic training images for deep reinforcement learning of a mobile robot," *Journal of Intelligent Informatics and Smart Technology*, vol. 2, pp. 16-20, 2017.

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

[16] Y. Xiao, F. Codevilla, A. Gurram, O. Urfalioglu, and A. M. López, "Multimodal end-to-end autonomous driving," *IEEE Transactions on Intelligent Transportation Systems*, pp. 1-11, 2020.

[17] Z. Chen and X. Huang, "End-to-end learning for lane keeping of self-driving cars," *2017 IEEE Intelligent Vehicles Symposium (IV)*, pp. 1856-1860, 2017.

[18] A. Seth, A. James, and S. C. Mukhopadhyay, "1/10th scale autonomous vehicle based on convolutional neural network," *International Journal on Smart Sensing and Intelligent Systems*, vol. 13, no. 1, pp. 1-17, 2020.

[19] D. Molchanov, A. Ashukha, and D. Vetrov, "Variational dropout sparsifies deep neural networks," *International Conference on Machine Learning*, pp. 2498-2507, 2017.

[20] S. Phiphitphatphaisit and O. Surinta, "Deep feature extraction technique based on Conv1D and LSTM network for food image recognition," *Engineering and Applied Science Research*, vol. 48, no. 5, pp. 581-592, 2021.

[21] S. M. J. Jalali, P. M. Kebria, A. Khosravi, K. Saleh, D. Nahavandi, and S. Nahavandi, "Optimal autonomous driving through deep imitation learning and neuroevolution," *2019 IEEE International Conference on Systems, Man and Cybernetics (SMC)*, pp. 1215-1220, 2019.

[22] Y. H. Ko, K. J. Kim, and C. H. Jun, "A new loss function-based method for multiresponse optimization," *Journal of Quality Technology*, vol. 37, no. 1, pp. 50-59, 2005.

[23] J. Qi, J. Du, S. M. Siniscalchi, X. Ma, and C. H. Lee, "On mean absolute error for deep neural network based vector-to-vector regression," *IEEE Signal Processing Letters*, vol. 27, pp. 1485-1489, 2020.

[24] J. M. Martin-Donas, A. M. Gomez, J. A. Gonzalez, and A. M. Peinado, "A deep learning loss function based on the perceptual evaluation of the speech quality," *IEEE Signal processing letters*, vol. 25, no. 11, pp. 1680-1684, 2018.

[25] S. Fatimah, "Artificial neural network for modelling the removal of pollutants: A review," *Engineering and Applied Science Research*, vol. 47, no. 3, pp. 339-347, 2020.

[26] A. D. Rasamoelina, F. Adjailia, and P. Sinčák, "A review of activation function for artificial neural network," *2020 IEEE 18th World Symposium on Applied Machine Intelligence and Informatics (SAMI)*, pp. 281-286, 2020.

[27] J. Schmidt-Hieber, "Nonparametric regression using deep neural networks with ReLU activation function," *The Annals of Statistics*, vol. 48, no. 4, pp. 1875-1897, 2020.

[28] P. Bohra, J. Campos, H. Gupta, S. Aziznejad, and M. Unser, "Learning activation functions in deep (spline) neural networks," *IEEE Open Journal of Signal Processing*, vol. 1, pp. 295-309, 2020.

[29] M. Carvalho, B. Le Saux, P. Trouvé-Peloux, A. Almansa, and F. Champagnat, "On regression losses for deep depth estimation," *2018 25th IEEE International Conference on Image Processing (ICIP)*, pp. 2915-2919, 2018.

[30] X. Zhu, H. I. Suk, and D. Shen, "A novel matrix-similarity based loss function for joint regression and classification in AD diagnosis," *NeuroImage*, vol. 100, pp. 91-105, 2014.

[31] Z. Allen-Zhu, Y. Li, and Z. Song, "A convergence theory for deep learning via over-parameterization," *International Conference on Machine Learning (PMLR)*, vol. 97, pp. 242-252, 2019.

[32] D. O. Melinte, and L. Vladareanu, "Facial expressions recognition for human–robot interaction using deep convolutional neural networks with rectified Adam optimizer," *Sensors*, vol. 20, no. 8, pp. 2393, 2020.

[33] J. Yang, and G. Yang, "Modified convolutional neural network based on dropout and the stochastic gradient descent optimizer," *Algorithms*, vol. 11, no. 3, pp. 28, 2018.

[34] A. M. Taqi, A. Awad, F. Al-Azzo, and M. Milanova, "The impact of multi-optimizers and data augmentation on TensorFlow convolutional neural network performance," *2018 IEEE Conference on Multimedia Information Processing and Retrieval (MIPR)*, pp. 140-145, 2018.

Youwei Li is currently studying for the Master of Engineering Technology, Faculty of Engineering and Technology, Panyapiwat Institute of Management, Thailand. He received B.B.A from Nanjing Tech University Pujiang Institute, Chaina, in 2020. His research interests are Research direction is artificial intelligence, image processing, and autonomous driving.

Jian Qu is an Assistant professor at the Faculty of Engineering and Technology, Panyapiwat Institute of Management. He received Ph.D. with Outstanding Performance award from Japan Advanced Institute of Science and Technology, Japan, in 2013. He received B.B.A with Summa Cum Laude honors from Institute of International Studies of Ramkhamhaeng University, Thailand, in 2006, and M.S.I.T from Sirindhorn International Institute of Technology, Thammast University, Thailand, in 2010. He has been a house committee for Thai SuperAI since 2020. His research interests are natural language processing, intelligent algorithms, machine learning, machine translation, information retrieval and image processing.