Project ID:FYP_2122E_05

Deep Learning Applications for Autonomous Vehicle

by

Chan Pang Hoi
19099043D

Final Report

Bachelor of Engineering (Honours)
in
Transportation System Engineering

Of

The Hong Kong Polytechnic University

Supervisor: Prof. Fung Yu-fai, Isaac

Date:26/6/2022

Abstract

An autonomous vehicle (AV), as known as a driverless or self-driving vehicle, is a ground vehicle that is able to sense its environment for a safe auto movement with minor or even no human input. Navigation is vital for an AV's control system operation. Sensing technologies like GPS, Radar, Lidar, sonar, odometry, and even thermography are being applied in different ways on AV by engineers up to now. Among all available robotic approaches, camera vision is the most similar in simulating human sensing in driving – taking visual context as the major collecting function for the surrounding information, which collects the most information at a relatively low cost.

Meanwhile, due to the improvement of computational power and the development of mature image processing technology and artificial intelligence, tiny computer like Nano Jetson could also execute intelligent computer image processing functions that had never existed. With these improvements and evolution, through applying deep learning with Convolutional Neural Network (CNN) architecture for image processing of the navigation process in the control system, camera image could possibly be the only sensing tool for the development of the learning process of AV's control system and application in a real-world environment. CNN models are trained by images captured by a vehicle front camera as the input, and both steering angle and power input to wheels respectively as the output of the neural network.

Two methods were designed and trained for the system to automatically learn the road features and the designed guiding line when performing on straight and turning roads with camera images only, which is given with the steering angle and the separate speed value to both wheels respectively. Tests were made on a road plate with a cycling track. The performance was still not able to perform on real vehicles to operate on the real road. However, an impressive result of the model car was observed when moving according to the guiding road and demonstrates the possibility of performing autonomous driving with only camera image collection.

Acknowledgment

I would like to express my deep gratitude to Professor Fung, my research supervisor, for his patient guidance, enthusiastic encouragement, and useful critiques of this research work.

I would also like to extend my thanks to the technicians of the laboratory of the Electrical Engineering department for their help in offering me the resources in running the program.

Finally, I wish to thank my parents for their support and encouragement throughout my study.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background

Since the beginning of the 21st century, developing an efficient and reliable autonomous driving system for the purpose of fulfilling the enormous growth of city travel demand and reducing vehicle collision is one of the most challenging and competitive research areas in the automotive technology and AI field. The first-ever experiment on the automated driving system was conducted back in the 1920s [1], and the world's first semi-automated vehicle was developed and tested on the road with marks on the street and elevated rail in Japan in late 1970 by the Tsukuba Mechanical Engineering Laboratory [2]. In November 2017, an American autonomous driving technology development company, Waymo LLC, announced the operation of the test for driverless vehicles with no safety driver. In March 2021, Japan Motor Company, Honda started to release the newly equipped Level 3 automated driving equipment with the certification by the Japanese government's "Traffic Jam Pilot" technology sedans, which is the world's first sedans to allow drivers' sight off the road legally during the trip [3, 4].

Autonomous cars are able to offer huge benefits, such as significantly improved road safety and the reduction of pollution caused by the traffic, which are part of the ultimate goals of this technology. The National Highway Transportation Safety Administration (NHTSA) in the USA has released a research note about vehicle crashes, in which over 94% of car accidents are due to human error [5]. Applying an auto-driving system can largely reduce the involvement of human decision-making in a drive that potentially causes a traffic accident. Despite the rate of accidents for self-driving vehicles is currently higher than human-driven cars, the injuries are less severe [6]. Safety is the uppermost target for an autonomous driving system. With the invention of future technology, such as advanced traffic strategy technology and instant traffic image processing, it is believed that the rate of accidents for self-driving cars can be largely reduced. Simultaneously, the rate of accidents can also be decreased and lead to a reduction of traffic congestion on the road after the integration with the advanced travel strategy by the auto-driving system, which over 60% of harmful emissions can be

reduced [7]. Rand, an American nonprofit global policy think tank, has stated that fuel economy can be improved by around 4% to 10% with The Hong Kong Polytechnic University Department of Electrical Engineering self-driving technology, supported by the study from Ohio University that 3.1 billion gallons of fuel can be saved in the USA [8]. Rand also stated that it is possible to increase lane capacity by over 500% [8]. Another international organization, KPMG International Limited, released a research report which states that an autonomous driving system could reduce UK citizens' travel time up to 40%, with a saving of 20 billion UK dollars in time cost, 80 billion hours in the USA, which cost 1.3 trillion US dollars to support the statement of autonomous driving benefits the economy. The report from KPMG also stated that the economy can be saved not only through time cost, but also through insurance cost, running cost, parking cost by more accurate driving path, smart driving strategy, and advanced parking, in total 5 billion UK dollars a year in the UK [9].

Deep learning technology allows computers to "learn" how to drive by dividing the variability of the human's demonstration in driving in different situations on road. Other than applying codes in rules of autonomous driving, applying deep learning to autonomous driving is able to reduce the effort of developing a huge amount of codes and avoid the application of cost-intensive obstacle locating tools like radar and lidar.

In this study, I would like to propose the approach of applying deep learning technology by collecting driving parameters for the computer to "learn" and build up the well-trained model layers.

1.2 Objectives

This study aims to apply the deep learning method to develop the operating system for autonomous driving with:

i) High accuracy: The vehicle moves along the designed guiding line marked on the road.

ii) High stability: The vehicle moves smoothly without frequent or sudden unnecessary acceleration, deceleration or stop, and unreasonable turning, like moving in zig-zag motion on a straight road.

iii) High efficiency: The speed of the vehicle moves close enough to human driving.

As deep learning method applying to autonomous driving has a short history, this study also aims to provide experiment and explore the possibility of the future development of this idea.

Chapter 2

Literature Review

2.1 Deep learning

Deep learning, also named as deep structured learning, is a type of machine learning method with representation learning in either supervised, unsupervised or semi-supervised way [10]. The multiple layers network extracts high-level features from raw data input.

A deep learning system contains a deep credit assignment path (CAP) depth, which is the connection of how the input and output data are potentially connected. For example, in a feedforward neural network, since the output layer is also parameterized, the CAP of the network is n+1 (n is the number of hidden layers). Shallow learning and deep learning are similar and have no universal agreed-upon threshold to divide the differences, as it is widely believed that the depth of CAP higher than 2 will be considered as deep learning as any function can be emulated with this level of depth [11].

In supervised learning, feature extraction is removed, and replaced by the process of translating data to principal-components-like for redundancy removal.

2.2 Artificial neural network

Neural networks, similar to other networks like World Wide Web, transportation networks, formed by nodes (i.e. neurons in a neural network) and connected with links for data sharing between nodes are mathematical models for optimization of problem-solving.

A neural network simulates the neural system's architectural pattern in the human brain, which contain billions of neurons and nerves [12]. ANN contains the word "artificial" because it is created based on artificial neurons, not real neurons in the human brain. In real neurons, there is a branched protoplasmic extension called dendrite, which propagates the electrochemical stimulation received (i.e. input data). "Calculations" occurred when the data reaches the part called Soma and the processed data is then

transferred by another part of the neuron named Axon to another neuron's dendrite. "Thinking" is defined as the synaptic input being large enough to create an expected value.

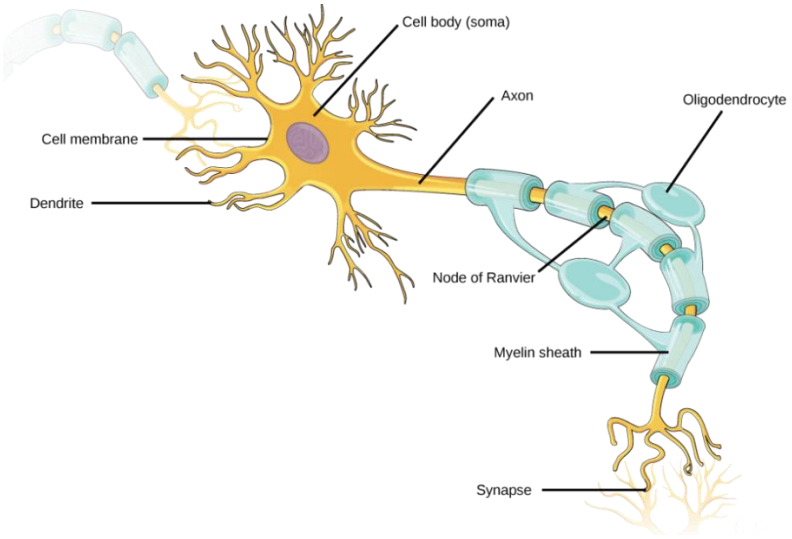


Figure 1. Structure of neurons in a human brain

Neurons are the main part of a neural network. Neurons act as the computation unit of the network, receiving the input, and producing an output which is the computed data according to the input to another neuron. The calculation method is called the activation function. Similarly, a simple graph with nodes and lines can simulates the performance of the activity of neurons in a human brain

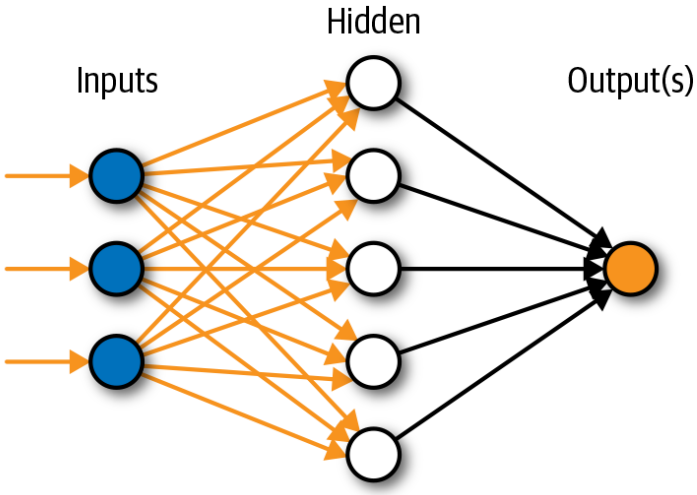


Figure 2. Layers in neural network

As shown in the above figure, the first layer with blue nodes is the input layer where the inputted raw data is passed to at first. The second layer with white nodes is for receiving data from the input layer, which is called the hidden layer(s). The final layer with orange node is to receive data from hidden layer(s) and is also the output layer of a neural network. Theoretically, more layers in the hidden layer tend to have higher accuracy as more features are extracted. A machine learning algorithm that applies a deep neural network to extract features from raw data is called a deep learning method [13].

2.2.1 Basic Artificial Neural Network

As mentioned, ANN is designed based on the nerve system's activity in a human brain, and the simulated procedure is called perceptron. The following is an example of solving a pattern recognition problem with perceptron and neural network.

	1st input	2nd input	3rd input	Output (either 1 or 0)
Input set a	0	0	0	0
Input set b	0	0	1	1
Input set c	0	1	1	0
Input set d	1	1	1	1
Input set e	1	0	1	0
Input set for guess	0	1	0	?

Table 1. Example of solving pattern recognition problem

With our human brain, we can notice that the pattern is outputting "1" when all three inputs together have odd number of "1", and outputting "0" for the rest, and the answer would be "1". By applying perceptron and neural network, a simple neural network can be created as below:

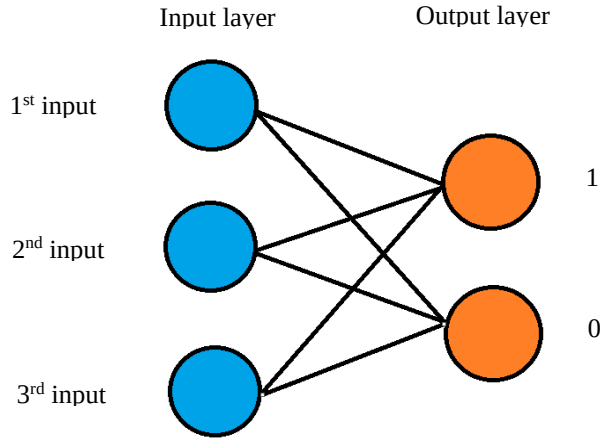


Figure 3. Structure of simple neural network

The input layer of the neural network contains three neurons that represent all three input data respectively, and two neurons at the output layer represent the output answer “0” and “1” respectively. The neural network above demonstrates a small set of neurons in the human brain for both learning and predicting the pattern.

2.2.2 Network training

Most of the training is under the supervised learning approach, which means the answer is provided and applied to the output layer. Before the training, links between neurons are given with weights. Input values are then multiplied with the weights and summed up before the activation function. The final output will be compared to the guiding output value. The difference between the output and the expected guiding output value is called error. The main part of the training is to reduce the error by adjusting the weights epodes by epodes. In each epode, three stages should be practiced and repeated in the next epode.

Step 1	Collect input data from the dataset and set original weights to links between neurons (only for the 1st epode). Sum up the value multiplied between input values and weights for the output value after applying the activation function.
Step 2	Calculate the value of error.

Step 3	Adjust the weight according to the error and linkage of the network.

Table 2. Procedures for network training

In step 1, the formula for the weights summing up progress is (assuming there is n input),

$$\sum weight * input = weight_0 + weight_1 * input_1 + weight_2 * input_2 + \cdots + weight_i * input_i$$

Equation (1)

There are various activation functions. ReLu and sigmoid are the famous functions that are commonly being used.

ReLu:

$$\max(0, x)$$

Equation (2)

Sigmoid:

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

Equation (3)

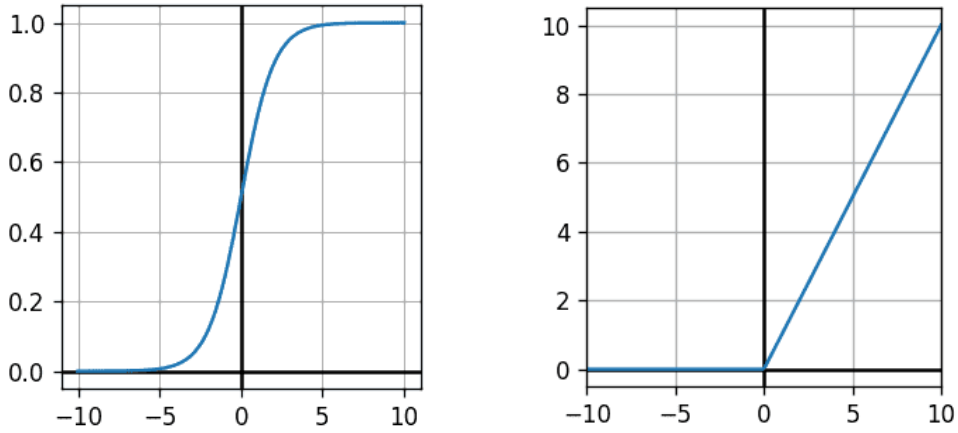


Figure 4. ReLu & Sigmoid

In step 2, the equation of error can be calculated by the following equation:

$$error = output_{expected} - output_{trained}$$

The update of weight in step 3 can be operated by the equation:

$$weight_{new} = weight_{old} - learning\ rate(\frac{\partial error}{\partial weight_{old}})$$

The calculation of the output value from the input layer, which goes forward through the hidden layer(s), is called the feed-forward; After the calculation of the output value, goes backward for weight update, called back-propagation.

2.2.3 Early stopping

Early stopping is a regularization for preventing the overfitting in machine learning during an iterative method of learning. With continuous training, the model should have a growing value in the test accuracy. However, with the increase of the epochs, the model might over trained with the unwanted features from the sample data, causing a decrease in the test accuracy, which is also called overfitting. Thus, a stopping action should occur before the model starts to overfit for keeping the high test accuracy value.

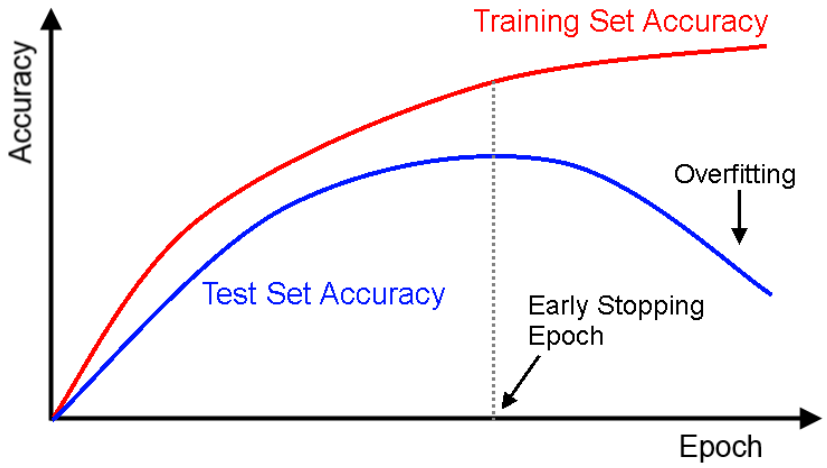


Figure 5. Early Stopping

2.3 Convolutional neural network

A digital image is a series of pixels combined with certain values that indicates color in a matrix, like a 2D matrix for greyscale photo, or a 3D matrix with RGB value for colourful photos. The detail of digital images can be performed in the mathematical function f(x,y), where x and y are the horizontal and vertical coordinates of the photo.

The value of colour is given to each of the xy pairs, ranging from 0 to 255 [14]. This characteristic makes digital images to be one of the most common objects applied in the deep learning field. A special type of deep neural network with a convolution technique specifically for extracting features from images was invented, named the convolutional neural network (CNN). Image recognition and image classification are the most common uses of CNN.

Convolutional Neural Network (CNN, or ConvNet), was inspired by the biological processes of animal visual context's neuron connecting pattern. CNN can be viewed as a regularized multilayer perceptron, which means all the neurons in a layer are fully connected to the next layer. Since multilayer perceptrons are easily resulted in overfitting to data because of the over-cloudy connection of neurons, regularizations, such as penalizing and trimming, are required to avoid the problem of overfitting. Different from regular approaches, CNN’s regularization applies hierarchical pattern technology to the data and gathers patterns for a higher level of complexity by applying patterns that are smaller in size and lower complexity to form filters.

The largest difference between CNN and other types of neuron networks is that the learning process is to optimize the filter (kernels) in the network. Thus, the filters are not hand-engineered as other types of neuron networks. A CNN consists of an input layer, hidden layers and an output layer that forms a classic forward-feeding neural network. Data are convoluted in hidden layers by the dot product of the input matrix. The product is mostly Frobenius inner product with ReLU (Rectified Linear Unit) as the activation function. The input data from the input layer is first reformed into a matrix, forming the input matrix for the first hidden layer for the convolution process. Various types of layers with different functions are used in CNN:

2.3.1 Convolution layer

The convolution layer aims to convolute the input from tensor in shape to feature map. The convolution in this layer is similar to the situation when the stimulus visual cortex reacts to special irritation. Neurons only process the data within their

sensory space into a single value. The data in the sensory space matrix passes through a filter and outputs data with activation function as the data at the relative part of the output matrix map.

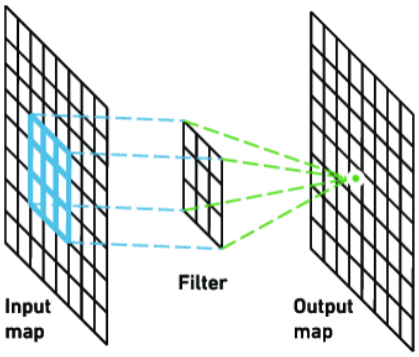


Figure 6. Outline of convolution layer

2.3.2 Pooling layer

Pooling layers are included in order to reduce the dimensions of the feature maps for allowing a fewer number of parameters to learn and the computation in the network. The pooling is mainly applied to summarize present features within a specific region instead of convoluting, and it can be expressed by averaging, maximizing, and globalling.

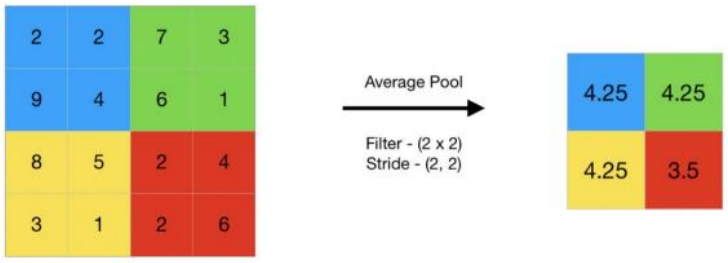


Figure 7. Demonstration of average pooling

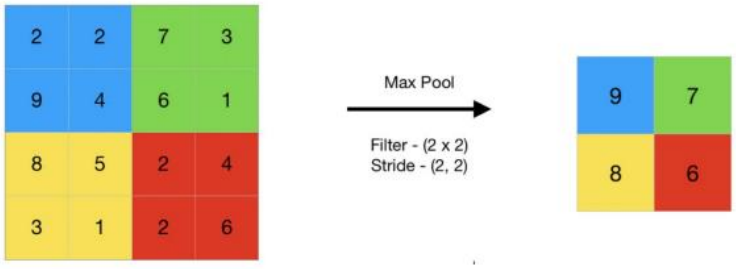


Figure 8. Demonstration of max pooling

2.3.3 Flatten layer & fully connected layer

The flatten layer is applied for reducing the dimension level of the data matrix. It mainly appears before the output layer and fully connected layers, which connect the hidden layer to the output layer, in full connection of every node in both layers.

Figure 9. Demonstration of flattening

2.4 AlexNet VS ResNet18

AlexNet is one of the earliest developed models for image processing. It contained eight layers with five convolutional layers at the beginning with three 3x3 max pooling layers, followed by three consecutive fully connected layers. Only ReLu is applied as the activation function in this model [15].

Figure 10. Structure layout of AlexNet

ResNet18, similar to the AlexNet, has the expected image size of 244x244, with 3 layers of RGB values. The model contains seventeen convolutional layers, with a max pooling layer and an average pooling layer, followed by a fully connected layer.

The accuracy of both networks is similar but the ResNet18has a wider range of usage [16].

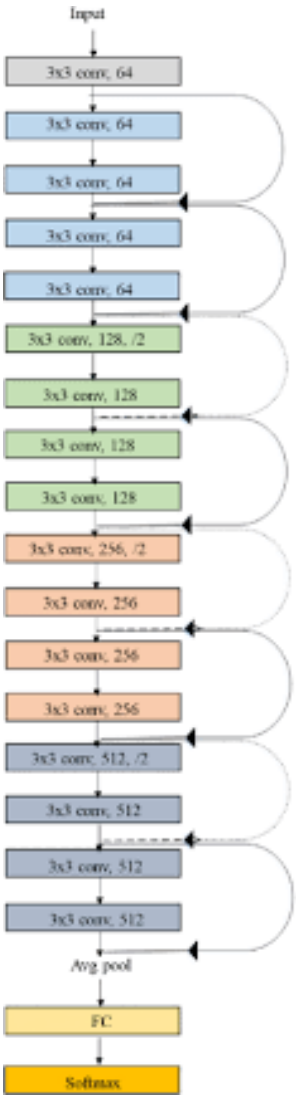


Figure 11. Structure layout of ResNet18

2.5 NVIDIA

As the leading company in developing deep learning technology, NVIDIA published their research result in the application of deep learning to autonomous vehicle technology. They disclosed an approach that the training and testing of the deep neural network successfully provides high accuracy steering angle of the autonomous car according to the input image captured from the front camera at completely new environment [17]. A system named “PilotNet”, which learns to emulate human behavior in driving and apply in the control of the real vehicle, were developed based on the method from the research.

2.6 Tesla

Founder of Tesla, Elon Musk, claimed the Tesla has developed the system for autonomous driving only with camera image. The system requires the driver to demonstrate their driving habit to the system, and the system learn through the deep learning approach and able to simulate the driving behavior of the demonstrator and drive automatically on road [18].

Chapter 3

Methodology

3.1 Model vehicle

3.1.1 Jetson Nano

Jetson Nano is a small, GPU computing platform with an embedded system design developed by NVIDIA that aims for providing designers or developers to run multiple neural networks in parallel for applications that run at around 5 watts. Jetson Nano was used as the computation platform for data collection and application of the trained model.

3.1.2 Camera

The Jetson Nano is connected to an IMX219-160 camera with the CSI port. It is a fisheye lens for capturing a larger area of surroundings compared to only using a single front camera. The height of the camera installed is set to be approximately 90 cm from the ground and the camera sight is horizontal to the ground. This camera is the only source that captures images for the input for both the training process and the instance image for the application test on the model.

Fig 12. IMX219-160 camera on Jetson Nano

3.1.3 Car body

The car body contains a 9.5 cm x 13.5 cm x 3.5cm metal box. Two motors

connecting to two wheels are respectively located on two sides of the metal box where slightly forward to the front. A back caster is installed at the bottom together with the two wheels forming triangular support to the model car, and for performing a smoother turning action. The width of the model car including the wheels is around 13.5cm. The Jetson Nano is located on the top of the metal box. The moving action is based on the two wheels at the side, which perform forward rotation or backward rotation for the forward and backward action, and different rotation speeds for the turning action.

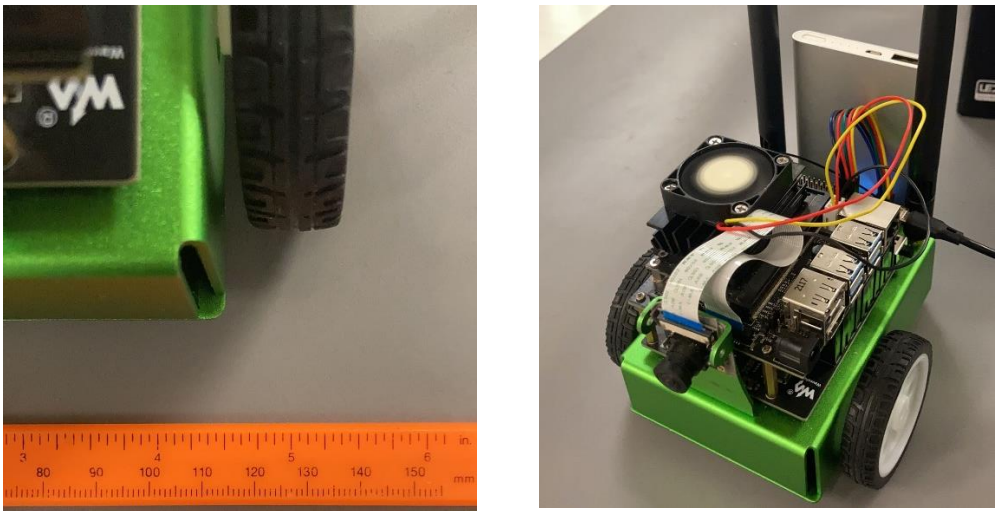


Fig 13. Width and outlook of the model car

3.1.4 Power system

Jetson Nano and the motors are powered by different power supplies respectively. Jetson Nano is powered by a portable charger, outputting 5.1V and 2.1A power to the Jetson Nano which is set to maximum 5W mode. Motors of the model car are powered by 3 INR 18650-25R rechargeable Li-Ion batteries, providing 9.62Wh (3.7V, 2600mAh) per battery.

3.2 Training/testing site

Both training and testing are on a man-made plate simulating driving in a real road situation. The size of the hole plate is 152 cm X 108.6 cm, with a 90 cm X 130 cm, a 24 cm radius with a 90° angle rectangular outer boundary, and 50cm X 90cm, a 4 cm radius

with 90° angle rectangular inner boundary as the track. The width of the track is 20 cm and the expected length of the model car to travel one cycle is

$$90 * 2 + 50 * 2 + \frac{2 * \pi * 14}{4} * 4 = 367.92cm$$

The background of the plate and the track are green and black in colour respectively with cardboard in texture. Masking tape is used as the boundary and the guiding line located in the middle of the track.

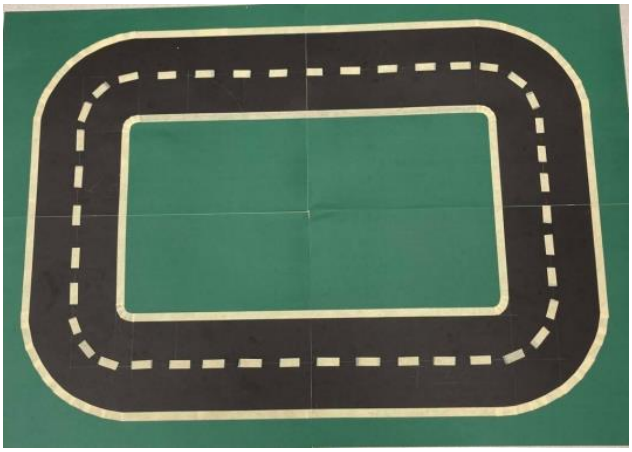


Figure 14. Training plate

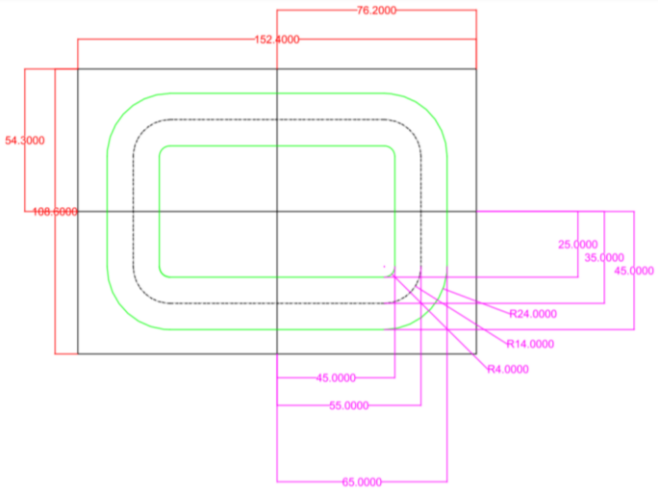


Figure 15. Size detail of the training plate

For a whole loop, the vehicle should pass two 14 cm radius 90° angle turning and 4 straight paths. As the model car only travels anti-clockwise in both the training and application process, only two travel situations are included, straight and left turning.

3.3 Jetbot

Jetbot is an open-source robot car operation system development code library developed for NVIDIA Jetson Nano. Jetbot allows the robot car with two wheels, which is based on the Jetson Nano as the core calculating machine and connects to accessories like a camera and controller, to control the model car and capture images. With the Jetbot library, simple actions can be performed by simple commands. For example:

robot.left(speed=0.3)

for performing counterclockwise spinning at 30% of the max speed the motors can perform.

3.4 Data collection

3.4.1 Steering angle method

Steering power is a parameter to supervise the training based on the assigned steering angle of the vehicle based on the surrounding. The model car was put on the track of the training plate at random angles to simulate the vehicle drove into different situations. Angle was plotted manually by the trainer on the model camera image shown on the screen for the supervised information. The image and the location of the dot were saved when the trainer clicked on the image. The angle was pointed according to the guiding lines of the plate, which is the expected line that the center point of the model car should be following while moving on the track in a normal situation. Together with the origin set at the middle of the bottom line of the image, the induced angle is the steering angle expected for the vehicle steer to. The horizontal and vertical distance between the green circle and the origin are stored at the name of the captured images' file name and the images were stored in size of 224*224. The data collection was done with a whole cycle of the track, with around 200 pairs of data collected.

Fig 16. Data collection in steering angle method

3.4.2 Software

The capture of camera image and the transformation from a colourful digital image to the jupyter notebook are operated by the Camera(), display(), and transform=bgr8_to_jpeg command.

```
camera = Camera()

camera_widget = ClickableImageWidget(width=camera.width, height=camera.height)
snapshot_widget = ipywidgets.Image(width=camera.width, height=camera.height)
traitlets.dlink((camera, 'value'), (camera_widget, 'value'), transform=bgr8_to_jpeg)

count_widget = ipywidgets.IntText(description='count')
count_widget.value = len(glob.glob(os.path.join(DATASET_DIR, '*.jpg')))
```

By default, captured camera images will be transformed into 224 x 224 digital images. The “click” event tells the system to operate the following actions when the clicking action happens and the curser is located to the displayed camera image. The coordinate of the point clicked on the image will be stored as “x” and “y” respectively, and the cv2.circle() command shows a green circle (0,225,0 for bgr colour value) at the pointed location. The coordinate and image are then stored by the os.path.join() command, which its name includes the coordination value and random value from uuid.

```
def save_snapshot(_, content, msg):
    if content['event'] == 'click':
        data = content['eventData']
        x = data['offsetX']
        y = data['offsetY']

        uuid = 'xy_%03d_%03d_%s' % (x, y, uuid1())
        image_path = os.path.join(DATASET_DIR, uuid + '.jpg')
        with open(image_path, 'wb') as f:
            f.write(camera_widget.value)

        snapshot = camera.value.copy()
        snapshot = cv2.circle(snapshot, (x, y), 8, (0, 255, 0), 3)
        snapshot_widget.value = bgr8_to_jpeg(snapshot)
        count_widget.value = len(glob.glob(os.path.join(DATASET_DIR, '*.jpg')))

camera_widget.on_msg(save_snapshot)
```

3.4.3 Wheel speed method

Unlike the previous method, the training-supervising data is the power input to the two motors separately. The motor was connected to a controller with two analog sticks and buttons. The two analog sticks control the power input to the two motors respectively. Both analog sticks were pushed to the maximum while the model car was moving in a straight direction; the left analog was pushed to a lower level than the right analog stick, causing the left wheel to rotate slower than the right wheel for performing a left-turning action. Buttons were connected to the system and pressed for photo taking of the camera image and the power input to the wheels respectively during the time the images were captured and saved. The value of the input power to both wheels was stored at the image file name and the images were stored in size of 224*224. Data were collected approximately every 0.3 seconds manually. Collected images were then separated into 5 classes (straight, turn_strong, turn_midstrong, turn_mid, and turnweak) according to the output power value level.

3.4.3.1 Software

The controller was connected by the widgets.Controller() command from the ipywidgets library. Same as the previous method, camera connection and setting, and the motor connection is executed by the Camera(), display() and transform=bgr8_to_jpeg command.

Fig 17. The controller

```
#create camera image
image = widgets.Image(format='jpeg', width=224, height=224)
camera = Camera.instance()
camera_link = traitlets.dlink((camera, 'value'), (image, 'value'), transform=bgr8_to_jpeg)

display(image)
```

```
Image(value=b'\xff\xd8\xff\xe0\x00\x10JFIF\x00\x01\x01\x00\x00\x01\x00\x01\x00\x00\xff\xdb\x00C\x00\x02\x01\x0…
```

The motors are connected to the controller's analog sticks by traitlets.dlink(). Controller.axes[1]&[5] are connected to the vertical axis of the left and right analog sticks respectively. The values of the analog sticks control the motors' rate of operation and direction of the rotation rating from -1 to 1 (negative is backward, positive is forward). As the wheels are not performing equally when given the same value, max operation rate to the left motor and right motor are limited at 0.4 and 0.37 respectively for performing straight movement when both analog sticks are pushed to the maximum together in the same direction.

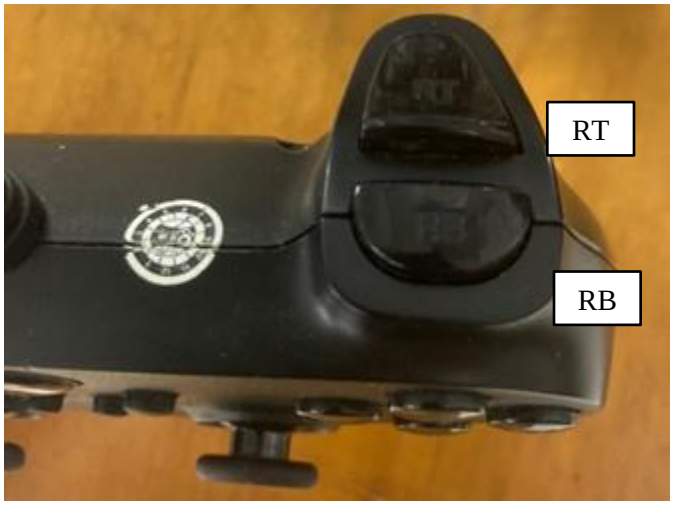


Fig 18. "RT" and "RB" buttons on the controller

The “RT” and “RB” buttons on the controller are set as the “shutter” for the snapshot of the camera image. When the “RB” button is triggered, a manually defined function named “save_snapshot_straight” is triggered for capturing the instant image from the camera by the f.write() command and given the name in the format of straight_l_r_uuid.jpg, where “l” and “r” are the operation value to the left and right motors respectively. Meanwhile, the “uuid” is a random value generated from the uuid1() function, and stored as a .jpg type file.

```
#creates new folder to store images
subprocess.call(['mkdir', '-p', 'snapshots'])

#image saving
snapshot_image = widgets.Image(format='jpeg', width=224, height=224)

#saving straight
def save_snapshot_straight(change):
    if change['new']:
        l = "{:.2f}".format(controller.axes[1].value)
        r = "{:.2f}".format(controller.axes[3].value)
        file_path = 'snapshots/' + 'straight_' + str(l) + '_' + str(r) + '_' + str(uuid.uuid1()) + '.jpg'

        # store snapshot to folder
        with open(file_path, 'wb') as f:
            f.write(image.value)

        snapshot_image.value = image.value
#saving turning
def save_snapshot_turn(change):
    if change['new']:
        l = "{:.2f}".format(controller.axes[1].value)
        r = "{:.2f}".format(controller.axes[3].value)
        file_path = 'snapshots/' + 'turn_' + str(l) + '_' + str(r) + '_' + str(uuid.uuid1()) + '.jpg'

        # store snapshot to folder
        with open(file_path, 'wb') as f:
            f.write(image.value)

        snapshot_image.value = image.value


controller.buttons[5].observe(save_snapshot_straight, names='value')
controller.buttons[7].observe(save_snapshot_turn, names='value')

display(widgets.HBox([image, snapshot_image]))
display(controller)
```

3.5 Training

Supervised learning approach is applied in the training of the neural network. The images collected will be converted into three layers storing the RGB values of the image separately as the data feed to the input layer of the training neuron network. The models were trained with tensorflow and pytorch respectively, and being compared.

3.5.1 TensorFlow

Even though Jetson Nano’s performance is close to an individual graphic card for

PC, running TensorFlow on PC is faster and less difficult in PC's 64-bit extended python than in the ARM architecture of the Jetson Nano. Therefore, the training was performed on PC, with image data extracted from the Jetson Nano

3.5.1.1 Steering angle method

349 data images are first separated into 6 classes, according to the arctan equation

$$\theta = \arctan\left(\frac{y}{x}\right)$$

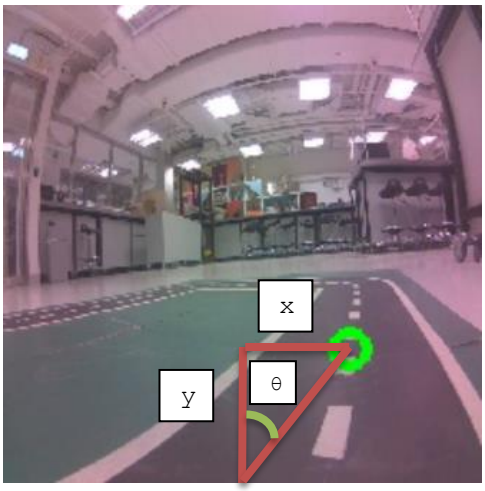


Fig 19. Application of arctan equation on image

, where theta is the steering angle, and x and y are the value stored in the image file's name representing the opposite line and adjacent line value of the theta angle respectively. The six classes are "0", "20", "40", "-20", "-40", and "-60", representing the designed steering angle for movement (positive represents right turn, negative represents left turn).

Theta	Designed angle
-90° to -50°	-60
-50° to -30°	-40
-30° to -5°	-20
-5° to 5°	0
5° to 30°	20
30° or above	40

Table 3. Transformation between theta and designed angle

J2 =IF(I2<-50,-60,IF(I2<-30,-40,IF(I2<-5,-20,IF(I2<5,0,IF(I2<30,20,40)))))

	A	B	C	D	E	F	G	I	J	K	L	M	N	O
1						x coordinate	y coordinate	steering angle	class					
2	C:	Us	ho	D	xy	25	198	-81.89	-60				width	224
3	C:	Us	ho	D	xy	26	182	-88.59	-60				length	224
4	C:	Us	ho	D	xy	27	183	-86.95	-60					
5	C:	Us	ho	D	xy	28	192	-81.51	-60					
6	C:	Us	ho	D	xy	29	184	-84.24	-60					
7	C:	Us	ho	D	xy	29	195	-79.14	-60					
8	C:	Us	ho	D	xy	29	197	-78.28	-60					
9	C:	Us	ho	D	xy	31	190	-79.27	-60					
10	C:	Us	ho	D	xy	32	183	-81.44	-60					
11	C:	Us	ho	D	xy	33	170	-86.99	-60					
12	C:	Us	ho	D	xy	34	193	-74.89	-60					
13	C:	Us	ho	D	xy	34	193	-74.89	-60					
14	C:	Us	ho	D	xy	37	194	-71.45	-60					
15	C:	Us	ho	D	xy	39	176	-76.98	-60					

Fig 20. Classification of the collected images with supervising data

The batch size was 8, and the train-test ratio was 9:1. AlexNet was applied as the model structure.

```
batch_size = 8
img_height = 224
img_width = 224
```

```
train_ds = tf.keras.utils.image_dataset_from_directory(
  data_dir,
  validation_split=0.1,
  subset="training",
  seed=123,
  image_size=(img_height, img_width),
  batch_size=batch_size)
```

```
Found 349 files belonging to 6 classes.
Using 315 files for training.
```

As the keras library in TensorFlow has no pre-trained model, or has to be additionally installed, the AlexNet structure was built manually with the Kerasin this training. The summary of the built model is shown below:

```
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_15 (Conv2D)          (None, 54, 54, 96)        34944

 batch_normalization_15 (Bat  (None, 54, 54, 96)       384
 chNormalization)

 max_pooling2d_9 (MaxPooling  (None, 26, 26, 96)       0
 2D)

 conv2d_16 (Conv2D)          (None, 26, 26, 256)       614656

 batch_normalization_16 (Bat  (None, 26, 26, 256)      1024
 chNormalization)

 max_pooling2d_10 (MaxPoolin  (None, 12, 12, 256)      0
 g2D)

 conv2d_17 (Conv2D)          (None, 12, 12, 384)       885120

 batch_normalization_17 (Bat  (None, 12, 12, 384)      1536
 chNormalization)

 conv2d_18 (Conv2D)          (None, 12, 12, 384)       1327488
batch_normalization_18 (Bat  (None, 12, 12, 384)      1536
 chNormalization)

 conv2d_19 (Conv2D)          (None, 12, 12, 256)       884992

 batch_normalization_19 (Bat  (None, 12, 12, 256)      1024
 chNormalization)

 max_pooling2d_11 (MaxPoolin  (None, 5, 5, 256)        0
 g2D)

 flatten_3 (Flatten)         (None, 6400)              0

 dense_9 (Dense)             (None, 4096)              26218496

 dropout_6 (Dropout)         (None, 4096)              0

 dense_10 (Dense)            (None, 4096)              16781312

 dropout_7 (Dropout)         (None, 4096)              0

 dense_11 (Dense)            (None, 6)                 24582

=================================================================
Total params: 46,777,094
Trainable params: 46,774,342
Non-trainable params: 2,752
```

Fig 21. Structure layout of the model

The optimizer was set as Adam, and the training loss, train accuracy, test loss, and test accuracy are shown for every epoch. With the ModelCheckpoint and EarlyStopping functions in the keras.callbacks library, the training of the model automatically stopped when the test accuracy did not improve more than 0.0001 for 5 consecutive epochs in preventing overfitting.

The EarlyStopping() function was set to be triggered when there was no improvement in val_accuracy (test accuracy) for ten epochs and call back the epoch with the best val_accuracy. The highest record of the test accuracy was 0.7059 after the 17^{th} epoch. The training loss, train accuracy and the test loss of the model called back were 0.9416, 0.6889, and 0.8130 respectively.

```
Epoch 13/120
79/79 [==============================] - 41s 525ms/step - loss: 1.0864 - accuracy: 0.5841 - val_loss: 0.9912 - val_accuracy: 0.
7059
Epoch 14/120
79/79 [==============================] - 41s 519ms/step - loss: 0.9590 - accuracy: 0.6254 - val_loss: 0.8718 - val_accuracy: 0.
7059
Epoch 15/120
79/79 [==============================] - 41s 521ms/step - loss: 1.0597 - accuracy: 0.6222 - val_loss: 1.6815 - val_accuracy: 0.
6176
Epoch 16/120
79/79 [==============================] - 44s 553ms/step - loss: 1.1915 - accuracy: 0.6222 - val_loss: 0.5488 - val_accuracy: 0.
Epoch 17/120
79/79 [==============================] - 43s 540ms/step - loss: 0.9416 - accuracy: 0.6889 - val_loss: 0.8130 - val_accuracy: 0.
7059
79/79 [==============================] - 43s 541ms/step - loss: 1.0169 - accuracy: 0.6476 - val_loss: 0.6969 - val_accuracy: 0.
6765
Epoch 19/120
79/79 [==============================] - 42s 525ms/step - loss: 1.1620 - accuracy: 0.6444 - val_loss: 1.4577 - val_accuracy: 0.
5882
Epoch 20/120
79/79 [==============================] - 42s 530ms/step - loss: 1.0398 - accuracy: 0.6635 - val_loss: 0.8066 - val_accuracy: 0.
6765
Epoch 21/120
79/79 [==============================] - 41s 515ms/step - loss: 0.9415 - accuracy: 0.6857 - val_loss: 0.7911 - val_accuracy: 0.
7059
Epoch 22/120
79/79 [==============================] - 42s 528ms/step - loss: 0.8106 - accuracy: 0.7016 - val_loss: 0.6297 - val_accuracy: 0.
```

```
earlystop_callback = EarlyStopping(
  monitor = 'val_accuracy', min_delta=0.0001,
  patience = 10)

epochs = 120
history = model.fit(
  train_ds,
  validation_data = val_ds,
  epochs = epochs, callbacks = [earlystop_callback]
)
```

The model with trained weights stored the structure and weight values together by a .json and .h5 file. Since the version of Tensor Flow in Jetson Nano is TensorRT, it is necessary to convert the .h5 file to .trt file for the application in Jetson Nano. ONNX is the bridge for this converting action by rewriting the data in the .h5 file in .onnx file format, then convert to .rtr file by the onnx2trt function.

3.5.1.2 Motor speed method

A total of 660 image data was collected for the database for training. The image data were first separated by the period of the collection (collected under moving on straight line or turning). The turning data were then once again separated into four classes according to the speed of the left wheel (the effect of the right wheel was ignored as the right analog stick was pushed to the maximum during the whole data collection process).

Input value to right motor	0 ~ 10	11 ~ 20	21 ~ 30	31 ~ 40	straight
class	turn_strong	turn_midstrong	turn_mid	turn_weak	straight

Table 4. Reaction when inputting different values to right motor

Same process as the previous method of storing the image data converted into 224x224 three layers bgr value data to a database before passing to the training model. The validation split value was 0.1 (594 as training data, 66 as testing data), batch size of 4, and same neural network (AlexNet) for training and application progress.

```
batch_size = 4
img_height = 224
img_width = 224
```

```
train_ds = tf.keras.utils.image_dataset_from_directory(
  data_dir,
  validation_split=0.1,
  subset="training",
  seed=123,
  image_size=(img_height, img_width),
  batch_size=batch_size)
```

```
Found 660 files belonging to 5 classes.
Using 594 files for training.
```

```
val_ds = tf.keras.utils.image_dataset_from_directory(
  data_dir,
  validation_split=0.1,
  subset="validation",
  seed=123,
  image_size=(img_height, img_width),
  batch_size=batch_size)
```

```
Found 660 files belonging to 5 classes.
Using 66 files for validation.
```

The optimizer was set as Adam. The same method was applied for performing early stopping in preventing overfitting. The best model called back was the 16th epoch with 0.7273 test accuracy, 0.7222 training accuracy, 0.7063 training loss, and 0.847 test loss.

```
Epoch 13/120
149/149 [==============================] - 77s 520ms/step - loss: 0.7434 - accuracy: 0.7323 - val_loss: 0.6985 - val_accuracy:
0.6364
Epoch 14/120
149/149 [==============================] - 80s 535ms/step - loss: 0.6322 - accuracy: 0.7609 - val_loss: 0.8051 - val_accuracy:
0.6515
Epoch 15/120
149/149 [==============================] - 79s 532ms/step - loss: 0.7708 - accuracy: 0.7357 - val_loss: 2.2482 - val_accuracy:
0.6818
Epoch 16/120
149/149 [==============================] - 80s 537ms/step - loss: 0.7063 - accuracy: 0.7222 - val_loss: 0.8470 - val_accuracy:
0.7273
Epoch 17/120
149/149 [==============================] - 79s 529ms/step - loss: 0.7934 - accuracy: 0.7088 - val_loss: 0.8036 - val_accuracy:
0.6212
Epoch 18/120
149/149 [==============================] - 79s 533ms/step - loss: 0.6845 - accuracy: 0.7458 - val_loss: 10.1608 - val_accuracy:
0.6212
Epoch 19/120
149/149 [==============================] - 78s 524ms/step - loss: 0.5883 - accuracy: 0.7727 - val_loss: 0.8631 - val_accuracy:
0.6667
Epoch 20/120
149/149 [==============================] - 79s 528ms/step - loss: 0.9708 - accuracy: 0.6919 - val_loss: 1.1012 - val_accuracy:
0.6212
Epoch 21/120
149/149 [==============================] - 78s 524ms/step - loss: 0.8288 - accuracy: 0.7054 - val_loss: 0.7853 - val_accuracy:
0.6515
```

3.5.2 Pytorch

Pytorch is a common library for the neural network training process in Jetson Nano. Unlike the Tensor Flow, Pytorch provides pre-trained models like VGG, ResNet…etc. The solution to prevent overfitting is also different in that pytorch allows automatically saving the best model according to the test accuracy. Since Pytorch is the official recommended method to operate training inside Jetson Nano, the training occurred in the Jetson Nano.

3.5.2.1 Steering angle method

This method has two significant differences from other methods with or without applying Pytorch's library for training and using Resnet18 instead of AlexNet. First, this is the only one doing regression training instead of classification. In regression training, the output is the predicted value through the calculation across the neural network instead of predicting a class. Thus, the output layer contains only one node. Since AlexNet was unable to perform regression training, a similar model Resnet18 was used.

```
model = models.resnet18(pretrained=True)
```

A manually set class named steer_Dataset was set to extract the x and y values from the name of the image files, transforming image files to data with bgr values, and stored as a dataset for the training.

```
def collect_x(path, width):
    return (float(int(path.split("_")[1])) - width/2) / (width/2)

def collect_y(path, height):
    return (float(int(path.split("_")[2])) - height/2) / (height/2)

class steer_Dataset(torch.utils.data.Dataset):

    def __init__(self, directory, random_hflips=False):
        self.directory = directory
        self.random_hflips = random_hflips
        self.image_paths = glob.glob(os.path.join(self.directory, '*.jpg'))
        self.color_jitter = transforms.ColorJitter(0.3, 0.3, 0.3, 0.3)

    def __len__(self):
        return len(self.image_paths)

    def __getitem__(self, idx):
        image_path = self.image_paths[idx]

        image = PIL.Image.open(image_path)
        width, height = image.size
        x = float(collect_x(os.path.basename(image_path), width))
        y = float(collect_y(os.path.basename(image_path), height))
        image = self.color_jitter(image)
        image = transforms.functional.resize(image, (224, 224))
        image = transforms.functional.to_tensor(image)
        image = image.numpy()[::-1].copy()
        image = torch.from_numpy(image)
        image = transforms.functional.normalize(image, [0.485, 0.456, 0.406], [0.229, 0.224, 0.225])

        return image, torch.tensor([x, y]).float()

dataset = steer_Dataset('dataset')
```

The validation split was set to 9:1, and the batch size was 4.

```
val_split = 0.1
num_test = int(val_split * len(dataset))
train_dataset, test_dataset = torch.utils.data.random_split(dataset, [len(dataset) - num_test, num_test])
```

```
train_loader = torch.utils.data.DataLoader(
    train_dataset,
    batch_size=4,
    shuffle=True,
    num_workers=0
)

test_loader = torch.utils.data.DataLoader(
    test_dataset,
    batch_size=4,
    shuffle=True,
    num_workers=0
)
```

The training process was then transferred to the GPU in the Jetson Nano to run the training process by CUDA, which is a parallel computing platform and application programming interface (API) that allows the software to use certain types of graphics processing units developed by NVIDIA.

```
device = torch.device('cuda')
model = model.to(device)
```

The train ended after the 70th epoch, Adam was set as the optimizer and automatically set the model with the highest accuracy named best_steering_model_xy.pth.

```
NUM_EPOCHS = 120
BEST_MODEL_PATH = 'best_steering_model_xy.pth'
best_loss = 1e9

optimizer = optim.Adam(model.parameters())
```

The final model recorded with the lowest test error was 0.000467 after the 82nd epoch. As the error is calculated by the difference between the actual output and predict output, based on the actual output's value,

$$error = \frac{\overline{output_{predicted} - output_{expected}}}{output_{expected}}$$

the accuracy is simply calculated by:

$$accuracy = 100\% - error$$

Thus, the training accuracy of the final model was 99.95%, and the test accuracy was 99.9531%.

0.006625, 0.005889
0.006316, 0.001139
0.001259, 0.001097
0.001396, 0.001448
0.000995, 0.000542
0.000898, 0.001249
0.000715, 0.000582
0.000711, 0.001457
[illegible]
Train error | 0.000500, 0.000469 | Test error
[illegible]
0.000648, 0.000467
0.000614, 0.001001
0.000551, 0.000596
0.000562, 0.002137
0.001298, 0.001514
0.000386, 0.000542
0.000414, 0.000826
0.000464, 0.000858
0.000376, 0.001401
0.001044, 0.001552
0.001980, 0.000799
0.000688, 0.001123
0.000456, 0.000939
0.001636, 0.000697
0.001128, 0.003090
0.001396, 0.000700
0.000782, 0.001360
0.000577, 0.000710
0.000847, 0.000615

3.5.2.2 Motor speed method

The image data were first recalled, resized, and stored in the database class by class, and split into training data and test data with the ratio of 9:1. Afterward, data is shuffled before each epoch in the training process. The batch size was set to 4, and the model used the pre-trained model Alexnet.

```
dataset = datasets.ImageFolder(
    'snapshots',
    transforms.Compose([
        transforms.ColorJitter(0.1, 0.1, 0.1, 0.1),
        transforms.Resize((224, 224)),
        transforms.ToTensor(),
        transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
    ])
)
```

```
train_dataset, test_dataset = torch.utils.data.random_split(dataset, [len(dataset) - int(len(dataset)*0.1), int(len(dataset)*0.1)])
```

```
train_loader = torch.utils.data.DataLoader(
    train_dataset,
    batch_size=4,
    shuffle=True,
    num_workers=0
)

test_loader = torch.utils.data.DataLoader(
    test_dataset,
    batch_size=4,
    shuffle=True,
    num_workers=0
)
```

A total of 120 epochs were performed by the training process in the GUP of the Jetson Nano with the linkage by CUDA. The model with the highest test accuracy appeared after the 18th epoch, with 0.848485. The set of weights after the 18th epoch was stored as "best_model.pth" for the application afterward.

```
16: 0.727273
17: 0.772727
18: 0.848485
19: 0.818182
20: 0.787879
21: 0.727273
22: 0.727273
23: 0.712121
24: 0.681818
25: 0.772727
26: 0.757576
27: 0.712121
```

The output layer was reset to 6 nodes which is one more than the number of classes. It is because during the pairing process of the classes to the nodes, the classes were remarked with numbers, starting from 1. However, in Python, the cells in the array are counted from "0". If the output layer's nodes were set as the same as the number of classes, the output layer structure will be unmatched in the database as the matching was done according to the given remarks and the serial number of the cells in the array.

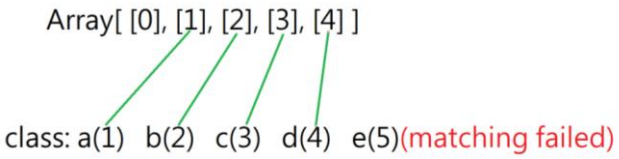


Figure X Demonstration of class matching to output layer in Python

Thus, one more node was added to fulfill the requirement of the matching process.

3.5.3 Tensor Flow or Pytorch

In this project, models trained from the Jetson Nano were applied for the application. There are three reasons that the TensorFlow method was not chosen. First, the weights potentially destroyed are changed through the process of transforming file formats. Second, the difficulty to convert .h5 file to .trt file is too high as the process requires lots of libraries that are not common and require specific versions of related libraries. Many different kinds of failure examples in performing the conversion are shown on the internet. An incompatible issue may easily happen if the same training is performed in the future with updated libraries. Even though training in a PC is faster than in the Jetson Nano, and the early stopping method helps reduce the time consumed for running too many unnecessary epochs, the time difference is not significant and decisive. Besides, the structure is developing the training program by Pytorch is much simpler than Tensor Flow, which is more friendly for future program optimization development.

3.6 Application

3.6.1 Steering angle method

The camera images captured by the model car on the plate's track were captured and converted into 224x224 3 layers bgr value as the input to the trained neural network. The model with trained weights stored in the format of .pth and the ResNet18 structure was called again and installed into the GPU of the Jetson Nano once again for the application of the trained network by CUDA. Four sliders were set for the control of the input value to the motors to perform at different speeds of the wheels. "Speed gain" slider controlled the base input value to the motor, while "steering gain" slider was the base value for the control of the extra value gain/loss for performing turning actions. "Steering kd" controlled the sensitivity value to perform turnings (i.e. if the calculated steering angle was resulting in low, the system did not perform turning), and the "steering bias" slider was the miner modifier of the speed of the wheels for fixing the wheels' unbalancing issue. The figure below shows the optimum setting for the best

performance of the model car.

Fig 22. Optimum setting for best performance

The motor's final input value was the sum of the value in the "speed gain" slide, which represents the basic input to both motors with a rating from 0 to 1, and the calculated external input to both motors for performing turning actions. The external input for the purpose of steering was calculated by the equation:

$$steering\ value = steering\ angle_{new} + steering\ gain\ value + (steering\ angle_{new} - steering\ angle_{old}) * sensitivity\ value$$

The value of each motor was calculated with the equation:

$$motor\ value_{left} = speed\ gain\ slider\ value + steering\ value$$

$$motor\ value_{right} = speed\ gain\ slider\ value - steering\ value$$

When the system predicted a left turn, the steering value would be negative. Oppositely, the steering value would be positive when a right turn was predicted.

```
angle = 0.0
angle_last = 0.0

def execute(change):
    global angle, angle_last
    image = change['new']
    xy = model(preprocess(image)).detach().float().cpu().numpy().flatten()
    x = xy[0]
    y = (0.5 - xy[1]) / 2.0

    x_slider.value = x
    y_slider.value = y

    speed_slider.value = speed_gain_slider.value

    angle = np.arctan2(x, y)
    pid = angle * steering_gain_slider.value + (angle - angle_last) * steering_dgain_slider.value
    angle_last = angle

    steering_slider.value = pid + steering_bias_slider.value

    robot.left_motor.value = max(min(speed_slider.value + steering_slider.value, 1.0), 0.0)
    robot.right_motor.value = max(min(speed_slider.value - steering_slider.value, 1.0), 0.0)

execute({'new': camera.value})
```

3.6.2 Motor speed method

Same as the previous method, the camera images captured by the model car on the plate's track were captured and converted into 224x224 3 layers bgr value as the input to the trained neural network. The model with trained weights stored in the format of .pth and the ResNet18 structure was called again and installed into the GPU of the Jetson Nano once again for the application of the trained network by CUDA. The neural network in the system predicted the most possible situation out of the five designed classes by giving the classes a predicted value, where the class with the highest predicted value becomes the prediction. The input value to motors was fixed for the prediction out of the five classes predicted by the system.

```
def update(change):
    x = change['new']
    x = preprocess(x)
    y = model(x)

    st = float(y.flatten()[1])
    mid = float(y.flatten()[2])
    mid_stg = float(y.flatten()[3])
    stg = float(y.flatten()[4])
    weak = float(y.flatten()[5])

    if max(st, mid, mid_stg, stg, weak) == st:
        robot.left_motor.value = 0.4
        robot.right_motor.value = 0.37
    elif max(st, mid, mid_stg, stg, weak) == weak:
        robot.left_motor.value = 0.35
        robot.right_motor.value = 0.37
    elif max(st, mid, mid_stg, stg, weak) == mid:
        robot.left_motor.value = 0.25
        robot.right_motor.value = 0.37
    elif max(st, mid, mid_stg, stg, weak) == mid_stg:
        robot.left_motor.value = 0.15
        robot.right_motor.value = 0.37
    elif max(st, mid, mid_stg, stg, weak) == stg:
        robot.left_motor.value = 0.05
        robot.right_motor.value = 0.37

update({'new': camera.value})
```

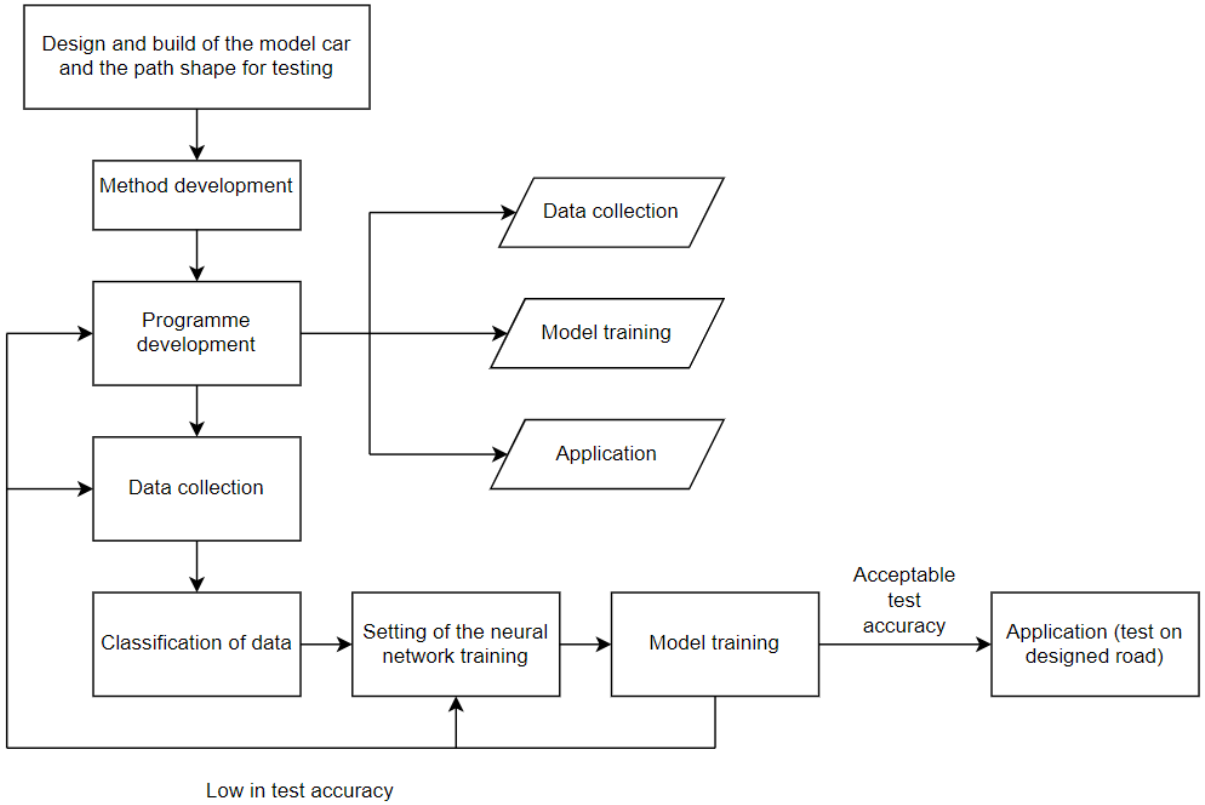
The table below shows the input value to both motors respectively according to the predicted classes.

Class	st (straight)	weak (turn_weak)	mid (turn_mid)	mid_stg (turn_midstrong)	Stg (turn_strong)
Left motor	0.4	0.35	0.25	0.15	0.05
Right motor	0.37				

Table 5. Input values applied to the left and right motor respectively

The value of the input to both motors was different due to the unbalance of the motors

3.7 System block diagram



Chapter 4

Result and Analysis

4.1 Steering angle method

The model car was controlled by the operation system under the result from the neural network and tested for three consecutive cycles without human interruption.

4.1.1 Accuracy

The model car successfully performed cycling as the returning position of the cycle was similar to the starting position of the cycle and did not leave the plate.

Figure 23.
Starting position of the cycle for cycle 1

Figure 24.
Returning position after the cycle for cycle 1

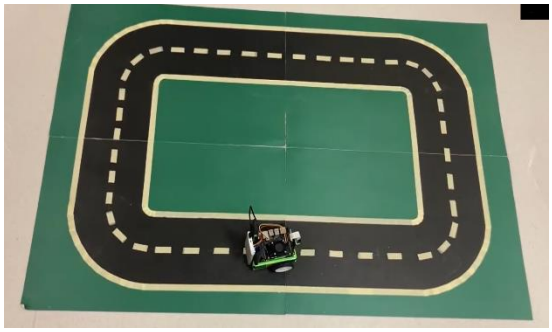


Figure 25.
Returning position after the cycle for cycle 2

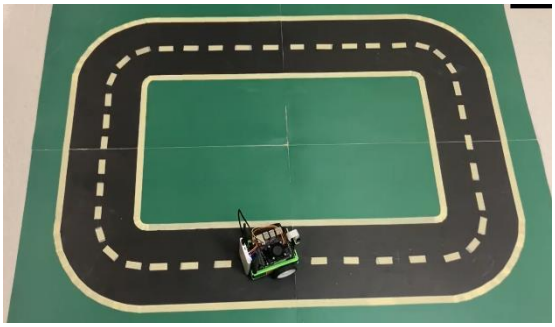


Figure 26.
Returning position after the cycle for cycle 3

Even though the shape of the path of the model car moved in a single cycle similar to the shape of the guiding line (i.e. rectangular shape, rounded in four corners), the path has deviated from the guiding line. Early turning and late turning happened in road turn, and the path passing the straight lines at the four sides also deviated to both sides.

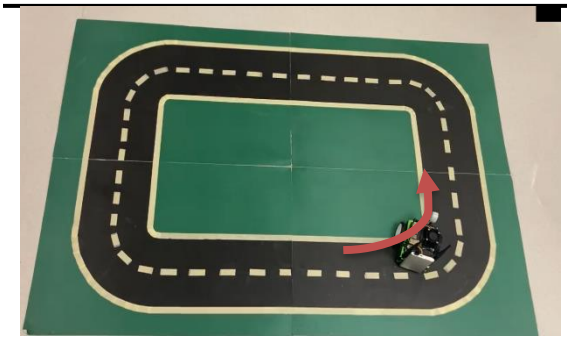


Figure 27. Example of early turning

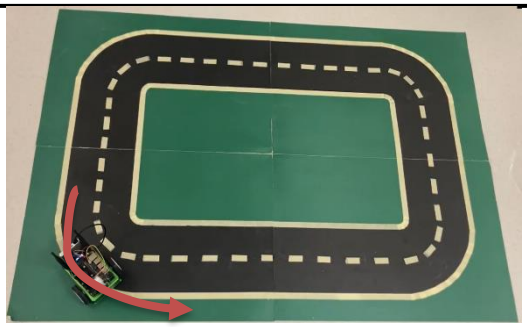


Figure 28. Example of late turning

The guiding line in general did successfully guide the shape of the path for the model car to follow. However, the model car failed to stick to the guiding line which the model car was expected to move right above. Thus, the accuracy was not satisfied.

The deviation of the moving path and the guiding line was potentially caused by the low quality of examples collected. As the guiding information was given under the distorted image collected by the camera with a fisheye lens, the judgment of the supervising steering angle was affected by the distortion and the real situation was easily being ignored, causing the given steering angle value for the supervising data to easily having bias, especially for situations near the turnings.

4.1.2 Stability

In turnings, the model car moved stably with constant steering angle and speed. However, a zigzag pattern appears when moving along straight lines. Below shows the direction the model was pointing at when moving on a straight line in a period.

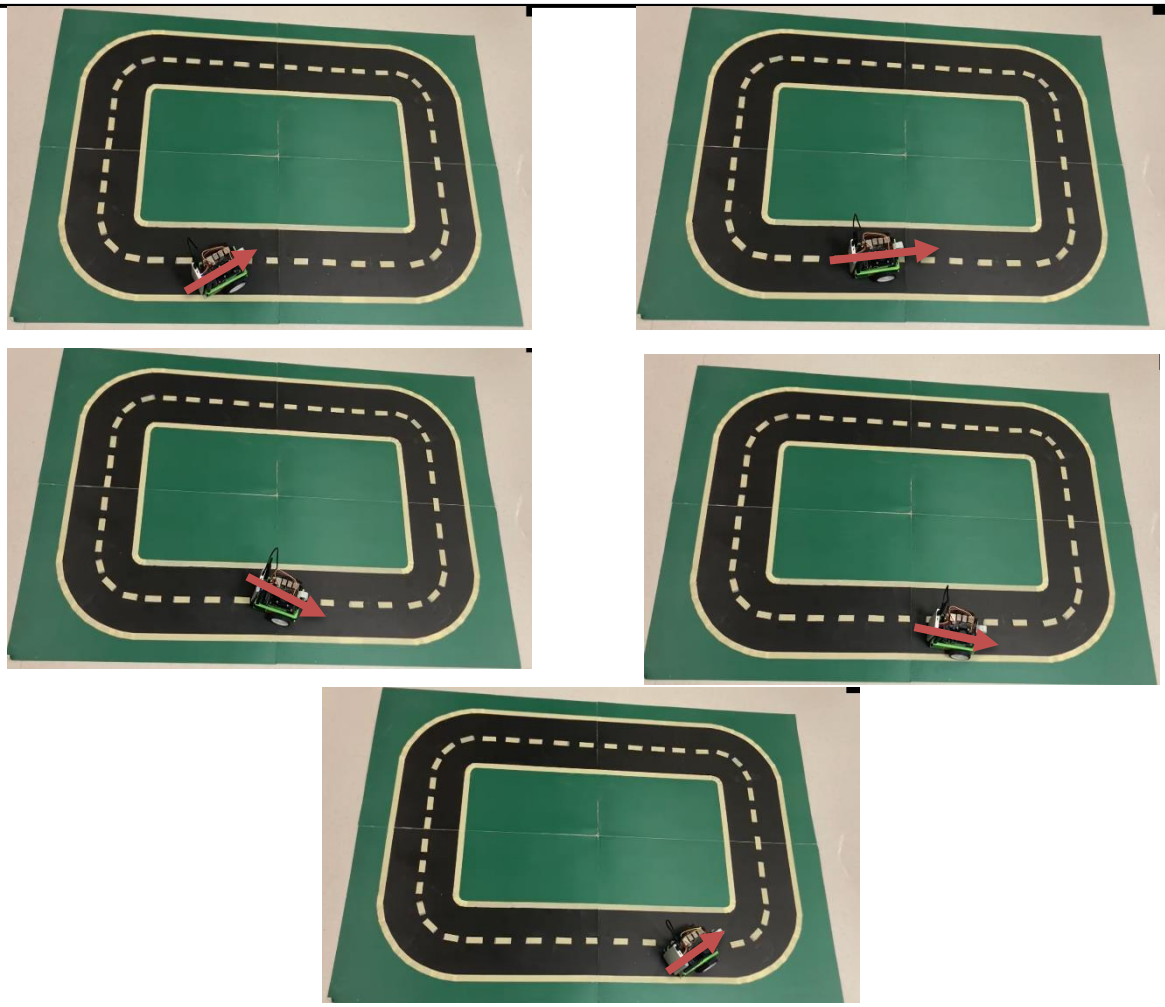


Figure 29. Direction and position of the model car perform on a straight road

A vehicle moving in zigzag motion on a straight line resulted in an increase in traveling time and brings passengers un-confident. Having a zigzag pattern appear on the real road may cause accidents, which should be prevented to be performed except for special cases. The performance of the model fulfilled the required stability for turnings but failed for straight lines.

The zigzag motion when moving on straight lines was potentially related to the poor refresh timing of the system and the principle for the system to control the model car. As the system calculates the steering angle repeatedly, and the refresh time of the system is too long, the ideal reacting timing to the changes in the surrounding was between the two refreshing moments. The time-consuming

process also restricts the system to perform the change instantly after the new situation is recognized, leading to a delay in the reacting changes of the motor values according to the surroundings. The other potential reason for the zigzag pattern performed on the straight line is the lack of straight-line samples collected for the database. As the model was located randomly in random directions, a relatively low amount of data in the situation of simulating moving straight was collected. The high amount of data collected simulates the situation requiring for modifying the direction, causing the model over-sensitive in collecting features required for turning. Therefore, the steering angle predicted by the system is usually relatively high. More image data with a low value of steering angle should be collected. Another possible solution is removing the steering value given to the motors when the steering angle calculated is lower than a certain amount.

4.1.3 Efficiency

The following table shows the time to cycle for each lap:

	Lap 1	Lap 2	Lap 3
Time (sec)	49	43	44

Table 6. Time spends in laps

The total length of the track was 367.92cm, and the average speed of the model car was:

$$speed = \frac{367.92}{(49 + 43 + 44)/3} = 8.116\ cm\ per\ sec$$

An average length of a vehicle is around 4.5m long, and the model car was 13.5 cm long. Expending the length of the track according to the ratio of an average vehicle and this model car's length, the length would become:

$$trach\ length_{simulate} = \frac{367.92}{13.5} * 450 = 122.64m$$

The speed of the vehicle would be:

$$\frac{122.64}{(49 + 43 + 44)/3} = 2.705\ m\ per\ sec = 9.74km\ per\ hr$$

The average road speed in Hong Kong is 50 km/hr, which is way faster than the

speed performed by this model car. The zigzag movement increased the total length the model car traveled, causing the time of each lap to increase. The refreshing time is also a concern of limiting the speed of the model car as the deviation could be serious due to the delay in performing changes in movement causing off-track.

4.2 Motor speed method

The model car was operated under the system with the trained neural network. The model was repositioned manually when the model car moved away from the plate and tends not to back to the plate automatically. The model car was tested for three cycles. The efficiency was ignored due to human support was needed to finish cycles.

4.2.1 Accuracy

The model car barely performed cycles, but most of the cycles were much smaller than preferred and in different shapes. The red line shows the proximate path the model car traveled for a relatively successful cycle. In this lap, the model car successfully performed straight lines and turning based on the image captured in different locations. Nonetheless, the changes in movement are always delayed and missed the ideal moment, which increases the distortion more time by time.

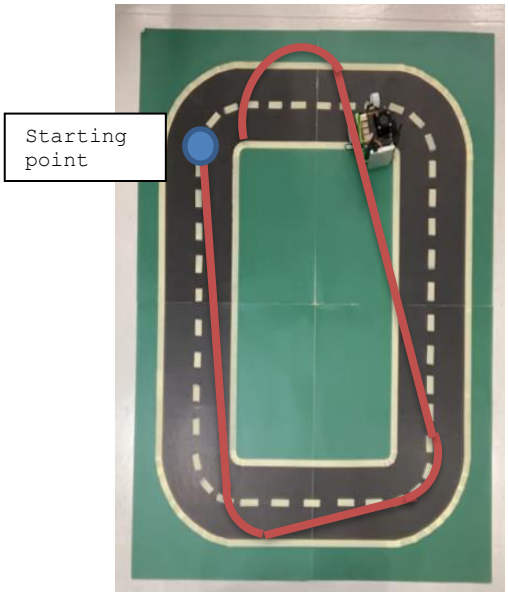


Fig 30. Performance of a successful lap

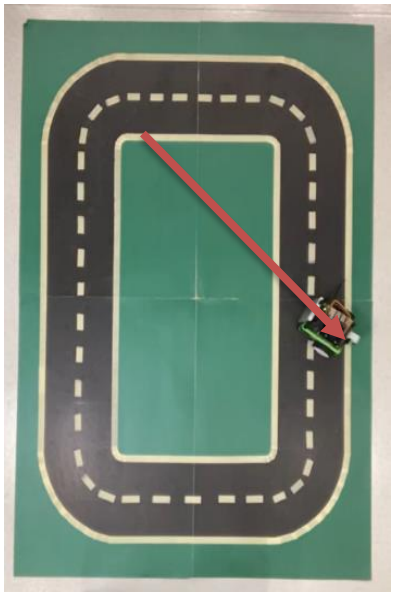


Fig 31. Demonstration of deviation

The model car completely distorted to the track right after the lap shown above.

This was potentially caused by the slow refreshing time or calculation time of the system. Due to the slow refreshing time, the ideal moment to perform the change of the movement is located a long distance between the previous and next refreshing time. When the system is refreshed again, the model might already be largely distorted due to the inappropriate movement performed before the refresh. In a similar theory, the long calculation time may cause the performance of the change much later than the situation was detected.

4.2.2 Stability

Overall, the movement was smooth and stable, and it is uncommon for zigzag problems or sudden large angle turning to appear. The speed changes when switching between forward and turning movements were instance, causing small shakes of the model car body. In general, stability is acceptable for having passengers.

4.3 Comparison

Overall, the steering angle method was the more preferred method in performance. However, the stability of the motor speed method was better as the number of movement sets was fewer and the method in the demonstration allowed the supervising data closer to the real human driving situation.

In difficulties, the steering angle method is a relatively simpler way for data collection as it only required demonstrator's choice of the guiding point, but the wheel speed method requires a decent driving skill of the demonstrator for positive sets of demonstrating data (static vs dynamic). Besides, the steering angle method can collect the data at the ideal position requires to operate changes in movement only by setting the accurate position and angle of the model car, but higher effort required for the wheel speed method to collect data at the right timing as high reaction and lots of experience or practice.

For straight roads, the movement under wheel speed method performs smoother and accurate, but a higher accuracy of steering angle performed by the steering angle method. The reason causing the differences due to the differences of operating according to the

range of potential moving combinations. For the steering angle method, the movement directly takes the predicted steering angle for the calculation, total of 180 sets of movement would be potentially performed. As a result, the performance in turnings were high in accuracy. However, as the method did not separate the situation of straight forward and turnings, the system keeps changing the steering angle, causing the model car failed to keep the decent input to both motors. On the other hand, the wheel speed method performs under classes, the model car abled to perform decent input to the motors when the class of straight road was detected. Unlike straight road, accuracy is more important than stability in turning, and the wheel speed pairs are different based on the direction and position of a vehicle on the road. Only four classes are not enough for performing high accuracy of turnings. Nonetheless, the increase in number of class result in lower test accuracy for the trained neural network model.

Chapter 5

Limitations and Improvements

5.1 Steering angle method

5.1.1 Level of simulation to human driving

The data collecting method is to capture the image of the vehicle in a random situation and supervise the system with the ideal steering angle based on the location and angle on the road. The level of "driving" is low that the system theoretically does not learn from a human's real driving action. The image pointing progress can be replaced by a steering wheel. Simulating the most common road vehicle driving method, the steering angle can be collected by the angle of the steering wheel that is being rotated in different situations. The system is able to learn the data from a more common way of human driving and more reasonable situations faced by drivers in real driving situations.

5.1.2 Overall steering rate

In the data collection process, the camera usually shows a certain distance length of the guiding line. Randomly selecting the point to plot on the guiding line may cause unstable performance of turning. The graph on the left-hand side below shows that the expected steering angle can be different while plotting on different points of the guiding line shown on the camera image. The red lines pair shows a smaller steering angle could be collected while a much bigger angle collected by the green lines pair, which has a smaller y-axis value. By the comparison of both images below, the surrounding situation shown on the camera on the right-hand side clearly requires a higher steering angle than the surrounding situation shown on the left image. If both steering angles are collected by the green lines pair, the expected steering angle would be unreasonable by comparison.

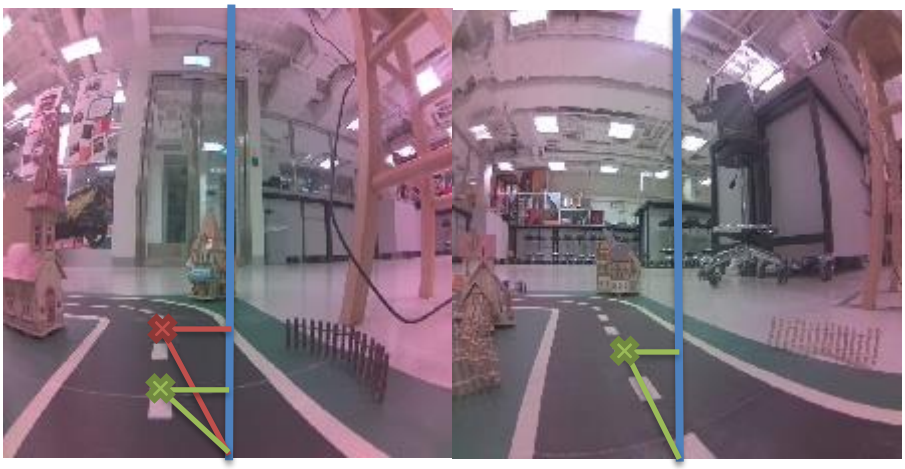


Figure 32. Effect to the steering angle with various guiding points

Fixing the y coordinate value would help to perform with more stability as when dealing with similar situations, the expected steering angle should be similar in theory if the y coordinate is taken in the same value. The other solution is to collect the point on the guiding line at a constant distance from the model car.

5.1.3 Reaction time (refresh rate) of the system

The whole process of the application includes image collection, converting images to digital data, passing through the designed model, and providing the command of the movement to the motors. The process is time-consuming that the system is unable to operate the next changes instantly based on the changes of surrounding right after the last “decision” from the autonomous driving system. This may cause the delay of the vehicle to operate the changing actions, like in the experiment, late turning action when the model goes from a straight road to a turning road or overturning when it goes from a turning road to a straight road, because of missing the best timing to provide the motors with accurate movement changing command. To reduce the system reaction time, reducing the complexity of the network helps to reduce the time required for the computer to operate the calculation. Applying a computer with a higher specification also helps to reduce the time required in the calculation. However, the price would be higher, or the size of the computer will be bigger and heavier, which give the vehicle a higher load consequently.

Another method is providing the system with predictivity for the output class type. When the output classes include predictivity, the delay effect can be reduced as the model simulated the “decision” for the instant and upcoming future. For example, when the system’s reaction time is 1 second, the model calculates the “decision” at the moment when it is moving straight forward and also predicts a turning is required in 0.5 seconds after. The command to the motor would be to move forward at present and turn 0.5 seconds later. The maximum delay to the ideal action timing would be improved from 1 second to 0.5 seconds. But this may require the neural network structure to be structured with higher complexity.

5.1.4 Fisheye lens

The lens on the camera is a fisheye lens, which is an ultra-wide-angle lens that produces visual distortion in creating a hemispherical image to show the 180° view in a single 2-d image. In order to collect an image with a wide-angle for more data with richer surrounding information captured in a single image, a fisheye lens is preferred. The distortion is more serious when further away from the center point of the image. The method of taking the point for the steering angle is based on the observation of the hemispherical image captured from the camera with the fisheye lens. If the ideal point for plotting on the image for the calculation of the expected steering angle is away from the center point of the image, the distortion may cause bias in the steering angle.

Fig 33. Fisheye lens view(left) normal human eye view

The images above compare the effect of the distortion. An obvious problem caused by distortion is that the direction of the guiding line on the right is different from the fisheye lens camera image and human eyes. Collecting data through demonstrations from real human driving eliminates the problem as the observation of the surrounding situation is no longer through the image collected from the camera but the human eyes and the steering angle collected is the angle that is actually performed.

5.2 Motor speed method

5.2.1 Human limit

5.2.1.1 Demonstrator's driving skill

This training method requires the demonstrator to control both wheels with two separate analog sticks at the same time. Different from the mainstream vehicle driving operation mode, both driving angle and speed are operated by finger action. This requires the demonstrator's excellent multi-tasking, coordination of fingers, and precise finger operation skills. Adequate training is required for continuously performing accurate driving actions like turning, stable acceleration, or deceleration. In this experiment, the samples collected are not fully accurate due to the demonstrator's ability in machine operation. Some turning actions are operated poorly. Even though over-turning and under-turning action data were being eliminated, the overall accuracy of the speed data connected to the image reaches an acceptable level as the overall operation by the demonstrator was not at an excellent level. One way to improve is to reduce the difficulty in operation and change the controlling tool from analog stick to stepping operation, like constant stepping up or down by button pressing. Having a demonstrator with more experience in machine operation may also help to collect stable and high-accuracy data.

5.2.1.2 Data collection timing

Data should be collected ideally at the moment of the change in operation,

which is when the surrounding condition forces a change in operation status to maintain a reasonable traveling afterward. Due to human limitations, the data could be collected slightly early or later than the ideal timing. Early collecting issue causes the failure in providing correct supervising value as the value would be captured right before the change of the demonstrator's change in operation. Late collecting issues may cause the operation system to have low sensitivity to the changes in the surrounding. Therefore, setting auto image and supervising data capturing function are great approaches for automatically capturing data when a huge change to the model car is being sensed during operation. Operating the data collecting process with more cycles also helps improve the issue due to the increase of chance in collecting data at the ideal timing and results in a higher variety of data

5.2.3 Computer reaction time

Same as the steering angle method, the performance of the model car is limited by the reaction time because the procedure for the operation system to process the data and pass it through the neural network model is time-consuming. Other than simplifying the model and improving from providing single operation command for instance change in movement, to providing a set of operation commands for the movement of a period, a new class for slowing down can be included, which provides the operation system a deceleration choice to slow down before performing turning action and give the system more time to reduce the issue caused from the system's reaction time.

5.2.4 Design of classes

Regarding the values to both motors for different turning actions, the middle value of the range of the motor's output is adapted, which is covered for different turning example data. These sets of operation choices may not be able to perform more precise actions. A better combination of action command sets should be designed for performing a wider range of actions with higher accuracy, despite

having the same or even fewer classes for the supervising data.

5.2.5 Limited Speed

The action command directly controls the input to the power for a designed wheel rotation speed for forward and turning actions. As the input value for turning is designed and the provided output values to the motors are fixed, the model car always moves at the same speed in case of the same “decision” from the five-set classes from the operation system. Since the speed control is much easier for straightforward actions, a function like creating a slider in python can be implemented, which controls the speed by controlling the power input to the motors according to the ratio set in the slider during operating the “straight” class command.

5.3 Hardware

5.3.1 Dual motor system

The two motors on each side are used to drive the model car in this experiment. Unfortunately, the motors are low-cost, low-quality goods with consideration of the project cost, the performance and depreciation of both motors could be different. The differences in the performance of the motors increased the difficulty of performing precise actions. The motor that drives the right wheel in the model car performs better than the left motor. Therefore, even though having the same command and power inputted to both motors, the right wheel rotates faster than the left wheel, causing the movement of the model car tends to move to the left while performing the command that is expected to move straight. This hardware limitation causes neither the straight moving command from the Jetbot library nor the maximum speed commands to both motors able to let the model car move straight forward.

The difficulty of data collection by the motor speed method increased due to the necessity of finding the ratio of the speed input command to the motors separately in order to perform a forward movement. The application of the steering angle method is also largely affected as the model car always performs with a slightly

left-biased steering angle. Even if the difference between the expected angle and the real performance steering angle is small, the difference is significant after traveling for a certain period. Turn actions may be required for modifying the angle of the vehicle on the road. Turning or Zig Zag movement may happen while moving on a straight road. Re-designing the class classification and the operation command is also a solution for the unbalance-motors-pair issue.

5.4 Software

An ideal turning of a vehicle with two wheels can be performed by the setting the rotation speed of the two wheels by calculating the radius of both wheels traveled with unbalanced rotation speed by the instant center of rotation.

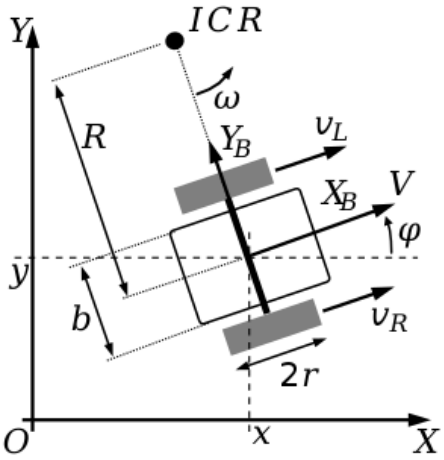


Fig 34. Presentation of Kinematics of Differential Drive Robots

$$\omega * \left(R + \frac{b}{2}\right) = v_R$$

$$\omega * \left(R - \frac{b}{2}\right) = v_L$$

$$V = \omega * R = \frac{v_R + v_L}{2}$$

Equation Kinematics of Differential Drive Robots

Jetbot library's movement commands are based on the power input to the motor. Although it is still possible to operate the ideal turning action by controlling wheels' speed respectively in this set of model car, additional gadgets like speed sensor and tests

for the relationship between the speed of the model car and power input to motor are required.

5.5 Method combination

The steering angle method's supervising parameter and the operation method are more similar to the most common actual driving method: providing steering angle to wheels by rotating the steering wheel and controlling the speed by an accelerator. Another advantage of this method is able to control the speed instantly at any time. However, the wheel speed method is collecting the actual performance in controlling the vehicle, higher level in the demonstration, and theoretically higher accuracy of the supervising data.

A new method from the combination of these two methods can take advantage of them both. A possible new method would be collecting the steering value by the demonstrator driving the model car with a steering wheel instead of the analog stick. The human decision in providing the supervision is no longer based on the distorted image captured from the camera with fisheye lens, but by the demonstrator's eyes.

Chapter 6

Conclusion

The project aims to develop a possible method to apply deep learning method with only front camera image as the input of the environment information. A model car with Jetson Nano as the system controlling device and two wheels installed at both sides controlled with individual motors were built to simulate normal vehicles on roads, and a model road with guiding lines were made to simulate the normal road in real situation.

Two methods, the steering angle method and the wheel speed method, were developed, trained and tested for the performance. Data collection, training of model, and the application were all operated in the Jetson Nano. The performance was ranked by the accuracy, stability and the efficiency.

The result shows that steering angle method was a more suitable method by the comparison of both methods and showed the possibility of deep learning method applied in autonomous vehicle with only camera image input to the system. A new possible method was designed with the combination of both methods.

Chapter 7

References

[1] M. Sentinel, "Phantom Auto'will tour city," Milwaukee Sentin., pp. 4, 1926.

[2] M. Weber, *Where to? A history of autonomous vehicles*, 2020. [Online]. Available: https://computerhistory.org/blog/where-to-a-history-of-autonomous-vehicles/. [Accessed: 22-Sep-2021].

[3] Honda Global, *Honda to Begin Sales of Legend with New Honda SENSING Elite*, 2021. [Online]. Available: https://global.honda/newsroom/news/2021/4210304eng-legend.html. [Accessed: 22-Sep-2021].

[4] C. Beresford, *Honda legend sedan with Level 3 Autonomy available for lease in Japan*, 2021. [Online]. Available: https://www.caranddriver.com/news/a35729591/honda-legend-level-3-autonomy-leasesjapan. [Accessed: 24-Sep-2021].

[5] NHTSA's National Center for Statistics and Analysis, *2016 Fatal Motor Vehicle Crashes: Overview*, 2017. [Online]. Available: https://crashstats.nhtsa.dot.gov/Api/Public/ViewPublication/812456. [Accessed: 22-Sep-2021].

[6] A. Kopestinsky, *Self driving car statistics For 2021: Policy advice*, 2021. [Online]. Available: https://policyadvice.net/insurance/insights/self-drivingcar-statistics/. [Accessed: 24-Sep-2021].

[7] Ohio University, *The future of Driving*, 2021. [Online]. Available: https://onlinemasters.ohio.edu/blog/the-future-of-driving/. [Accessed: 24-Sep-2021].

[8] J. Anderson, N. Kalra, K. Stanley, P. Sorensen, C. Samaras and O. Oluwatola, "Autonomous Vehicle Technology : A Guide for Policymakers". RAND Corporation, 2016.

[9] KPMG LLP, *Connected and Autonomous Vehicles - The UK Economic Opportunity*, 2015. [Online]. Available: https://assets.kpmg/content/dam/kpmg/images/2015/05/connected-and-autonomousvehicles.pdf. [Accessed: 24-Sep-2021].

[10] Y. Bengi, Y. Cun and G. Hinton, "Deep Learning". Nature. 52, 436–444. 2015.

[11] U. Muller, J. Ben, E. Cosatto, B. Flepp, and Y. Cun, "Off-road obstacle avoidance through end-to-end learning,". Advances in neural information processing systems. pp. 739–746. 2006.

[12] J. Hopfield, "Neural networks and physical systems with emergent collective computational abilities". Proc. NatL Acad. Sci. USA. 79, pp. 2554-2558. 1982.

[13] 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.

[14] L. Serran, *A friendly introduction to Convolutional Neural Networks and Image Recognition*, 2017. [Online]. Available: https://www.youtube.com/watch?v=2-Ol7ZB0MmU. [Accessed: 20-Jun-2022].

[15] A. Krizhevsky, I. Sutskever & G. Hinton, "ImageNet classification with deep convolutional neural networks". Communications of the ACM. 60 (6): 84–90, 2017.

[16] S.Aziz, M. Bilal, M. Khan & F. Amjad, "Deep Learning-based Automatic Morphological Classification of Leukocytes using Blood Smears.". 2020. [Online]. Available: https://www.researchgate.net/publication/343955468_Deep_Learning-based_Automatic_Morphological_Classification_of_Leukocytes_using_Blood_Smears.

[17] Bojarski, Mariusz, Davide Del Testa, Daniel Dworakowsk, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D. Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, Xin Zhang, Jake Zhao and Karol Zieba. 2016.

[18] Thomas, Andrew, Tesla, MobileEye, and deep learning, 2016. https://medium.com/@andrewt3000/tesla-mobileeye-and-deeplearning-b7ceb882848