THE UNIVERSITY OF WESTERN AUSTRALIA

End-to-End learning: Autonomous driving system based on PilotNet

Zhihui Lai

22598312

Supervisor: Prof. Dr. Thomas Bräunl

GENG5011/GENG5012 Engineering Honours Research Project

Submitted: 22th Oct. 2021

Word Count: 12035

Faculty of Engineering and Mathematical Sciences

School of Electrical, Electronic and Computer Engineering

Declaration

I, **Zhihui Lai**, certify that:

This thesis is my own work, and I have appropriately referenced any sources used in preparing it.

This thesis does not contain any material which has been submitted under any other degree in my name at any other tertiary institution.

In the future, no part of this thesis will be used in a submission in my name, for any other degree or diploma in any university or other tertiary institution without the prior approval of The University of Western Australia and where applicable, any partner institution responsible for the joint-award of this degree.

This thesis does not infringe on any copyright, trademark, patent or any other rights for any material belonging to another person.

Date: 22th Oct. 2021

Abstract

This thesis aims to continue the research of the ModelCar-2 project and develop a more functional and reliable autonomous driving system utilizing deep learning. In the previous ModelCar-2 project, a fully end-to-end method that applies PilotNet neural network was developed, involving a current image as input (the dataset was collected by Lidar drive mode and preprocessed) and a steering command and speed as outputs. This method achieved the Lidar-based methods' performance while presenting advantages like higher and consistent frame rate and low cost. However, a significant drawback was no past information, imposing an error-sensitive performance, especially in high speed and complex environments. Spurred by this deficiency, this thesis introduces two new models: CNN + LSTM and 3DCNN, aiming for autonomous driving in high-speed and complex environments. This is because in deep learning methods, despite the current image being damaged by external factors like vibrations and obstructions on the camera, they can still exploit past images to produce the right steering decisions. Furthermore, deep learning can learn more complicated driving skills like recovering from failures and turning around. While building these models, multiple influencing factors are discussed: memory capacity, dataset balance, TensorFlow version compatibility with Raspberry Pi, frame rate, and Raspberry Pi's computing power. The three experiments investigated in this thesis challenge the CNN + LSTM and 3DCNN model against the original PilotNet and Lidar models. The first one involves a maze map in the EyeSim simulator, which can run several models simultaneously and compare their speed, autonomy, and reliability. The second experiment considers the 4^{th} floor of the EECE building, where the models circle within a rectangular area separately, and we compare their speed, reliability and autonomy. Finally, the third experiment is in the corridor of the CME building, where the models turn around in a narrow corridor separately, and we compare their speed, reliability and autonomy.

Table of Contents

List of figures

List of tables

Acronyms

AVs Autonomous Vehicles

NHTSA National Highway Traffic Safety Administration

US United States

Lidar Light Detection and Ranging

CNN Convolutional Neural Network

UWA the University of Western Australia

LSTM Long Short-Term Memory

RoBIOS Robotics Basic Input/Output System

SAE Society of Automotive Engineers

API Application Programming Interface

SLAM Simultaneous Localisation and Mapping

EECE Electrical, Electronic and Computer Engineering

CME Civil and Mechanical Engineering

DNN Deep Neural Network

CPS Cyber-Physical Systems Link

IMU Inertial Measurement Unit

PC Personal Computer

GPS Global Positioning System

PWM Pulse Width Modulation

GPIO General Purpose Input/Output

GPU Graphics Processing Unit

RMSE Root Mean Square Error

DNF Did Not Finish

TBF Mean Time Between Boundary Failures

DBF Mean Trajectory Distance Between Boundary Failures

NBF Mean Number of Boundary Failures

LT Lap Turns

MDBF Mean Distance Between Failures

ReLU Rectified Linear Unit

PID Proportional-Integral–Derivative

RC Radio Control

GUI Graphical User Interface

3DCNN Three-dimensional Convolutional Neural Network

ELU Exponential Linear Unit

ResNet Residual Neural Network

TF TensorFlow

FPS Frames Per Second

1. Introduction

1.1 Background

The interest in autonomous vehicles (AVs) has increased exponentially in recent years. Even though AVs could fully replace human drivers at the current stage, there is an irreversible trend towards full driving automation. In 2018, the US Department of Transportation National Highway Traffic Safety Administration (NHTSA) reported that 94% of severe traffic accidents are caused by human error in the US [1]. AVs or computer-assisted driving systems can significantly reduce fatalities in accidents and thus are promising to solve this issue. However, there are still many challenges to overcome, as the demand for AVs is not only to be better than human drivers but also to be economical. Hence, the technology that can solve these two problems simultaneously is still an open research case.

Currently, most AVs companies are using Lidar to collect data. It is accurate at sensing distances of the surrounding environment but extremely expensive [2], while an alternative solution is using a camera and exploiting machine learning algorithms. Since computers nowadays can learn from a large dataset, using the camera to collect self-driving data is cheap and severely reduces the time to program complex driving algorithms. If a camera with machine learning can reach the performance of Lidar, then there is a high probability that we can afford to market AVs.

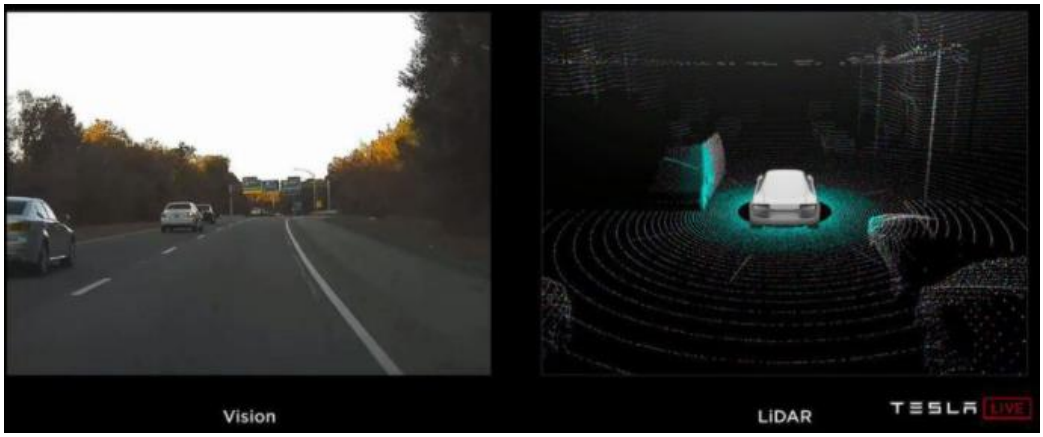

Figure 1 Camera view (left), Lidar view (right), copied from [3]

Table 1 Lidar vs Camera, pros and cons

	Lidar with a driving algorithm	**Camera with a neural network**
Pro	• Stable and mature •Not sensitive to weather •Accurate 3D measurement	•Can keep learning from new data •Cheap •Can drive in any area •Nice appearance •Can read traffic signs
Con	•can only drive in a precisely defined area •exceptionally costive •Need extra space for sensor	•Need large amounts of data for training •Sensitive to weather •Rely on powerful machine learning •Need a lot of computing power (GPUs)

In 2016 NVIDIA introduced PilotNet, an end-to-end Convolutional Neural Network that extracts raw pixels from a single front camera image as input and produces steering commands as output [4]. This network has been proved incredibly powerful, but there is still much space for improvement, e.g., adding memory to increase driving continuity. This upgrade to PilotNet can be implemented in several ways: at the input side by using optical flow to replace raw image as input or at the neural network side by employing Long Short-Term Memory (LSTM) to combine it with CNN. Another example is to add future information to increase driving reliability. To implement this, trajectories like waypoints and Bezier curves can replace the direct control command at the output side. All these methods have the potential to achieve better steering results with more profound information.

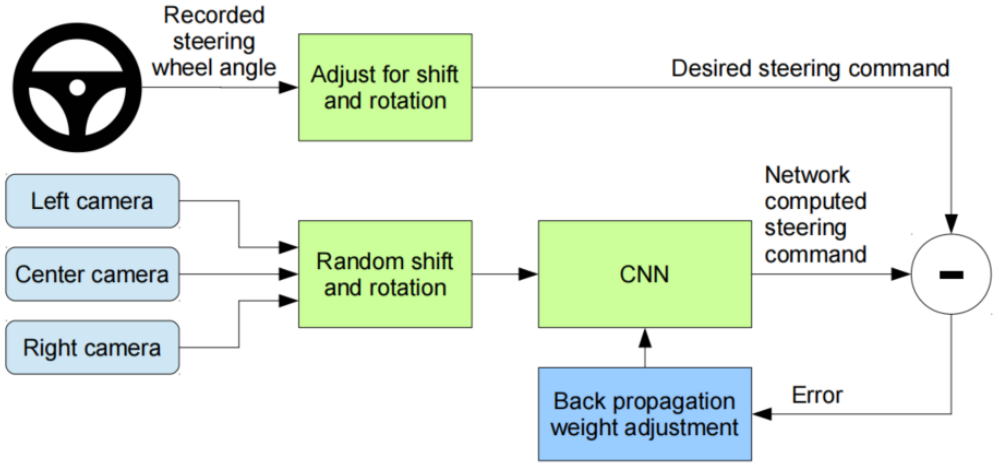

Figure 2 Training the PilotNet, copied from [4]

In 2017, an RC car equipped with a Lidar sensor and a Raspberry Pi named ModelCar was developed as a research project by a previous student at the University of Western Australia (UWA). This ModelCar project provided a Lidar-based autonomous driving baseline [5]. Three years later, another student at UWA continued this project utilizing NVIDIA PilotNet and performed end-to-end deep learning research by modifying ModelCar with a wide-range camera

and a plastic plate. This project named ModelCar-2 provided a camera-based autonomous driving baseline [6]. The ModelCar-2 now has Lidar drive mode, manual mode, and PilotNet-drive mode. These features satisfy the prerequisites for further autonomous driving research.

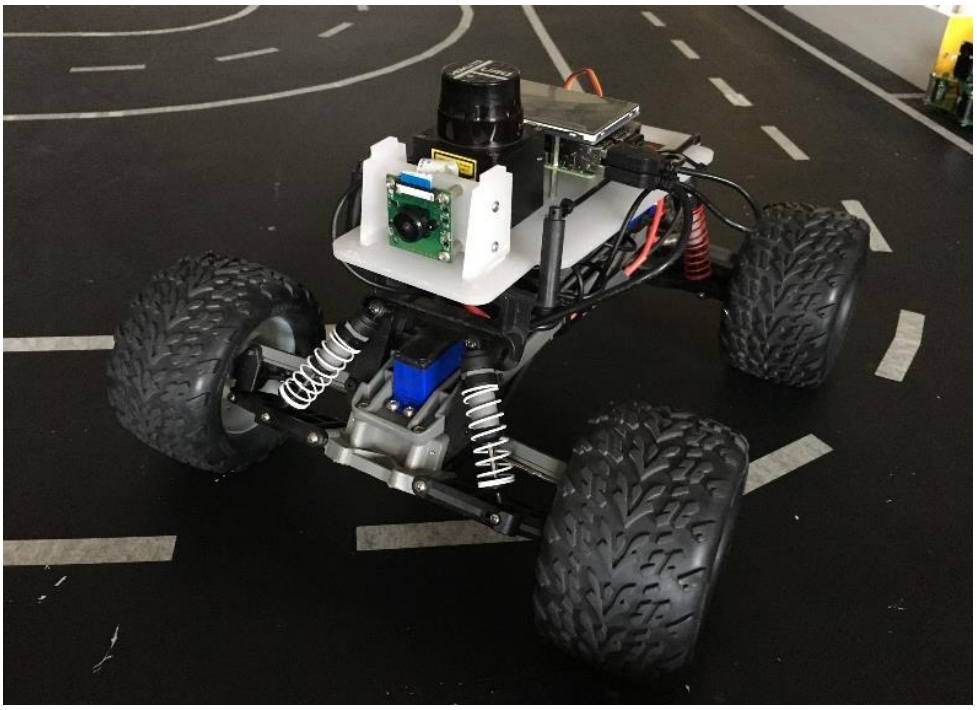

Figure 3 ModelCar-2 Autonomous Driving Robot, copied from [6]

1.2 Problem Statement

This thesis aims to continue the research on the ModelCar-2 project and find a more functional and reliable autonomous driving system employing deep learning. In the previous ModelCar-2 project, a fully end-to-end method applies the PilotNet neural network was developed, exploiting a current image as input and providing a steering command and speed as outputs. This method reached the performance of Lidar-based methods, affording some advantages like higher and consistent frame rate and low cost. However, it has memory constraints and does not exploit future information for path planning. These flaws lead to prone-overfitting and overall error-sensitive performance, especially in high speed and complex environments [7], [8].

There are two approaches to introduce memory in the system. One is to replace raw pixels of the input image with optical flow imagery. This strategy solves the overfitting problem, as, during training utilizing the original image, the neural network struggles to memorize every detail. However, filtering the critical information by optical flow means the neural network has less data to process. An alternative approach to increase the network's memory is adding LSTM or 3D CNN in the PilotNet neural network. In this case, the neural network efficiently operates within a high-speed and complex environment, as despite the current image being damaged by external factors like vibrations and obstructions on the camera, the networks still have the past images to produce the proper steering and speed decisions.

Figure 4 Camera image (left), optical flow grayscale image (right)[1]

To introduce future information in the system, we use a trajectory to replace the direct control command as output and add an additional controller that transfers the trajectory to control the commands. This thesis discusses two trajectory types: waypoints and the Bezier curve. Waypoints are the points represented in the ground 2D coordinates, while a Bezier curve is defined by a set of control points that fit the waypoints. With the involvement of future information, the indirect control commands are much more stable than the direct control commands.

Several advanced neural network models, i.e., the CNN + LSTM + optical flow (pixels to control), CNN+LSTM (pixels to waypoints), and CNN+LSTM (pixels to Bezier Curves trajectory), will be discussed. Regarding the CNN+LSTM and 3DCNN, these networks will be constructed and compared against the PilotNet under the same experimental setup and various performance metrics, including autonomy, processing speed, mean lap time, and reliability.

1.3 Document Structure

The thesis consists of the following structure: Chapter 2 briefly reviews the development of AVs, the concepts and models of deep learning; Chapter 3 describes the design preparation for this project, including hardware, software, and neural network models' architecture; Chapter 4 presents the methods to implement this project and the methods to evaluate them; Chapter 5 introduces the results of three experiments and compares the influencing factors; Chapter 6 derives the conclusion and future work;

[1] Original video from https://www.youtube.com/watch?v=7BjNbkONCFw

2. Literature Review

2.1 Autonomous Vehicles

Society of Automotive Engineers (SAE) has defined a standard for AVs, ranging from Human drive (no automation) to Full automation, six different levels in total. This standard clears the goal of autonomous driving research, and apparently, no AVs have reached the final destination-Level 5 Full automation yet. The highest level of automation widely recognized to date is Waymo, formerly the Google self-driving car project, which reached Level 4 High automation. However, Waymo can only drive in a precisely defined area like Phoenix with the Lidar-based technique and is exceptionally costive. On the other side, Tesla cars, electric vehicles that utilize the camera-based approach, are currently at Level 2 Partial automation but learning very fast with an enormous amount of data coming worldwide [9]. While Waymo can only get data in Phoenix and is limited by its fleet size and cost, Tesla needs to improve its computer vision technology with machine learning. Therefore, it is reasonable to think that Tesla will take over the market in the foreseeable future. Elon Musk is even adamant about saying Tesla will reach Level 5 Full automation at the end of 2021 [10] and can drive on any raw road without modern infrastructure. While many experts do not believe him, this message still shows the incredible highness machine learning can achieve.

Figure 5 Tesla Model S (left), Waymo (right), copied from [9]

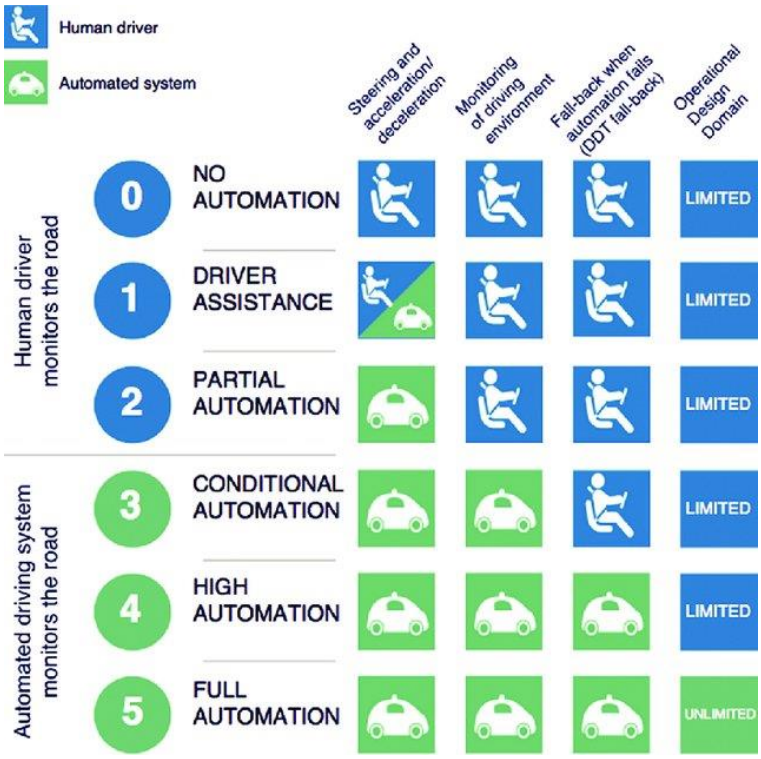

Figure 6 SAE J3016 levels of driving automation, copied from [11]

2.2 Deep Learning concepts and models

2.2.1 Convolutional Neural Networks

A Convolutional Neural Network (CNN) is a subset of deep neural networks (DNNs) widely used in image analysis. Inspired by the brain's visual cortex, CNNs have two core elements: partially connected layers and weight sharing. Compared with conventional DNNs that use fully connected layers, these elements significantly reduce the number of parameters and connections of CNNs.

CNNs basically consist of several convolutional layers, pooling layers, and fully connected layers. Convolutional layers apply multiple filters to the inputs simultaneously. These filters (or convolution kernels) are different sets of weights used for filtering the receptive field (part of the previous layer), so they can map multiple features no matter where they are. Pooling layers reduce the computational load by subsample (shrink) the size of input images. Two pooling approaches are commonly used: max-pooling to select only the max value in a receptive field as output and average-pooling to average the whole receptive field as output. Fully connected layers are the same as in regular DNNs. They connect all neurons in the current layer to every neuron in the previous layer [12].

2.2.2 2016 NVIDIA PilotNet

In 2016, NVIDIA launched its end-to-end CNN architecture called PilotNet. End-to-end means PilotNet can output steering command directly from image input and successfully drive an actual

vehicle on public roads [4]. This network is relatively simple, so it has a bunch of space for improvement.

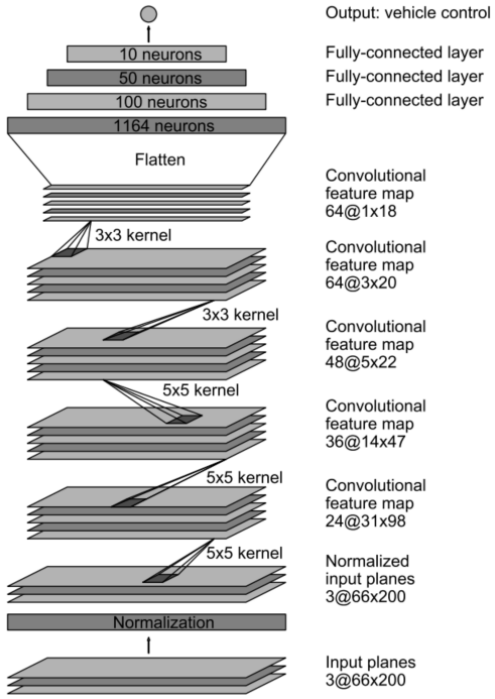

Figure 7 PilotNet Architecture, copied from [4]

2.2.3 Recurrent Neural Networks

A Recurrent Neural Network (RNN) is another subset of DNNs, different from CNN, a feedforward neural network. RNN has connections that feed backward. This feature allows it to deal with sequential data. However, the standard RNN can only remember a few sequences back, and the exponentially decaying gradient during backpropagation causes the "Vanishing Gradient Problem," so it has not been widely used [12].

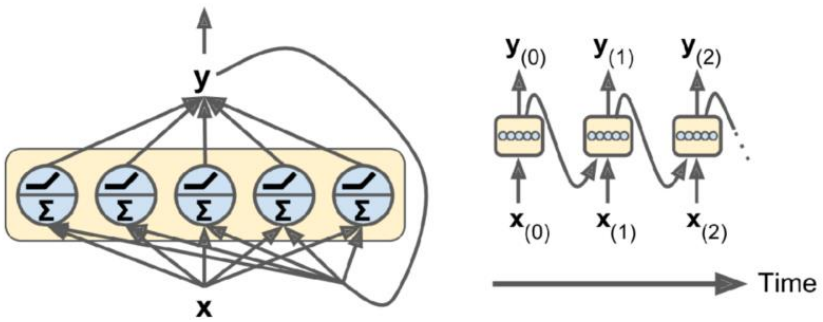

Figure 8 A layer of recurrent neurons (left) unrolled through time (right), copied from [12]

2.2.4 Long Short-Term Memory

Long Short-Term Memory (LSTM) is developed to solve common RNNs shortages. LSTM Complicates its calculations with a cell, an input gate, an output gate, and a forget gate. These features allow it to achieve long-term memory and eliminate the "Vanishing Gradient Problem" [13]. LSTM networks are usually used to classify and predict time information. If there is a lag of unknown duration between the basic features in the time series, the LSTM network can also capture it well. In addition, the relative insensitivity to gap length is a significant advantage of LSTM over RNN and other sequence learning methods in many fields.

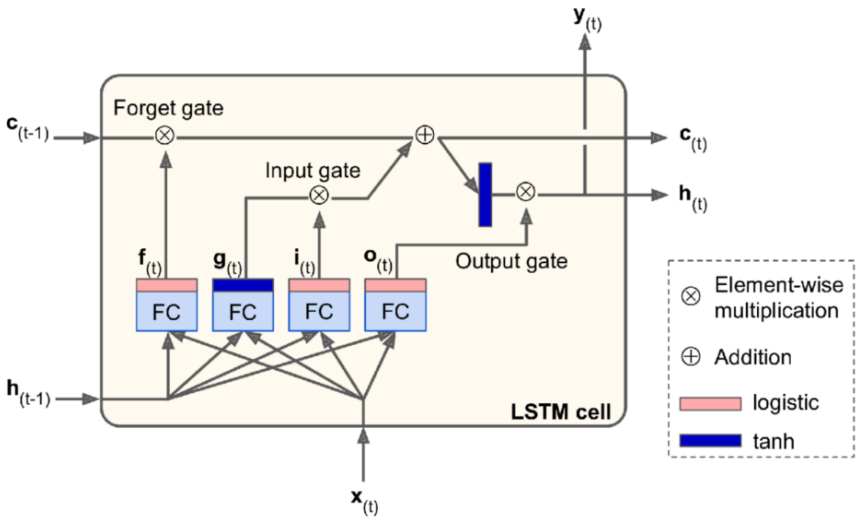

Figure 9 LSTM cell structure, FC is Fully Connected Layer, copied from [12]

2.2.5 3D Convolutional Neural Networks

A 3D Convolutional Neural Network is a neural network that uses 3D filters (kernels). This network is beneficial in image processing because 3D filters can extract spatial and temporal features from a sequence of image inputs hence obtaining action information [14].

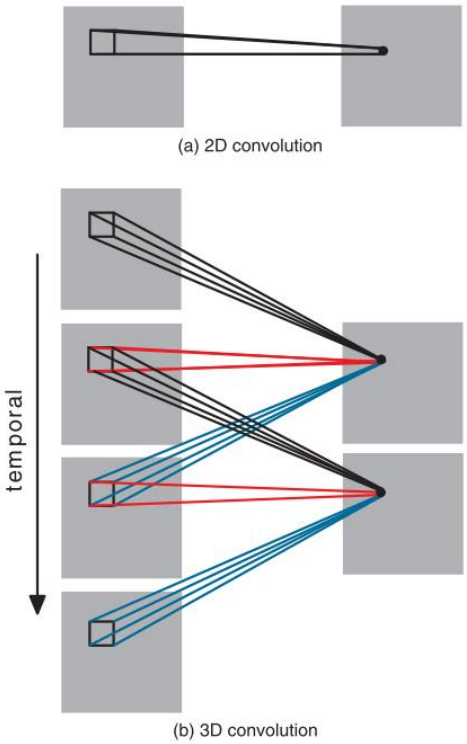

Figure 10 Comparison of 2D and 3D convolution, copied from [14]

2.2.6 AdmiralNet Bezier Curve Predictor

In 2020, the Cyber-Physical Systems Link (CPS) Lab at University of Virginia (UVA) developed a novel end-to-end deep learning model named AdmiralNet, combining the original PilotNet, LSTM, and 3D convolution. This network works well on both the photo-realistic F1 racing simulator and the 1/10 scale racecar testbed. Unlike the original PilotNet predicting steering command, they use AdmiralNet to predict Bezier Curve from pixels directly. CPS Lab compared four different deep learning approaches: PilotNet (pixels to control), CNN-LSTM (pixels to control), CNN-LSTM (pixels to waypoints), and CNN-LSTM (pixels to Bezier Curves trajectory). The results are impressive. AdmiralNet Bezier Curve Predictor outperforms all other approaches[7]. This experiment shows that the autonomous driving system's performance can be significantly improved by adding memory and predicting trajectory instead of the control

command.

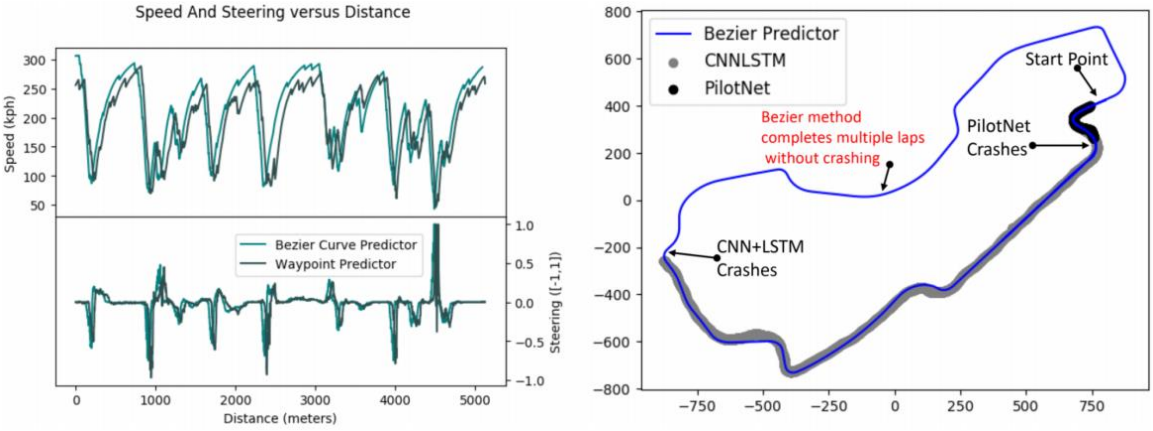

Figure 11 A F1 style control plot for a test run (left), A plot of the path followed by several deep learning approaches, copied from [7]

Table 2 F1 racing game closed-loop testing results, copied from [6]

Model Configuration	Lap Time (s)	TBF (s)	DBF (m)	Number of boundary failures	Successful laps
PilotNet	DNF	4.0	181.4	1,800	0
CNN-LSTM	DNF	6.4	304.5	3,600	0
Waypoint Predictor	106.7	16.7	855.8	6	4
Bezier Curve Trajectory Predictor	**101.7**	**33.6**	**1786.36**	**2**	**5**

AdmiralNet Waypoint Predictor outputs predicted future waypoints with a sequence of images as input. The loss function is composed of position loss and velocity loss [7].

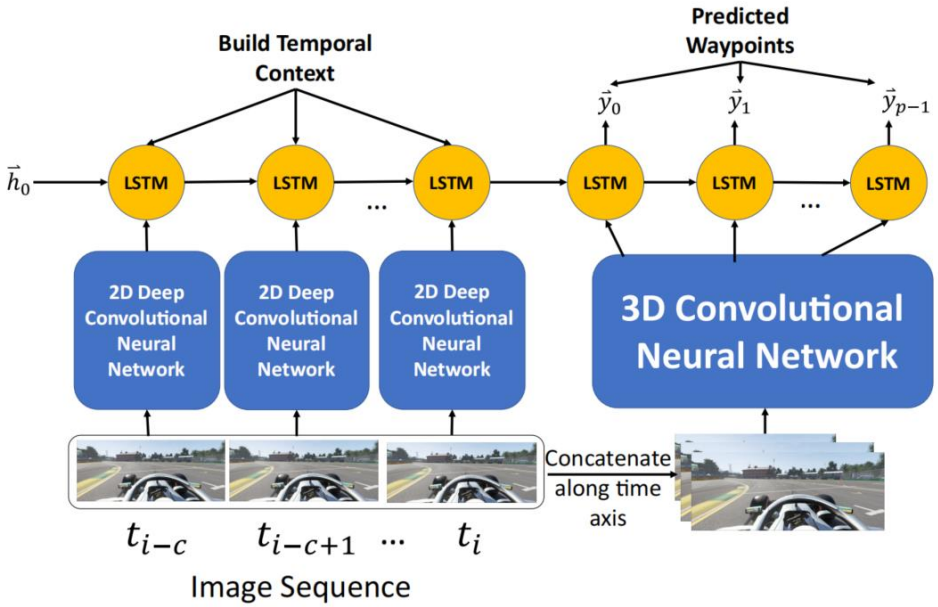

Figure 12 AdmiralNet Waypoint Predictor architecture, copied from [7]

AdmiralNet Bezier Curve Predictor outputs predicted Bezier Curve control points of future waypoints with a sequence of images. This model's loss function is particular: besides position loss and velocity loss, it appends control-point loss to find the perfect control points for a sequence of waypoints [7].

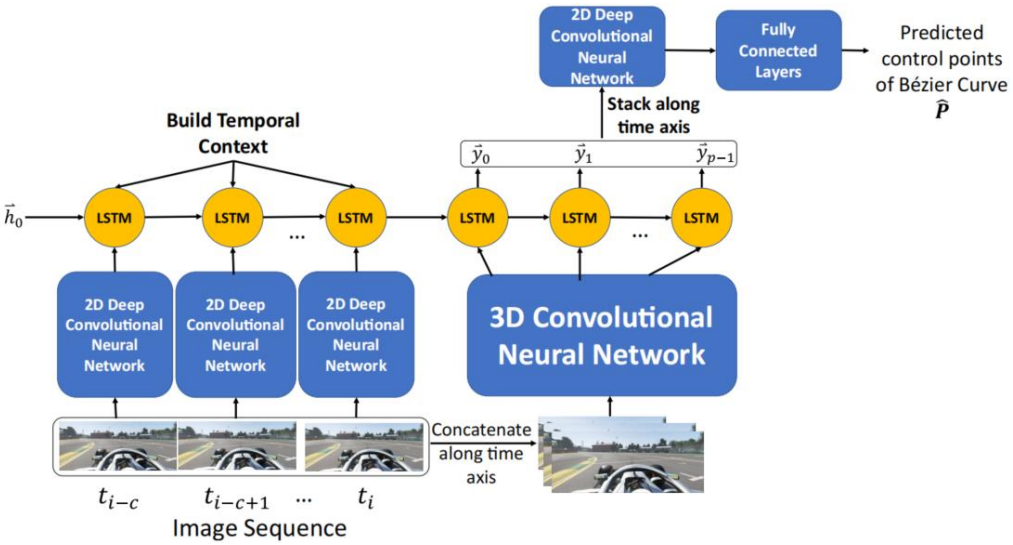

Figure 13 AdmiralNet Bezier Curve Predictor architecture, copied from [7]

2.2.6.1 Bezier curve principle

A Bezier curve is a parametric curve based on Bernstein Polynomials. It has the following characteristics: use a few control points to fit a set of points into a smooth and continuous curve; reshape curve by changing control points; additional control points in part of the curve will not influence the overall shape [15]. Predicting Bezier curve control points instead of waypoints in path planning can vastly reduce the computational load and smooth the trajectory.

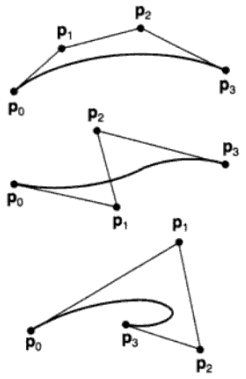

Figure 14 Bezier curves and their control points, copied from [16]

2.2.6.2 Trajectory Control

Unlike fully end-to-end models that output direct control commands, trajectory outputs must be fitted into an extra control algorithm to generate steering and throttle commands. Traditional control algorithms like PID control, Bang-bang control are not suitable for high-speed driving because these controls cannot handle sharp turns. Pure pursuit control is capable of this task. It is first introduced in 1985 [17] and has an extensive application in vehicle path following. It defines a lookahead distance as a product of a tunable parameter and vehicle current speed, so the faster, the longer vehicle will look ahead. Once the lookahead point is selected from the waypoints, we can use simplified Ackermann forward kinematics to drive a curve toward it [18].

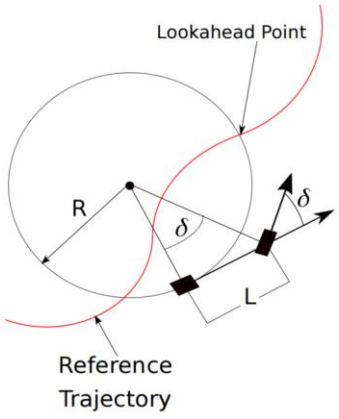

Figure 15 Pure Pursuit Control illustration, copied from [18]

2.2.7 Residual Neural Networks

A Residual Neural Network (ResNet) is a neural network with skip connections (or shortcuts) between layers. A skip connection and several convolutional layers compose a residual unit. This unit typically uses Batch normalization as regularization and rectified linear unit (ReLU) as activation function. This neural network is often used in more profound neural network training and can solve the "Vanishing Gradient Problem," hence speeding up learning [12].

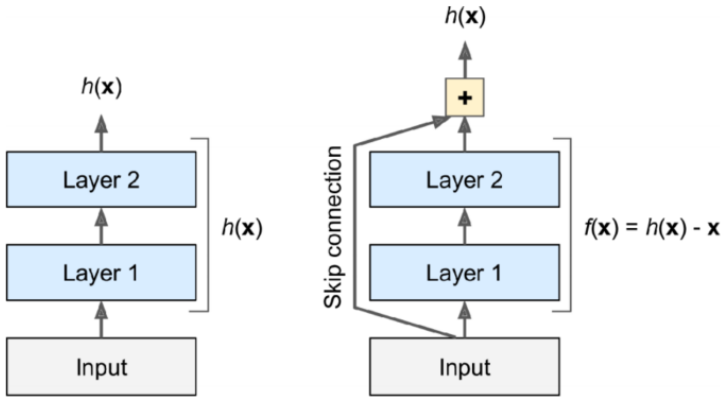

Figure 16 ResNet structure, copied from [12]

2.2.8 2020 NVIDIA new PilotNet

Surprisingly, NVIDIA also experimented "pixels to trajectory" deep learning method and started just two years after their original PilotNet. In 2020, NVIDIA shared their current achievements in the paper *The NVIDIA PilotNet Experiments*. This paper introduces a new PilotNet method that outputs a predicted trajectory in the car's local coordinate system. The ground-truth trajectory is derived from vehicle odometry, inertial measurement unit (IMU), and GPS, then the data and pixel input are filled in a modified ResNet. PilotNet can also be trained to output up to 7 different trajectories, including lane stable, change to left lane (first half), change to left lane (second half), change to right lane (first half), change to right lane (second half), split right

branch, split left branch. An advantage of the new PilotNet is that it can be easily integrated with other systems like obstacle detection systems [19].

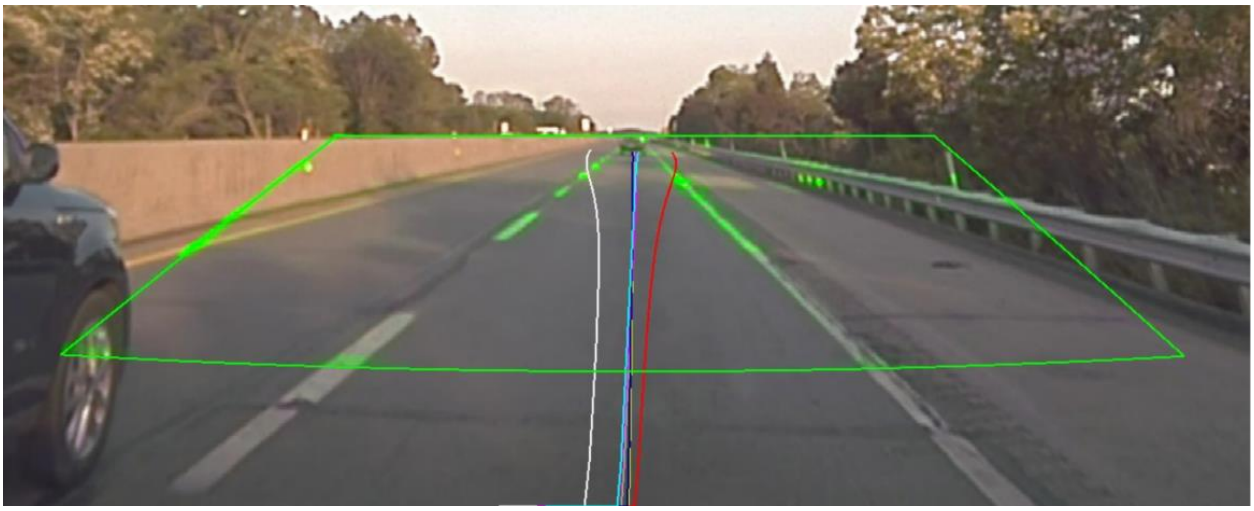

Figure 17 New PilotNet displaying the Region of Interest (ROI) and 7 desired trajectories, copied from [19]

NVIDIA's new PilotNet has the following layers in sequence: NVIDIA's new PilotNet has the following layers in sequence:

- One Batch norm year
- Four residual layers
- Two convolution layers
- One flattened layer
- Five linear layers

It takes a sequence of images as input and outputs various trajectories. For simplicity, we choose line stable as the only output trajectory [19].

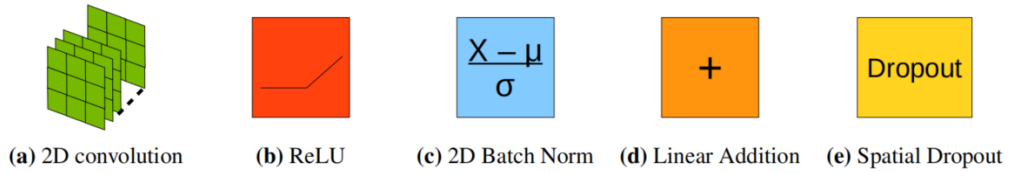

Figure 18 Basic blocks of new PilotNet, copied from [19]

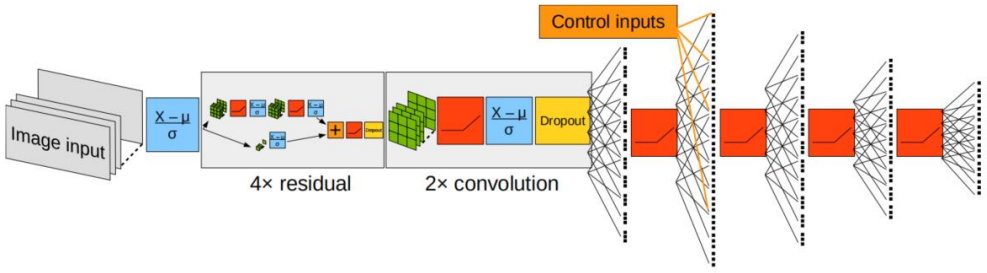

Figure 19 Full new PilotNet architecture, copied from [19]

2.2.9 Optical Flow Principle

Optical or optic flow is the apparent movement velocities of objects attributable to relative motion between viewer and scene. It is extracted from two consecutive frames with space and time information and can be roughly divided into two categories: sparse optical flow and dense optical flow. Lucas–Kanade method can calculate Sparse optical flow. This method is suitable for high noise images. Other methods like Horn–Schunck method and Gunnar Farneback method can calculate Dense optical flow. Horn–Schunck method uses flow smoothness assumption, while the Gunnar Farneback method uses polynomial expansion. Optical flow has extensive use in motion detection and video classification. As shown in Fig. 20, the motion of pixels is captured.

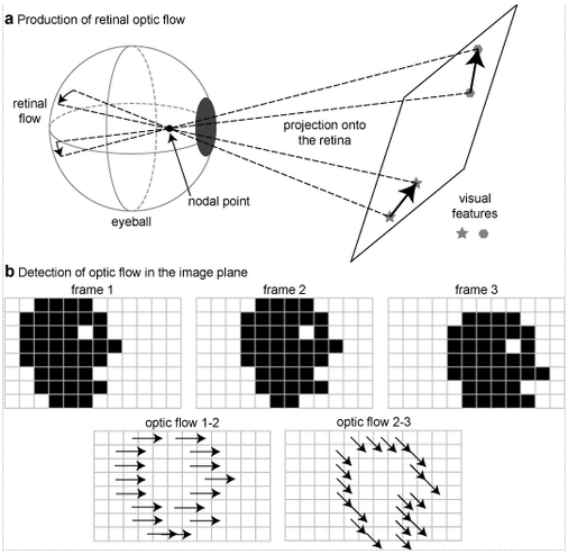

Figure 20 Optical flow explanation, copied from [20]

2.2.10 AdmiralNet with optical flow

CPS Lab also tried optical flow with their AdmiralNet: optical flow + CNN + LSTM (pixels to control). Instead of only taking raw image pixels as input, they computed the optical flow vector field as additional input. The results are astounding. The RMSE of steering angle prediction was reduced 4 to 6 times [21].

Table 3 Comparison between steering angle predictions, copied from [21]

	Australia	Bahrain
	Testing (RMSE)	Testing (RMSE)
PilotNet	0.1699	0.3309
CNN-LSTM	0.1485	0.40255
AdmiralNet	**0.0368**	**0.067**

AdmiralNet with optical flow contains original PilotNet, LSTM, and 3D convolution. To add optical flow, firstly calculate the optical flow vector fields by Farneback method, secondly

separate it into horizontal vector fields and vertical vector fields, finally combine these two channels with a grayscale image as three channel input [21].

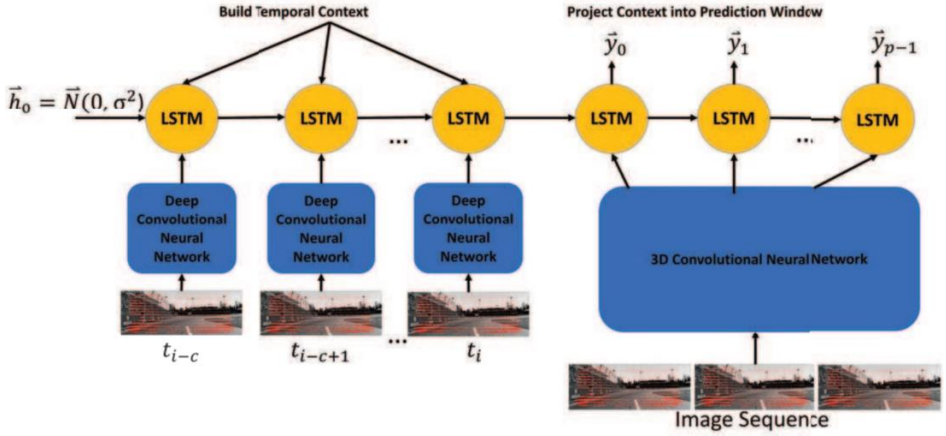

Figure 21 AdmiralNet+optical flow architecture, copied from [21]

3. Design

3.1 Hardware

3.1.1 ModelCar-2 Autonomous Driving Robot

ModelCar-2 is available as a result of a previous student research project [6]. The components are as follow:

Table 4 Hardware table

Name	**Type**	**Function**
RC car	Traxxas Stampede XL-5	Actuator, platform
Embedded Controller	Raspberry Pi 4 Model B	Central controller that connects all hardware
Camera	5MP OV5647 Digital Camera with LS-40180 Fisheye Lens	Deep learning data collector
2D LIDAR Unit	Hokuyo URG-04LX-UG01 LIDAR	Data collector, Lidar autonomous driving mode
Handheld Controller	Logitech F710 Gamepad	Manual Driving control and program menu navigation
Power Bank	Rock space 10000 mAh power bank	Power Raspberry Pi and the LIDAR unit
Screen	3.5 Inch WaveShare LCD Touchscreen	Display the user interface
Battery	Traxxas Battery, 3000mAh (NiMH, 7-C hump, 8.4V)	Power RC car

3.2 Software

Table 5 Software table

Name	Function
RoBIOS[2]	The RoBIOS-7 library is an application programming interface (API) designed for the 'EyeBot' mobile robot family [22] and the 'EyeSim' mobile robot simulator [23], available programming languages including C, C++, and Python.
TensorFlow/Keras[3]	Keras is an API for Python deep learning. In this project, we need Keras to build architecture for several neural network models.
OpenCV[4]	OpenCV is an API for image processing. It can resize, rotate, crop, and re-format images.
Pygame[5]	Pygame is an API for video game programming. The function needed here is joystick control commands.
ServoBlaster[6]	ServoBlaster is software for Raspberry Pi servo control. It can send Pulse width modulation (PWM) signals to multiple servos via the general-purpose input/output (GPIO) pins.
BreezyLidar[7]	BreezyLidar is software for receiving Lidar data, supporting Python and C++ in Linux computers.
BreezySLAM[8]	BreezySLAM is software for processing Lidar data, supporting Python in Linux computers. It can generate SLAM maps from Lidar data.

3.3 ModelCar-2

Except for replacing the new battery and wrapping the tires with tape to reduce friction, the overall components of the ModelCar-2 have not changed. It should be noted that the cable

[2] https://robotics.ee.uwa.edu.au/eyebot/Robios7.html

[3] https://github.com/keras-team/keras

[4] https://github.com/opencv/opencv

[5] https://github.com/pygame/pygame

[6] https://github.com/richardghirst/PiBits/tree/master/ServoBlaster

[7] https://github.com/simondlevy/BreezyLidar

[8] https://github.com/simondlevy/BreezySLAM

connecting the power bank and the Raspberry Pi must transmit a stable current of 2A. Otherwise, the SD card will be damaged due to an insufficient power supply.

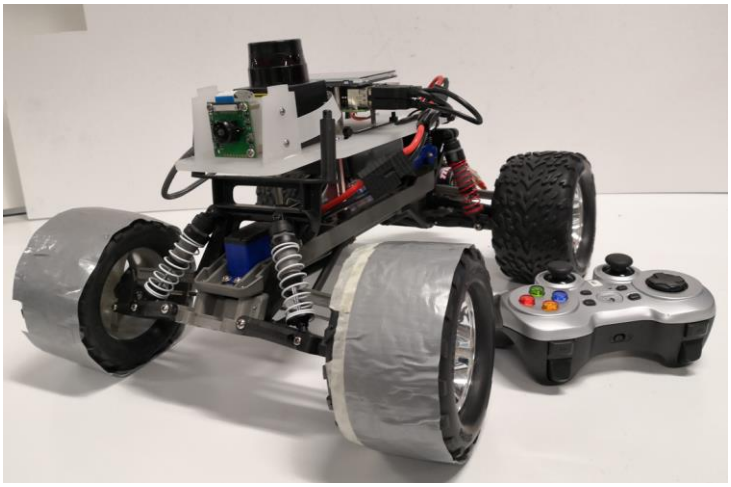

Figure 22 ModelCar-2 Appearance

The ModelCar-2 uses the RoBIOS GUI as the low-level control program, so users can efficiently operate it via a touch screen or gamepad. There are three driving modes: manual, camera neural network, and Lidar. The manual drive mode uses the gamepad to control the speed and steer the robot, and image recordings are possible at a 30Hz frame rate. Lidar drive mode uses the Lidar sensor to measure the distance of the surrounding environment and then inputs these distances into a fine-tuned algorithm to calculate the speed and steering of the robot. The camera neural network drive mode uses the camera to capture images and then feeds these images into a trained neural network to generate the speed and steering of the robot.

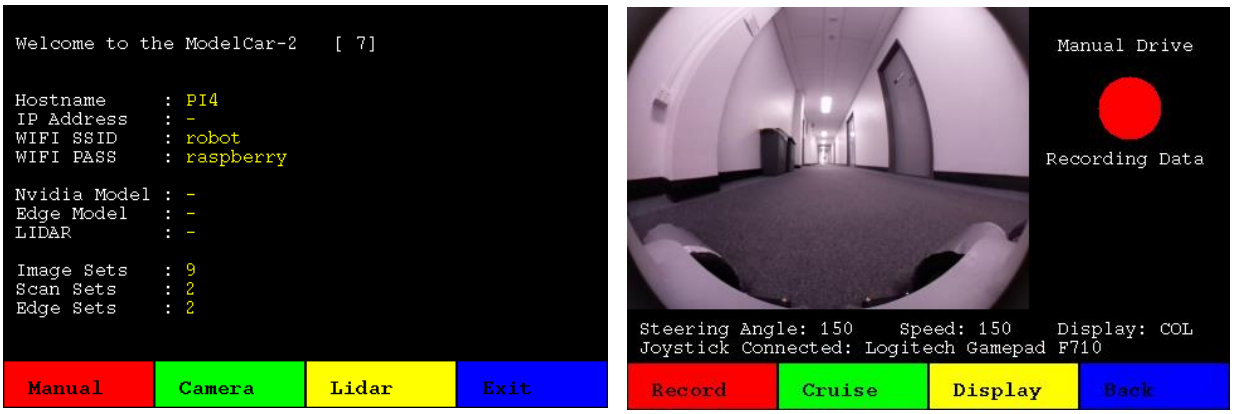

Figure 23 ModelCar-2 Program Main interface (left) Manual Drive Mode (right)

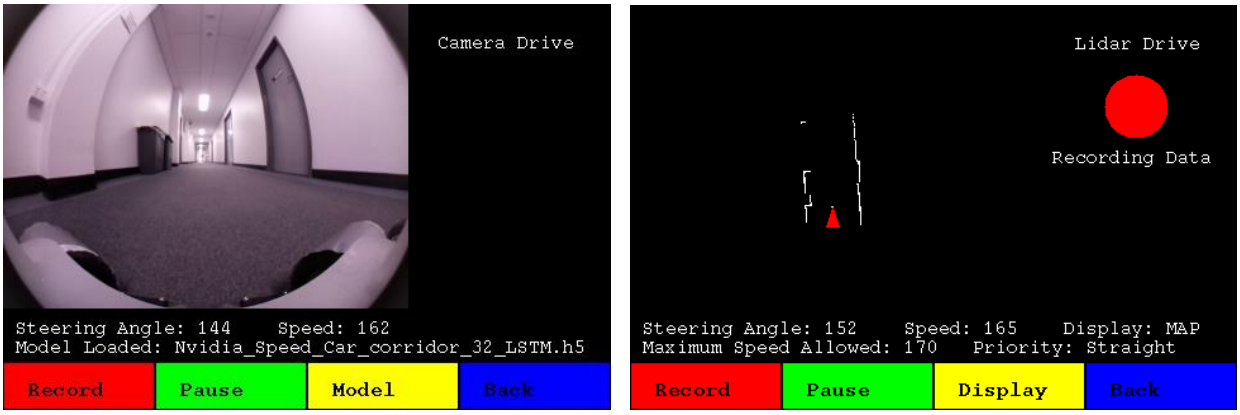

Figure 24 Camera Neural Network Drive Mode (left) Lidar Drive Mode (right)

Arrows of different colors indicate the control flow of the three driving modes in Fig. 25.

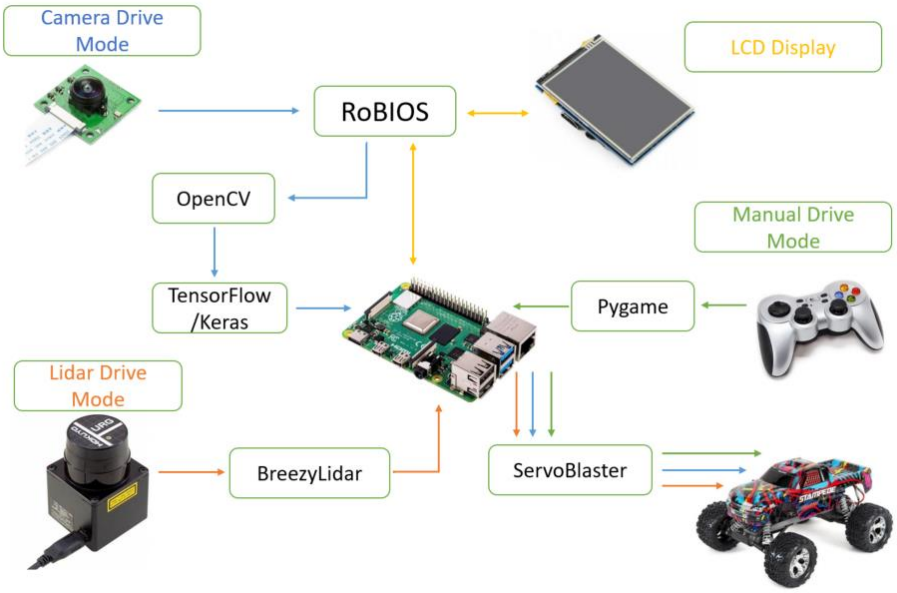

Figure 25 Simplified Control Flow for ModelCar-2

3.4 Neural network models

This part introduces the neural network models intend to design in the project.

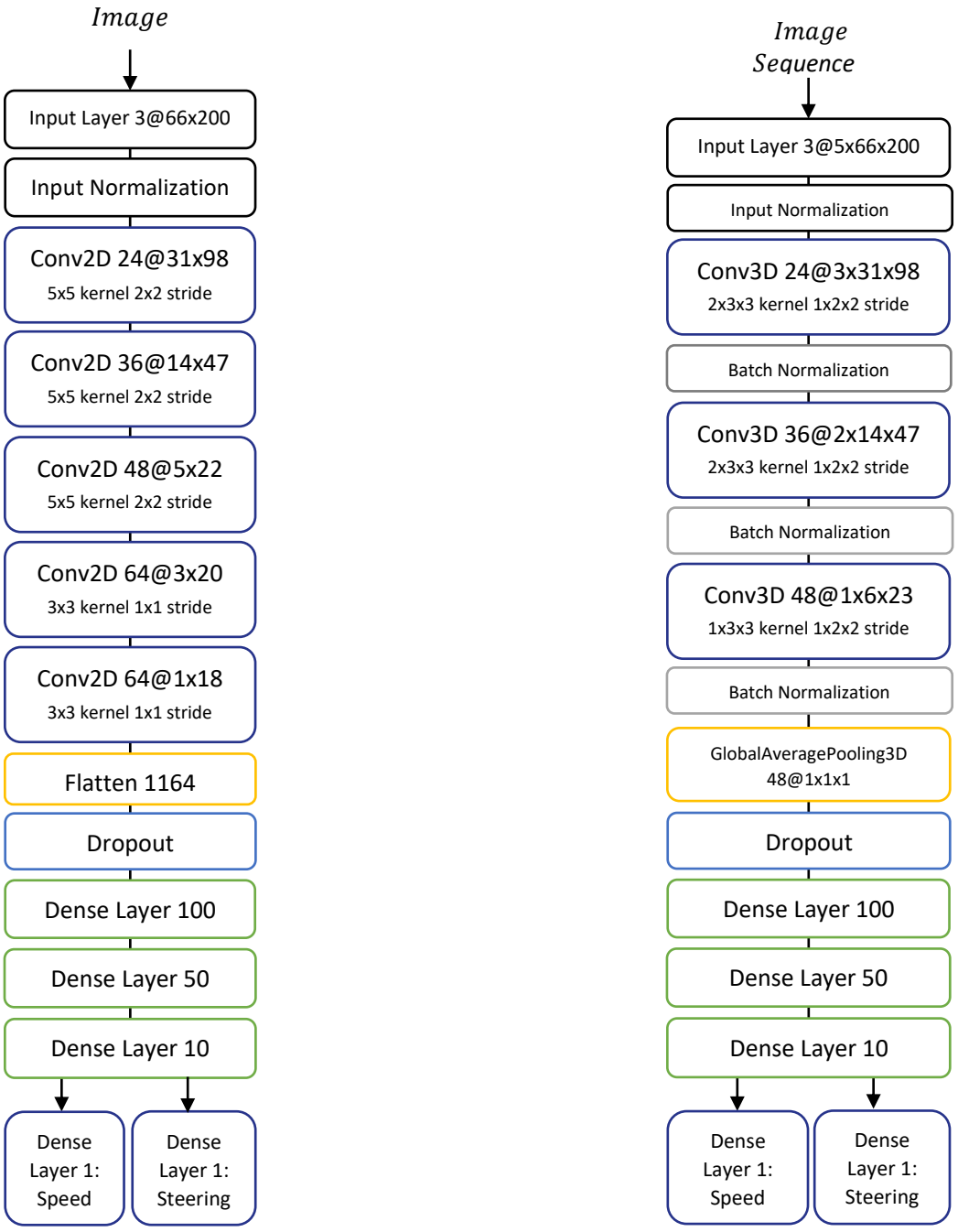

Figure 26 Original PilotNet Architecture (left) 3DCNN Architecture (sequence length=5) (right)

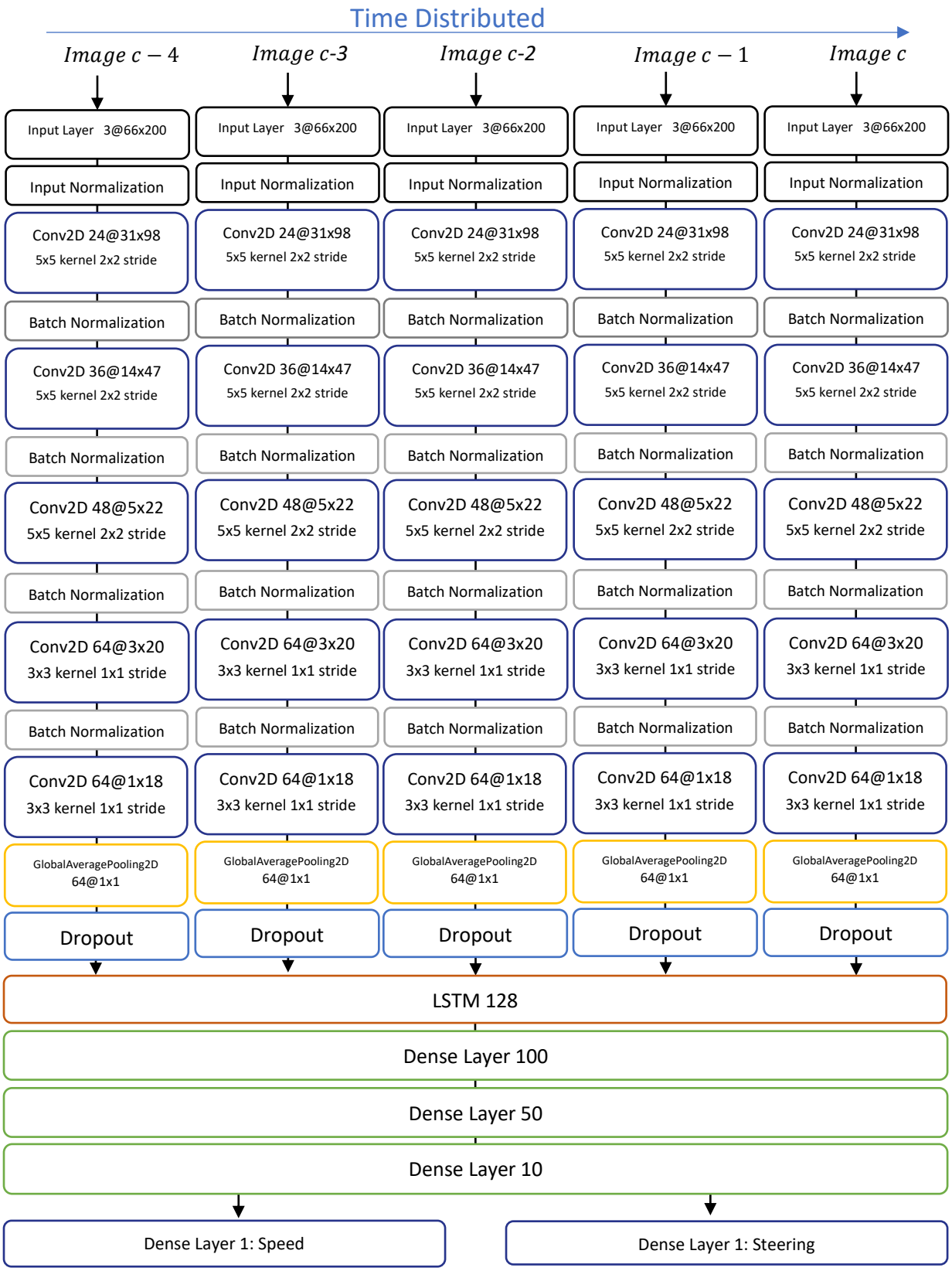

Figure 27 CNN+LSTM Architecture (image c: current image, image c-i, i=1,2,3,4: past 4 images)

3.4.1 Original PilotNet

The previous work experimented with the original PilotNet using speed as additional output and dropout for the deep network's regularization [6]. We adopt this architecture, which is illustrated in Fig.26 left diagram. Input normalization converts the image pixel values from (0-255) ‘uint8’ type to (0-1) ‘float’ type. The activation function used by the model is ‘ELU,’ and its output range is [0, inf]. If the input is not normalized, the activation function will encounter the “Exploring Gradient Problem”, so the next two models also applied input normalization.

3.4.2 CNN+LSTM

The CNN+LSTM model (Fig. 27) involves an LSTM layer between the fully connected layers (dense layer) and the 2D convolutional layers. Furthermore, the input changes from a single image to an image sequence (in this case, five images in a row), where the current image and past four images are input into five convolutional layers separately, which are then input into the LSTM layer together. The LSTM layer can extract information along the timeline and assist the model in making better decisions. This model also uses GlobalAveragePooling2D layers to replace the Flatten layers, where the latter layer transforms a multi-dimensional tensor into a one-dimensional tensor. This transformation has a risk of overfitting, especially for complex models. However, the GlobalAveragePooling2D layer solves this problem because it sums out spatial information and has no parameters to optimize. Batch Normalization transforms the data to a zero mean and a unit variance, reducing the “Vanishing or Exploring Gradient Problem,” affording a higher learning rate and speeding up the entire training process.

3.4.3 3DCNN

The 3DCNN model (Fig. 26 right diagram) uses 3D convolutional layers instead of 2D, with the additional dimension employed as a timeline. Specifically, when the input is an image sequence, it obtains both spatial and temporal information. The GlobalAveragePooling3D layer is similar to the GlobalAveragePooling2D layer but with an additional dimension for pooling.

3.4.4 Other models

Initially, several models were planned to be built, due to multiple reasons, the following ones were not developed:

1. CNN+Optical flow
2. ResNet (pixels to waypoints)
3. CNN+LSTM (pixels to waypoints)
4. CNN+LSTM (pixels to Bezier curve)

The CNN+optical flow model was intended to be built but was not due to time shortage. Nevertheless, the preliminary results show the potential of this model, with Fig.28 presenting the optical flow capturing the essential features, i.e., the walls' edges.

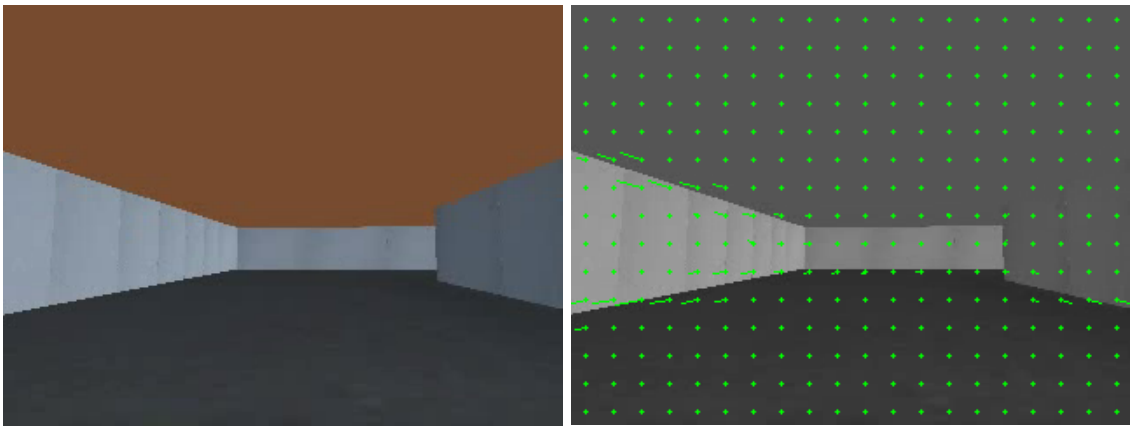

Figure 28 Optical flow in EyeSim Maze Drive View (Left: original image, right: grayscale image with optical flow)

The ResNet (pixels to waypoints), CNN+LSTM (pixels to waypoints), and CNN+LSTM (pixels to Bezier curve) models require an IMU or GPS device to record waypoints for neural network training. These models were not built because no such devices were available during this project. Additionally, even if these models were built, the computing power required exceeds the available platform one as Raspberry Pi 4 can barely run the CNN+LSTM (pixels to control) model, with only input sequences. However, the pixel-to-waypoint and pixel-to-Bezier-curve models have input and output sequences, further increasing the computational load.

4. Method

4.1 Simulation

Before any practical experiments, it is mandatory to try the neural network models on simulation, e.g., the EyeSim platform. The latter is a mobile robot simulator for the EyeBot family that can simulate a mobile car, which has attached a camera, Lidar, a PSD sensor, and an LCD screen for the camera and data display.

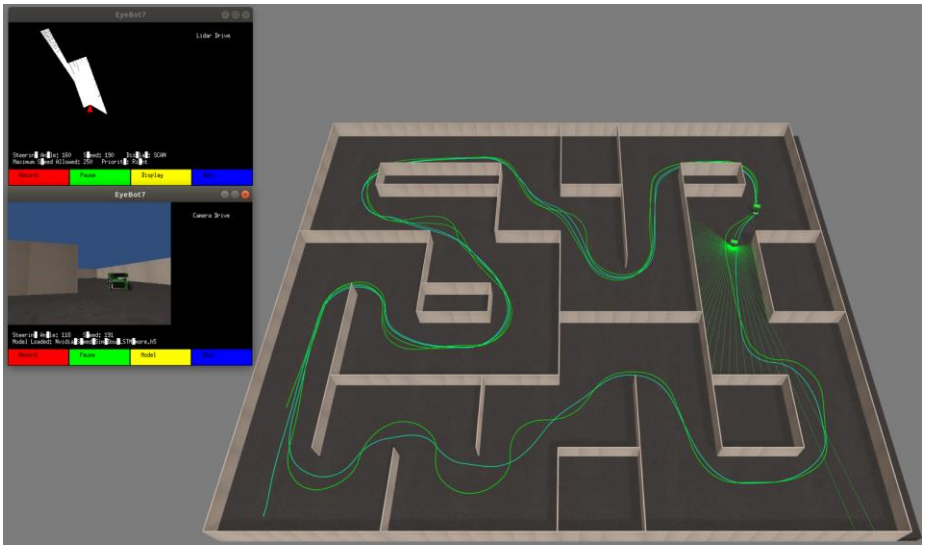

Figure 29 EyeSim Maze Map

4.1.1 Data collection, preprocessing and augmentation

Data collection

A well-designed autonomous driving algorithm is preferred against a button control algorithm to reduce human error when obtaining data for the neural network's training. Thus, we apply a simulated Lidar-based algorithm to drive the ModelCar-2 in the maze map. During this process, we exploit the camera to capture images and store them with a labeled steering angle and speed until adequate training images are captured. 5,110 clockwise driving images and 5,166 anti-clockwise driving images (10,276 images in total) were collected in the simulation experiment. It should be noticed that the Lidar drive mode captures 700 images per lap with a frame rate of 10Hz, and thus, 10,276 images require 15 laps to collect, presenting an adequate training data density. The captured images with labeled steering angle and speed are stored in the npy format, a NumPy-array format that is small and expandable with additional information like labels. Considering the speed, 150 is the corresponding stop value. If the speed exceeds 150, the robot will drive forward. Otherwise, the robot will drive backward. In any case, the higher the value, the faster the robot moves. Regarding the steering angle, 150 is the value corresponding to a straight motion. If the steering angle exceeds 150 the robot turns left (maximum value is 200, corresponding to a 90-degrees left turning). Otherwise, the robot will turn right (100 is the minimum value, corresponding to 90 degrees right turning). Before the neural network training, the images are randomly split into three datasets: training, validation, and testing, with the latter

involving 0.4% of the total images, and the remaining imagery is split into a 4:1 training-to-validation ratio.

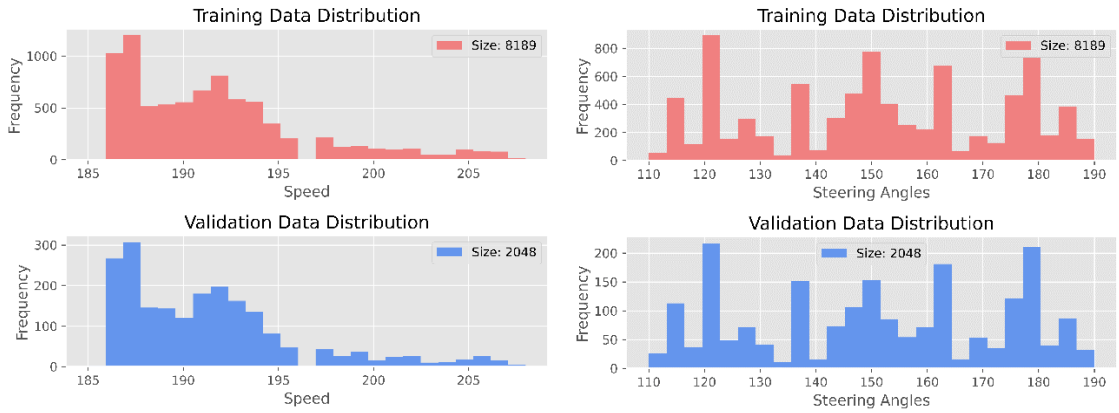

Figure 30 EyeSim Maze Map: Speed Data (left) and Steering Angle Data (right) Distribution

Data preprocessing

The driving program using RoBIOS API captures 320x240 RGB images. Each image is first converted to the YUV color space, then passes through a Gaussian blurring filter, and is finally resized to 200x66. The YUV color space helps adjust the brightness (further details on this process will be mentioned later). The Gaussian blurring filter smooths the resizing process and prevents image distortion [24]. Given that 200x66 is the input size of the original PilotNet, the entire PilotNet architecture is built based on this size, so there is no need to modify that. Additionally, a smaller size can help reduce the computational load, especially when generating image sequences for CNN+LSTM and 3DCNN models.

Data augmentation

Data augmentation increases the size of the dataset by generating variants of the data. This can improve the model's generalization and robustness [25], but excessive augmentation generates much noise preventing the model from learning. Thus, the extent of data augmentation needs to be well considered. This thesis considers two data augmentation types: blurring and flipping. Blurring helps the model learn more holistic features without overlearning on the minutiae, while image flipping balances the uneven distribution of left and right steering data, eliminating the dataset's inherent bias assisting the models' better learning. Each augmentation is randomly applied to 5% of the training data in a single batch.

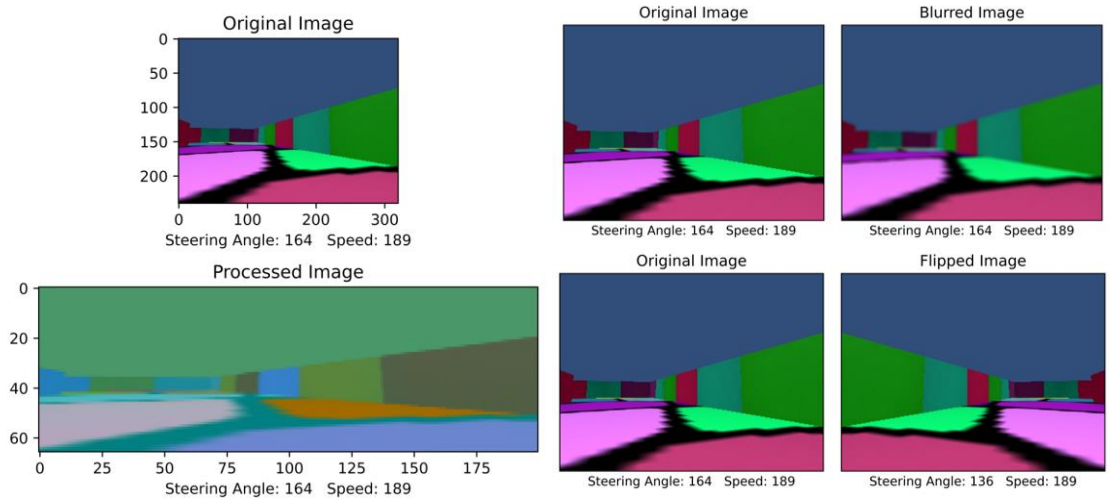

Figure 31 Simulation Image preprocessing (left), Simulation Image Augmentation (right)

4.1.2 Image sequence generation

To generate an image sequence for the CNN+LSTM and 3DCNN models, we arrange the collected pictures in order and use a sequence-length window to slide from the beginning to the end by sliding one image at a time. Each sequence uses the labels of the last image.

4.1.3 Model training

The model training employs the bagging strategy. Bagging or bootstrap aggregation refers to the process of randomly sampling the dataset with replacement in each batch [12], significantly reducing the dataset's variance. Avoiding overfitting and saving training time can be achieved by including early stopping, learning rate decaying, batch normalization, and dropout. This project has a total of three neural network models to train. Thus, we first fit the data into each model with its specific inputs and then train it on Ubuntu 19.04 with two NVIDIA 1080Ti GPUs. Once training finishes, we copy the models to the EyeSim model folder for further exploitation.

Table 6 Basic parameter settings for model training

Batch size	**Training steps**	**Validation steps**	**Epochs**	**Early Stopping**	**Learning rate schedule**
32	Number of raining image /batch size (If using data augmentation then times three)	Number of validation image /batch size	100	If the validation loss does not improve within 10 epochs, stop.	If the validation loss does not improve within four epochs, reduce it to one-fifth of the original. (Initial: 0.001, minimum: 0.000001)

4.1.4 Validation and comparison methods

Open-loop metrics consider the following**:**

i. Separate the training set and validation set with an 0.8 training ratio and then calculate the Mean Square Error (MSE) of each model.
ii. Calculate the steering angle and speed accuracy of each model.
iii. Display the saliency map to show the inner layer of each neural network model, with the highlighted saliency map being the model's focus. If the model is appropriately learning, the walls' edges should be the focus.

The training loss (MSE), validation loss (MSE), steering angle accuracy, and speed accuracy are calculated in each epoch during the model training by Keras API.

Saliency maps are shown after training using the Keras.vis API [9].

Closed-loop metrics consider the following**:**

We separately run the models in the EyeSim maze map and apply the following metrics:

i. Mean Lap time[18]: if a lap is note finished, the label "did not finish" (DNF) is placed.
ii. $Autonomy$ [3]: The percentage of time the network model drives the vehicle without human intervention.

$$Autonomy = \left(1 - \frac{Number\ of\ Interventions \times\ human\ reset\ time[seconds]}{Elapsed\ Time\ [seconds]}\right) \times 100\%$$

The ModelCar-2 driving program has a pause function. Each time the robot hits a wall or stops moving, the pause button will be pressed and counted as an intervention. We set the manual reset time to 5 seconds. The elapsed time is the total time minus the pause time.

4.2 Practical experiments

This project considers two practical experiments: driving in a rectangular loop and turning around at a corridor with two dead ends.

4.2.1 Data collection, preprocessing and augmentation

Data collection

We apply the Lidar-based algorithm to drive ModelCar-2 on the 4th floor of the EECE building and the Ground floor of the CME building.

The 4th floor of the EECE building has a rectangular loop for the Lidar drive mode to run continuously (Fig.32 red zone). In the Lidar drive mode, we prioritize right-turning first and collect 30,679 clockwise driving images. Then we prioritized left-turning first and collected 30,518 anti-clockwise driving images (61,197 images in total). The Lidar driving mode captures

[9] https://github.com/raghakot/keras-vis

590 images per lap at a frame rate of 10Hz, and thus, 61,197 images require 104 laps to be collected. This data density far exceeds the simulation because the natural environment is more complicated, and data is unevenly distributed. As illustrated in Fig. 33, the steering angle in most pictures is marked as 150, i.e., the car is driving in a straight line. This unbalance causes the model to ignore a few turns and fails a correct learning process. Thus, more data are needed to compensate for this bias.

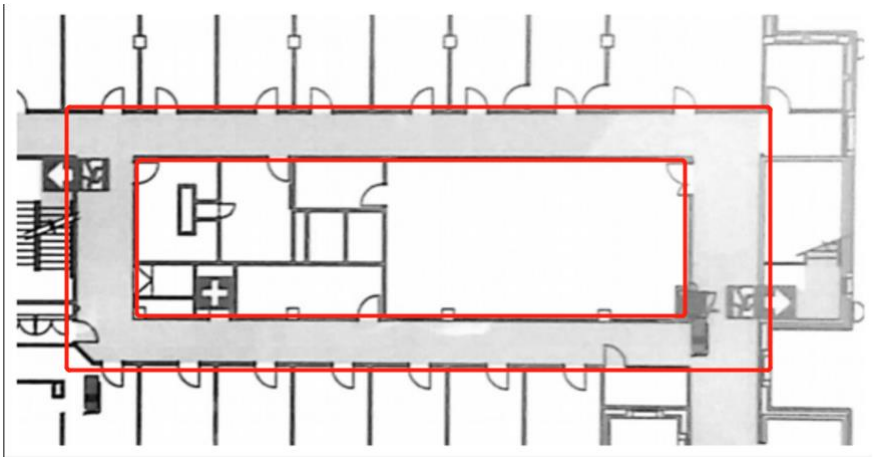

Figure 32 UWA EECE 4th Floor Plan, copied from [5]

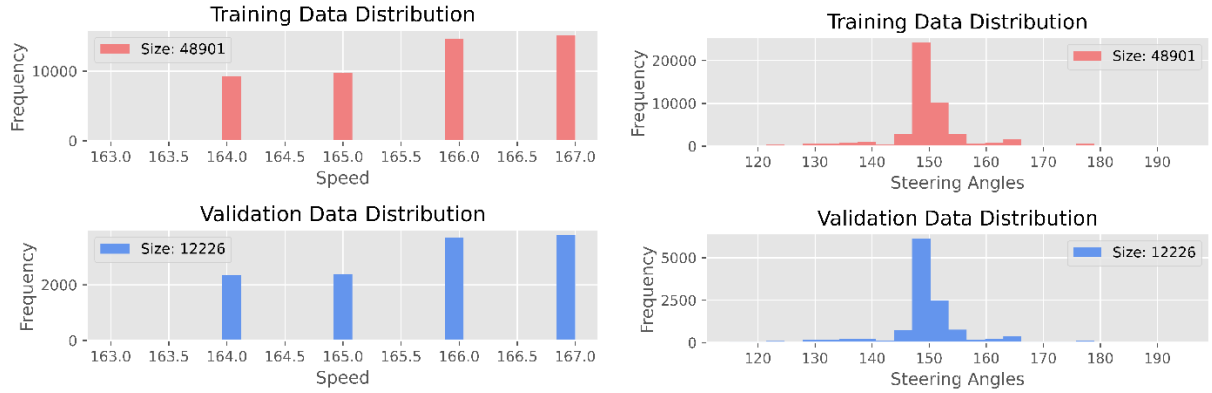

Figure 33 EECE Building: Speed Data (left) and Steering Angle Data (right) Distribution

The Ground floor of the CME building has a corridor with two doors (Fig.34 red zone). If we close the doors on both sides, the Lidar drive mode can drive and turn around in the corridor without stopping. This corridor is not wide enough for the robot to make a complete U-turn, so the robot moves as follows. It stays in the middle of the walls on both sides. If it encounters a dead end, it turns left until it approaches the wall, turns right, reverses, and finally turns left to the center of the road to repeat the initial steps. For this scenario, we collected 47,838 images. The Lidar drive mode captures 560 images in one round trip with a frame rate of 10Hz. Thus,

47,838 images need 86 rounds to be collected. The data density is sufficient, but due to the design defects of the Lidar's driving program, when encountering a dead end, the robot has a 50% chance to turn left and a 50% chance to turn right. This inconsistent behavior is not conducive to the training of the neural network models, with data cleaning reducing this effect. Hence, we manually delete the images where the car turns to the right.

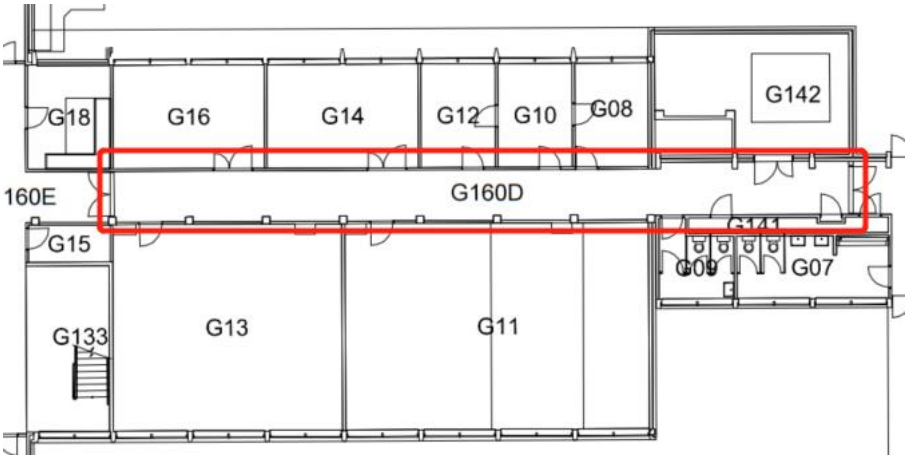

Figure 34 UWA CME Ground Floor Plan

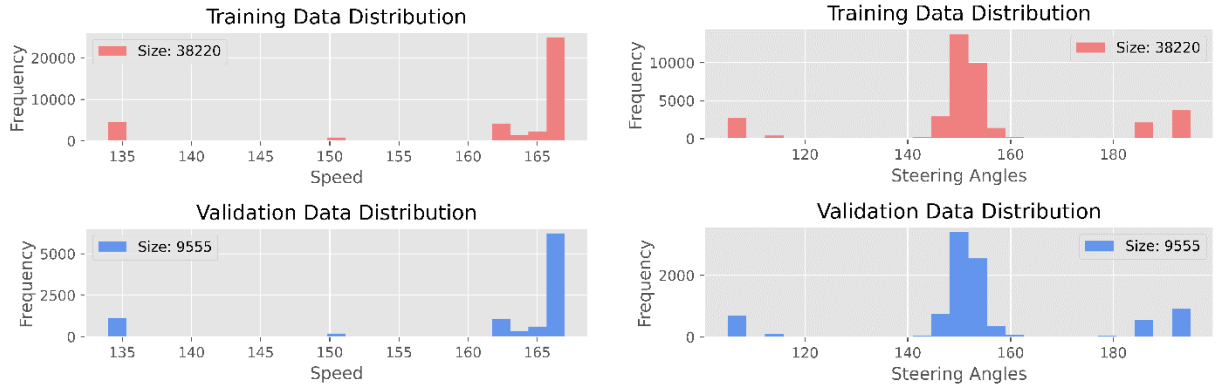

Figure 35 CME Building: Speed Data (left) and Steering Angle Data (right) Distribution

Data preprocessing

Practical data preprocessing is mainly the same process as for simulation, except that the images are taken by a fisheye lens, requiring additional image processing to eliminate lens distortion. Precisely, the images are cropped 25% of the top and 25% of the bottom before being converted into YUV color space.

Data augmentation

Practical experiments consider an additional data augmentation process, as the brightness of both places is changing due to sunlight. We randomly modify the image brightness within the range of [-20%, 20%] and utilize brightness modification on 5% of the training data in a single batch.

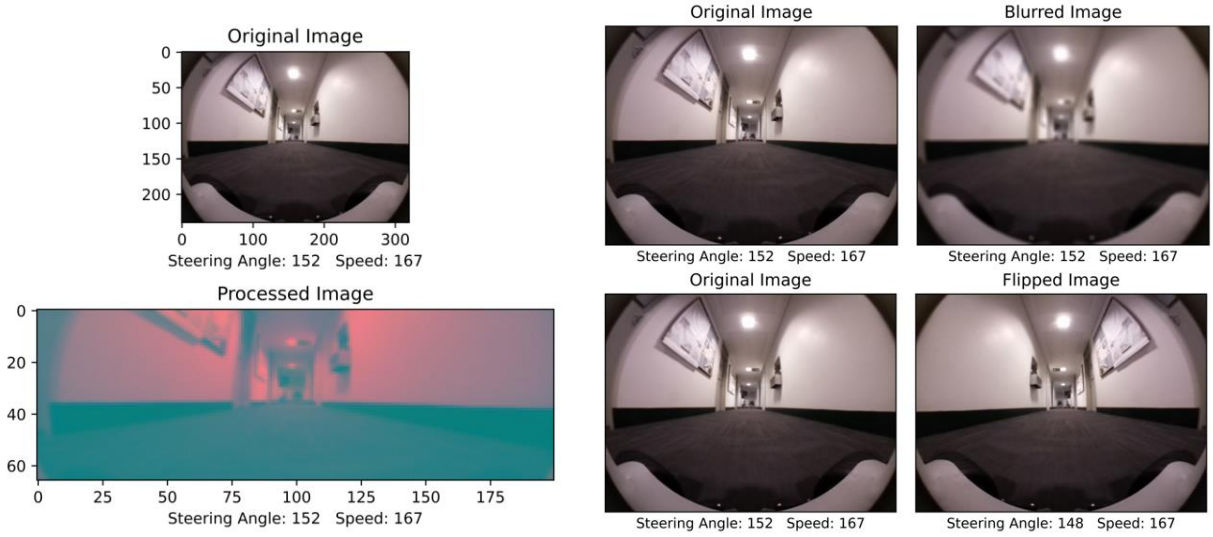

Figure 36 Practical Image preprocessing (left), Practical Image Augmentation (right)

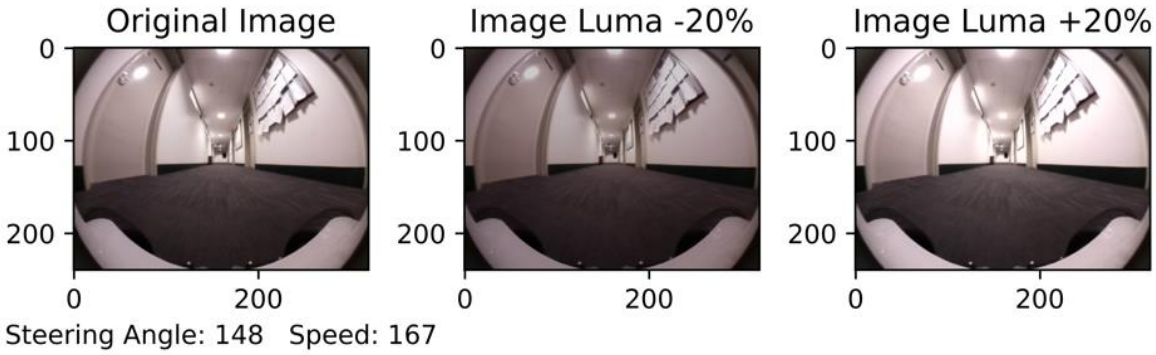

Figure 37 Practical Image Augmentation: Brightness

4.2.2 Model training

Practical model training is identical to the simulation. Once finished, we copy the models to the Raspberry Pi model folder for further manipulation.

4.2.3 Validation and comparison method

The practical validation and comparison methods are identical to the simulation ones.

5. Results

5.1 Simulation Results

Table 7 Simulation models comparison.

Test \ Model	CNN+LSTM	3DCNN	PilotNet	Lidar
Training Loss (MSE)	**11.3484**	34.6803	29.1005	
Validation Loss (MSE)	**4.4548**	17.8612	8.1348	
Speed Accuracy	**32.94%**	14.19%	24.28%	
Steering Accuracy	**32.78%**	13.72%	22.44%	
Mean Lap Time in maze (s)	**73.45**	73.61	73.94	76.89
Autonomy	100%	100%	100%	100%

Other settings: Batch size=32, Data augmentation=False, Training steps=3*(Number of training image/batch size), Validation steps=1.6*(Number of validation image/batch size).

Open-loop test

The CNN+LSTM model affords the best performance in the open-loop test, but the metrics utilized do not necessarily correlate with the real driving performance. This is because the Lidar drive mode used to collect data is not the optimal solution, as the neural network model may surpass the performance of Lidar during the learning process. The more the neural network model mimics the Lidar drive mode, the more it is restricted by the Lidar driving performance.

Closed-loop test

In the closed-loop test, each model is applied on six laps to calculate the mean lap time, and then each model is subjected to two 10-minute autonomy tests. The results show that the CNN+LSTM model affords the shortest time per lap, but each model learns from the same speed samples, suggesting that the CNN+LSTM model attains a better solution and travels a shorter route. Overall, all neural network models can drive perfectly in a clockwise and a counterclockwise 10-minute autonomy test without requiring any human. Additionally, all neural network models are much faster than the Lidar model they learned from. Furthermore, the results indicate that the Lidar drive algorithm does not apply optimal route planning, deliberately keeping a distance from each obstacle that significantly limits its driving performance. It is worth noting that amending to this method optimal route planning requires significant effort, while in contrast, neural networks learn optimal route planning automatically.

Prediction

The prediction process of all neural network models is very similar. Here we use the CNN+LSTM model as an example. Although the last image of the sequence is illustrated, the corresponding predictions exploit five consecutive images as input. As presented in Fig.38, the predicted steering angles are significantly different from the actual ones, without suggesting that these predictions are not good, as these "wrong" predictions allow the neural network models to drive the robot better than the Lidar model.

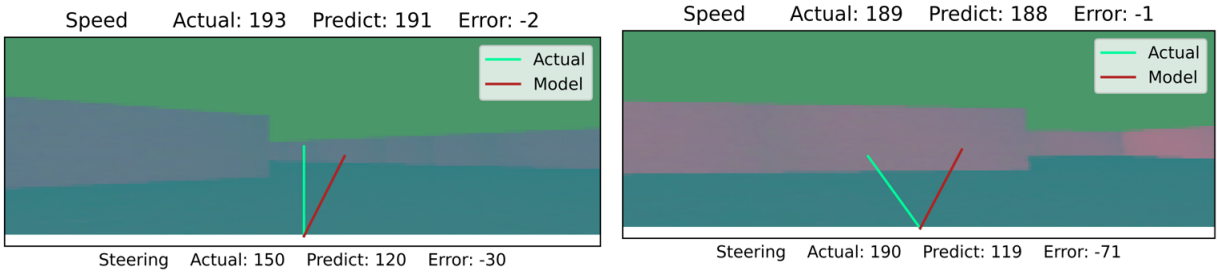

Figure 38 CNN+LSTM prediction in EyeSim maze map

Obstacle avoidance test

In this trial, we add new walls as obstacles to test the model's ability to avoid obstacles in the simulation map. All neural network models react to obstacles and change their routes, indicating that the neural network models have successfully learned to recognize walls and plan routes based on them. For illustration purposes, the new walls (obstacles) in Fig.39 are marked in red. Despite the Lidar model avoiding these obstacles, this model is sometimes too sensitive and forces the robot to turn around. This performance suggests that to avoid these obstacles ideally, several adjustments have to be made to the Lidar algorithm every time it encounters a new environment limiting its effectiveness. While the neural network models adapt to the new environment well without any adjustment.

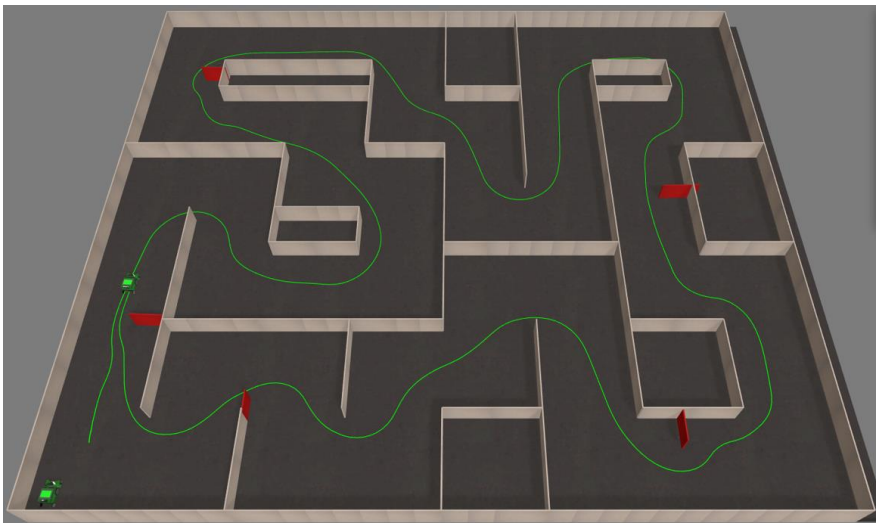

Figure 39 Simulation models obstacle avoidance test

Saliency Map

All neural network models have successfully highlighted the walls in the saliency map. For the PilotNet saliency map, the red dots (high saliency) are on the walls' edges (Fig. 40). Regarding the CNN+LSTM and the 3DCNN, Figs. 41 and 42, respectively, illustrate the outline of the walls on each saliency map.

In terms of wall recognition, 3DCNN manages a poorer performance than CNN+LSTM because 3DCNN employs only three convolutional layers, while CNN+LSTM utilizes five. Additionally, CNN+LSTM has the same weight for each image within the image sequence, while 3DCNN gives the middle image the highest weight. This is because CNN+LSTM processes the temporal and spatial dimensions separately by first extracting the spatial features in the 2D convolutional layers and then inputting these features into the LSTM layer to extract the temporal features. In comparison, 3DCNN extracts both temporal and spatial features utilizing the 3D convolutional layers [26].

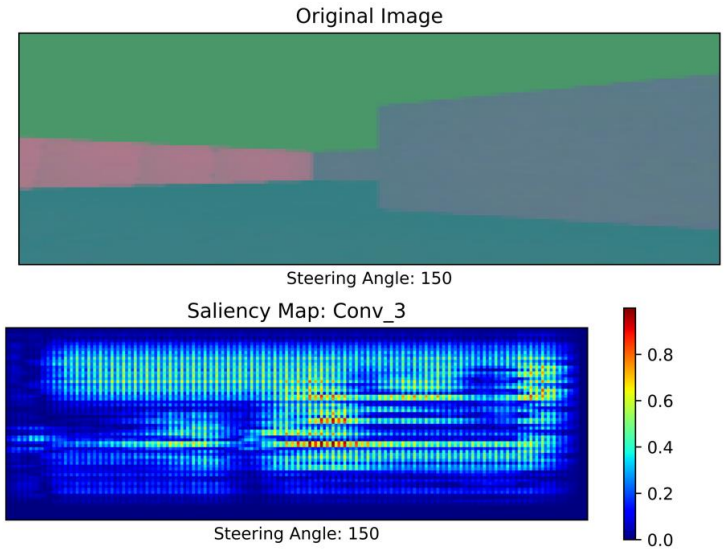

Figure 40 PilotNet saliency map in EyeSim maze map

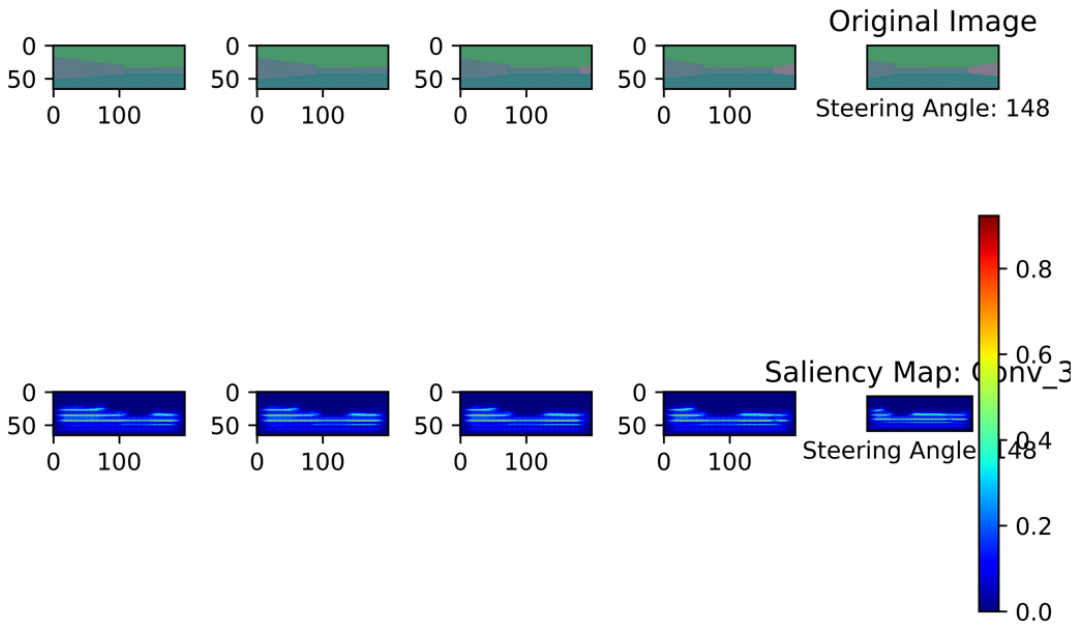

Figure 41 CNN+LSTM saliency map in EyeSim maze map

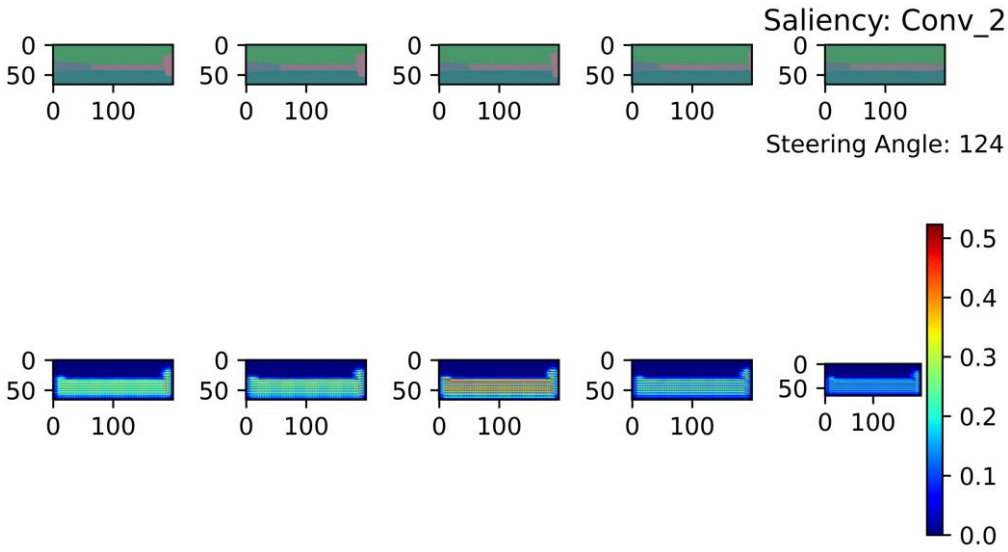

Figure 42 3DCNN saliency map in EyeSim maze map

5.2 Practical Results

5.2.1 EECE rectangular loop

Table 8 Rectangular loop models comparison.

Test / Model	CNN+LSTM	3DCNN	PilotNet	Lidar
Training Loss (MSE)	2.5974	4.8579	**2.5347**	
Validation Loss (MSE)	**0.7065**	1.9746	0.8791	
Speed Accuracy	**82.89%**	75.67%	81.86%	
Steering Accuracy	53.27%	40.90%	**54.84%**	
Mean Lap Time(s)	86.63	93.85	**49.34**	50.65
Autonomy	64.64%	44.10%	**100%**	**100%**

Other settings: Batch size=32, Data augmentation=False, Training steps= (Number of training image/batch size), Validation steps= (Number of validation image/batch size).

Open-loop test

In the open-loop test, the CNN+LSTM model attains the best performance. However, the models' performance does not match the closed-loop tests due to data imbalance and lack of computing power.

Closed-loop test

In the closed-loop test, each model runs three laps clockwise and three laps anti-clockwise to calculate the mean lap time and autonomy. Among the neural network models, only PilotNet can drive steadily, with the remaining two models being affected by past images and either turned too early or too late. Hence, except PilotNet, all methods could not stay in the middle of the road when going straight.

Prediction

Here we use the CNN+LSTM model predictions to analyze its failure. In Fig.43, the left picture shows the model did not learn the fine adjustments, as, without fine adjustments, the robot cannot stay in the middle of the road due to hardware bias. The right picture depicts the case where the model turned prematurely because the actual size of the robot is relatively large compared to the camera (it is possible that the camera does not see the wall, but the tire is blocked).

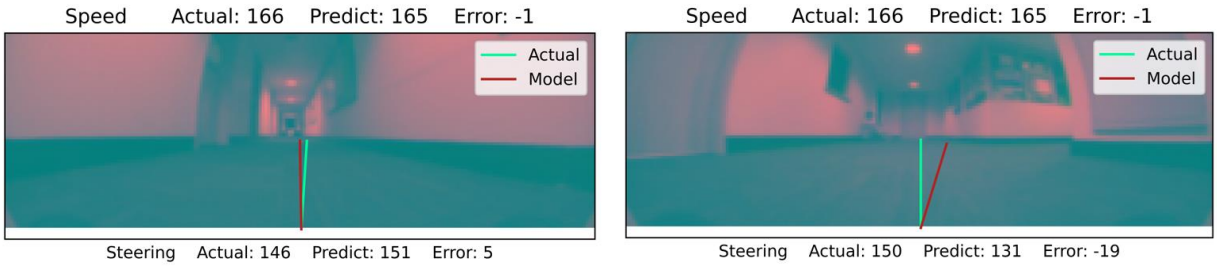

Figure 43 EECE CNN+LSTM predictions

Saliency Map

All neural network models have successfully highlighted the walls' edges in the saliency map.

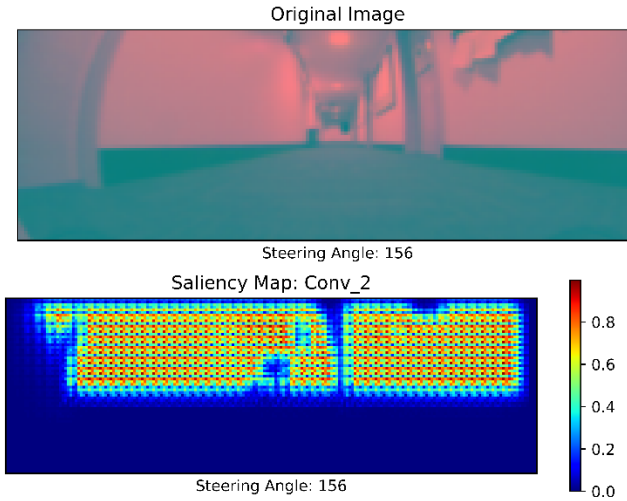

Figure 44 EECE PilotNet Saliency map

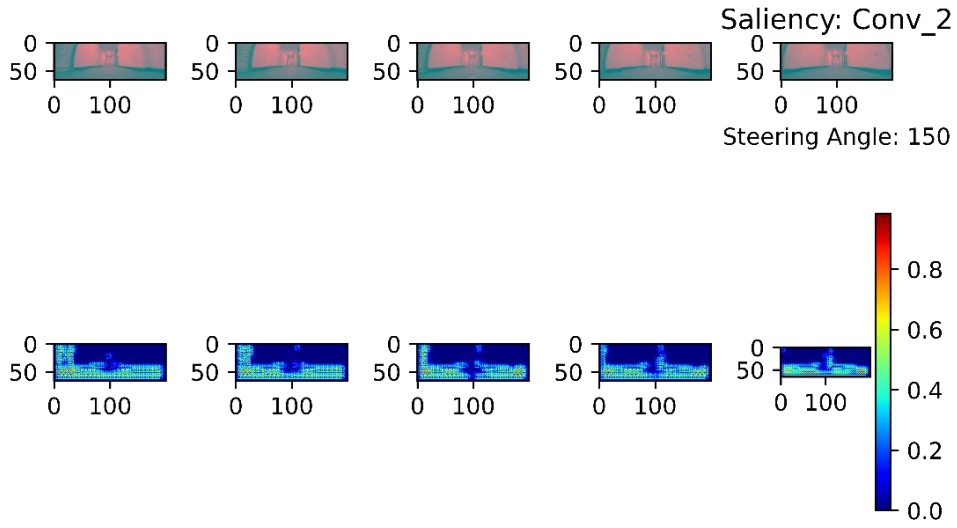

Figure 45 EECE CNN+LSTM Saliency map

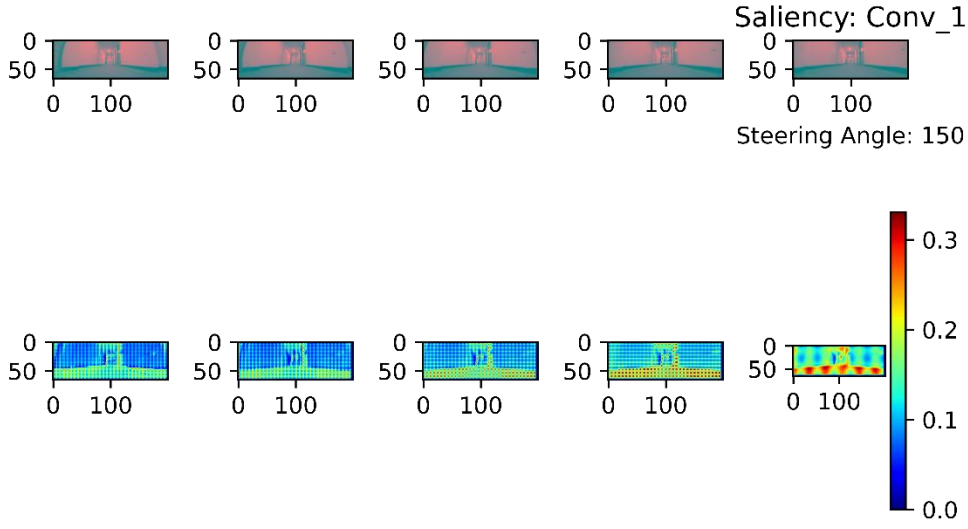

Figure 46 EECE 3DCNN Saliency map

5.2.2 CME corridor

Table 9 Corridor models comparison.

Test \ Model	CNN+LSTM	3DCNN	PilotNet	Lidar
Training Loss (MSE)	**19.5104**	22.2323	51.1504	
Validation Loss (MSE)	**1.637**	2.0445	7.3155	
Speed Accuracy	**72.17%**	60.03%	63.98%	
Steering Accuracy	**36.05%**	34.35%	33.72%	
Mean Lap Time(s)	107	108	143	**56**
Autonomy	84.11%	78.70%	69.23%	**100%**

Other settings: Batch size=32, Data augmentation=False, Training steps=1.7*(Number of training image/batch size), Validation steps=1.3*(Number of validation image/batch size).

Open-loop test

In the open-loop test, the CNN+LSTM model performs best, matching the performance of the closed-loop tests.

Closed-loop test

In the closed-loop test, each model runs six laps to calculate the mean lap time and autonomy. Among the neural network models, none can complete a lap without human interventions. Despite both CNN+LSTM and 3DCNN models effectively turning the car around at the dead-end without human interventions, the car hit the left wall multiple times when driving straight. On the contrary, PilotNet affords the car to stay in the middle of the wall when going straight, but it is trembling at the dead end and cannot turn around.

Prediction

Fig.47 illustrates two predictions where the image is labeled with the wrong speed due to Lidar algorithm flaws, but the CNN+LSTM model predicts it correctly. In the latter figure, the left picture presents the case where the robot should drive forward, but the image is labeled as backward, and the right picture illustrates the opposite case. These interferences increase the training difficulty, but models with image sequences manage a greater tolerance.

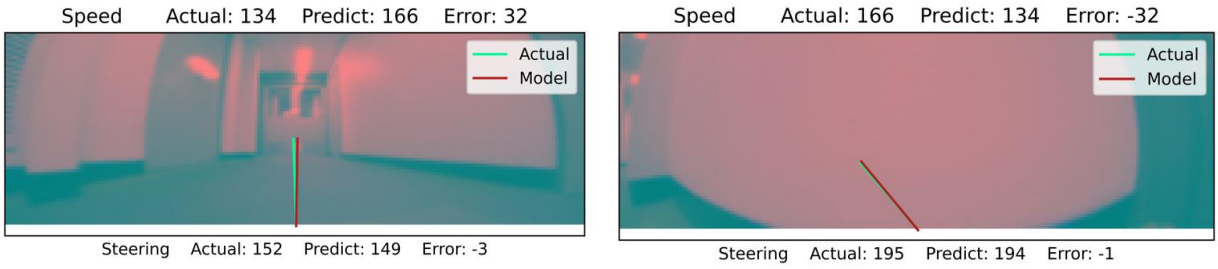

Figure 47 CME CNN+LSTM predictions

Saliency Map

All neural network models have successfully highlighted the walls' edges in the saliency map.

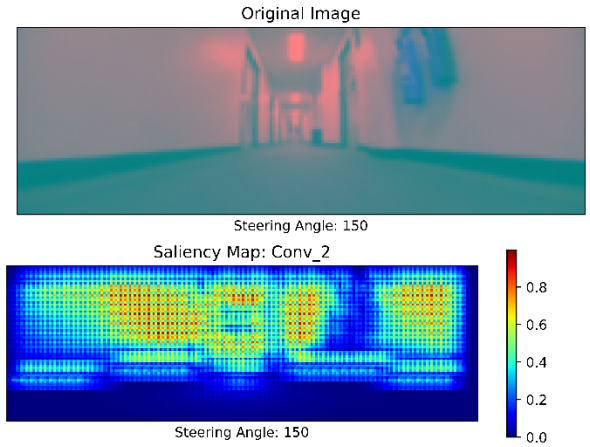

Figure 48 CME PilotNet saliency map

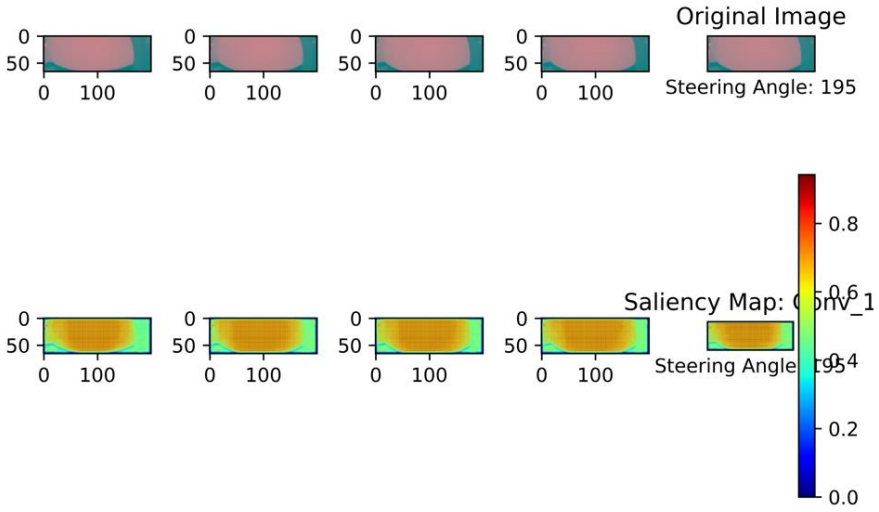

Figure 49 CME CNN+LSTM saliency map

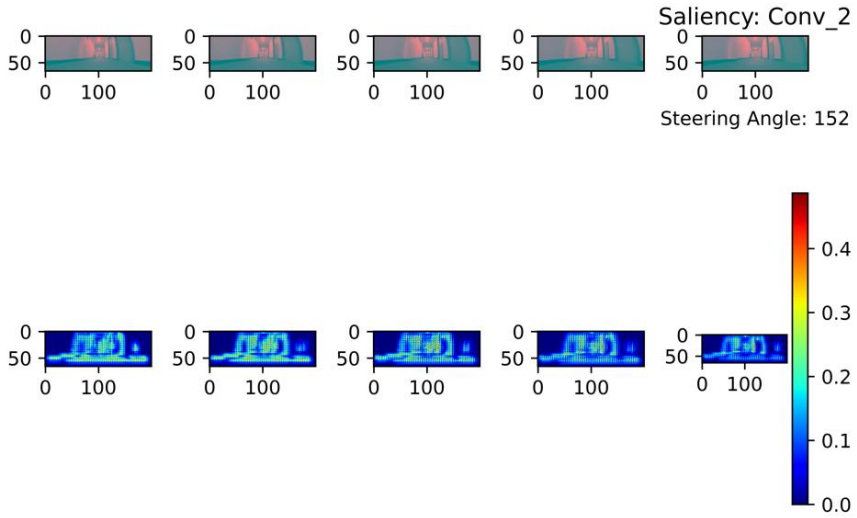

Figure 50 CME 3DCNN saliency map

5.3 Comparison of influencing factors

5.3.1 Data augmentation

Simulation

Table 10 Simulation models with (second) or without (first) data augmentation comparison.

Test \ Model	Sim_CNN+LSTM	Sim_CNN+LSTM_A
Training Loss (MSE)	**46.8248**	116.4086
Validation Loss (MSE)	20.433	**14.7844**
Speed Accuracy	14.71%	**24.09%**
Steering Accuracy	10.82%	**11.47%**

Other settings: Batch size=32, Training steps= (Number of training image/batch size), Validation steps= (Number of validation image/batch size). Training steps times three if data augmentation=True.

In the simulation experiment, data augmentation successfully reduces the validation loss of the model and improves its speed and steering accuracy. However, the actual performance of the two (with and without data augmentation) is not that different.

EECE rectangular loop

Table 11 EECE models with (second) or without (first) data augmentation comparison.

Test / Model	EECE _CNN+LSTM	EECE _CNN+LSTM_A	EECE_PilotNet	EECE_PilotNet_A
Training Loss (MSE)	**2.5974**	13.8415	**2.5347**	2.7068
Validation Loss (MSE)	**0.7065**	0.9791	**0.8791**	1.1525
Speed Accuracy	**82.89%**	79.82%	81.86%	**82.85%**
Steering Accuracy	**53.27%**	44.31%	**54.84%**	54.20%

Other settings: Batch size=32, Training steps= (Number of training image/batch size), Validation steps= (Number of validation image/batch size). Training steps times three if data augmentation=True.

In the EECE rectangular loop experiment, none of the data augmentations schemes impacted the model metrics, presenting a similar performance to not utilizing data augmentation.

CME corridor

Table 12 CME model with (second) or without (first) data augmentation comparison.

Test / Model	CME_CNN+LSTM	CME_CNN+LSTM_A
Training Loss (MSE)	**62.716**	414.071
Validation Loss (MSE)	**4.382**	241.3056
Speed Accuracy	**59.71%**	26.17%
Steering Accuracy	**28.69%**	10.76%

Other settings: Batch size=32, Training steps= (Number of training image/batch size), Validation steps= (Number of validation image/batch size). Training steps times three if data augmentation=True. **Data augmentation applied on 20% of data instead of 5% in a batch.**

In the CME corridor experiment, we applied excessive data augmentation, with the results indicating that excessive data augmentation preserves a high validation loss and does not afford the model to converge.

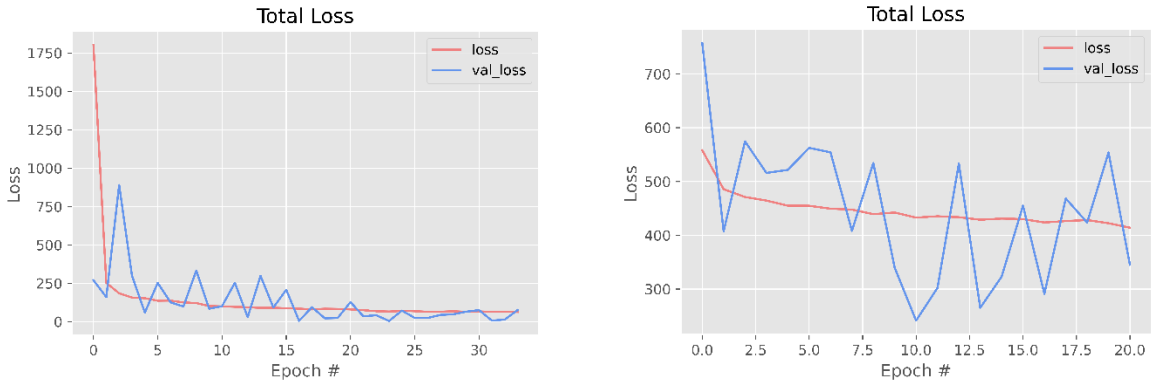

Figure 51 CME_CNN+LSTM model loss curve with (second) or without (first) data augmentation comparison.

5.3.2 Batch size

Table 13 EECE_CNN+LSTM with batch size 1024, 128, 32 (left to right) comparison

Test / Model	EECE_CNN+LSTM_B1024_A	EECE_CNN+LSTM_B128_A	EECE_CNN+LSTM_B32_A
Training Loss (MSE)	15.5774	14.5048	**13.8415**
Validation Loss (MSE)	6.2114	3.2392	**0.9791**
Speed Accuracy	59.64%	72.19%	**79.82%**
Steering Accuracy	25.46%	37.94%	**44.31%**

***** B stand for batch size, A stand for data augmentation.

Other settings: Data augmentation=True, Training steps=3*(Number of training image/batch size), Validation steps= (Number of validation image/batch size).

The debate on batch size has never been conclusive. Many researchers use large batch sizes because it improves the efficiency of GPU usage (parallel processing), but many experiments show that small batch sizes can produce good results in a shorter time [12]. In this experiment, the batch size has been adjusted many times, with Table 13 highlighting that a smaller batch size produces better models. This is reasonable, as the more significant the batch size, the more accurate the learning during each epoch, but the longer the learning time. The large batch size model will likely be stopped by early stopping before reaching the global optima. Although a model with a large batch size is more likely to produce the best results, the research time for this project is limited, and the goal of this experiment is not to reduce the validation loss but to attain the best real driving performance, so a small batch size is preferred.

5.3.3 Training and validation steps

Table 14 Simulation models with or without more training/validation steps comparison

Model / Test	Sim _CNN+LSTM	Sim _CNN+LSTM_ More_steps	Sim_ PilotNet	Sim_ PilotNet_ More_steps
Training Loss (MSE)	46.8248	**11.3484**	18.1957	29.1005
Validation Loss (MSE)	20.433	**4.4548**	9.2673	**8.1348**
Speed Accuracy	14.71%	**32.94%**	**25.20%**	24.28%
Steering Accuracy	10.82%	**32.78%**	**24.77%**	22.44%

Normal steps: Training steps= (Number of training image/batch size), Validation steps= (Number of validation image/batch size).

More steps: Training steps=3*(Number of training image/batch size), Validation steps=1.6*(Number of validation image/batch size).

Other settings: Batch size=32, Data augmentation=False

The model training uses the bagging method, and therefore data can be sampled multiple times within a batch. If (steps = size of dataset/batch size), then only 63.19% of the data will be sampled in an epoch, while if (steps =2*size of dataset/batch size) then 86.19% of the data will be sampled in an epoch, and if (steps =3*size of dataset/batch size) then 95.64% of the data will be sampled in an epoch.

More training/validation steps ensure the best model training performance, but the training time increases substantially. From Table 14, the PilotNet presents the best training performance because more training/validation steps do not significantly reduce the model validation loss. However, the validation loss of the CNN+LSTM model drops significantly with more training/validation steps. This also shows that different models require different steps, with a more complex model usually requiring more steps.

5.3.4 Memory capacity

The 61,197 images collected in the EECE rectangular loop occupy 14.1GB of memory. 14.1x5 (sequence length) = 70.5GB of memory is required to generate image sequences, but our computer has only 64GB, so we crop the images before creating the sequence (data preprocessing). The 320x340 RGB image is cropped to 200x66 YUV color space and is only 17% of its original size, so the final image sequences are only 12GB in total, which is within the computer's memory capacity.

The computer's memory size used to train the model determines the upper limit of the dataset, while more memory is needed to continue increasing the sequence length or sample size.

5.3.5 Quality of the dataset

Table 15 Practical models with different input data.

Test \ Model	EECE_ PilotNet_ 30k	EECE_ PilotNet _ 2x30k	CME_ CNN+LSTM	CME_ CNN+LSTM _ Data_cleaning
Training Loss (MSE)	39.6818	**3.0879**	62.716	**19.5104**
Validation Loss (MSE)	29.5848	**2.3128**	4.382	**1.637**
Speed Accuracy	15.64%	**81.67%**	59.71%	**72.17%**
Steering Accuracy	18.86%	**50.94%**	28.69%	**36.05%**

EECE other settings: Batch size=1024, Data augmentation=False, Training steps=300, Validation steps=200.

CME other settings: Batch size=32, Data augmentation=False, Training steps=1.7*(Number of training image/batch size), Validation steps=1.3*(Number of validation image/batch size).

The data quality plays a decisive role during training the neural network model. The data quality of the simulation experiment is outstanding because the number of images labeled as a left turn, straight and right turn is the same.

However, in reality, both datasets present several problems. The EECE PilotNet model is first trained with 30k images collected by Lidar driving counterclockwise, but the trained model is prone to turn left and hit the wall. After adding 30k images of clockwise driving with Lidar, the model enhances its driving stably. The CME CNN+LSTM model is first trained with 47k images collected by Lidar drive mode, but this Lidar algorithm has a 50% chance of doing left-turning and a 50% chance of doing right-turning when encountering a dead end. Thus, the trained model will tremble when it encounters a dead end because it cannot decide whether to turn left or right. After data cleaning (removing all the right-turning images), the model is trained on 43k images affording a smoother and faster turn at the dead end.

5.3.6 TensorFlow compatibility

Initially, this project employed TensorFlow 2.6.0, but due to the incompatibility of Raspberry Pi 4, TensorFlow 1.13.0 was used. TensorFlow 1 can be installed on Raspberry Pi directly with the “pip install tensorflow” command, but TensorFlow 2 needs to install additional plug-ins. The CNN+LSTM model trained on TensorFlow 1 has 13Hz FPS on Raspberry Pi, but only 3Hz when trained on TensorFlow 2. The accuracy of TF1 and TF2 are the same, but the training time of TF2 is only half of TF1, so if in the future the Raspberry Pi can be fully compatible with TF2, then TF2 is a better choice.

5.3.7 Speed decaying

The speed used in this project is not the real speed but the control command to the motor, which has no effect on the simulation but has a significant impact on the actual experiment. The actual

experiment uses the ServoBlaster API to send PWM signals to control the speed of the robot. With 100% power, the LIDAR autopilot takes only 50 seconds to complete the rectangular loop of the EECE, but when the power drops to 60% it takes 59 seconds to complete. The drop in speed makes the time interval between the collected images inconsistent, with the models exploiting image sequences as input are very sensitive to it, which largely affects the training results of these models.

The battery currently utilized is only 3000mAh, replacing it with a larger capacity battery eliminates the impact of insufficient power supply, but the speed still drops. The best way is to add an odometer to the robot, then employ it to get the actual speed and maintain the speed by a PID control algorithm.

5.3.8 Steering bias

In the actual experiment, the robot's steering is not perfectly balanced, and the manual driving mode reveals that the robot will slightly deviate to the left or right even under the command of straight ahead. Despite the neural network models are capable of correcting minor deviations during driving, only the Lidar algorithm can achieve 100% autonomy when the deviation is too large. Therefore, during the experiment, the front wheel screws need to be manually adjusted to keep the deviation within the tolerable range.

In fact, this hardware bias can be used to build a more diverse dataset for neural network training. The current Lidar driving mode stays in the middle of the road when driving straight, so the corrective behavior after the deviation is not included in the dataset. By adding hardware deviations, the neural network model has more samples for extreme situations to learn from, which will significantly increase the stability and reliability of the neural network model.

5.3.9 Frame rate

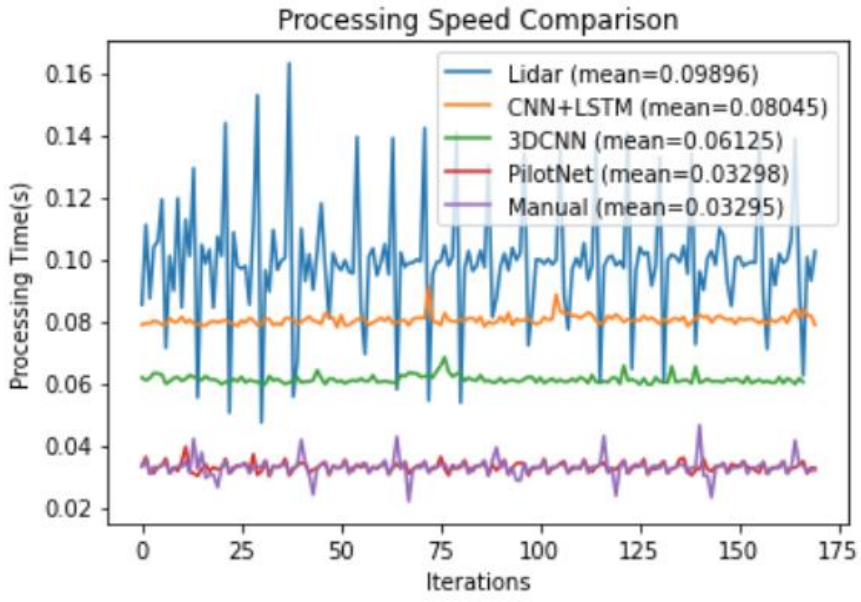

Figure 52 Processing speed of all driving modes and neural network models

Corresponding FPS from Fig 52: Manual FPS= 30.35Hz, PilotNet FPS= 30.32Hz, 3DCNN FPS=16.33Hz, CNN+LSTM FPS=12.43Hz, Lidar FPS=10.11Hz

The frame rate has a significant impact on the real-time robot driving performance. As the complexity of the model increases, the longer it takes for the model to predict. After comparing the input sequence length of the model, the number of neural network model layers, the TensorFlow version, and other factors, we find that the number of neural network model layers directly impacts FPS. The 3DCNN model of this experiment uses only three convolutional layers, while CNN+LSTM uses five convolutional layers and an additional LSTM layer. The gap in the number of layers imposes the FPS of the CNN+LSTM (12.43Hz) to be less than the 3DCNN (16.33Hz). Increasing the convolutional layer of 3DCNN to five layers afforded the same FPS as CNN+LSTM.

All neural network models have more minor fluctuations in processing than the Lidar model, which shows that the neural network models are more stable during operation. The stability will be a considerable advantage, especially when driving in high-speed and complex environments.

Without replacing the raspberry pi with a higher-performance microcontroller, reducing the number of model layers can improve the model's actual performance while not affecting the open-loop test results.

To replace the Raspberry Pi with a more powerful microcontroller, the jetson with GPU is a good choice. CPS Lab at UVA, as mentioned in the literature review, also used it to do experiments and achieved remarkable results in its F1/10 autonomous racing research[10]. In the simulation, the FPS of CNN+LSTM and 3DCNN models are the same, if the neural network model is equipped with a microcontroller with GPU, then the FPS will no longer be a problem, and I believe that the results will be as good as the simulation experiment.

[10] https://deepracing.ai/

6. Conclusion & Future Work

6.1 Conclusion

This research verifies the significance of End-to-end learning for Autonomous driving systems and the original PilotNet model improvement by adding LSTM or 3D convolutional layers. The improvements are:

The model can make more complex motions like turning around at a dead end: In the CME corridor experiment, the robot needs to make a series of moves to turn around at the dead ends. These actions are temporally sequenced, but PilotNet does not have access to temporal information, and these actions for it are like a single input corresponding to multiple outputs. Both CNN+LSTM and 3DCNN models solve this problem perfectly by virtue of their processing of temporal information.

Recovery from failures: In the corridor, the robot may not succeed in a single turnaround at a dead end, but the CNN+LSTM and 3D CNN models repeatedly try until they succeed while PilotNet stops there.

Lowering the MSE (Mean Square Error): In all experiments, the CNN+LSTM model achieved the best open-loop test performance. While the 3DCNN only had better open-loop test performance than PilotNet in the CME experiment, but this is because the 3DCNN tested only used three convolutional layers, if the 3DCNN uses the same number of convolutional layers as the other two models, it can achieve similar open-loop test performance as the CNN+LSTM model.

Suggesting an ideal route and a faster speed: In the simulation experiment, all neural network models run through a lap faster than the lidar model, with the CNN+LSTM and 3DCNN models being slightly faster than the PilotNet model. This indicates that the neural network models optimize the route by themselves, and the addition of LSTM layer or 3D convolutional layers helps to optimize.

It should be noticed that, in this experiment, the performance of the CNN+LSTM and 3DCNN models is limited by Raspberry Pi's computing power, as with low FPS, the model cannot turn in time if the speed is high. Additionally, the inconsistency of the gap between time steps due to speed decaying increases the difficulty of temporal information extraction.

Besides, the data quality is more important than the model structure. No useful information can be obtained regardless of the model’s performance if a dataset is affected by too much noise.

6.2 Future Work

Future work shall include the following:

Predicting waypoints using the Inertial Measurement Unit (IMU) to obtain future locations and optimize the path. IMU is necessary to obtain an accurate vehicle pose and comprises accelerometers, gyroscopes, and magnetometers that provide an attached object's angular rate,

acceleration, and orientation. This device uses a local coordinate system instead of a global coordinate system like GPS. GPS is unavailable for indoor applications, but IMU does not have this constraint. Using IMU-odometry fusion, we can derive the pose and direction of the attached vehicle, with MPU9250 being an excellent choice, as it is small, cheap, and compatible with Raspberry Pi.

Adding different types of input, i.e., past steering angles and speeds. One feature of Lidar drive mode that no model can currently emulate is braking. The robot in this project can simulate the braking effect by reversing the motor. Still, when using images with brakes as a dataset, even a model using image sequences as input cannot tell when to brake. The driver considers the immediate view and the speed and direction when judging the brakes, so adding past speed and steering can help the neural network model learn more complex driving behavior.

Exploiting optical flow and grayscale images as input instead of image sequences. Image sequences acquire temporal information but significantly increase the complexity of the neural network model and thus increase the processing time. In contrast, optical flow models can also acquire temporal information, but with lighter weight inputs and higher frame rates.

Training neural network models to park. The action of turning around at a dead end is similar to parking. It may be possible to learn parking actions for CNN+LSTM and 3DCNN models. Still, it is more challenging to learn only with the front camera because the driver usually needs to look at the scene behind the vehicle and judge when parking. Installing an additional camera at the robot's rear, and using images from multiple cameras as input can solve this problem.

7. References

[1] (2018). *Automated Vehicles for Safety*. [Online] Available: https://www.nhtsa.gov/technology-innovation/automated-vehicles-safety

[2] N. Mugunthan, S. Balaji, C. Harini, V. H. Naresh, and V. V. Prasannaa, "Comparison Review on LiDAR vs Camera in Autonomous Vehicle," *International Research Journal of Engineering and Technology (IRJET),* vol. 07, no. 08, 2020.

[3] B. Templeton. "Elon Musk's War On LIDAR: Who Is Right And Why Do They Think That?" https://www.forbes.com/sites/bradtempleton/2019/05/06/elon-musks-war-on-lidar-who-is-right-and-why-do-they-think-that/?sh=7a7b61142a3b (accessed 15 Apr. 2021.

[4] M. Bojarski *et al.*, "End to End Learning for Self-Driving Cars," 2016.

[5] M. R. Mollison, "High Speed Autonomous Vehicle for Computer Vision Research and Teaching," Bachelor of Engineering, School of Electrical, Electronic and Computer Engineering, The University of Western Australia, 2017.

[6] A. Ryan, "End-to-End Learning for Autonomous Driving Robots," School of Electrical, Electronic and Computer Engineering, The University of Western Australia, 2020.

[7] T. Weiss, V. SureshBabu, and M. Behl, "DeepRacing AI: Agile Trajectory Synthesis for Autonomous Racing," Department of Computer Science, University of Virginia, 2020. [Online]. Available: https://www.madhurbehl.com/newpubs/trent2020iros.pdf

[8] L. Bahl. "Speed estimation of car with optical flow." https://github.com/laavanyebahl/speed-estimation-of-car-with-optical-flow (accessed 6 Apr. 2021.

[9] A. BRAYLON. "Tesla could beat Waymo in the race to self-driving future." https://www.wheelsjoint.com/tesla-could-beat-waymo-in-the-race-to-self-driving-future/ (accessed 7 Apr. 2021.

[10] J. Klender. "Tesla CEO Elon Musk talks Level 5 Full Autonomy for 2021." https://www.teslarati.com/tesla-full-autonomy-2021/ (accessed 7 Apr. 2021.

[11] A. Serban, E. Poll, and J. Visser, "A Standard Driven Software Architecture for Fully Autonomous Vehicles," *Journal of Automotive Software Engineering* vol. Vol. 00(0), 2020, doi: https://doi.org/10.2991/jase.d.200212.001.

[12] A. Géron, *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow : Concepts, Tools, and Techniques to Build Intelligent Systems*, Second edition ed. O'Reilly Media, 2019.

[13] S. Hochreiter and J. Schmidhuber, "Long Short-term Memory," *Neural Computation,* vol. 9, pp. 1735-80, 1997, doi: 10.1162/neco.1997.9.8.1735.

[14] S. Ji, W. Xu, M. Yang, and K. Yu, "3D Convolutional Neural Networks for Human Action Recognition," *IEEE TRANSACTIONS ON PATTERN ANALYSIS AND MACHINE INTELLIGENCE,* vol. 35, 2013, doi: 10.1109/TPAMI.2012.59.

[15] N. E. Pour, "PATH PLANNING OF UNMANNED AERIAL VEHICLES USING PARTICLE SWARM OPTIMIZATION AND B-SPLINE CURVES," 2009. [Online]. Available: https://search-proquest-com.ezproxy.library.uwa.edu.au/docview/305136936?pq-origsite=primo. ProQuest Dissertations Publishing.

[16] M. E. Mortenson, *Mathematics for Computer Graphics Applications*. Industrial Press Inc., 1999.

[17] R. Wallace, A. Stentz, C. Thorpe, H. Moravec, W. Whittaker, and T. Kanade, "First Results in Robot Road-Following," 1985. [Online]. Available: https://www.researchgate.net/publication/220815976_First_Results_in_Robot_Road-Following.

[18] T. Weiss and M. Behl, "DeepRacing: Parameterized Trajectories for Autonomous Racing," 2020. [Online]. Available: https://arxiv.org/pdf/2005.05178.pdf.

[19] M. Bojarski *et al.*, "The NVIDIA PilotNet Experiments," 2020. [Online]. Available: https://arxiv.org/abs/2010.08776.
[20] F. Raudies, "Optic flow," in *Scholarpedia* ed, 2013.
[21] T. Weiss and M. Behl, "DeepRacing: A Framework for Autonomous Racing," 2020, doi: 10.23919/DATE48585.2020.9116486. IEEE.
[22] T. Bräunl, M. Pham, F. Hidalgo, R. Keat, and H. Wahyu, "EyeBot-7 User Guide," ed. University of Western Australia, School of Electrical, Electronic and Computer Engineering, 2020.
[23] E. V. Team. "Eyesim vr user's manual." https://robotics.ee.uwa.edu.au/eyesim/ftp/EyeSim-UserManual.pdf (accessed Oct. 22, 2021).
[24] E. Gedraite and M. Hadad, "Investigation on the effect of a Gaussian Blur in image filtering and segmentation," presented at the ELMAR, 2011 Proceedings, 2011.
[25] S. C. Wong, A. Gatt, V. Stamatescu, and M. D. McDonnell, "Understanding data augmentation for classification: when to warp?," Defence Science and Technology, Edinburgh, 2016. [Online]. Available: https://arxiv.org/abs/1609.08764.
[26] J. Mänttäri, S. Broomé, J. Folkesson, and H. Kjellström, "Interpreting video features: a comparison of 3D convolutional networks and convolutional LSTM networks," 2020. [Online]. Available: https://arxiv.org/abs/2002.00367.