# Autonomous lane navigation of a robotic car using deep learning models

Lakshmikar R. Polamreddy[1], Harsha Koduri[2], and Jatin Kayasth[3]

[1] Yeshiva University, New York City, NY 10016, USA,
lpolamre@mail.yu.edu
[2] Yeshiva University, New York City, NY 10016, USA,
hkoduri@mail.yu.edu
[3] Yeshiva University, New York City, NY 10016, USA,
jkayasth@mail.yu.edu

**Abstract.** Autonomous navigation has gained significant attention in recent years, with applications ranging from self-driving cars to robotic systems. In this project, we focus on constructing a physical robotic car capable of autonomously navigating lanes using a Raspberry Pi and deep learning techniques. We first collect training data by driving this robotic car on a lane and then utilize the state-of-the-art NVIDIA model architecture for predicting steering angle. Then, we develop our own Convolutional Neural Network(CNN) model with fewer parameters. After conducting extensive experiments, we realize that our model's performance is on par with that of the NVIDIA model in terms of $R^2$ value for a smaller dataset. In the case of a more diverse dataset, our model outperforms the NVIDIA model.

**Keywords:** Robotic car · deep learning models · Lane navigation · NVIDIA model · CNN model.

## 1 Introduction

Currently, a lot of big companies are all working on developing their autonomous vehicles that are legal to drive on public roads. According to several observers, fully autonomous cars will begin to operate in our cities within the next five years, and within thirty years, almost all automobiles will be entirely driverless. Vehicles having autonomous navigation systems, often referred to as self-driving cars or driverless cars, represent a transformative technology with the potential to revolutionize transportation. These vehicles leverage advanced technologies to navigate and operate without human intervention. The development and deployment of autonomous vehicles involve a convergence of various disciplines, including computer science, robotics, artificial intelligence, sensor technologies, and automotive engineering.

The remainder of the paper is organized as follows. Section 2 covers seminal contributions made in the deep learning field related to autonomous driving. Section 3 shows information about project setup with hardware and software

components. Section 4 explains steering angle calculation, loss function, and the deep learning model architectures. Section 5 presents the dataset and Section 6 compares performance of the models and their results. Section 7 discusses our data preparation, training process of the models and the loss plots. Section 8 discusses the challenges we face during this work. Section 9 captures concluding remarks of the proposed work and scope for future work.

## 2 Related Work

Bojarski et al. [1], as part of the NVIDIA research team, proposed a new convolutional neural network (CNN) architecture for end-to-end deep learning for self-driving cars. In the new automotive application, they used convolutional neural networks (CNNs) to map the raw pixels from a front-facing camera to the steering commands for a self-driving car. Zhenye-Na [5] also implemented NVIDIA model architecture for training self-driving vehicles using Udacity's simulation environment to generate training data and test the model. In addition, Smolyakov et al. [2] explored various CNN architectures to obtain better results with a minimum number of parameters for predicting the steering angles to drive the car in autonomous mode in the Udacity simulation environment. Tian [4] published articles related to building a DeepPiCar.

## 3 Project setup

Building an autonomous robotic car involves a combination of hardware and software components to enable perception, decision-making, and control. The following is the list of the key hardware components we use for our project setup.

### 3.1 Hardware components

**Raspberry Pi:** A Raspberry Pi serves as the brain of the robotic car. Its GPIO pins are utilized for connecting motor controllers and sensors. Raspberry Pi 3 Model B+ kit with 2.5A Power Supply, This is the brain of our robotic car. This latest model of Raspberry Pi features a 1.4 Ghz 64-bit Quad-Core processor, dual-band Wi-Fi, Bluetooth, 4 USB ports, and an HDMI port. It also comes with a power adapter, which we need to plug in for charging, and two chip heat sinks, which will prevent the Raspberry Pi CPU from overheating.

**USB drive:** This is where your Raspberry Pi's operating system and all of our softwares are stored. We use a 64 GB USB drive for storage purposes.

**SunFounder PiCar-V kit:** This is the main body of the robotic car. It comes with everything we need in a robotics car, except for the Raspberry Pi and the batteries. There are a number of Raspberry Pi car kits on the market, we chose

this car kit because it comes with an open source Python API to control the car, whereas other vendors have its proprietary API or C-based API. As we know, python is now the language of choice for machine learning and deep learning, and open source is important.

**Batteries:** We make use of 2× 18650 3.7V rechargeable batteries that are required for high-drain applications, such as driving the Raspberry Pi board and the robotic car.

**Camera module:** The camera that is compatible with Raspberry Pi comes with the SunFounder kit. It is a 120-degree wide-angle camera with a resolution of 640x320 pixels and is mounted on the front wheel zone to capture the lane video during the car's motion.

**Lane preparation:** We use blue tape to prepare a lane on the wooden floor for the car to navigate under proper lighting conditions. It is required for collecting training data and for testing the car in autonomous mode.

The physical robotic car and the prepared lane are shown in Fig. 1 and Fig. 2. We assemble the robotic car by following the instructions mentioned in the SunFounder document [3]

### 3.2 Software components

**Operating system:** Raspberry Pi OS (formerly Raspbian) is installed on the Raspberry Pi to provide a stable operating system for the project. It is tailored specifically for the Raspberry Pi's ARM architecture, ensuring optimal performance on these single-board computers. It includes a variety of pre-installed software, including the Chromium web browser, Python programming language, and educational tools like Scratch.

**Other software components:** We connect to Raspberry Pi terminal using Wi-Fi and utilize open source software PuTTY on our laptop for a convenient and remote terminal interface. We then install VNC on the laptop which allows us to remotely access Raspberry Pi desktop.

**Python virtual environment setup:** We create a virtual environment with Python 3.9 to provide an isolated environment for development and then install necessary libraries like OpenCV, Tensorflow, and others with compatible versions.

**Testing the software and hardware components:** We validate the working of the front and the rear wheels by controlling them through the pi terminal. In addition, we need to make sure that OpenCV is also working by visualizing the black and color images of the lane on our laptop.


Fig. 1: Robotic car with Raspberry Pi

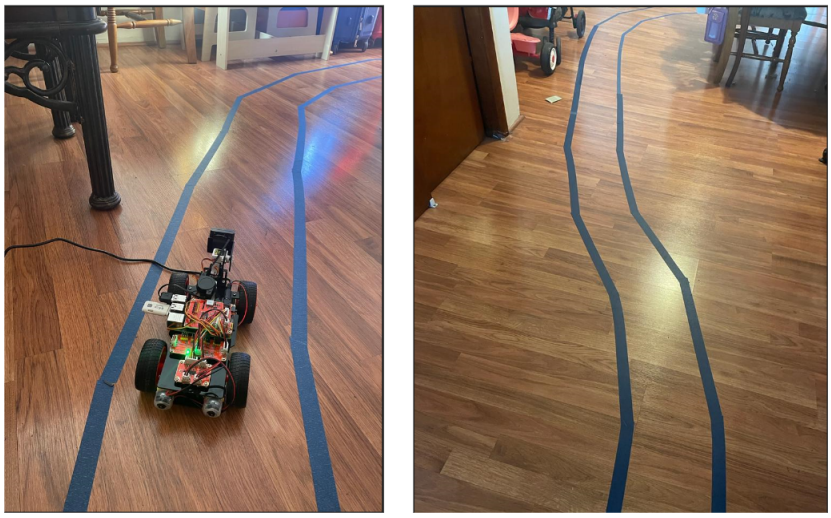

Fig. 2: Images of the lane on the wooden floor

## 4 Methods

### 4.1 Steering angle calculation

The steering angle is calculated based on the detected lane lines in a given frame using the OpenCV library. A sample of the detected track is shown in Fig. 3. If no lane lines are detected, it returns a default value of -90. If either one or two lane lines are detected, the deviation from the center of the lane is calculated. It uses trigonometric functions to determine the angle between the vehicle's navigation direction and the center of the lane as shown below [4]. The resulting steering angle is then adjusted to obtain the final value.

$$\text{mid} = \frac{\text{width of the frame}}{2}$$

$$x\_offset = \frac{\text{left\_x2} + \text{right\_x2}}{2} - \text{mid}$$

$$y\_offset = \frac{\text{height of the frame}}{2}$$

where $x\_offset$ refers to the distance between the center line and the detected track, and $y\_offset$ refers to half of the frame height.

$$\text{angle\_to\_mid\_radian} = \tan\left(\frac{x\_offset}{y\_offset}\right)$$

$$\text{angle\_to\_mid\_deg} = \text{angle\_to\_mid\_radian} \times \frac{180.0}{\pi}$$

$$\text{steering\_angle} = \text{angle\_to\_mid\_deg} + 90$$

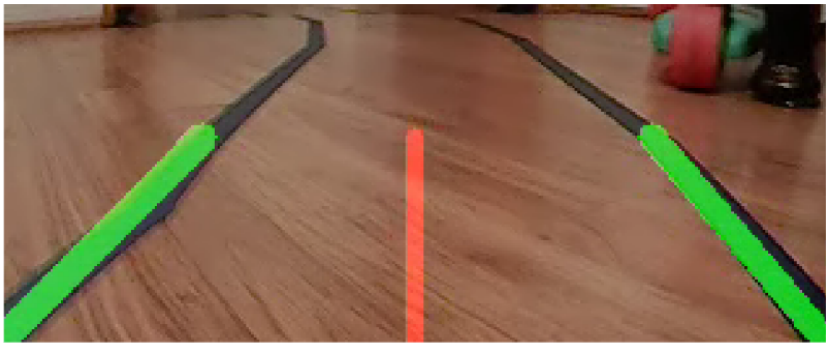

Fig. 3: A frame of the detected track using OpenCV

### 4.2 Loss function

Firstly, the images and corresponding steering angles are collected from the simulator in the training mode and passed onto the CNN model as inputs. Secondly, the CNN model is trained for a sufficient number of epochs till the convergence in loss is achieved. Thirdly, the outputs from the CNN model, i.e., steering angles, are passed onto the simulator in an autonomous mode. Then the car drives on its own on the selected track in the simulator.

As the model is expected to predict the steering angles, we consider the regression loss calculated using the equation below.

$$\text{Regression Loss} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2, \tag{1}$$

where $y_i$ is the actual steering angle and $\hat{y}_i$ is the predicted steering angle.

### 4.3 Model Architecture

Our CNN model consists of four convolutional layers, four max-pooling layers, two dropout layers, and two fully connected layers, with one being an output layer, as shown in Fig. 4. We use 3×3 kernels for the first three convolutional layers and 5×5 kernels for the last convolutional layer. This model has a total of 208,481 training parameters.

On the other hand, the model developed using the NVIDIA architecture [1] consists of five convolutional layers, one dropout layer, and four fully connected layers, with one being an output layer, as shown in Fig. 5. The total training parameters of this model are 252,219.

### 4.4 Implementation details

Both of these models are trained based on the following hyperparameters:

1. Number of epochs: 20

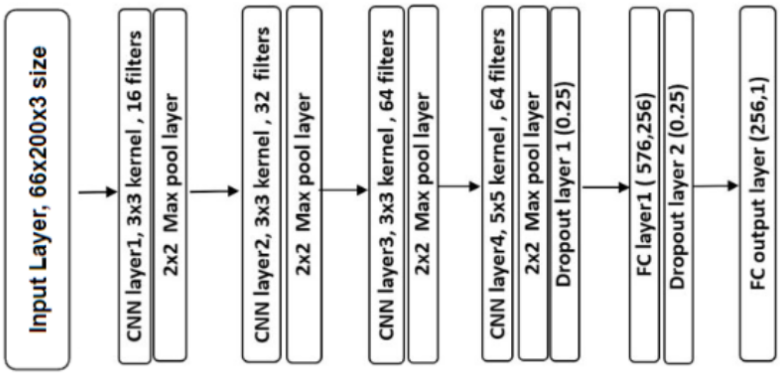


Fig. 4: Our CNN model architecture

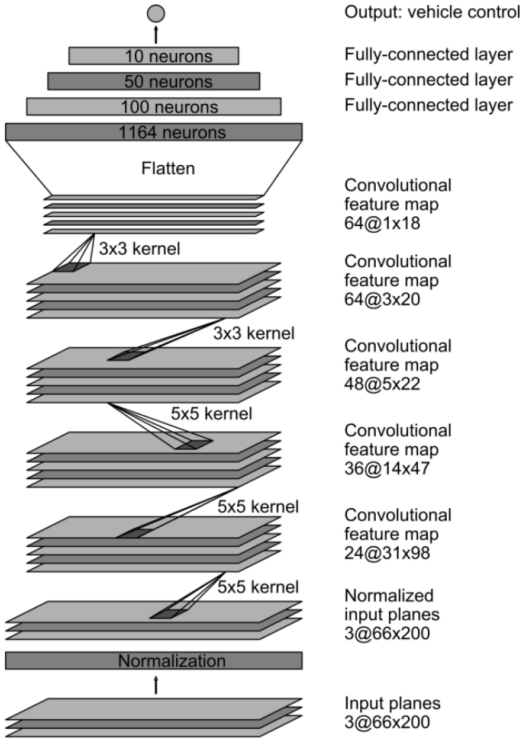


Fig. 5: NVIDIA model architecture [1]

2. Number of steps per epoch: 300
3. Batch size: 100
4. Optimizer: Adam
5. Learning rate: 0.001

## 5 Datasets

We create our dataset by driving the car on the prepared lane using OpenCV. We utilize the Pi camera mounted on the vehicle to capture video of the lane. The car is able to successfully navigate on the lane as it detects lanes and computes steering angles. Images are then extracted from these videos which serve as the training data for the model. The steering angle values are included in the names of the images as the last 3 characters. This dataset is used for training and testing our deep-learning models. The size of the dataset is as shown in Table. 1. We record three videos of the same lane to make our data more diverse. We first use the images extracted from the first video and then train the model to observe the result. Then, we combine the first two videos to check the performance and then with all three videos together.

Table 1: Dataset size

| Video | Total number of images | Train data | Test data |
|---|---|---|---|
| Video 1 | 448 | 358 | 90 |
| Video 1 and Video 2 | 807 | 701 | 106 |
| Video 1, Video 2 and Video 3 | 1321 | 1056 | 265 |

## 6 Results

Our results are consolidated and shown in the Table. 2 and Table. 3. We obtained these results after training the models for 20 epochs and then testing with 20 percent of the data from the images extracted from videos 1, 2, and 3. We notice that both models have shown similar performance in terms of $R^2$ when trained with the images of videos 1 and 2. Our CNN model has outperformed the NVIDIA model when combined with images of all three videos. However, both models show relatively poor results in the latter case. It is due to the reason that the loss did not converge in 20 epochs and so, we further train the model for 50 epochs and these results are shown in the Table.Table. 4.

## 7 Discussions

We first collect the images of size 320x240x3 from videos 1, 2, and 3 that serve as training data. Then, we augment the data for better diversity using techniques

Table 2: Performance of the models in terms of MSE(Mean Squared Error) with 20 epochs

| Model | Video 1 | Videos 1,2 | Videos 1,2,3 |
|---|---|---|---|
| NVIDIA model | 5.5 | 6.6 | 36 |
| Our CNN model | 7.7 | 8.6 | 16 |

Table 3: Performance of the models in terms of $R^2$ with 20 epochs

| Model | Video 1 | Videos 1,2 | Videos 1,2,3 |
|---|---|---|---|
| NVIDIA model | 93.4 | 94.2 | 65.2 |
| Our CNN model | 92.9 | 93.3 | **80** |

like zoom, pan, adjusting brightness, crop, random flip, and blur. After training both the models for 20 epochs with the images from videos 1 and 2, we observe that the loss converges. When the model is trained with the images from video 1, the loss plots look as shown in Fig. 6 and Fig. 7.

## 8 Challenges

The challenges that we face during this project are listed below.

1. Setting up the front wheels: After physically assembling the car with Raspberry Pi, we noticed that the front wheels did not work properly. we had to dismantle the car and re-assemble by ensuring that the servos positions were properly aligned.
2. Installation of OpenCV and Tensorflow on Raspberry Pi: Raspberry Pi supports only particular versions so, could not successfully install these libraries and dependencies on our first attempt.
3. Driving the car autonomously using OpenCV: Edge detection of lanes is challenging sometimes due to lighting conditions. When the speed of the car is faster, it is again challenging so we try different speeds. In addition, a steering angle calculation task is also required for the car to drive on the track.
4. Failure of SD card: We initially installed all software and libraries including Raspberry Pi but it failed to work for no reason. So, we later used a USB drive to install everything from the start.

Table 4: Performance of the models in terms of $R^2$ with 50 epochs

| Model | Training videos | MSE | $R^2$ |
|---|---|---|---|
| NVIDIA model | 1,2,3 | 28 | 65.6 |
| Our CNN model | 1,2,3 | 10 | **90.6** |

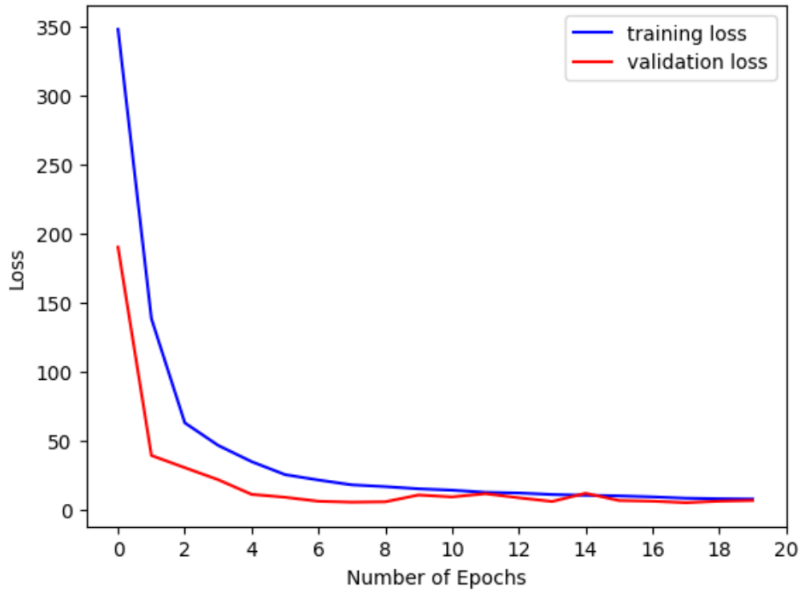


Fig. 6: Loss versus epochs for NVIDIA model when trained with images of video 1

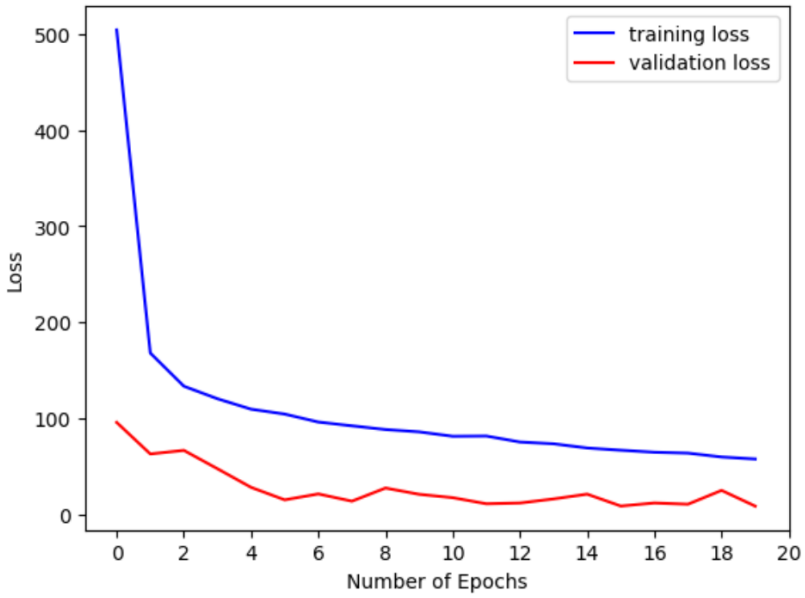


Fig. 7: Loss versus epochs for Our CNN model when trained with images of video 1

5. Limited training data: As we built a track of just 10 meters on the wooden floor using tape, we could generate only a smaller dataset. So, we have taken multiple videos on the same track.

## 9 Conclusion

In this project, we first build a physical car and prepare a lane on the wooden floor for navigation. We then make use of OpenCV to drive the car autonomously by detecting the lanes and recording these videos. Images are then extracted from these videos for the training data. After training with the NVIDIA model and our own CNN model, we conclude that our CNN model with fewer parameters demonstrated similar performance to that of the NVIDIA in terms of $R^2$. Our future work aims to test the performance of these models in terms of autonomous lane navigation by deploying them on the physical robotic car. In addition, we want to focus on testing the performance of these models by generating more training data on multiple tracks.

## References

1. Bojarski, M., Del Testa, D., Dworakowski, D., Firner, B., Flepp, B., Goyal, P., Jackel, L.D., Monfort, M., Muller, U., Zhang, J., et al.: End to end learning for self-driving cars. arXiv preprint arXiv:1604.07316 (2016)
2. Smolyakov, M., Frolov, A., Volkov, V., Stelmashchuk, I.: Self-driving car steering angle prediction based on deep neural network an example of carnd udacity simulator. In: 2018 IEEE 12th international conference on application of information and communication technologies (AICT). pp. 1–5. IEEE (2018)
3. SunFounder: Sunfounder picar-v smart car kit for raspberry pi (2019), `https://docs.sunfounder.com/projects/picar-v/en/latest/`
4. Tian, D.: Deeppicar series (2019), `https://towardsdatascience.com/deeppicar-part-1-102e03c83f2c`
5. Zhenye-Na: End-to-end learning for self-driving cars — udacity's simulation env (2019), `https://github.com/Zhenye-Na/e2e-learning-self-driving-cars`