Received Date:11th January, 2025
Revision Date: 15th March, 2025
Accepted Date:3rd April, 2025

End-to-End Steering Prediction and Object Detection for Self-Driving Car using Machine Learning

Binaya Dhakal [1*], Ankit Mallik[2], Sandesh Khanal[3], Shyam Tamang[4], Gaurav Gautam[5], Smriti Nakarmi[6]

[1] Department of Electronics, Communication and Information Engineering, Kathmandu Engineering College. E-mail: binayadhakal15@gmail.com

[2] Department of Electronics, Communication and Information Engineering, Kathmandu Engineering College. E-mail: mallikankit00@gmail.com

[3] Department of Electronics, Communication and Information Engineering, Kathmandu Engineering College. E-mail: khanal.sandesh.012@gmail.com

[4] Department of Electronics, Communication and Information Engineering, Kathmandu Engineering College. E-mail: shyamt273@gmail.com

[5] Associate Professor, Department of Electronics, Communication and Information Engineering, Kathmandu Engineering College. E-mail: gaurav.gautam@kecktm.edu.np

[6] Associate Professor, Department of Electronics, Communication and Information Engineering, Kathmandu Engineering College. E-mail: smriti.nakarmi@kecktm.edu.np

Abstract — In general, Self-Driving car uses a combination of numerous sensors, cameras, radars, LIDAR, high-performance processors and artificial intelligence (AI) technology to travel between destinations without the need of a human operator. This research utilizes a Raspberry pi 4 (2GB) as the processing element and a camera module connected to it acts as the input for the car. Machine learning models are used for lane detection, steering angle prediction, and object detection, focusing on traffic light and stop sign detection, to make various decisions for the car. Raspberry pi is the main part of the system with the task of processing the video provided by the camera by extracting individual image frames, detecting the road lanes and the objects in front of it, and finally taking various decisions for the car such that the car follows the lane lines and doesn't go off-track, maintains appropriate speed and turning angles, and follows traffic rules by recognizing the status of traffic lights and detecting stop sign. For the lane detection and steering angle prediction model, lane images were collected by manually controlling the car through the track lane. Those thousands of lane images were transferred to the computer, and after pre-processing, the model was trained using Google Colab. Similarly, for the object detection model, images were collected for the required classes and then used, to train a pre-trained YOLOv8 model for detecting only the few required classes of object. Finally, both the machine learning models were saved in Raspberry Pi. In the inferencing process, the latest image frame from the camera was fed into the raspberry pi after conversion into appropriate format as required by the steering prediction model (YUV Format) and object detection model (RGB Format). After interpretation of the output of these models, the decision for the car movement was sent to the motor driver module which then controlled the rotation and speed of the Battery-Operated (BO) motors. In the end, the car prototype successfully followed the predefined lane tracks and detected objects within its view, achieving near real-time inferencing at low speeds. In conclusion, this paper presents the integration of two machine learning models to develop a miniature self-driving car prototype within a resource-constrained environment capable of autonomously navigating road lanes and making precise decisions.

Keywords — Self-driving, Raspberry pi, Machine learning, Lane detection, Object detection, Traffic light detection, Stop sign detection, Google Colab, YOLOv8, Near real-time inferencing

Introduction

Background Theory

A self-driving car (sometimes called an autonomous car or driverless car) is a vehicle that uses a combination of sensors, cameras, radar and artificial intelligence (AI) to travel between destinations without a human operator. To qualify as fully autonomous, a vehicle must be able to navigate without human intervention to a predetermined destination over roads that have not been adapted for its use. An autonomous car can go anywhere a traditional car goes

* *Corresponding Author*

and do everything that an experienced human driver does. These cars use vast amount of data for machine learning and neural networks, to build the system that can drive by itself.

Autonomous cars rely on sensors, actuators, complex algorithms, machine learning systems, and powerful processors to execute software. It creates and maintains a map of their surroundings based on a variety of sensors situated in different parts of the vehicle. Radar sensors monitor the position of nearby vehicles. Video camera detects traffic lights, read road signs, track other vehicles, and look for pedestrians. Lidar sensors bounce pulses of light off the car's surroundings to measure distances, detect road edges, and identify lane markings. Ultrasonic sensors in the wheels detect curbs and other vehicles when parking. Sophisticated software then processes all this sensory input, plots a path, and sends instructions to the car's actuators, which control acceleration, braking, and steering. Hard-coded rules, obstacle avoidance algorithms, predictive modelling, and object recognition help the software follow traffic rules and navigate through obstacles.

In the race to design vehicles that are fully independent of drivers, the automotive industry looks to the Society of Automotive Engineers' (SAE) system of categorization that defines the six levels of automated driving[1] as follows:

- Level 0 (No Automation): The human driver is in constant and complete control of the car.
- Level 1 (Driver Assistance): System provides continuous assistance with either acceleration/braking OR steering, while driver remains fully engaged and attentive. Cruise control, lane keeping and parking assist are other such common place features found in autonomous car of this level.
- Level 2 (Additional Assistance): More than one function is automated at the same time such as a combination of adaptive cruise control and lane centering. However, the driver must still remain constantly attentive.
- Level 3 (Conditional Automation): System actively performs driving tasks while driver remains available to take over. If the system can no longer operate and prompts the driver, the driver must be available to resume all aspects of the driving task.
- Level 4 (High Automation): System is fully responsible for driving tasks within limited-service areas while occupants act only as passengers and do not need to be engaged.
- Level 5 (Full Automation): The car can completely drive itself without a human operator. The vehicle is designed to perform all driving function and monitor roadway conditions for an entire trip.

Problem Statement

Self-driving technology relies on accurate lane detection and object recognition to navigate safely and follow traffic rules. However, most autonomous systems require high-performance hardware, making them costly and, implementing such technology in a resource-constrained environment remains a significant challenge.

This research focuses on developing a low-cost miniature self-driving car prototype using a machine learning-based approach on a Raspberry Pi 4 (2GB). The key challenge is achieving real-time lane detection and steering prediction, and object recognition while ensuring efficient processing on limited hardware. The system must accurately detect lane markings, identify critical traffic signs such as stop signs and traffic lights, and make appropriate driving decisions. By addressing these challenges, this study demonstrates the feasibility of implementing machine learning-based steering prediction and object detection on low-power hardware, contributing to research and education in autonomous vehicle technology.

Objectives

- To build a miniature car-based prototype that can navigate through road lanes accurately.
- To detect and identify object in front of the car, and take decisions accordingly.
- To integrate and operate two machine learning models on a Raspberry Pi while minimizing inference time.

A. Scope and Applications

The scope of this paper involves developing a self-driving car prototype using a Raspberry Pi and a camera module that can autonomously follow the marked lanes and detect objects in front of it, by integrating two machine learning models while achieving high accuracy.

This paper serves as a low-cost implementation for learning and research in lane detection and object detection using machine learning. It enables students and researchers to explore computer vision techniques for identifying lane markings and detecting critical road signs, such as stop signs and traffic lights by implementing machine learning models on a resource-constrained system. Its concepts can also be applied to smart robotic systems, such as warehouse automation, where small autonomous vehicles follow designated paths to transport goods efficiently.

Literature Review

This paper proposed an autonomous car that was capable of sensing its environment and navigating without human input using components like Camera, Ultrasonic sensor, Raspberry pi, and GPS module. It consisted of three subsystems:

(i) Client systems, such as hardware platform;

(ii) Algorithms for localization, perception, and planning and control; and

(iii) The cloud platform, which included data storage.

Using the cloud platform, better recognition, tracking, and decision models were trained and tested. [2]

This study produced a successful model that navigates through the traffic and tackles all the obstacles placed in its path. This self-drive car was implemented using Machine Learning algorithms and Convolutional neural networks in a way that provides complete automation to the driver. This car integrated and performed the following functions: self-park, avoid traffic congestion, follow lane markings, avoid obstacles and detect vehicles and moved accordingly based on its distance from other vehicles.[3]

This paper discussed the use of CNN deep learning algorithm for recognizing the surrounding environment in creating the automatic navigation required by autonomous cars. The system designed creates and learns the data set taken in advance from Udacity Self-driving car simulator and the learning outcomes were implemented in an open simulation system. This simulation showed high accuracy in learning to navigate the autonomous car by observing the surrounding environment.[4]

In this paper, a monocular vision-based self-driving car prototype using Deep Neural Network on Raspberry Pi was proposed. First, the CNN model parameters were trained by using data collected from vehicle platform built with a 1/10 scale RC car, Raspberry Pi 3 Model B computer and front facing camera. The training data were road images paired with the time-synchronized steering angle generated by manually driving. Second, road test the model on Raspberry to drive itself in the outdoor environment around oval-shaped and 8-shaped with traffic sign lined track. The experimental results demonstrated the effectiveness and robustness of autopilot model in lane keeping task. Vehicle's top speed was observed about 5-6km/h in a wide variety of driving conditions, regardless of whether lane markings are present or not.[5]

This paper proposed a working model of self-driving car which was capable of driving from one location to the other or to say on different types of tracks such as curved tracks, straight tracks and straight followed by curved tracks. A camera module was mounted over the top of the car along with Raspberry Pi sent the images from real world to the Convolutional Neural Network which then predicted one of the following directions i.e. right, left, forward or stop which was then followed by sending a signal from the Arduino to the controller of the remote controlled car and as a result of it the car moved in the desired direction without any human intervention.[6]

This paper addressed the problem of using GPS in the roads of India for implementation of a self-driving car. So the idea was to use special pattern deployed on the road and those pattern were used for detection of pathway and type of road. The prototype used a modelled car which has a Raspberry pi to process the captured images from the camera and send it remotely on remote computer process it and send back. Similarly, various sensors were used around the car to detect the surrounding obstacles. The camera captured the specific pattern on the road that made it easier to drive on roads in India.[7]

This paper proposed a convolutional neural network (CNN) approach to implement a level 2 autonomous vehicle by mapping pixels from the camera input to the steering commands. The network automatically learns the maximum variable features from the camera input, hence requires minimal human intervention. Given realistic frames as input, the driving policy trained on the dataset by NVIDIA and Udacity could adapt to real-world driving in a controlled environment. The CNN was tested on the CARLA open-source driving simulator. Arduino Mega and Raspberry Pi are used for motor control and processing respectively to output the steering angle, which was converted to angular velocity for steering. In case of obstruction in the path, three ultrasonic sensors were used to decide in which direction the vehicle should turn to continue on its path. Once this was achieved, the vehicle resumed its normal function in terms of manoeuvring based on the steering angle given by the CNN.[8]

For Object Detection, YOLO used a totally different approach. It applied a single neural network to the full image. This network divided the image into regions and predicted multiple bounding boxes and probabilities for each region. These bounding boxes were weighted by the predicted probabilities. This model had several advantages over classifier-based systems. It looked at the whole image at test time so its predictions were informed by global context in the image. It also made predictions with a single network evaluation unlike systems like R-CNN which required thousands for a single image. This made it extremely fast, more that 1000x faster than R-CNN and 100x faster than Fast R-CNN.[9]

HydraNet was introduced by Ravi Teja and team in 2018. It was developed for semantic segmentation, for improving computational efficiency during inference time. HydraNet is dynamic architecture so it can have different CNN networks, each assigned to different tasks. These blocks or networks were called branches. The idea of HydraNet was to get various inputs and feed them into a task-specific CNN network. Take the context of self-driving cars. One input dataset can be of static environments like trees and road-railing, another can be of the road and the lanes, another of traffic lights and road, and so on. These inputs were trained in different branches. During the inference time, the gate chose which branches to run, and the combiner aggregates branch outputs and makes a final decision. In the case of Tesla, they modified this network slightly because it was difficult to segregate data for the individual tasks during inference. To overcome that problem, engineers at Tesla developed a shared backbone. The shared backbones were usually modified ResNet-50 blocks. This HydraNet was trained on all the object's data. There were task-specific heads that allow the model to predict task-specific outputs. The heads were based on semantic segmentation architecture like the U-Net. The Tesla HydraNet could also project a birds-eye, meaning it could create a 3D view of the environment from any angle, giving the car much more dimensionality to navigate properly. On ImageNet, applying the HydraNet template improved accuracy up to 2.5% when compared to an efficient baseline architecture with similar inference cost.[10]

This paper proposed a RNN-based neural network named as ChauffeurNet, used by Google Waymo, however, CNN was actually one of the core components here and used to extract features from the perception system. The idea behind this network was to train a self-driving car using imitation learning. The authors decided to train the network by adding synthetic data, along with original data to create variations. This synthetic data introduced deviations such as introducing perturbation to the trajectory path, adding obstacles, introducing unnatural scenes, etc. They found that such synthetic data was able to train the car much more efficiently than the normal data. Usually, self-driving has an end-to-end process where the perception system is part of a deep learning algorithm along with planning and controlling. In the case of ChauffeurNet, the perception system was not a part of the end-to-end process; instead, it was a mid-level system where the network could have different variations of input from the perception system.[11]

Methodology

Block Diagram

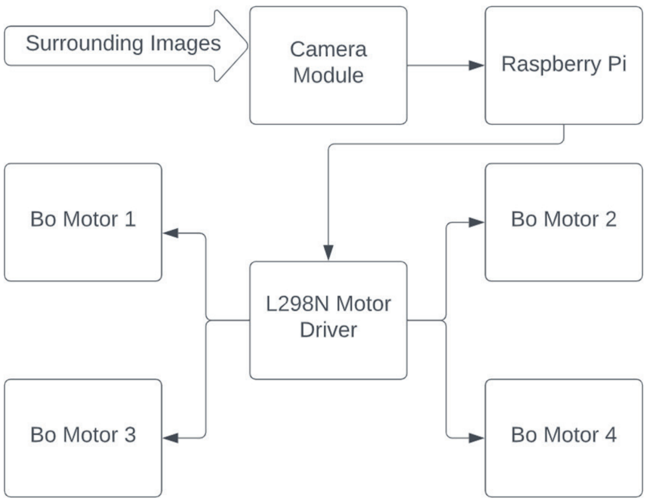


Fig. 1 Block Diagram

The self-driving car consists of components such as the Raspberry Pi, camera module, motors, and motor driver, all connected as illustrated in Fig. 1. The Raspberry Pi, a single-board computer, processes the images from the camera module and computes the necessary adjustments for the motors. Four BO motors are connected to the L298N motor driver, which sends control signals to the motors.

Steering Angle Prediction

1) *Data Collection:* In the track lane prepared over a plywood, collection of images with steering angle data was done. The car was manually controlled by using a joystick and the camera module was used to capture the images in real time. The raspberry pi stored the captured images inside a folder and the location of images along with corresponding steering values (from -1 to +1) were recorded in a comma-separated values (CSV) file. Finally, the collected images and the CSV files were uploaded to computer for training the model.

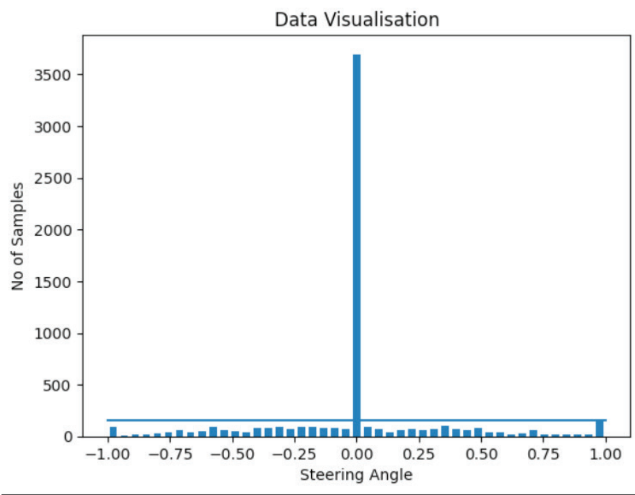
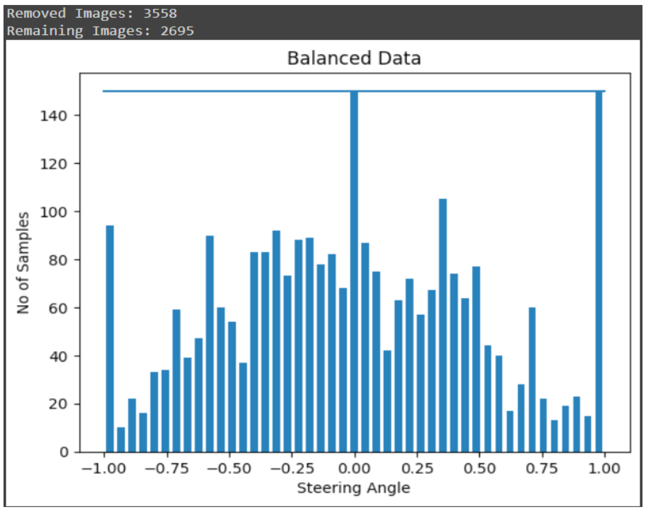


Fig. 2 Histogram Plot

Data Balancing: At first, the collected data was uploaded to Google Colab for training and building the model. Then, reading of data was done using pandas to manipulate and analyse the data of CSV file. A histogram was plotted to visualize the distribution and see what steering angles were most frequent throughout the dataset.



```
Removed Images: 3558
Remaining Images: 2695
```

Fig. 3 Balanced Data

Horizontal axis represents the normalised steering angles and the Vertical axis represents the frequency of the steering angle used while collecting data. Plotting this histogram of training data showed that the data obtained from driving in the track had more zero angles because of the nature of track. Also, the left and right angle data seemed to be somewhat balanced which was a major requirement for the data. Biased data is not preferred, therefore, to solve this issue, reject all the samples above a certain threshold and make sure that the data is more uniformed and not biased towards a specific steering angle.

1) *Data Pre-processing:* The next step was to process the original image such that extraction of steering angle was simplified. The image was cropped so that unwanted areas of the image were removed such as the background and the hood of the car. As the NVIDIA Model architecture suggested using YUV colour format, conversion of original image from RGB to YUV colour format was done. Next gaussian blur function was applied to smoothen the image and remove noise.

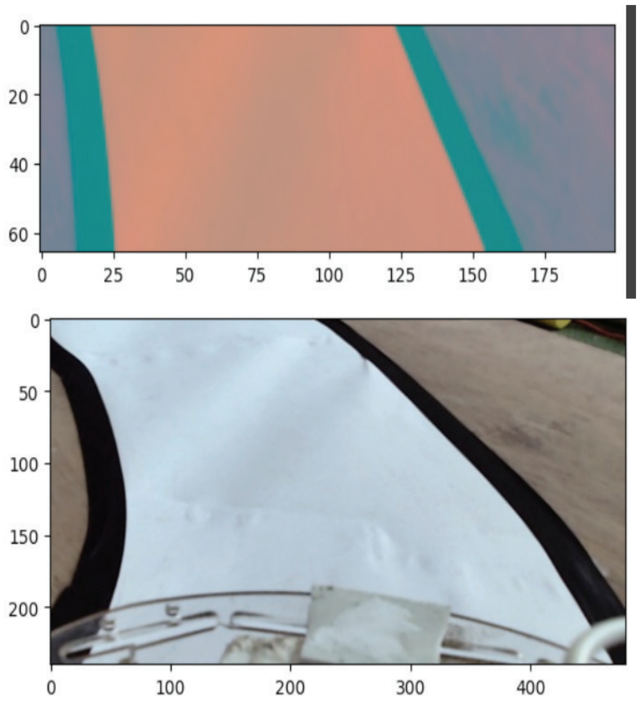


Fig. 4 Lane Image Pre-processing

Finally, the lane images were suitable for feeding into the NVIDIA CNN architecture[12], which consisted of 9 layers, including a normalization layer, 5 convolutional layers, and 3 fully connected layers. The input image was split into YUV planes and passed to the network. The first layer of the network performed image normalization. The normalizer was hard-coded and thus, was not adjusted in the learning process. The convolutional layers were designed to perform feature extraction, and were chosen empirically through a series of experiments. The fully connected layers were designed to function as a controller for steering, but it was noted that by training the system end-to-end, it was not possible to make a clean break between which parts of the network function primarily as feature extractor, and which serve as controller. The final output of the model was the predicted normalised steering angle value.

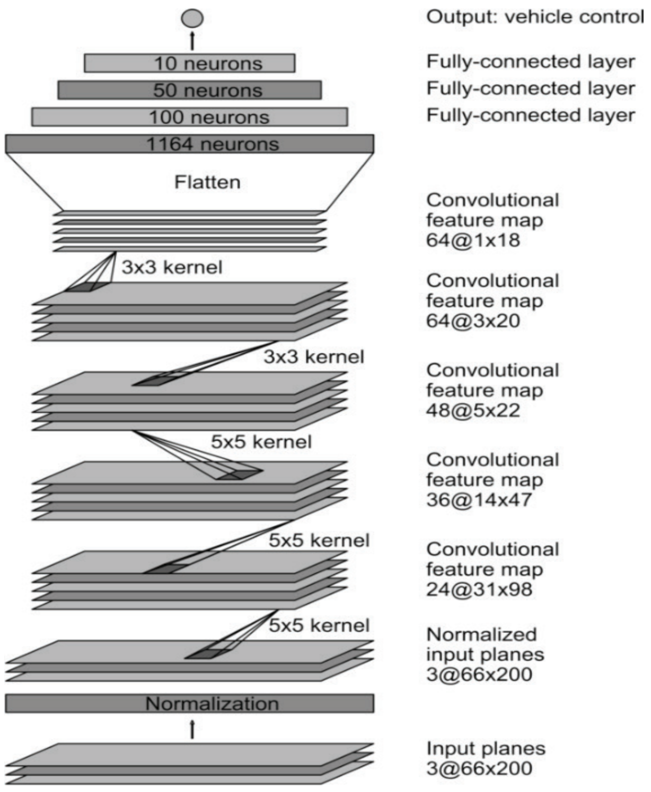


Fig. 5 CNN architecture

Object Detection

1) *Data Collection:* Firstly, the image data for the various classes i.e. traffic lights (red and green) and stop sign from Google were collected and the unwanted and unclear images were filtered out.

2) *Data Augmentation:* Then, the original image data was augmented to create new, modified samples of images by applying various transformation to artificially increase the diversity and size of a training dataset. Image was resized to 640*640 for feeding into YOLO v8 model and different transformation such as flipping, rotation, zoom, crop and resize, brightness and contrast adjustment, noise addition were applied. Such transformations in the dataset were done with the help of a python library 'Albumentations'.

3) *Data Annotation:* After creating a larger dataset of about 2700 images, it was then split into train (80%) and test (20%) set with each folder that would contain different set of images. Then the objects to be detected were labelled in every image using 'Makesense.ai' tool which is free and open source, and easy to use. Annotation file of YOLO format was generated for each image and stored in labels folder.

Finally, the images were ready for training and then fed to the pre-trained YOLO v8n (Nano) object detection model. The reason for using a pre-trained model is due to the fact that the model can capture complex features easily and allows model to leverage pre-trained weights, significantly reducing computational time, with increased accuracy even in smaller datasets. This technique is known as transfer learning where a pre-trained deep learning model is adapted to a new task by fine-tuning it on a smaller dataset. Transfer learning helps prevent overfitting by retaining generalized features, making the model more robust to variations in the new dataset. It also optimizes computational resources, enabling efficient model adaptation without the need for extensive hardware.

Flowcharts

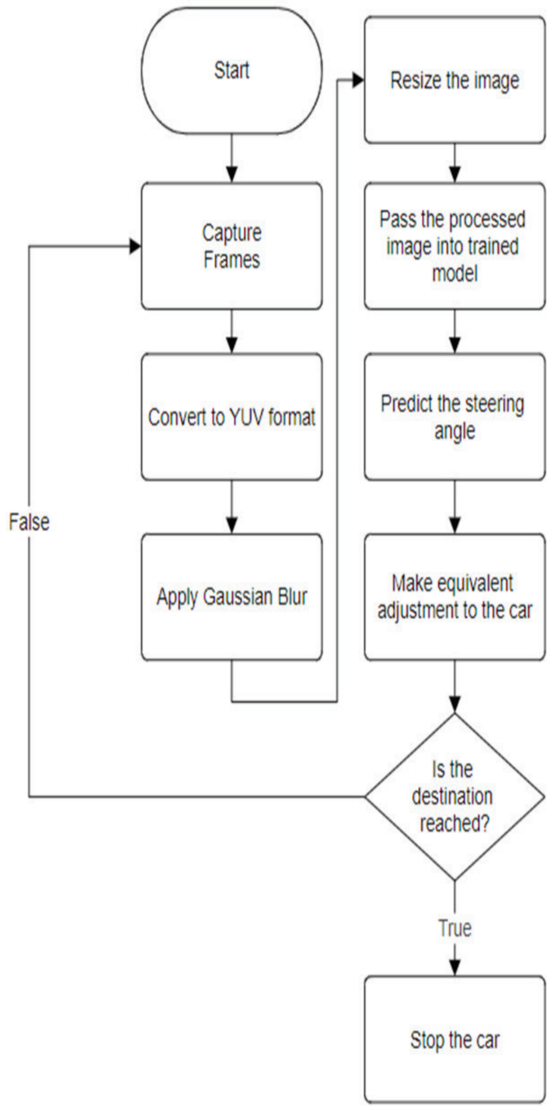

Start
Capture Frames
Convert to YUV format
Apply Gaussian Blur
Resize the image
Pass the processed image into trained model
Predict the steering angle
Make equivalent adjustment to the car
Is the destination reached?
False
True
Stop the car

Fig. 6 Flowchart for steering angle prediction

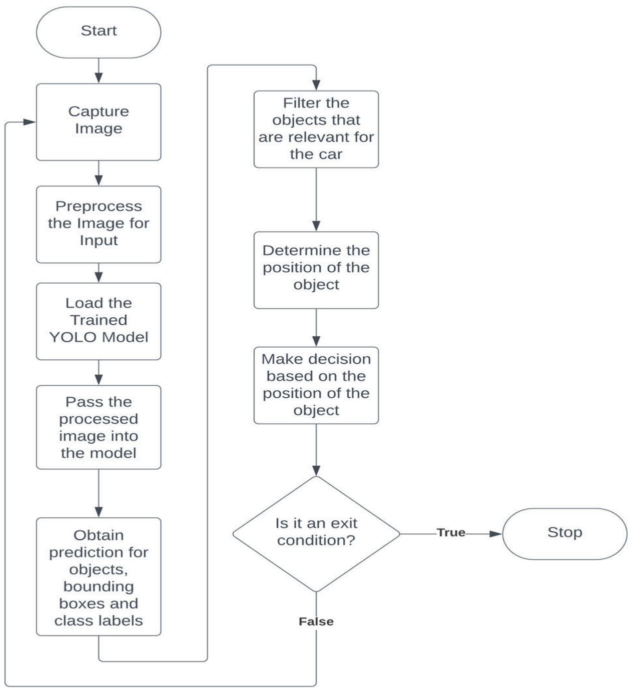

Start
Capture Image
Preprocess the Image for Input
Load the Trained YOLO Model
Pass the processed image into the model
Obtain prediction for objects, bounding boxes and class labels
Filter the objects that are relevant for the car
Determine the position of the object
Make decision based on the position of the object
Is it an exit condition?
True
Stop
False

Fig. 7 Flowchart for Object Detection

Results

Steering Angle Prediction

After training the model multiple times while varying the hyper parameters such as batch size, number of epochs, steps per epochs, and learning rate, along with addition of early stopping criteria, a model capable of navigating properly through the track was obtained. The model was trained with following parameters:

- Batch Size: 128
- No. of Epochs: 100
- Steps per epoch: 30
- Learning rate: 0.001

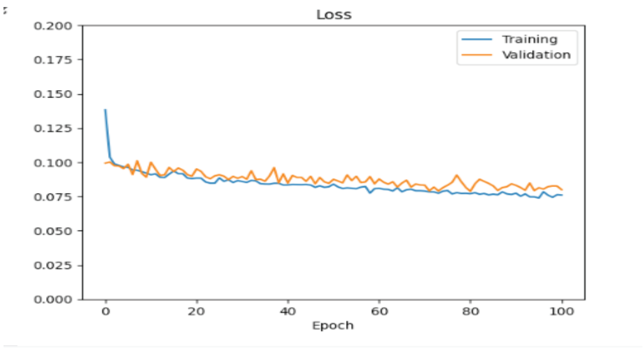


Fig. 8 Loss Plot for Steering Angle Prediction

Object Detection

The pre-trained YOLO model is re-trained to predict only 3 classes: Red traffic light, Green traffic light and Stop sign using the train folder data. Also, accuracy test was done using the test folder data which contained different images that those contained in train folder. A train folder is created automatically after completion of training and it contains all the results of training such as F1 score curve, Precision-Recall curve, Confusion matrix, validation batch label and predictions in image format as well as csv file containing all the value results for every epoch. As seen in Fig. 11, the losses are gradually decreasing and the values of different performance metrics are gradually increasing as the epoch increases which is a good sign that the model is increasing in accuracy.

Table 1

Performance Metrics Result

Performance Metric	**Values**
Precision	0.96282
Recall	0.96875
mAP50	0.9928
mAP50-95	0.7803
F1-Score	0.9658

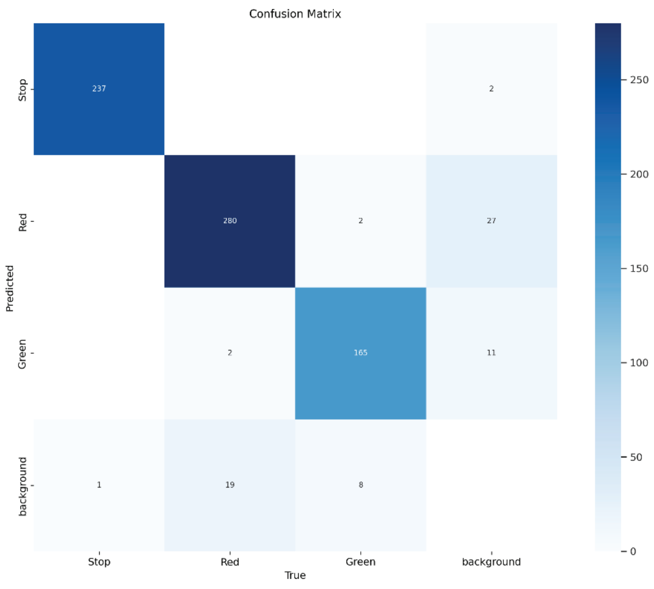


Fig. 9 Confusion Matrix

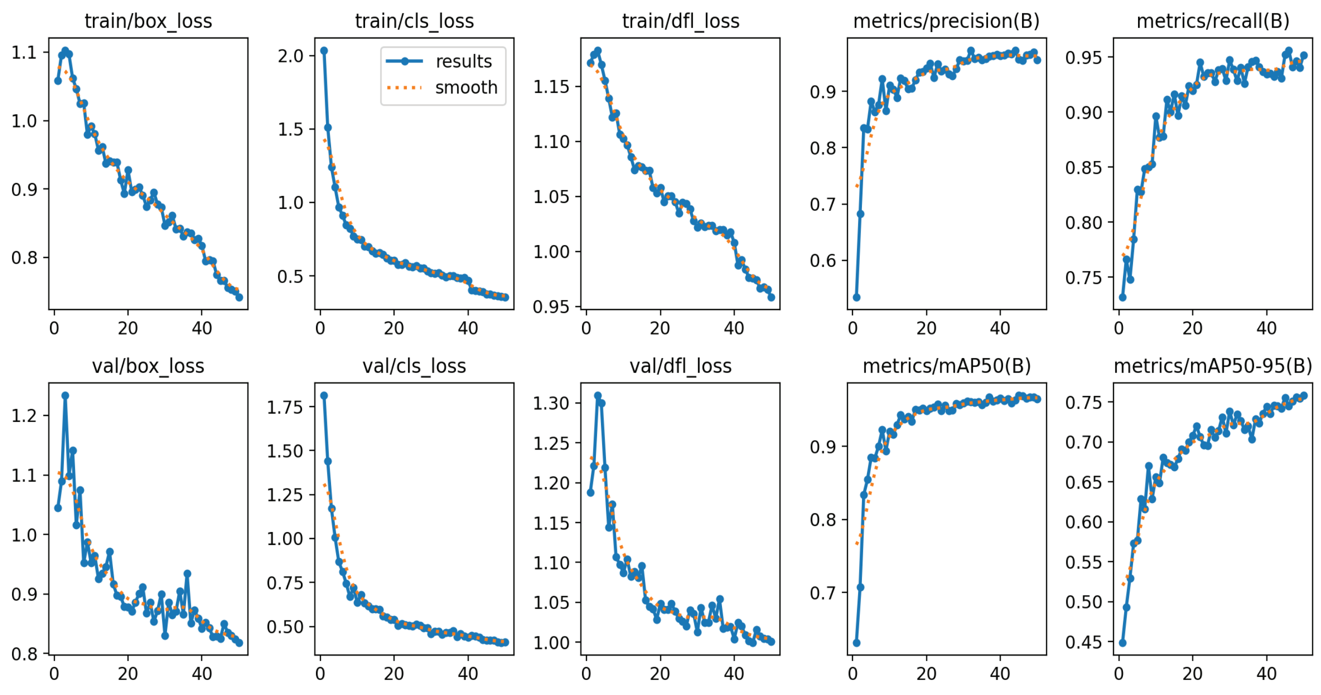
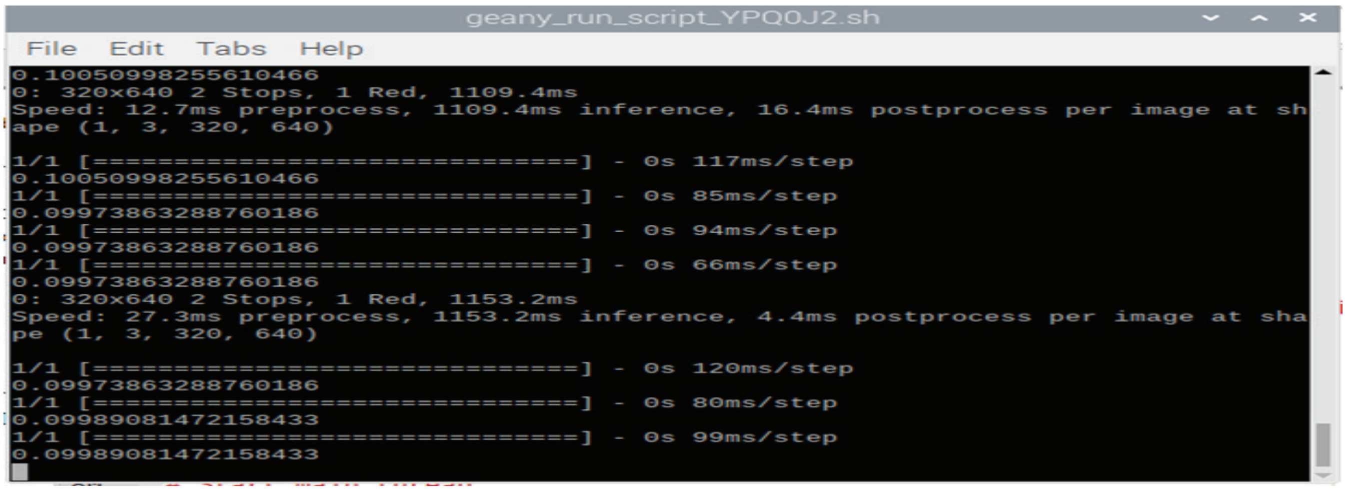


Fig. 10 Curve Plot of various performance metric values (vertical) against epochs (horizontal)



```
0.10050998255610466
0: 320x640 2 Stops, 1 Red, 1109.4ms
Speed: 12.7ms preprocess, 1109.4ms inference, 16.4ms postprocess per image at sh
ape (1, 3, 320, 640)

1/1 [==============================] - 0s 117ms/step
0.10050998255610466
1/1 [==============================] - 0s 85ms/step
0.09973863288760186
1/1 [==============================] - 0s 94ms/step
0.09973863288760186
1/1 [==============================] - 0s 66ms/step
0.09973863288760186
0: 320x640 2 Stops, 1 Red, 1153.2ms
Speed: 27.3ms preprocess, 1153.2ms inference, 4.4ms postprocess per image at sha
pe (1, 3, 320, 640)

1/1 [==============================] - 0s 120ms/step
0.09973863288760186
1/1 [==============================] - 0s 80ms/step
0.09989081472158433
1/1 [==============================] - 0s 99ms/step
0.09989081472158433
```

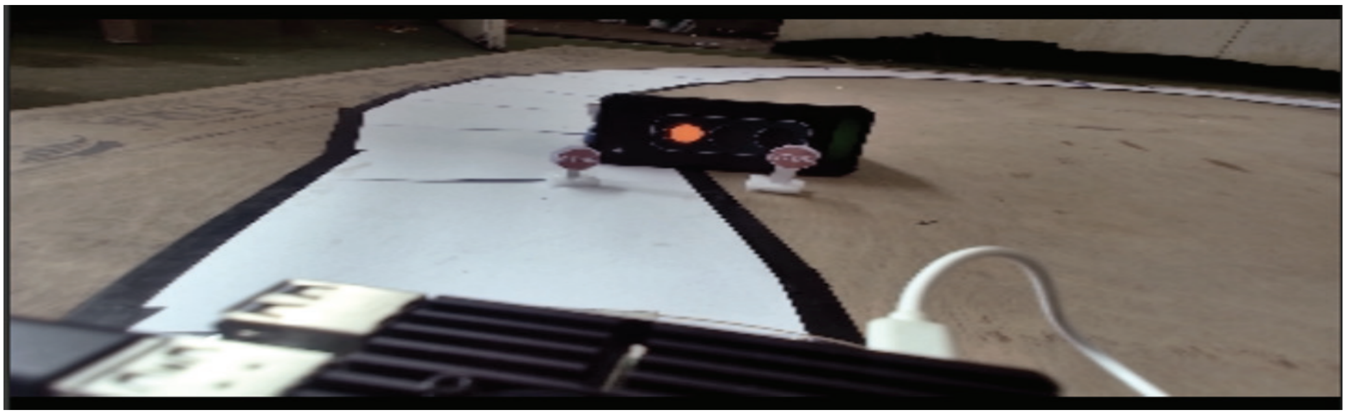


Fig. 11 Lane Image and Corresponding Prediction

Finally, after training of both the models, YOLO object detection model and CNN model were constructed successfully, these were integrated in the Raspberry Pi program where both the models had to run simultaneously in real time and predict the steering value for the car in various situations. The captured image was converted into different formats (YUV or RGB) for the respective model before giving them as inputs and based on the output from those models, the decisions were taken for the movement of the car.

Discussion

The self-driving car prototype successfully navigated the track it was trained on but encountered difficulties when attempting to navigate an unknown track. There were issues during the implementation primarily due to the limited processing power of the Raspberry Pi (2GB RAM), which struggled to run both models simultaneously. As a result, there was an average delay of nearly 5 seconds between processing of consecutive frames, leading to inaccurate decision-making, as proper navigation requires near real-time responses. To address this, multithreading was implemented to enable the simultaneous execution of both models, while reducing the image resolution improved processing speed and decision-making time. Additionally, a heat sink with a fan was installed to prevent performance degradation caused by overheating, ensuring stable operation of the Raspberry Pi.

Conclusion

The main aim of this paper was to develop a miniature self-driving car prototype by integrating two machine learning models in a resource-constrained environment, enabling it to autonomously navigate lane tracks without any assistance. Towards the end, the prototype was able to make accurate decisions to navigate the track it was trained on, while also successfully recognizing traffic signs and responding accordingly.

Acknowledgement

We would like to express our sincere gratitude to the faculty and staff of the Department of Electronics, Communication, and Information Engineering for their constant encouragement, support, and constructive feedback throughout the course of this research. Our sincere appreciation is also directed to Er. Suramya Sharma Dahal, Head of the Department of Electronics, Communication and Information Engineering, for providing the necessary resources, motivation, and steadfast support that facilitated the successful completion of this work.

We are immensely grateful for the assistance, guidance, and supervision we received from numerous individuals who contributed to the successful completion of this research. Our sincere appreciation goes to the faculty, teachers, and friends who helped and supported us along the way.

References

[1] "What are the six levels of autonomous driving? | HERE." Accessed: Jan. 15, 2025. [Online]. Available: https://www.here.com/learn/blog/6-driving-automation-levels

[2] S. K. Kokate, "Review on Autonomous Car using Raspberry Pi," *IJRASET*, vol. 6, no. 1, pp. 3090–3094, Jan. 2018, doi: 10.22214/ijraset.2018.1427.

[3] P. Holla, S. N, S. C, N. S. N, and A. M. B, "Automated Driving Car using RPI Board and CNN Algorithm," *Journal of Network Security Computer Networks*, vol. 5, no. 2, pp. 24–30, Jul. 2019, doi: 10.5281/zenodo.3345261.

[4] I. Sonata, Y. Heryadi, L. Lukas, and A. Wibowo, "Autonomous car using CNN deep learning algorithm," *J. Phys.: Conf. Ser.*, vol. 1869, no. 1, p. 012071, Apr. 2021, doi: 10.1088/1742-6596/1869/1/012071.

[5] T.-D. Do, M.-T. Duong, Q.-V. Dang, and M.-H. Le, "Real-Time Self-Driving Car Navigation Using Deep Neural Network," in *2018 4th International Conference on Green Technology and Sustainable Development (GTSD)*, Ho Chi Minh City: IEEE, Nov. 2018, pp. 7–12. doi: 10.1109/GTSD.2018.8595590.

[6] A. K. Jain, "Working model of Self-driving car using Convolutional Neural Network, Raspberry Pi and Arduino," in *2018 Second International Conference on Electronics, Communication and Aerospace Technology (ICECA)*, Coimbatore: IEEE, Mar. 2018, pp. 1630–1635. doi: 10.1109/ICECA.2018.8474620.

[7] M. N. A. Shetty, "Autonomous Self-Driving Car using Raspberry Pi Model," *International Journal of Engineering Research*, vol. 7, no. 08, 2019.

[8] "A Convolutional Neural Network Approach Towards Self-Driving Cars | IEEE Conference Publication | IEEE Xplore." Accessed: Jun. 22, 2023. [Online]. Available: https://ieeexplore.ieee.org/document/9030307

[9] "YOLO: Real-Time Object Detection." Accessed: Jan. 04, 2024. [Online]. Available: https://pjreddie.com/darknet/yolo/

[10] N. Shazeer, K. Fatahalian, W. R. Mark, and R. T. Mullapudi, "HydraNets: Specialized Dynamic Architectures for Efficient Inference," in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, Salt Lake City, UT: IEEE, Jun. 2018, pp. 8080–8089. doi: 10.1109/CVPR.2018.00843.

[11] M. Bansal, A. Krizhevsky, and A. Ogale, "ChauffeurNet: Learning to Drive by Imitating the Best and Synthesizing the Worst," in *Robotics: Science and Systems XV*, Robotics: Science and Systems Foundation, Jun. 2019. doi: 10.15607/RSS.2019.XV.031.

[12] M. Bojarski *et al.*, "End to End Learning for Self-Driving Cars," Apr. 25, 2016, *arXiv*: arXiv:1604.07316. doi: 10.48550/arXiv.1604.07316.