Received Date: 19th March, 2024
Revision Date: 15th May, 2024
Accepted Date: 21st July, 2024

A Comprehensive Study on Implementation of Deep Learning on Autonomous Vehicle for Steering Angle Prediction and Stability

Swodesh Sharma[1*], Puskar Neupane[2], Shashwot Shrestha[3], Sushil Phuyal[4], Sanjivan Satyal[5]

[1]*Dept of Electrical Engineering, Pulchowk Campus, Tribhuvan University, Email:* xarmaswodesh@gmail.com
[2] *Dept of Electrical Engineering, Pulchowk Campus, Tribhuvan University, Email:* puskarneupane4321@gmail.com
[3] *Dept of Electrical Engineering, Pulchowk Campus, Tribhuvan University, Email:* 076bel043.shashwot@pcampus.edu.np
[4] *Dept of Electrical Engineering, Pulchowk Campus, Tribhuvan University, Email:* sushilphuyal.sp@gmail.com
[5]*Assoc. Professor, Dept of Electronics and Computer Engineering, Pulchowk Campus, Tribhuvan University, Email:* sanziwans@gmail.com

Abstract— **The development of autonomous vehicles has recently enhanced the transportation industry and opened up a variety of opportunities and problems that can be solved with the aid of current methods and technology. In this study, three separate algorithms were used to predict the steering angle with a track image: Artificial Neural Network (ANN), Convolution Neural Network (CNN), and a combination of CNN and Long-Short Term Memory (CNN-LSTM). The PID controller was employed for benchmarking, which takes Cross-Track Error (CTE) provided by the simulation to steer the vehicle. To achieve improved performance, the standard NVIDIA CNN self-driving model was slightly altered by feeding it with sequential frames. The comparison analysis was conducted using the OpenAI Gym Donkey Simulator.**

Keywords— Self Driving Car, ANN, CNN, LSTM, CTE, PID

Introduction

Humans strive to incorporate intelligence and learning into every system with the intent of making it automatic. An automated system has the benefit of being not only affordable and simple to use but also safer. The use of autonomous vehicles has also been on the rise recently in an effort to prevent traffic accidents and driver fatigue.

The history of self-driving cars can be traced back to 1925 when Francis Houdini demonstrated a radio-controlled car with no one inside [1]. Following that, various prototypes are tested for various speeds and environments. Deep learning frameworks have recently been used to make significant progress in the development of self-driving cars. The use of digital hardware in Electric Vehicles (EVs) is gaining momentum, propelling them as one of the most cutting-edge and energy-efficient ways of transportation. Unlike internal combustion (IC) engines, which have a fuel efficiency of just 40%, electric vehicles (EVs) use digital computation to enhance performance. Because of the fast integration of digital electronics, EVs now have great opportunities to enter into the area of autonomous driving, making them even more desirable in terms of transportation safety and innovation. Many approaches were applied in the course of developing autonomous EVs. Traditionally, these approaches were based on a robotic-based approach that split the autonomous driving task into subsequent models, namely perception, planning, and control. These approaches were difficult and needed more computational capacity. With the advancement of chip manufacturing in recent decades the traditional robotic-based approach is being replaced by machine-learning approaches. These machine learning approaches include the Artificial Neural Network Multilayer Perceptron (ANN-MLP), Convolutional Neural Network (CNN), Support Vector Machine (SVM), and Convolutional Neural Network Long Short-Term Memory (CNN-LSTM).

Though different modern techniques are used for improving the performance of autonomous vehicles, which require high computational power, there has been ongoing research on PID controllers to ensure the stability of vehicles in lanes. Different optimization algorithms are also used in the PID controller, which is proven to be efficient [2]. In this paper, we have done a comparative study between the different deep learning algorithms and validated their performance with PID controllers that utilize CTE for path tracing. In Section II, there is a review of numerous publications using various self-driving vehicle techniques. Section III describes several deep neural network techniques used in this experimental study. whereas Section IV presents the PID implementation. Section V describes the procedure used and the steps taken. Comparative Study is demonstrated in Section VI and concluding statements are in Section VII.

Literature review

Over the last two decades, research has been conducted to make self-driving vehicles safer and more practical in any situation. The different research papers are studied to

* *Corresponding Author*

explore new and creative approaches. The steering control is the major part of the control used to move vehicles along the lane.

One of the major catalysts for the evolution of self-driving cars was the Defense Advanced Research Projects Agency (DARPA) challenge, aimed at accelerating the development of autonomous vehicle technologies, which took place in 2004 and 2005. The challenge was won by Stanley, the autonomous vehicle developed by the Stanford Racing Team. Stanley's achievement demonstrated the feasibility and potential of autonomous driving technology as it successfully navigated a 175mile desert course without human intervention in 6 hours and 53 minutes. This remarkable feat was made possible by employing advanced AI technologies such as machine learning and probabilistic reasoning. The success of Stanley accelerated the world's move towards autonomous driving, showcasing the immense potential of intelligent robots in tackling complex environments [3].

Many rule-based as well as machine learning-based approaches to self-driving cars have been developed. One of the famous rule-based autonomous driving methods is to detect lane features and mathematically formulate the steering angle. For lane detection, the well-known Canny edge detection algorithms are employed. These algorithms open up a plethora of possibilities for feature extraction, image analysis, and computer vision [4]. The computer vision-based techniques help to formulate the path for the vehicle and PID is then employed for the motion. The majority of recent articles used deep learning for lane recognition and PID controllers to regulate steering angle stability. The measurements of yaw rate and lateral offset are critical for adjusting the steering angle and keeping the autonomous vehicle on course [5].

Emirler et al. proposed a parameter-space-based robust PID steering controller that has been experimented with in real prototype vehicles, which validates the theoretical data [6]. Various open platforms are also being developed to study and design autonomous vehicle algorithms and test their performance [7]. Achieving stability for a wide range and different speeds of vehicles is also one of the problems that are overcome by using an adaptive PID controller. Zhao et al. have used an adaptive PID controller that gives us flexibility and simplifies the system's software and hardware [8]. The tuning of the PID parameter is also crucial to eliminating the error and moving along the path. For the best tuning of parameters, different algorithms are being tested and implemented. A new algorithm like" WAF tune" was also introduced and proved to be the best among other algorithms for different vehicle speeds [9]. PID and pure pursuit control are also used for lane detection, using three different methodologies: edge detection, Hough transformation, and bird's eye view [10].

Bojarski et al. presented a CNN-based approach that maps pixel data of the car's front camera image directly to steering commands without the need for complex rule-based decisions. The data were recorded by manually driving the car under diverse weather conditions and were further augmented. The results proved the CNN-based system to be able to drive successfully on the track even without lane markings [11].

Lade et al. conducted an extensive study on the simulation of a self-driving car using the Udacity simulator. All the data were recorded by manually driving the car, and later balanced, augmented, and cropped to eliminate unnecessary details. The authors performed a comparison of the replica of the Nvidia model by increasing the CNN complexity and found that the more complex architecture performed better [12].

Gu et. al introduced an LSTM-based model designed to imitate Waymo's self-driving cars' behavior using the Waymo Dataset. The model predicts the acceleration and steering angle of the car by utilizing sensor data and camera images. Specifically, it processes ten sequential front images of the car through ResnetV2. The output from ResnetV2, along with twelve input features, is then passed into an encoder-decoder structure with an LSTM model. The authors conclude that their LSTM-based model, incorporating front camera images, is an effective and efficient approach for predicting acceleration [13].

Deep Neural Network

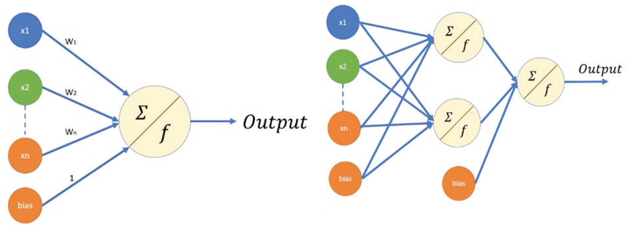
A. Artificial Neural Network (ANN)

ANN Multi-Layer Perceptron (MLP) consists of different nodes connected together, where each connection has its own weights, and the information is passed from input nodes to output nodes through a feed-forward network, and the weights can be adjusted to minimize the error through backpropagation. This is shown in Fig.1 where each node produces output by summarizing the product of its weight and input from another node, and the total sum is passed to the activation function.

ANN MLP is highly effective in the task of classification, where a large number of features have to be given as input. The mathematical formula for the output of nodes can be given by the equation (1).

$$y = \sigma\left[\sum_{i=1}^{n}(w_i \cdot x_i) + b\right] \quad (1)$$

where σ is the activation function.



(a) (b)

Fig. 1. (a) Perceptron (b) Multilayer Perceptron

B. Convolution Neural Network (CNN)

CNN is used for extracting features from the image. Depending upon those features, items can be categorized. CNN is based on the convolution function which can be represented as

$$(f x g)(t) = \int_{-\infty}^{\infty} f(\tau)\, g(t-\tau)\, d \qquad (2)$$

Equation 2 simply shows that convolution is the combined integration of two functions and shows how one function modifies the other function. In the case of self-driving cars, the first function is the input image and the second function is the feature detector also called the filter or kernel. These two functions generate convolved features also called feature maps. This process is depicted in Fig. 2.

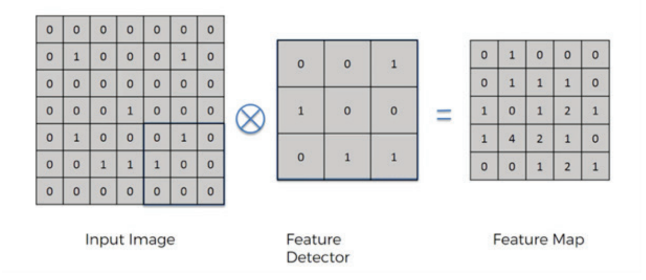


Fig. 2. Convolution Process

The feature map is activated by an activation function, e.g., ReLU, which increases non-linearity in the images. The convolution step reduces the size of the input image as the size of the feature map gets significantly smaller. The size of the feature map can be further reduced by using a feature of CNN known as" pooling." Even though the size of the feature map is reduced, the key features are preserved. The output of the pooling process is converted into a 1D vector, which is then fed into an ANN with dense layers, and finally, the output is obtained.

C. Long Short-Term Memory

LSTM is a type of recurrent neural network (RNN) architecture that is designed to process sequential data because they have a hidden state that can retain information from previous time steps and use it as context for predicting the next step in the sequence. Unlike traditional RNN, LSTM has a more sophisticated cell structure that incorporates gating mechanisms. These gating mechanisms control the flow of information within the LSTM cell, allowing it to selectively remember or forget information at different time steps [14]. LSTM networks can be employed to analyze and understand temporal patterns within a sequence of images. By treating each image as a time step in the sequence, LSTMs can capture the dependencies and context between consecutive frames.

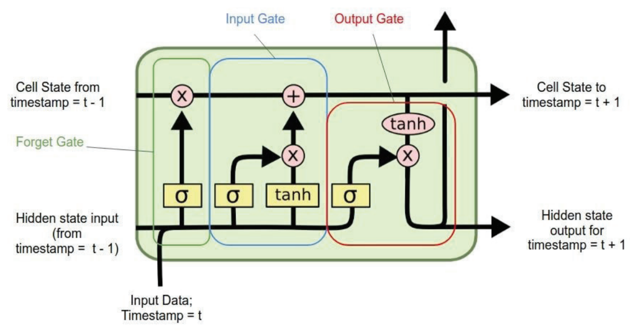

Input Gate
Output Gate
Cell State from timestamp = t - 1
Cell State to timestamp = t + 1
Forget Gate
tanh
σ
σ
tanh
σ
Hidden state input (from timestamp = t - 1)
Hidden state output for timestamp = t + 1
Input Data; Timestamp = t

Fig. 3. LSTM architecture [17]

LSTM has mainly three types of gates: Forget gate, Input gate and Output gate as in Fig.3. The purpose of these gates is to control information at various stages of the network. The equation for the output of forget gate is:

$$f_t = \sigma(W_f * [h_{t-1}, x_t] + b_f) \qquad (3)$$

where, $[h_{t-1}, x_t]$ is the concatenation of input vector x_t and previous hidden state h_{t-1}. The output of the input gate is

$$i_t = \sigma(W_i * [h_{t-1}, x_t] + b_f) \qquad (4)$$

This network selectively retains important information and discards unimportant data. The output of new cell state with candidate memory (C_e) using current state x_t and previous hidden state h_{t-1} is

$$Ct = Ct{-}1 * ft + it * C_t' \qquad (5)$$

where,

$$C_t' = tanh(W_C * [h_{t-1}, x_t] + b_C) \qquad (6)$$

To decide the portion of (C_t) to be passed to the output, the input is passed to the sigmoid layer

$$o_t = \sigma(W_o * [h_{t-1}, x_t] + b_o) \qquad (7)$$

The final output of the cell will be,

$$h_t = o_t * tanh(C_t) \qquad (8)$$

PID

The basic functional block diagram of the PID controller is shown in Fig.4. The P, I, and D are the bias hyper-parameters that are multiplied by the error obtained and then fed to the process. The hyper-parameters were continuously updated by the algorithm till the desired output is obtained.

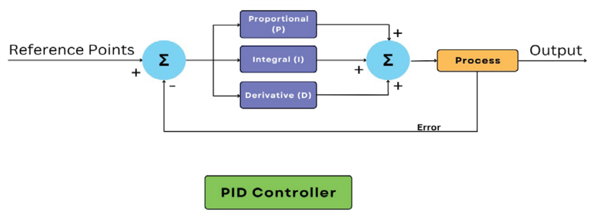


Fig. 4. PID Controller

A. P Controller

The output of the P controller is the product of error and constant term(K_P). The constant term K_P is known as the proportional gain constant. The output is directly proportional to the proportional gain constant. So the output can be controlled by controlling the value of k_p. The output equation of the P controller is given as in equation 9.

$$Output = K_P * error \quad (9)$$

The response of the system after implementing the P controller only is shown in Fig.5.

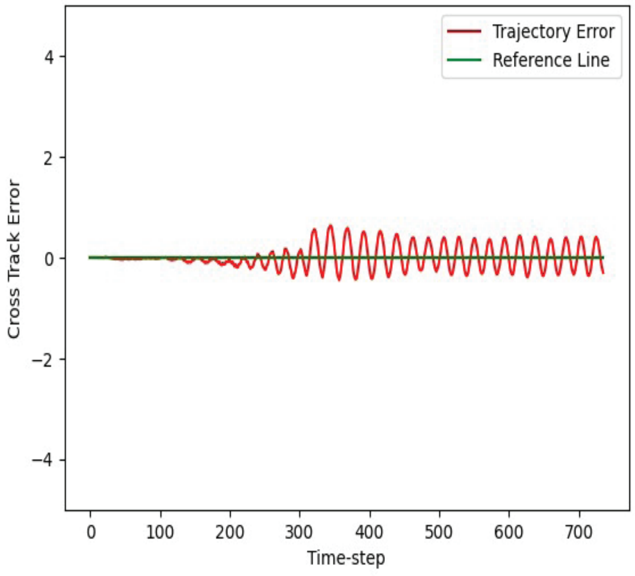


Fig. 5. Response of the car using only P controller

From Fig.5, it is seen that the P controller tries to converge to the reference line but cannot align itself along the reference. So, it overshoots and starts to oscillate around the reference line.

B. PD Controller

To overcome the oscillation caused by the P controller, another term is introduced which is known as Derivative(D). The output of the D controller is the product of the slope of error over time and derivative gain K_D. D controller helps to minimize the error by reducing overshoot and settling time. The output equation of the PD controller is given as:

$$Output = \quad K_P * error \; + \; K_D * \frac{d}{dt} error \quad (10)$$

Combining D with P eliminates the oscillation and our system reaches a steady state within a short time. The response of the system after using the PD Controller is shown in Fig.6.

As seen from Fig.6, despite following the reference line quite nicely, the PD controller is not robust. When there is noise or glitch in the system, then the derivative term amplifies the noise and our system will oscillate far from our target path.

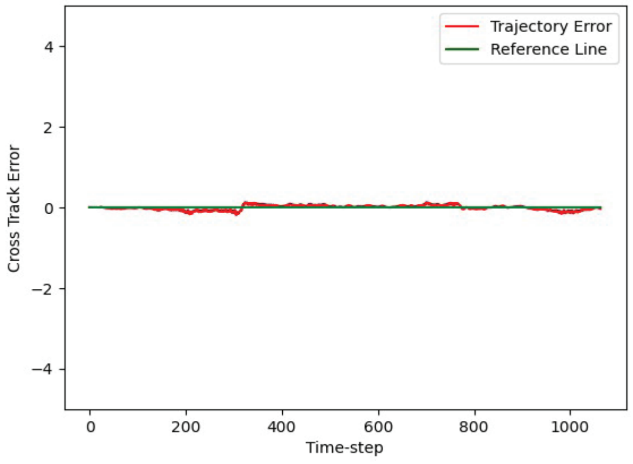


Fig. 6. Response of the car using PD controller

C. PID Controller

Proportional(P), Integral(I) and Derivative(D) combined together to form a PID controller. PID controller is widely used in industrial appliances. The output of the P and D controller is already been discussed and the output of the I controller is the sum of the instantaneous error over time. The output equation of the PID Controller is given as:

$$Output = \quad K_P * error \; + \; K_D * \frac{d}{dt} error \; + \; K_I \quad (11)$$

In an autonomous vehicle, P controls the direction of the steering, D monitors the P and makes counter-steering to P if required, and I check the whole path and error and makes decisions according to it. The response of the system using the PID controller is shown in Fig.7.

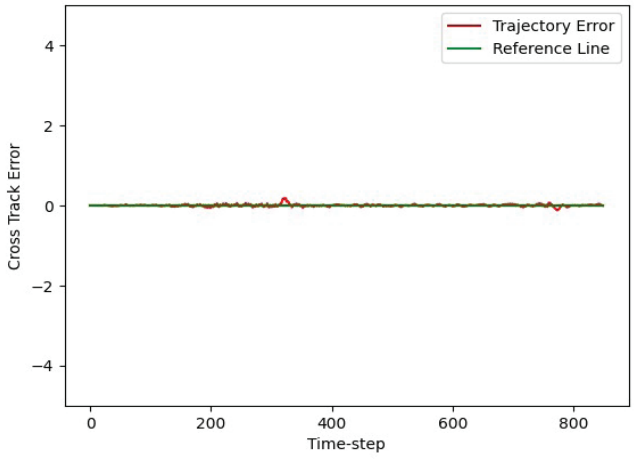


Fig. 7. Response of the car using PID controller

Fig. 7 shows that by using the properly tuned hyperparameters, the car followed the reference line accurately with very less error.

D. Twiddle Algorithm

Twiddle Algorithm [15] is a recent technique to tune the hyper-parameters of the PID controller. Twiddle Algorithm was used so as to minimize the average cross-track error for the vehicle system. Its flowchart is shown in Fig.8. In Twiddle, the set of hyperparameters is put in a matrix P, and based on the following algorithm, updating the values of P based on the average cross-track error obtained till the point where the parameters of P start to converge. The final values of P are the properly tuned parameters of the PID controller.

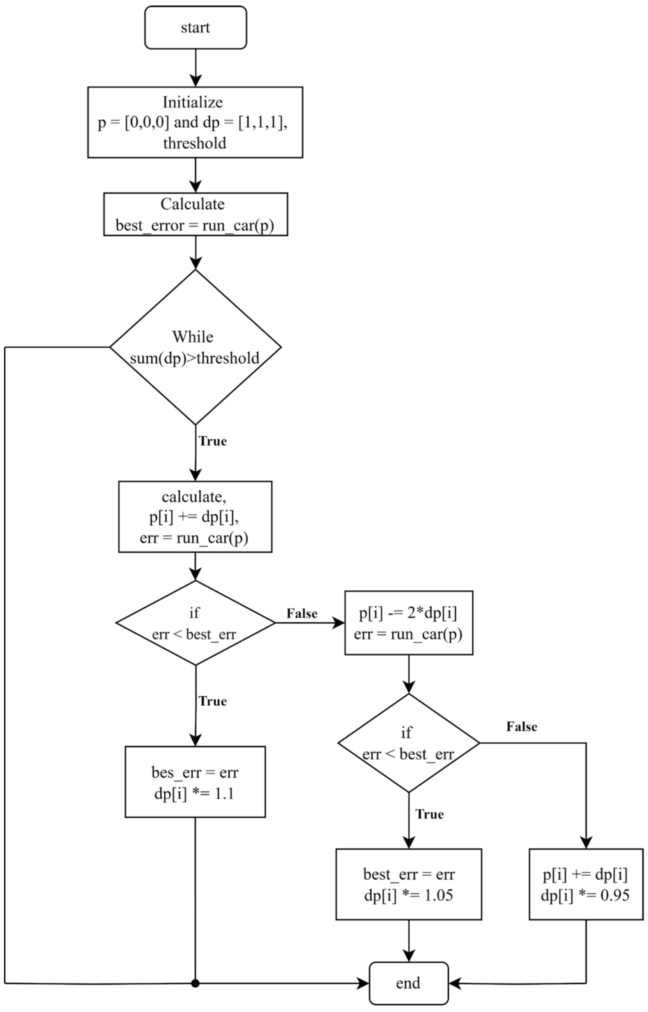


Fig. 8. Flowchart of Twiddle algorithm for tuning PID hyper-parameteRS

Methodology

A. Data Collection

For the data collection, an open-gym donkey car simulator [16] was used as shown in Fig. 9. This simulator consists of a car with a front camera, which is used by the simulator to record the images of the track, at regular instant of time. For the data collection, the inbuilt auto-driving mode of the simulator was used. Here, in this mode, the car drove around the track in constant throttle. The car was driven across three different tracks and a total of 50000 images were recorded and stored on the computer along with their corresponding JSON files, which contain the key information of the images like steering angle. The car being driven for three tracks has more data with 0 as the steering angle as seen from the graph in Fig. 10 which will make the model biased towards predicting 0 so we balance the data. Now the balanced data was split into two sets, training and validation sets.

B. Data Prprocessing and Augmentation

The data was augmented with the 'imgaug' augmentation library and applied only to the training data. The data augmentation includes various techniques such as image panning, brightness alteration, zooming, and flipping. These techniques make the data more versatile, allowing the model to generalize for various scenarios despite the limited amount of data. Next, the unwanted features of the image are removed through four preprocessing steps: cropping, changing the color format, applying Gaussian

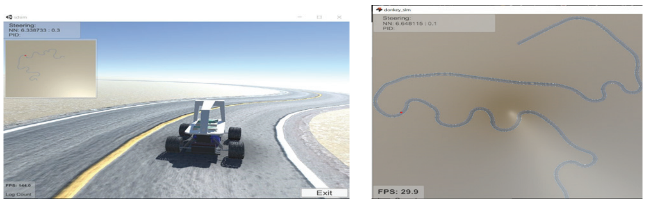
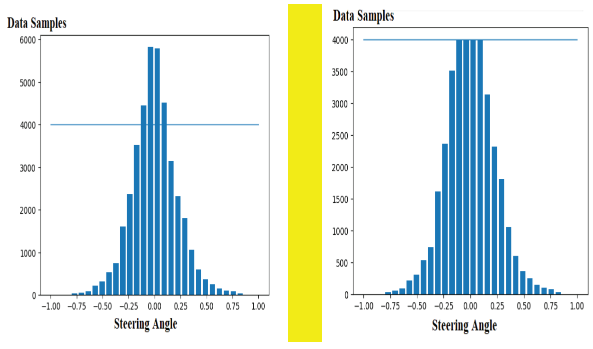

(a) (b)

Fig. 9. (a) Donkey Simulator (b) Track Sample



(a) (b)

Fig. 10. (a) Before Balancing (b) After Balancing

blur, and normalizing the image. The primary purpose of cropping is to eliminate unwanted portions of the image, such as the sky portion. In the second preprocessing step, the image format is changed from RGB to YUV. This conversion helps to highlight the lane features of the track. Subsequently, a Gaussian blur is applied, which smooths the images and removes disturbances. The final steps in image processing are resizing and normalization, which resizes the image to the desired shape and translates data in the range [0, 1] helping speed up the training process. The resized image frame (IN) with an intensity value in the range [0,255] is normalized to an image frame (IN') with intensity values in the range [0,1] by the relation as given in the equation (12).

$$I_{N'} = \frac{I_N}{255} \qquad (12)$$

Fig. 11 shows the difference between the original and the pre-processed image.

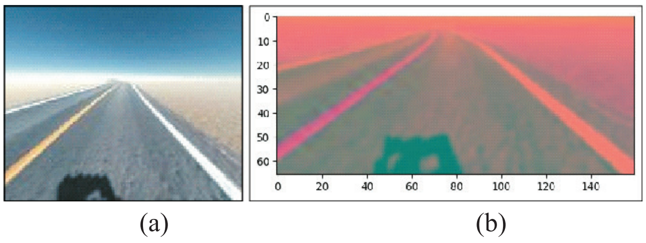


Fig. 11. (a) front camera image (b) preprocessed image

E. Model Architecture

Three different algorithms were employed for behaviour cloning. ANN architecture involved an input layer of fully connected layers that accepts flattened image. Dropout layers were added for regularization. The output layer had one neuron for regression. Mean absolute error (MAE) was the loss metric and the Adam optimizer was employed. Each hidden layer had Exponential Linear Unit (ELU) activation.

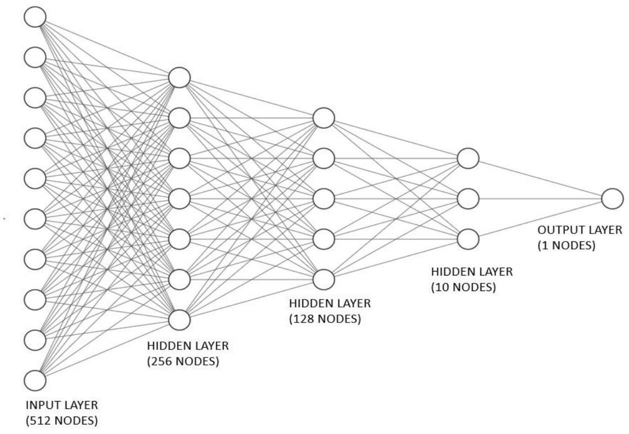


Fig. 12. ANN Architecture

The CNN architecture used was the Nvidia Self Driving Car CNN model [11], which architecture is shown in Fig.13.

Now, to improve the performance of the Nvidia model, LSTM was incorporated. The proposed model architecture consists of a Time Distributed CNN with ELU activation, followed by a Time Distributed max-pooling layer with a pool size of (2, 2). This pattern is then followed by increasing the size of CNN filters to 32, 64, and 128, respectively. The output of this stack is then flattened and fed to an LSTM layer with 128 units. Additional Dense layers are applied with ELU activation, containing 1000, 100, 50, and 10 units, respectively. Finally, the last layer is a Dense layer with a single unit to perform the regression task of predicting the steering angle. The model architecture is as shown in Fig.14. This comprehensive architecture aims to effectively capture spatiotemporal features in the track frame sequence.

The total trainable parameters along with the time taken for training (with 100 epochs with a batch size of 50 and 20 steps per epoch) in google collab is mentioned in table I.

F. PID Implementation

The PID controller is implemented to drive the car following a central lane properly with some smooth left and right turns. A good set of PID parameters enables our vehicle to stay in the central portion of the lane and take a smooth left and right

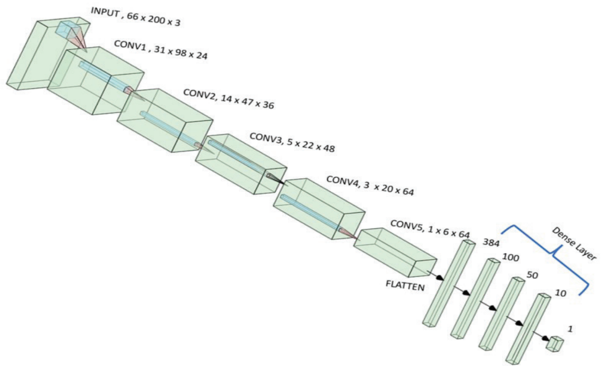


Fig. 13. CNN Architecture

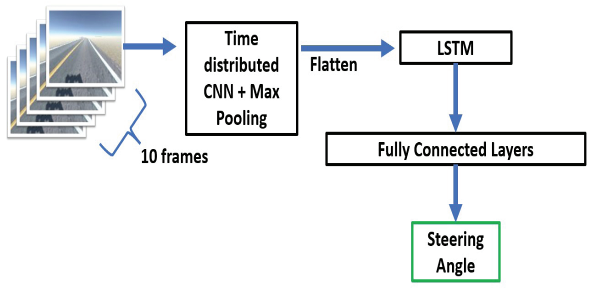


Fig. 14. CNN + LSTM Architecture

TABLE I

TRAINING HISTORY OF DIFFERENT MODELS

SN	Model	Total Trainable Parameters	Time for training
1	ANN	1,446,037	1.2 Hours
2	CNN	220,219	0.5 hours
3	CNN + LSTM	7,471,855	1.7 Hours

turns accordingly without touching the extremities of the lane. The simulation of PID implementation was done in a donkey simulator as shown in Fig.9. The cross-track error (CTE) is the error distance between the current lateral position of the vehicle and the center of the lane. The CTE was extracted from the default built-in function from the simulator. With the following mathematical equation, the CTE was used with hyperparameters of PID to finally obtain the steering angle δ as in equation (13).

$$\delta = K_p * cte + K_d * \frac{d}{dt}(cte) + K_i * \sum cte \tag{13}$$

Comparative study

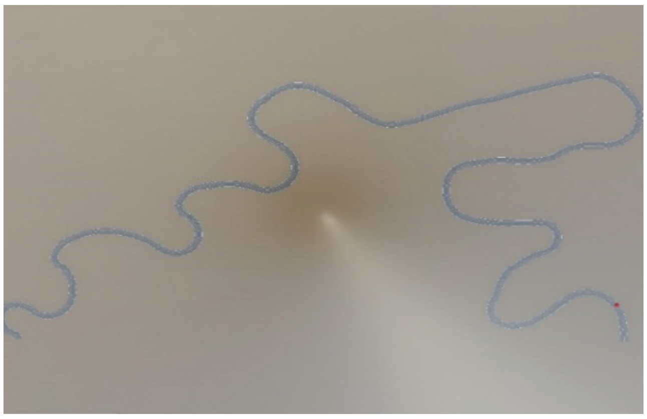


Fig. 15. Track for evaluation

After manually running the vehicle down the track to collect training data, the algorithms were ran on the fresh track which consists of two lanes and the car has to follow the track by maintaining its position in the right lane. To evaluate the model accuracy, the CTE was plotted and its absolute mean was taken for the evaluation. The track on which testing was done is shown in Fig.15.

After performing the simulation using all the models described in the earlier sections, the final graphical performance of all the models is shown in Fig.16.

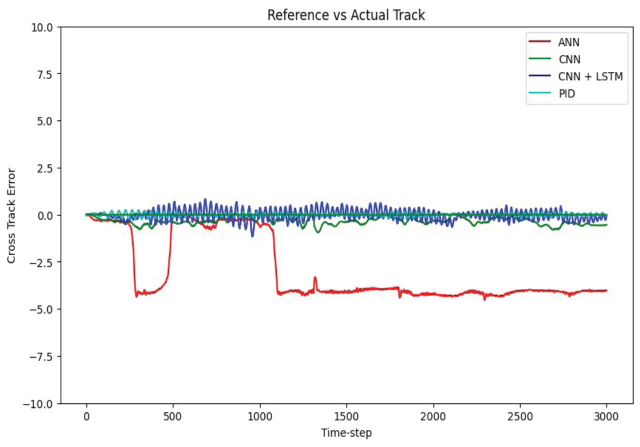


Fig. 16. CTE vs time steps plot of Different models

TABLE II

MEAN CTE OF DIFFERENT MODELS

SN	Model	Mean Absolute CTE
1	ANN	2.99639
2	CNN	0.34638
3	CNN + LSTM	0.23837
4	PID	0.07438

From the observation of the Mean CTE table in Table II and CTE plot in Fig.15, the following remarks can be put forward.

a The ANN model has a mean CTE of 2.99, indicating that it is not reliable and doesn’t capture the lane feature accurately. Though it follows the road, it went outside the desired lane.

b The CNN model shows an improvement with a mean CTE of 0.34. It extracts the lane features from the image and runs smoothly with little jerky motion when lighting conditions vary.

c The combination of CNN and LSTM performs significantly better with a mean CTE of 0.23. This model aligns perfectly with the track, indicating a better understanding of the track’s dynamics and accurate positioning. There is a minor oscillation resulting from the time it takes to predict the steering angle.

d The PID model achieves the lowest mean CTE of 0.07. Since the simulation itself provides feedback, it is expected to have less error. The PID model utilizes this feedback effectively to achieve precise alignment with the track. The tuning of PID was carried out through the twiddle algorithm.

Conclusion

In this paper, tests, and comparisons of various algorithms for automatic steering were conducted. The study utilized the Gym Donkey Car simulator to gather data and evaluate the trained models. The ANN failed to grasp the track features, whereas the CNN did a good job in feature extraction but lacked the temporal information from previous frames. The CNN + LSTM model demonstrated the best performance among the tested models, achieving accurate alignment with the track. On the other hand, the PID model, though it had low CTE, was used solely for benchmarking purposes and proved challenging to utilize in real-life scenarios due to the absence of direct CTE feedback, as obtained in simulations. The findings suggest that combining CNN with LSTM, where 10 frames are continuously fed, is optimal for predicting the steering angle. This model effectively captures track features and achieves precise alignment. However, due to the model’s longer prediction duration, minor oscillation was introduced which can be reduced with higher processing power.

References

[1] R. Glon and S. Edelstein, “History of self-driving cars milestones,” Jul 2020. [Online]. Available: https://www.digitaltrends.com/cars/history-of-selfdriving-cars-milestones/.

[2] K. Kobatake, T. Okazaki, and M. Arima, “Study on optimal tuning of pid autopilot for autonomous surface vehicle,” *IFAC-PapersOnLine*, vol. 52, pp. 335–340, 01 2019.

[3] S. Thrun, M. Montemerlo, H. Dahlkamp, D. Stavens, A. Aron, J. Diebel, P. Fong, J. Gale, M. Halpenny, G. Hoffmann, K. Lau, C. Oakley, M. Palatucci, V. Pratt, P. Stang, S. Strohband, C. Dupont, L.-E. Jendrossek, C. Koelen, and P. Mahoney, “Stanley: The robot that won the darpa grand challenge.” *J. Field Robotics*, vol. 23, pp. 661–692, 01 2006.

[4] J. Canny, “A computational approach to edge detection,” *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. PAMI-8, pp. 679 – 698, 12 1986.

[5] R. Marino, S. Scalzi, and M. Netto, “Nested pid steering control for lane keeping in autonomous vehicles,” *Control Engineering Practice*, vol. 19, no. 12, pp. 1459–1467, 2011.

[6] M. T. Emirler, ˙I. M. C. Uygan, B. A. Guvenc¸, and L. G¨ uvenc¸, “Robust PID steering control in parameter space for highly automated driving,”¨ *International Journal of Vehicular Technology*, vol. 2014, pp. 1–8, Feb. 2014. [Online]. Available: https://doi.org/10.1155/2014/259465

[7] S. Kato, E. Takeuchi, Y. Ishiguro, Y. Ninomiya, K. Takeda, and T. Hamada, “An open approach to autonomous vehicles,” *IEEE Micro*, vol. 35, no. 6, pp. 60–68, 2015.

[8] P. Zhao, J. Chen, Y. Song, X. Tao, T. Xu, and T. Mei, “Design of a control system for an autonomous vehicle based on adaptive-PID,” *International Journal of Advanced Robotic Systems*, vol. 9, no. 2, p. 44, Jan. 2012. [Online]. Available: https://doi.org/10.5772/51314

[9] W. Farag, “Complex trajectory tracking using PID control for autonomous driving,” *International Journal of Intelligent Transportation Systems Research*, vol. 18, no. 2, pp. 356–366, Sep. 2019. [Online]. Available: https://doi.org/10.1007/s13177-019-00204-2

[10] M. K. Diab, H. H. Ammar, and R. E. Shalaby, “Self-driving car lane-keeping assist using PID and pure pursuit control,” in *2020 International Conference on Innovation and Intelligence for Informatics, Computing and Technologies (3ICT)*. IEEE, Dec. 2020. [Online]. Available: https://doi.org/10.1109/3ict51146.2020.9311987

[11] M. Bojarski, D. D. Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. D. Jackel, M. Monfort, U. Muller, J. Zhang, X. Zhang, J. Zhao, and K. Zieba, “End to end learning for self-driving cars,” 2016.

[12] S. Lade, P. Shrivastav, S. Waghmare, S. Hon, S. Waghmode, and S. Teli, “Simulation of self driving car using deep learning,” in *2021 International Conference on Emerging Smart Computing and Informatics (ESCI)*, 2021, pp. 175–180.

[13] Z. Gu, Z. Li, X. Di, and R. Shi, “An lstm-based autonomous driving model using a waymo open dataset,” *Applied Sciences*, vol. 10, no. 6, 2020. [Online]. Available: https://www.mdpi.com/2076-3417/10/6/2046

[14] S. Hochreiter, J. Schmidhuber *et al.*, “Long short-term memory [j],” *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[15] S. Thrun, “Cs373: Artificial intelligence for robotics. udacity, san francisco, california,” 2018.

[16] T. Kramer, R. Sokolkov, and L. Johnson, “Openai gym environments for donkey car,” Online, Jul. 2019. [Online]. Available: https://gym-donkeycar.readthedocs.io/en/latest/?badge

[17] Yan Xu. Understand recurrent neural network with four figures, Sep 2018.