Enhancing Autonomous Vehicle Control: Adaptive Neural Network-Based Control in AirSim Simulator

Nishant Pandey
UID: 119247556
npandey2@umd.edu

Aaqib Barodawala
UID: 119348710
abarodaw@umd.edu

Rishikesh Jadhav
UID: 119256534
rjadhav1@umd.edu

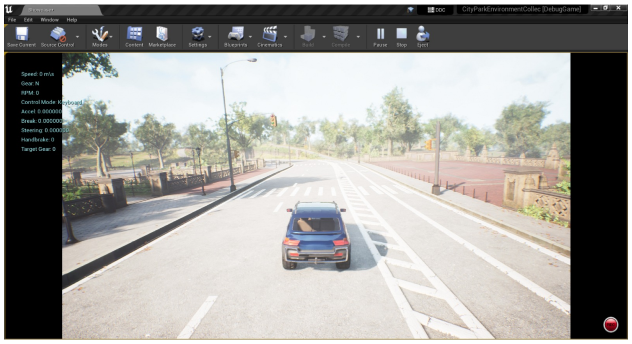
Hrugved Pawar
UID: 120118074
hpawar@umd.edu

Abstract

Ensuring seamless, safe navigation in intricate, real-world settings is the main problem facing autonomous driving. This is converting complex mathematical formulas into exact physical steps in the face of variable traffic circumstances. Our goal is to improve the safety and effectiveness of autonomous cars in various driving scenarios through the development and testing of sophisticated control systems and deep learning techniques inside a realistic simulation. This research goes into the use of the AirSim simulator [2] for training neural networks to steer a car independently. The study begins by utilizing the courses/environments available in Unreal Engine using AirSim and utilizing a neural network to maneuver an automobile around it. The second section builds on this foundation by investigating the effect of data source selection on neural network performance. A proportional-integral-derivative (PID) controller and a Model Predictive Control (MPC) are used to collect data and train the designed model on them. The report also explores the difficulties encountered, such as model selection and the limits of mean squared error as a performance indicator, laying the groundwork for future research in real-world applications and more complicated control scenarios. In this work, we report the outcomes of gathering data with MPC and PID controllers and then applying a trained model to this dataset. Among the results are video outputs that show how well the model performs in actual-world situations.

1. Introduction

Among the primary obstacles in the realm of autonomous driving is to make sure that cars moving through complicated real-world situations move smoothly. The issue at hand is the smooth conversion of complex algorithm-based high-level decision-making processes into accurate and fluid physical motions in the dynamic road environment. This translates, practically speaking, to the requirement for a strong simulation framework that faithfully captures the complexities of real-world settings. We tackle this problem by simulating the autonomous driving experience in our simulation environment using sophisticated control systems and deep learning techniques. This methodology serves as a testing environment for tackling the inherent problems connected with attaining jitter-free mobility in autonomous vehicles on real roads, in addition to facilitating the development of safer and more dependable autonomous systems. This implementation also tackles the problems related to sharp and gradual turns, the vehicle does not have enough space for mobility on a single stretch of a road, which shows the generalizability of our system.



Speed: 0 m\s
Gear: N
RPM: 0
Control Mode: Keyboard
Accel: 0.000000
Break: 0.000000
Steering: 0.000000
Handbrake: 0
Target Gear: 0

Figure 1: AirSim Environment featuring the Utilized Vehicle

Significant progress has been made in the incorporation of machine learning into autonomous vehicle simulation, especially when using Microsoft's AIRSIM simulator [2]. Using AIRSIM's capabilities, this study presents novel ideas and methodology for training neural networks for autonomous driving. To test and develop autonomous driving algorithms, AIRSIM is an open-source simulator that runs on top of game engines such as Unreal Engine 4. With capabilities like depth maps, semantic segmentation data, LIDAR, and realistic urban layouts, it's the perfect platform for in-depth study on autonomous vehicles.

This study primarily focuses on teaching neural networks to operate a vehicle on their own in the AIRSIM environment. To do this, data must first be gathered using basic controllers like Model Predictive Control (MPC) and Proportional-Integral-Derivative(PID) controllers. Next, the data must be used to train neural network models. The neural networks are made to forecast the steering angles and throttle of the car depending on input data, such as pictures taken by the simulator's cameras.

The selection of suitable neural network topologies and training protocols to guarantee the model's efficacy in directing the vehicle in diverse driving conditions is one of the main difficulties tackled. In addition, instead of using RGB data as is often done, the project investigates using image data along with annotations to train the neural network.

Moreover, the study explores the use of these approaches outside of simulation. It talks about how to train a neural network with a comparable machine learning pipeline used for the AIRSIM project so that it can operate an autonomous remote-controlled automobile in real life. This illustrates how information and models may be used from simulation to the actual world.

Using Model Predictive Control (MPC) and Proportional-Integral-Derivative (PID) controllers—which are fundamental to the field of autonomous vehicle navigation—we have methodically collected data for our study. The creation of an extensive dataset that represents a variety of driving circumstances and controller reactions required the completion of this data-collecting phase. Using this dataset as a foundation, we trained an advanced model that aims to replicate these control schemes. Our method's efficacy is demonstrated in video outputs where the trained model is used on the dataset to illustrate how well it can mimic and even improve the traditional MPC and PID controllers' decision-making processes. These outcomes not only support our technique but also open the door to more sophisticated autonomous driving technologies.

2. Related Work

The study "Neural Network Based Model Predictive Control for an Autonomous Vehicle" (Vianna et al., 2021) [14] makes a substantial addition to the field of autonomous vehicle control that is quite similar to our work. The use of Model Predictive Control (MPC) in autonomous vehicle management is the main topic of both research. Vianna et al. investigate the possibility of learning-based controllers by using neural networks in place of conventional MPC. Their method stands out in particular for simultaneously exploring supervised learning and reinforcement learning (RL) techniques—imitation learning for cloning the behaviour of MPC and PID controller.

The paper "A Survey of Deep Learning Applications to Autonomous Vehicle Control" by Sampo Kuutti, Richard Bowden, Yaochu Jin, Phil Barber, and Saber Fallah (Kuutti et al., 2019) [15] offers a thorough overview of deep learning's application in the field of autonomous vehicle control. This survey work, which explores the difficulties and developments in the design of controllers for autonomous vehicles—a crucial area of our research—is very pertinent to our project. The concepts covered by Kuutti et al. and our study are comparable, especially when it comes to the use of deep learning for challenging control tasks in dynamic contexts. Our study is primarily concerned with the actual implementation and optimization of a hybrid control system that makes use of the advantages of both deep learning and traditional control systems, whereas Kuutti et al. offer a comprehensive assessment of the subject.

3. Data Collected and Used

Data collected is a substantial amount of 256x144 pixel dimensional images recorded when running scenarios of the vehicle navigating in the AirSim environment. Waypoints are first generated in AirSim which provides a reference trajectory that the vehicle should follow. The controllers then optimize the vehicle's control inputs (throttle and steering) over a finite time horizon while taking into account the desired trajectory represented by these waypoints. To generate and save these waypoints, the car is first simulated in AirSim, and using manual input, its coordinates in the environment are then saved in a text file using a Python script.

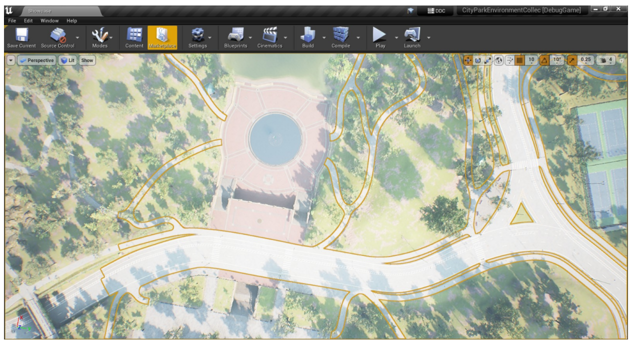


Figure 2: Top View of the Utilized Environment

Before using these waypoints in the optimization problem, they are transformed into the vehicle's local coordinate system. This transformation makes it easier to define the cost and constraint functions relative to the vehicle's state. A function then transforms these waypoints from the global coordinate system to the vehicle's local coordinate system, considering the current vehicle pose. The transformed waypoints are then used to fit a polynomial curve. This curve represents the desired trajectory that the vehicle should follow. The polynomial fitting is done so that the shape of the desired path is captured. The coefficients of this polynomial are then used as parameters in the optimization problem.

The images are collected when running PID and MPC controllers on the car and each image consists of the ground truth data that includes timestamps, position, orientation, throttle and steering inputs, breaking, and speed. The dataset created is then separated into test, training, and validation datasets along with their labels to create three splits in the ratio of 7:2:1.

The total size of the annotated dataset collected using the MPC controller is 1017 images. Also for the PID controller 1825 images with annotation files.

3.1. Optimization using PID

We first define error terms for both throttle and steering. The error is the difference between the desired state (setpoint) and the current state of the system. For example, velocity error, cross-track error, and heading error are used as errors for the throttle and steering control of the car.

The proportional component is immediately responsive to the current error, dictating the necessary adjustments for acceleration or deceleration in the case of throttle, or the required degree of turning based on lateral distance error in steering. The integral component addresses the cumulative effect of past errors, mitigating persistent steady-state errors. In throttle control, it incorporates a cumulative speed error, while in steering, it accounts for the accumulated lateral distance error.

On the other hand, the derivative component is forward-looking, factoring in the rate of change of error to anticipate future behavior. In throttle control, it assesses the speed error's changing rate, while in steering, it gauges the rate of change in lateral distance error.

We then combine the proportional, integral, and derivative components to calculate the controller output for both throttle and steering. The PID control output $u(t)$ is then computed as follows, where K_p is the proportional gain, K_i is the integral gain, K_d is the derivative gain, $e(t)$ is the error value calculated for steering and throttle, $de(t)$ is the change in error and $d(t)$ is the change in time:

$$u(t) = K_p e(t) + K_i \int_0^t e(\tau) d\tau + K_d \frac{de(t)}{dt}$$

The control outputs are then ensured that they are within acceptable limits. This means limiting the control output at each iteration to prevent excessive acceleration or steering, which is important for vehicle stability. Finally, the PID gains are fine-tuned to achieve the desired trade-off between responsiveness and stability.

Figure 3: Sample Frame from the Collected Dataset

3.2. Optimization using MPC

Model Predictive Control (MPC) is an advanced control strategy used in various engineering applications, including autonomous vehicles. MPC [1] operates by repeatedly solving an optimization problem over a finite time horizon, where the optimization variables are the control inputs. It applies the best control inputs that minimize a cost function and makes predictions about future states based on the present state using a dynamic model of the system. The process is repeated at each time step, making MPC a receding horizon control strategy.

The idea is to predict how the system of the car will behave in the next N steps. The MPC then returns actuators (steer and throttle) for the next N steps, but only actuators predicted for the next time step are used for control.

The cost function in the MPC is formulated based on the error between the actual vehicle state and the desired trajectory represented by the polynomial curve. The values of the actuators are then determined by minimizing a cost function given below:

$$\text{cost} = \sum_{i=1}^{N} \{c_1 \cdot \text{CTE}_i^2 + c_2 \cdot (V_i - V_0)^2 + c_3 \cdot \delta_i^2 + c_4 \cdot a_i^2 + \ldots\}$$

where δ and a are the steering angle and throttle (actually, acceleration), the c_i are coefficients chosen by the user, specifically, coefficients for penalizing consecutive acceleration differences and steering angle differences, V_0 is the desired speed at which we wish the vehicle to operate.

The MPC minimizes the cost subject to a set of constraints dictated by the kinematic model, which makes this a nonlinear optimization problem. We use the SLSQP

(Sequential Least Squares Programming) method available in the Scipy.optimize.minimize function for solving this problem. These constraints ensure that the vehicle's state evolves according to the vehicle dynamics and follows the desired trajectory.

Once we find an optimal set of actuators, $((\delta_i, a_i))$, only the first pair, (δ_1, a_1), is used to control the car. Finally, the optimal control inputs obtained from the optimization are used to update the control commands for the vehicle. These control commands are then sent to the simulated vehicle to execute the desired trajectory.

4. Method

To enhance autonomous vehicle control using Neural Network-Based Control in the AirSim simulator, we are following a systematic approach. This approach aims to augment the conventional PID and Model predictive controller with adaptive multi-layered neural networks to improve steering inputs for the vehicle. The reason why our method is better than the other imitation learning methods is that it imitates a PID or Model predictive controller behaviour rather than a human (keyboard/Gamepad). This provides us with a lot better distribution of data free of any sort of bias and also gives a jittery-free output after training. The neural network architecture used in this method also uses an architecture that is well-suited for tasks that benefit from both detailed image analysis and supplemental information that cannot be gleaned from the image alone. This is because the architecture combines convolutional layers, which are adept at processing and extracting features from image data, with the capability to integrate additional non-image data. The variety of Hyperparameters used also helps in the generalizability of the model over different scenarios.

4.1. Input/Output of the model

The neural network model is fed sensor data from the AirSim simulator, notably pictures from the simulated vehicle's front-facing camera. The model's output will be the vehicle's steering instructions, which will be continuous values corresponding to the steering angle required to negotiate the course.

4.2. Evaluation metrics

The neural network model's performance is assessed using the Mean Squared Error (MSE) between the predicted steering angles and the ground truth values from the dataset. Furthermore, the model's capacity to effectively manoeuvre the car around the course without human involvement is used as a qualitative performance indicator.

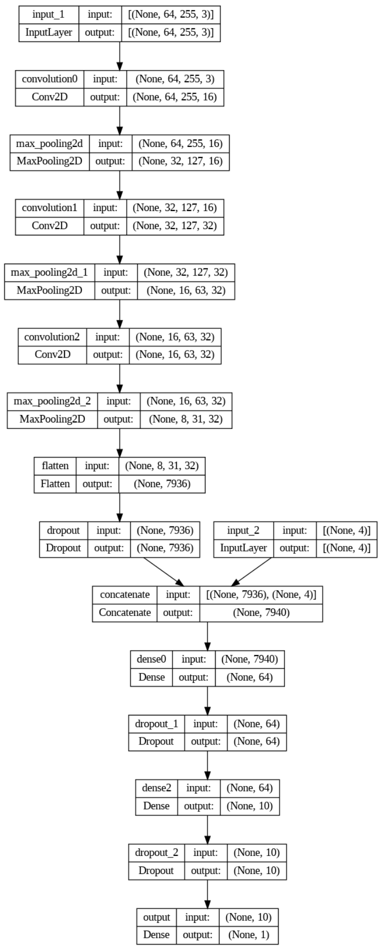

input_1 | input: | [(None, 64, 255, 3)]
InputLayer | output: | [(None, 64, 255, 3)]
convolution0 | input: | (None, 64, 255, 3)
Conv2D | output: | (None, 64, 255, 16)
max_pooling2d | input: | (None, 64, 255, 16)
MaxPooling2D | output: | (None, 32, 127, 16)
convolution1 | input: | (None, 32, 127, 16)
Conv2D | output: | (None, 32, 127, 32)
max_pooling2d_1 | input: | (None, 32, 127, 32)
MaxPooling2D | output: | (None, 16, 63, 32)
convolution2 | input: | (None, 16, 63, 32)
Conv2D | output: | (None, 16, 63, 32)
max_pooling2d_2 | input: | (None, 16, 63, 32)
MaxPooling2D | output: | (None, 8, 31, 32)
flatten | input: | (None, 8, 31, 32)
Flatten | output: | (None, 7936)
dropout | input: | (None, 7936)
Dropout | output: | (None, 7936)
input_2 | input: | [(None, 4)]
InputLayer | output: | [(None, 4)]
concatenate | input: | [(None, 7936), (None, 4)]
Concatenate | output: | (None, 7940)
dense0 | input: | (None, 7940)
Dense | output: | (None, 64)
dropout_1 | input: | (None, 64)
Dropout | output: | (None, 64)
dense2 | input: | (None, 64)
Dense | output: | (None, 10)
dropout_2 | input: | (None, 10)
Dropout | output: | (None, 10)
output | input: | (None, 10)
Dense | output: | (None, 1)

Figure 4: Architecture diagram

4.3. Neural Network Architecture

We have designed a multi-layered neural network architecture 4 suitable for control tasks. The final architecture of the framework remains abstract, however, the neural network using deep learning frameworks will be implemented. The architecture depicted in the image is a convolutional neural network (CNN) with a multi-input structure, which is often used for tasks that require the integration of different types of data. Here is a detailed explanation of the architecture:

- Input Layers:

  Input 1: This layer accepts input images with a shape of (64, 255, 3), which corresponds to the width, height, and color channels (RGB) of the images.
  Input 2: This layer takes in a separate input vector of length 4, which could be a form of numerical data or embeddings that are meant to be used alongside the image data.

- First Convolutional Block:

  Convolution 0: A Conv 2D layer that applies 16 filters to the input image, maintaining the image dimensions due to padding.
  Max pooling 2d: A MaxPooling 2D layer that reduces the spatial dimensions by half, to (32, 127), while keeping the same number of channels (16).

- Second Convolutional Block:

  Convolution 1: Another Conv 2D layer with 32 filters, keeping the spatial dimensions the same.
  Max pooling 2d 1: A Max Pooling 2D layer reducing the spatial dimensions further to (16, 63), with 32 channels.

- Third Convolutional Block:

  Convolution 2: A Conv 2D layer with 32 filters that do not alter the spatial dimensions due to padding.
  Max pooling 2d 2: The last Max Pooling 2D layer in the sequence, which downsamples the spatial dimensions to (8, 31), maintaining 32 channels.

- Flattening and Dropout:

  Flatten: This layer flattens the 3D output from the previous layer into a 1D vector with 7936 features.
  Dropout: A Dropout layer is applied to the flattened output to prevent overfitting by randomly setting a fraction of the input units to 0 during training.

- Concatenation:

  After the image data is processed through convolution and pooling layers, its output is concatenated with the output of input 2. This suggests that the model is designed to learn from both image data and additional numerical data.

- Dense Layers and Dropout:

  Dense0: A fully connected (Dense) layer that reduces the number of features to 64.
  Dropout 1: Another Dropout layer is applied after the first Dense layer. Dense2: Another Dense layer that outputs 10 features, which could correspond to the number of classes in a classification task.
  Dropout 2: A final Dropout layer is applied to the output of the second Dense layer. Output Layer:

- Output: The final Dense layer with a single unit, which might be used for binary classification tasks. The output may be a probability that the input belongs to a certain class, which is commonly used with a sigmoid activation function for binary classification, as suggested by the usage of a single unit.

4.4. Training the Neural Network

We have split the data into training, validation, and test sets. Using the training data, we will train the neural network so it can learn to predict the appropriate steering commands given the current vehicle state and environment perception. The input for the model is the RGB [1] images and state variables like steering, throttle, speed, and brake. For training as the model deals with feature extraction and we do not want any sort of bias or mislearning to happen because of the background sceneries a relevant ROI is selected. The process of selecting a correct ROI has also been challenging as a lot of iterations of training had to be made to fine-tune this to get an implementation from the model. The network is trained for a maximum of 500 epochs, however, due to early stopping (patience = 30) the model only trained for 258 epochs for the data procured by the model predictive controller. We kept monitoring the training and validation losses under check to draw the loss curves.

5. Experiments

We tested the network on a wide range of scenarios to ensure it can handle various driving conditions and challenges.

5.1. Fine-Tuning and Optimization

We did not play around much with the dropout and consistently kept it low, as the size of our dataset was not very huge. Also, we had early stopping to save our time and avoid overfitting so the dropout value was kept consistent. Also for optimization the Nadam optimizer was used

Experiment 1 We first collected the dataset using the keyboard however the distribution of steering data using the keyboard was in extremes,i.e. We obtained only three values 0.0, 0.5, and -0.5. This led the training to produce inefficient models which could not imitate the behavior properly.

5.2. Experiment 2

We designed a PID controller using LIDAR data to detect the nearest obstacle. After getting this data the controller used to provide a steering angle such that the car moves in the opposite direction to that of the nearest obstacle. Although we could produce a small dataset without any collision, however, this dataset was not enough in size to get a generalizable model after training. Also taking sharp turns using this method is not possible hence we had to find an alternative. This alternative was to input waypoints which are a collection of car poses on a track. The PID controller had the task of going from one waypoint to another while also getting the current state data. This process was more viable and produced better datasets. The same idea is then extended to Model predictive controllers and they can produce even better results with their computationally expensive optimization methods.

5.3. Experiment 3

Tuning the zero percentage drop was a tedious task as the right value for this parameter would be different for different datasets. However, for most datasets, we concluded that the correct range for this parameter is between 75-95 percent. For the MPC data which had a good distribution, the perfect model was obtained keeping this parameter value as 95.

5.4. Experiment 4

ROI selection was another challenge as well. We started by selecting an ROI of [76,140;0,255] pixels in figure 5, which produced decent results. However, ROI [50:140;0:255] pixels in figure 6 also produced good results as the height of the image input in the model increased the model could extract features of the lane slightly farther away from the car and thus gave better predictions most of the time. As seen in figure 7a and figure 7b the model may be learning from the training data because the photos demonstrate how the loss and validation loss decrease with time. The loss curves show that most learning happens in the earlier epochs; they start higher, decline abruptly, and subsequently level out. Good generalization is shown by the fact that loss and validation loss fall simultaneously and stay relatively near to one another, indicating that the model is not overfitting to the training set.

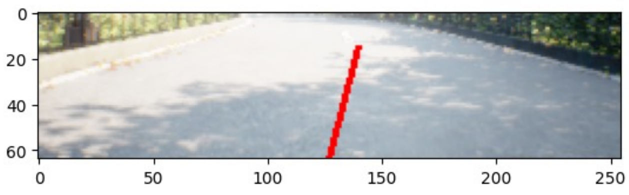


Figure 5: Smaller Region of Interest (ROI)

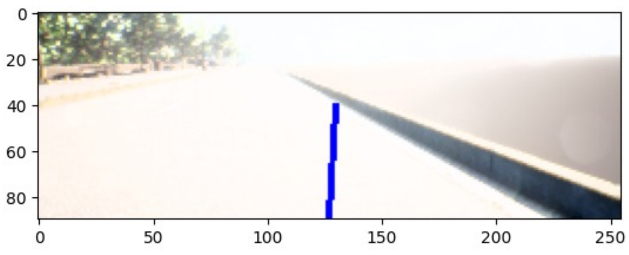
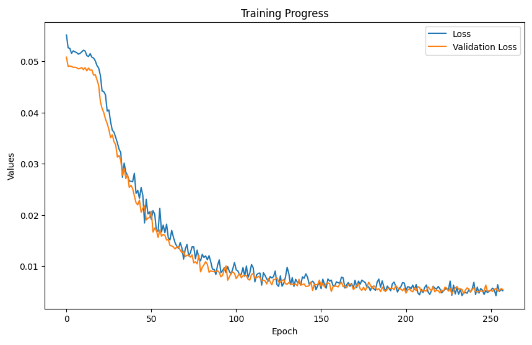
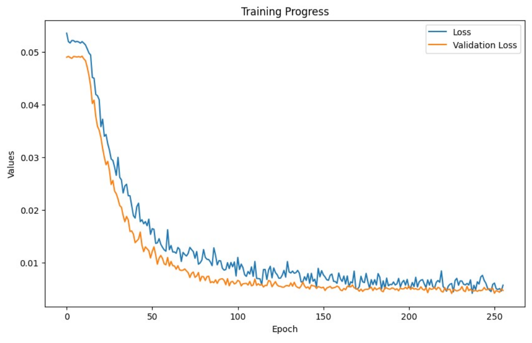


Figure 6: Larger Region of Interest (ROI)



(a) Loss Curve for Training with Larger ROI



(b) Loss Curve for Training with Smaller ROI

Figure 7: Loss graphs

6. Limitations

- **Cross road decision making**
  - The car struggles to decide at the crossroad where to turn although to facilitate the process this process we have intentionally given a bias in our data to take a right turn always in crossroad scenarios however, it does not convincingly take any sharp turns at the crossroad.
  - The car struggles to climb uphill without colliding on either side of the road. This is because the images in uphill scenarios are not up to mark. After all, most of the region of interest is covered by the road itself, unlike other cases where the image has lanes on both sides visible in the ROI.
- **Simulation-to-Reality Gap:**
  - The model performs well in scenarios that closely resemble the conditions encountered during training in the simulation.
  - Performance may degrade when exposed to real-world conditions that deviate significantly from the simulated environment.

- **Environmental Variability:**
  - The model adapts effectively to a diverse range of environmental conditions encountered during training.
  - Extreme or uncommon scenarios not adequately represented in the training data may challenge the model's generalization.
- **Real-time Challenges:**
  - The model responds effectively to dynamic scenarios within the constraints of its training setup.
  - Real-time adaptability may be constrained by the model's training environment, impacting responses to rapid changes in road conditions.
- **Sensor Dependency:**
  - The model effectively utilizes vision-based perception for decision-making.
  - Limitations may arise when the system encounters scenarios where additional sensor modalities, beyond vision, are crucial for comprehensive understanding.
- **Adversarial Scenarios:**
  - The model demonstrates resilience to common driving challenges.
  - Limitations may become apparent when the model encounters intentional adversarial attacks or scenarios not well-represented in the training set.

7. Conclusion

In conclusion, this project successfully integrates imitation learning enhanced by a Model predictive controller for autonomous driving in a simulated environment, showcasing a meticulous and comprehensive approach to neural network architecture, training, and deployment.

The hybrid neural network architecture, combining convolutional and dense layers, has demonstrated its efficacy in synthesizing visual and contextual information, thereby proving its potential for informed decision-making. Techniques such as dropout, data augmentation, and specific ROI selection have been incorporated to address challenges inherent in real-world applications and diverse dataset characteristics.

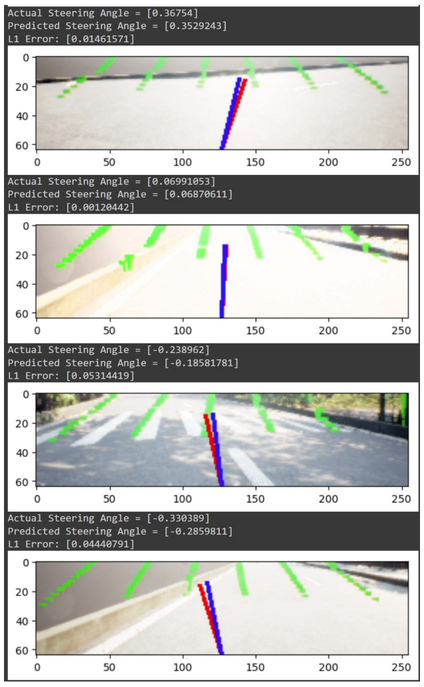
The model's robustness and adaptability are further underscored by the thoughtful integration of callbacks, optimization algorithms, and a systematic hyperparameter tuning approach, including the use of grid search. These practices significantly contribute to enhancing the model's reliability and robustness.

As the project progresses, the identified challenges in ROI selection, hyperparameter tuning, and model architecture serve as valuable guideposts for future exploration. Addressing these challenges is crucial for achieving seamless transitions from simulation to reality. Future work could involve employing more advanced techniques for ROI selection, exploring alternative architectures to enhance generalization, and conducting a rigorous analysis of interpretability and robustness in diverse driving scenarios. The model still struggles to make decisions at crossroads driving uphill is still a struggle however, taking gradual and sharp turns is no issue at all. The jittery-free execution shows the capabilities of our solution.

The inclusion of real-time adaptation and learning will be pivotal for addressing dynamic and unpredictable road scenarios. By acknowledging these challenges and proposing avenues for future exploration, this project contributes to the current body of knowledge and establishes a strong foundation for ongoing research and development in the dynamic field of deep learning for autonomous vehicles.

7.1. Testing

The figure showcases the testing results on the testing data, as can be seen, the accuracy is high and this is done by using an L1 error for its computation. The red line determines the actual steering angle blue stands for the predicted steering angle and green is the LIDAR point clouds 8 being detected.


```
Actual Steering Angle = [0.36754]
Predicted Steering Angle = [0.3529243]
L1 Error: [0.01461571]

Actual Steering Angle = [0.06991053]
Predicted Steering Angle = [0.06870611]
L1 Error: [0.00120442]

Actual Steering Angle = [-0.238962]
Predicted Steering Angle = [-0.18581781]
L1 Error: [0.05314419]

Actual Steering Angle = [-0.330389]
Predicted Steering Angle = [-0.2859811]
L1 Error: [0.04440791]
```

Figure 8: Testing Results

8. Going above and beyond

One of the major dilemmas we faced was to make a model that could make concrete decisions in case of crossroads. Often the model we trained on the PID controller is used to give varied steering outputs when the car is at the crossroads. Due, to this the car was never able to take a sharp turn at intersections if the two steering inputs were positive then the other two steering inputs would be positive. This made the car get stuck in between and collide in front. To stop this from happening the data we collected using a Model predictive controller ensured that the car always takes a left turn on any type of crossroads. This way we intentionally instilled a bias in the dataset. Although this is not a method applicable in real-life scenarios, however, in real life, the car would also have the destination data and path to follow to reach that destination making the decision-making process straightforward. However, without any such data, we were able to replicate the behaviour of the model predictive controller and the car many times successfully took left turns at crossroads.**Video of one such case we have put in the supplementary material.**

9. Contributions:

- Nishant Pandey - Architecture design and coding of the model and report
- Hrugved Pawar - Data preparation and report
- Rishikesh Jadhav: PID controller and Testing of the model
- Aaqib Barodawala- model predictive controller, architecture designing and testing of the data

References

[1] Carla Simulator, ”carla/racetrack/PythonClient/racetrack,” 2023. Link

[2] Microsoft, ”AutonomousDrivingCookbook/AirSimE2EDeepLearning,” 2023. Link

[3] Albert Shalumov et al 2021 Bioinspir. Biomim. 16 066016 LiDAR-driven spiking neural network for collision avoidance in autonomous drivingLink

[4] Deep convolutional neural network based autonomous drone navigation Karim Amer,1 Mohamed Samy,1 Mahmoud Shaker,1 Mohamed ElHelw1 1Nile Univ. (Egypt)

[5] X. Chen, S. Leng, J. He and L. Zhou, ”Deep-Learning-Based Intelligent Intervehicle Distance Control for 6G-Enabled Cooperative Autonomous Driving,” in IEEE Internet of Things Journal, vol. 8, no. 20, pp. 15180-15190, 15 Oct.15, 2021, doi: 10.1109/JIOT.2020.3048050.

[6] iADA*-RL: Anytime Graph-Based Path Planning with Deep Reinforcement Learning for an Autonomous UAV Konkuk Aerospace Design-Airworthiness Research Institute (KADA), Konkuk University, Seoul 05029, Korea 2 Department of Aerospace Engineering, Konkuk University, Seoul 05029, Korea

[7] V. Sadhu, S. Zonouz and D. Pompili, ”On-board Deep-learning-based Unmanned Aerial Vehicle Fault Cause Detection and Identification,” 2020 IEEE International Conference on Robotics and Automation (ICRA), Paris, France, 2020, pp. 5255-5261, doi: 10.1109/ICRA40945.2020.9197071.

[8] Chapter 15 - Deep learning for vision-based navigation in autonomous drone racing Author links open overlay panelHuy Xuan Pham a c, Halil Ibrahim Ugurlu a c, Jonas Le Fevre a, Deniz Bardakci a b, Erdal Kayacan a

[9] J. G. C. Zuluaga, J. P. Leidig, C. Trefftz and G. Wolffe, ”Deep Reinforcement Learning for Autonomous Search and Rescue,” NAECON 2018 - IEEE National Aerospace and Electronics Conference, Dayton, OH, USA, 2018, pp. 521-524, doi: 10.1109/NAECON.2018.8556642.

[10] What is Unreal Engine 4, pp. 1, 2017, [online] Available: https://www.unrealengine.com/en-US/whatis-unreal-engine-4.

[11] R. S. Sutton and A. G. Barto, ”Reinforcement learning: an introduction”, UCLComputer Science Department Reinforcement Learning Lectures, pp. 1054, 2017, [online] Available: Link.

[12] F. Zhong, W. Qiu, T. Yan, A. Yuille and Y. Wang, Gym-UnrealCV: Realistic virtual worlds for visual reinforcement learning, 2017

[13] K. Farkhodov, S. -H. Lee, J. Platos and K. -R. Kwon, ”Deep Reinforcement Learning Tf-Agent-Based Object Tracking With Virtual Autonomous Drone in a Game Engine,” in IEEE Access, vol. 11, pp. 124129-124138, 2023, doi: 10.1109/ACCESS.2023.3325062.

[14] Neural Network Based Model Predictive Control for an Autonomous Vehicle, Vianna et al., 2021

[15] A Survey of Deep Learning Applications to Autonomous Vehicle Control, Sampo Kuutti, Richard Bowden, Yaochu Jin, Phil Barber, and Saber Fallah (Ku- utti et al., 2019)