Lateral Control for Autonomous Vehicles

A Project Report Submitted by

Udit Kandpal (M24CSE027)

Om Patel (M24CSA019)

Vishvaskumar Patel (M24CSE029)

Rahul Maurya (M24CSA025)

Patil Divya Kailash (M24CSE018)

Indian Institute of Technology Jodhpur

Computer Science Department

April, 2025

Contents

List of Figures

List of Tables

Abstract

This project targets the improvement of the lateral control system of self-driving cars through the incorporation of state-of-the-art deep learning methods for perception and control. The system uses convolutional neural networks (CNNs) and Vision Transformers (ViTs) to carry out lane detection and object detection based on camera-based data. These perception units are input to the lateral control logic, which steers the vehicle to ensure safe and accurate lane keeping. By integrating visual understanding in real-time with exact control algorithms, the project promises to enhance stability and responsiveness in autonomous driving under various driving environments. Experimental studies prove the suitability of CNN and ViT perception pipelines in efficient lane detection and dynamic object recognition, allowing robust and smooth lateral control in scenarios involving complex road conditions.

Objective

This project aims to develop and evaluate deep learning-based lateral control systems for autonomous vehicles. The primary objectives include:

- Implementing and comparing the performance of Convolutional Neural Networks (CNNs) and Vision Transformers (ViTs) for lane detection and steering angle prediction using simulated driving datasets (Udacity and AirsimNH).
- Designing and applying data preprocessing and augmentation techniques to enhance the robustness and generalization of the models.
- Developing a multi-task learning model to simultaneously predict steering and throttle control.
- Implementing steering smoothing and throttle reduction techniques to improve the stability and safety of the autonomous vehicle control.
- Deploying the trained model in a web application for demonstration and evaluation.

1 Overview of Simulators

1.1 Udacity Simulator

This simulator, often integrated with Udacity's online courses, provides focused environments and scenarios designed for learning specific self-driving concepts and algorithms. It typically features simplified physics and sensor models, prioritizing pedagogical clarity and ease of use for educational purposes. The simulator often utilizes Python and libraries specific to the Udacity curriculum.

Figure 1.1: Udacity Stimulator

1.2 AirSimNH Simulator

Based on Microsoft AirSim, this simulator offers a more photorealistic and complex environment with detailed physics and support for a wider range of realistic sensors like lidar and radar. It allows for greater flexibility in creating custom scenarios and integrates well with industry-standard tools and frameworks such as ROS and Unreal Engine, making it suitable for more advanced research and development. AirSimNH enables the simulation of diverse and challenging real-world conditions.

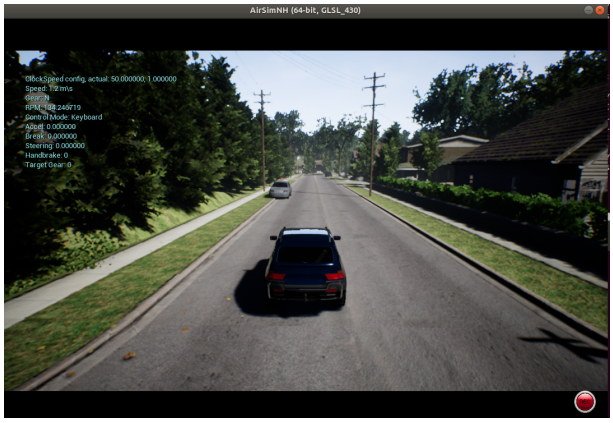

Figure 1.2: AirsimNH Stimulator

2 Dataset: Udacity

The dataset includes images captured from three different perspectives: center, left, and right cameras of a simulated vehicle, along with corresponding driving data.

2.1 Image Data

- **Center Camera:** Positioned directly at the center of the simulated vehicle, capturing the primary view of the road.
- **Left Camera:** Positioned on the left, providing a view that could be helpful for the model to understand potential off-center positions.
- **Right Camera:** Positioned on the right, assisting the model in learning recovery maneuvers if the vehicle is off-center to the right.

2.2 Driving Data

- **Steering Angle** The primary label for training, representing the angle of the vehicle's steering wheel.
- **Throttle** Indicates the acceleration at each frame.
- **Reverse** A binary feature indicating whether the vehicle is in reverse mode.
- **Speed** Speed data that affects the throttle control.

2.3 Preprocessing Steps

The following preprocessing steps were applied to the images to prepare them for model training:

1. **Balancing:** To address the skew in steering angles (with a large portion centered around zero), a balancing technique was applied. The steering angles were divided into 25 bins, and the number of samples in bins with excessive representation were reduced to a target of 400 samples per bin.

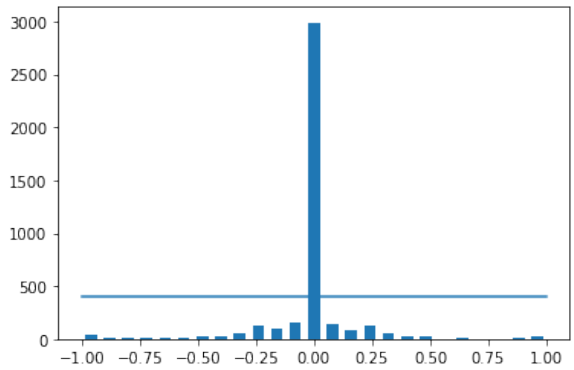

Figure 2.1: Udacity Dataset before Balancing

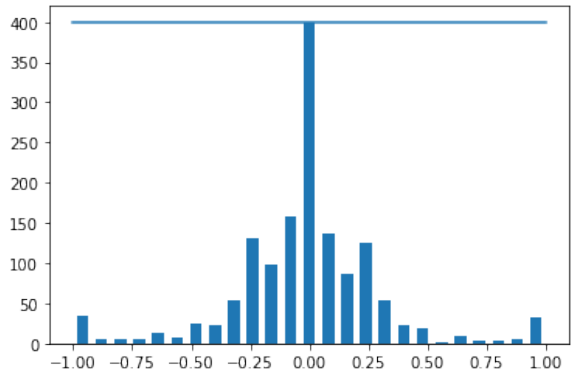

Figure 2.2: Udacity Dataset after Balancing

2. **Cropping:** Unnecessary portions of the image, such as the sky and car hood, were cropped out to focus on the relevant road section.

3. **Color Space Conversion:** The images were transformed to the YUV color space to enhance road features.

4. **Gaussian Blur:** A Gaussian blur was applied to reduce noise and smooth the images.

5. **Resizing:** The images were resized to 200×66 pixels to match the input dimensions of the CNN model.

6. **Pixel Value Normalization:** Pixel values were normalized by dividing by 255 to scale them to the range $[0, 1]$.

2.4 Data Augmentation

Data augmentation techniques were employed to enhance the robustness of the model:

1. **Zooming:** Images were slightly enlarged to simulate the car approaching closer to the scene.

2. **Panning:** Images were shifted slightly in the horizontal or vertical direction to simulate lateral or forward shifts.

3. **Random Brightness Adjustment:** The brightness of the images was randomly adjusted to mimic varying lighting conditions.

4. **Horizontal Flipping:** Images were flipped horizontally to simulate scenarios where the car might be approaching a curve or making turns.

The dataset undergoes several transformations, summarized in Table 2.1.

Table 2.1: Dataset Transformations

Transformation	Number of Samples
Total initial data	4053
Data removed for balancing	2590
Remaining data after balancing	1463
Training set after data augmentation	3511
Validation set after data augmentation	878

The dataset is split into training and validation sets to facilitate model development and evaluation. The initial dataset contains images from the center, left, and right cameras. After balancing and augmenting the data, the dataset is expanded to include multiple views and variations, significantly increasing the total number of images.

3 Dataset: AirsimNH

Since there is no publicly available dataset for this simulator, we generated our own dataset consisting of 34,707 images. The AirsimNH dataset comprises the following information:

3.1 Image Data

These images were captured from the simulated environment, with only one centered image captured per frame from the simulator.

3.2 Driving Data

- **Vehicle Name:** Identifier for the vehicle in the simulation.
- **Throttle** Indicates the acceleration applied to the vehicle.
- **Steering** The steering angle applied to the vehicle, which is the primary label for training.
- **ImageFile** The file path or identifier for the corresponding image.

3.3 Balancing

The initial dataset exhibited a skew in the distribution of steering angles. To mitigate this, a balancing step was applied by reducing the number of samples for the predominant steering angle (0.0). The distribution of steering angles before and after balancing is summarized in Table 3.1.

Table 3.1: AirsimNH Steering Angle Distribution Before and After Balancing

Steering Angle	Value Counts (Initial)	Value Counts (Balanced)
-0.5	1124	1124
0.0	26012	15000
0.5	7571	7571

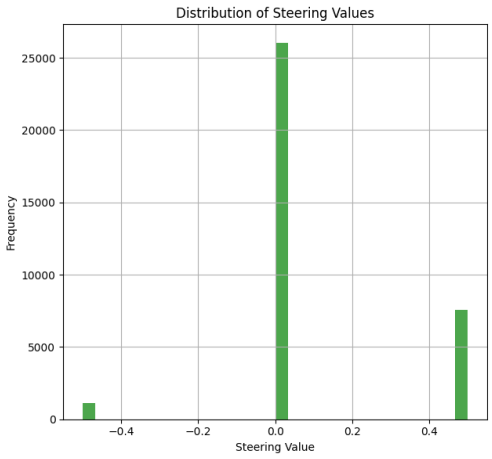

Figure 3.1: AirsimNH Dataset before Balancing

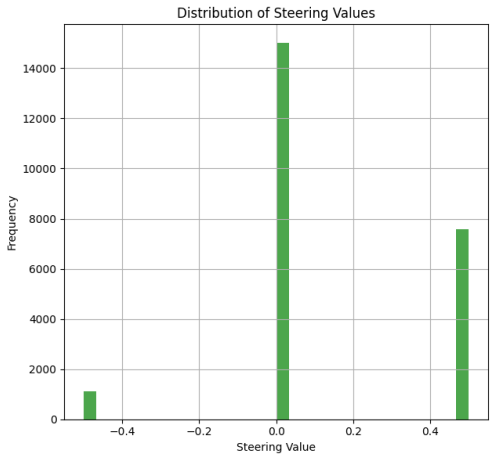

Figure 3.2: AirsimNH Dataset after Balancing

4 Approach 1: CNN (Udacity)

The first approach to autonomous driving implemented on the Udacity simulator in this project uses a Convolutional Neural Network (CNN). CNNs are well-suited for image-based tasks due to their ability to automatically learn hierarchical features from raw pixel data.

4.1 Model Architecture: NVIDIA CNN with Modifications

The CNN architecture used in this project is based on the NVIDIA "End to End Learning for Self-Driving Cars" model. The original NVIDIA model was designed to map raw pixels from a single front-facing camera directly to steering commands. It consists of a series of convolutional layers that extract increasingly complex features from the input images, followed by fully connected layers that output the predicted steering angle.

Table 4.1: CNN Model Architecture

Layer (Type)	Output Shape	Param #
conv2d (Conv2D)	(None, 31, 98, 24)	1,824
conv2d_1 (Conv2D)	(None, 14, 47, 36)	21,636
conv2d_2 (Conv2D)	(None, 5, 22, 48)	43,248
conv2d_3 (Conv2D)	(None, 1, 18, 64)	76,864
flatten (Flatten)	(None, 1152)	0
dense (Dense)	(None, 100)	115,300
dense_1 (Dense)	(None, 50)	5,050
dense_2 (Dense)	(None, 10)	510
dense_3 (Dense)	(None, 1)	11
Total params:		**264,443 (1.01 MB)**
Trainable params:		**264,443 (1.01 MB)**
Non-trainable params:		**0 (0.00 B)**

4.2 Results

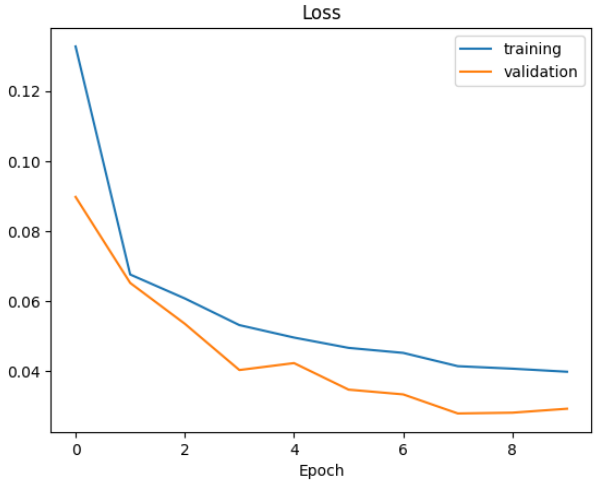

Figure 4.1: Result of CNN On udacity Stimulator

5 Approach 2 - Autoencoders (Udacity)

The second approach to autonomous driving implemented on the Udacity simulator uses autoencoders in conjunction with a CNN.

5.1 Model Architecture: Autoencoders

In this approach, an autoencoder is used to process the input images. The encoder part of the autoencoder is then integrated into the CNN architecture. This allows the CNN to work with a more refined feature representation learned by the autoencoder.

Table 5.1: Autoencoder Architecture

Layer (Type)	Output Shape	Param #
input_layer_3 (InputLayer)	(None, 100, 320, 3)	0
conv2d_11 (Conv2D)	(None, 100, 320, 8)	224
max_pooling2d_4 (MaxPooling2D)	(None, 50, 160, 8)	0
conv2d_12 (Conv2D)	(None, 50, 160, 8)	584
up_sampling2d_4 (UpSampling2D)	(None, 100, 320, 8)	0
conv2d_13 (Conv2D)	(None, 100, 320, 3)	219
Total params:		**1,027 (4.01 KB)**
Trainable params:		**1,027 (4.01 KB)**
Non-trainable params:		**0 (0.00 B)**

5.2 Results

The CNN with autoencoders achieved a lower validation loss compared to the CNN-only model. This indicates that the enhanced feature representation from the autoencoder helped the model make more accurate predictions.

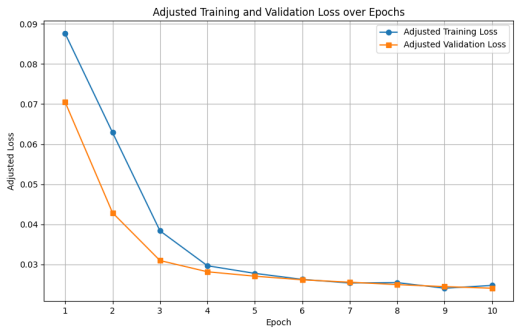

Figure 5.1: Result of Autoencoders On udacity Stimulator

6 Approach 3: CNN On AirsimNH (Predicting only Sterring)

The third approach to autonomous driving implemented in this project uses a Convolutional Neural Network (CNN). This model is designed for end-to-end steering angle prediction from RGB camera images in an autonomous driving simulation environment (Airsim). The architecture consists of a feature extraction module followed by a regression module, with weight initialization for improved training stability.

6.1 CNN Architecture

Table 6.1: Detailed SteeringCNN Architecture

Layer (Type)	Output Shape	Param #
Conv2d-1	[-1, 32, 64, 64]	2,432
BatchNorm2d-2	[-1, 32, 64, 64]	64
ReLU-3	[-1, 32, 64, 64]	0
MaxPool2d-4	[-1, 32, 32, 32]	0
Conv2d-5	[-1, 64, 32, 32]	18,496
BatchNorm2d-6	[-1, 64, 32, 32]	128
ReLU-7	[-1, 64, 32, 32]	0
MaxPool2d-8	[-1, 64, 16, 16]	0
Conv2d-9	[-1, 128, 16, 16]	73,856
BatchNorm2d-10	[-1, 128, 16, 16]	256
ReLU-11	[-1, 128, 16, 16]	0
MaxPool2d-12	[-1, 128, 8, 8]	0
Conv2d-13	[-1, 256, 8, 8]	295,168
BatchNorm2d-14	[-1, 256, 8, 8]	512
ReLU-15	[-1, 256, 8, 8]	0
MaxPool2d-16	[-1, 256, 4, 4]	0
AdaptiveAvgPool2d-17	[-1, 256, 4, 4]	0
Linear-18	[-1, 512]	2,097,664
BatchNorm1d-19	[-1, 512]	1,024
ReLU-20	[-1, 512]	0
Dropout-21	[-1, 512]	0
Linear-22	[-1, 256]	131,328
BatchNorm1d-23	[-1, 256]	512
ReLU-24	[-1, 256]	0
Dropout-25	[-1, 256]	0
Linear-26	[-1, 1]	257
Total params:		**2,621,697**
Trainable params:		**2,621,697**
Non-trainable params:		**0**

6.2 Steering-Dependent Throttle Reduction:

The throttle is reduced proportionally to the magnitude of the steering angle to slow the vehicle during turns. The reduction is governed by a parameter, $R = 0.2$, termed the steering throttle reduction factor. The adjusted throttle, T_{adjusted}, is calculated using the formula:

$$T_{\text{adjusted}} = T_{\text{base}} \cdot (1 - |\theta| \cdot R)$$

where:

- θ is the smoothed and clipped steering angle, $\theta \in [-1.0, 1.0]$,
- $|\theta|$ is the absolute value of the steering angle,
- $R = 0.2$ is the reduction factor.

6.3 Steering Smoothing:

The raw steering prediction from the CNN is smoothed using an exponential moving average to reduce jitter and ensure smooth control transitions. The smoothing is implemented as:

$$\theta_{\text{smoothed}} = \alpha \cdot \theta_{\text{new}} + (1 - \alpha) \cdot \theta_{\text{prev}}$$

where:

- $\alpha = 0.3$ is the smoothing factor,
- θ_{new} is the current predicted steering angle,
- θ_{prev} is the previously smoothed steering angle.

The smoothed steering angle, θ_{smoothed}, is clipped to the range [-1.0, 1.0] to ensure it remains within the vehicle's steering limits.

6.4 Result

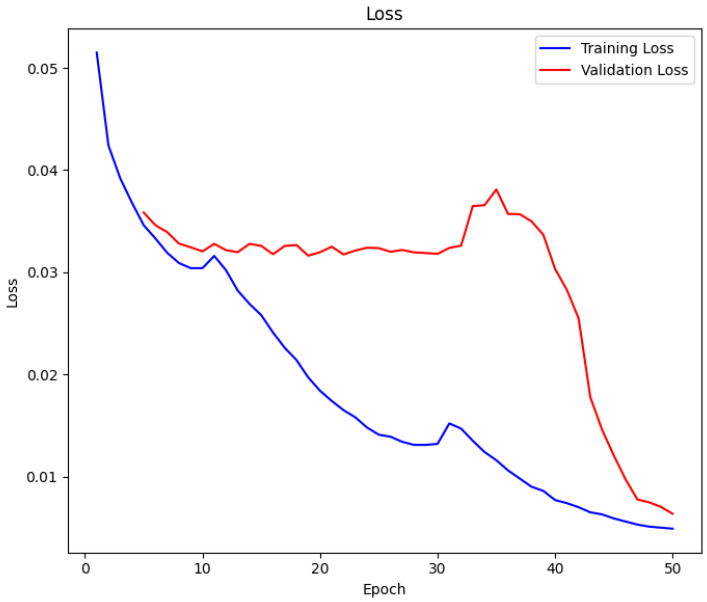

Figure 6.1: Result of CNN On AirsimNH Stimulator prediction steering only

7 Approach: 4 CNN On AirsimNH (Predicting Steering and Throttle)

The fourth approach implemented in this project uses a Convolutional Neural Network (CNN) to predict both steering and throttle values. This model is designed for end-to-end control of an autonomous vehicle in the Airsim simulation environment.

7.1 CNN Architecture

The CNN architecture for predicting both steering and throttle is detailed in Table 7.1.

Table 7.1: CNN Architecture for Steering and Throttle Prediction

Layer (Type)	Output Shape	Param #
Conv2d-1	[-1, 32, 64, 64]	2,432
BatchNorm2d-2	[-1, 32, 64, 64]	64
ReLU-3	[-1, 32, 64, 64]	0
MaxPool2d-4	[-1, 32, 32, 32]	0
Conv2d-5	[-1, 64, 32, 32]	18,496
BatchNorm2d-6	[-1, 64, 32, 32]	128
ReLU-7	[-1, 64, 32, 32]	0
MaxPool2d-8	[-1, 64, 16, 16]	0
Conv2d-9	[-1, 128, 16, 16]	73,856
BatchNorm2d-10	[-1, 128, 16, 16]	256
ReLU-11	[-1, 128, 16, 16]	0
MaxPool2d-12	[-1, 128, 8, 8]	0
Conv2d-13	[-1, 256, 8, 8]	295,168
BatchNorm2d-14	[-1, 256, 8, 8]	512
ReLU-15	[-1, 256, 8, 8]	0
MaxPool2d-16	[-1, 256, 4, 4]	0
AdaptiveAvgPool2d-17	[-1, 256, 4, 4]	0
Linear-18	[-1, 512]	2,097,664
BatchNorm1d-19	[-1, 512]	1,024
ReLU-20	[-1, 512]	0
Dropout-21	[-1, 512]	0
Linear-22	[-1, 256]	131,328
BatchNorm1d-23	[-1, 256]	512
ReLU-24	[-1, 256]	0
Dropout-25	[-1, 256]	0
Linear-26 (Steering)	[-1, 1]	257
Linear-27 (Throttle)	[-1, 256]	131,328
BatchNorm1d-28	[-1, 256]	512
ReLU-29	[-1, 256]	0
Dropout-30	[-1, 256]	0
Linear-31 (Throttle)	[-1, 1]	257
Total params:		**2,953,794**
Trainable params:		**2,953,794**
Non-trainable params:		**0**

7.2 Multi-Task Loss Function for Steering and Throttle Prediction

In the proposed model, a custom multi-task loss function is implemented to jointly optimize predictions for steering and throttle control. The loss function uses mean squared error (MSE) to quantify prediction errors for both steering and throttle outputs. The total loss is a weighted sum of the individual steering and throttle losses:

$$\text{Total Loss} = w_s \cdot \text{Steering Loss} + w_t \cdot \text{Throttle Loss}$$

where w_s and w_t are hyperparameters that modulate the relative importance of each task. This multi-task loss formulation ensures that the model optimizes both steering and throttle predictions simultaneously.

7.3 Steering Smoothing:

The same steering smoothing technique was used as in the previous approach.

7.4 Result

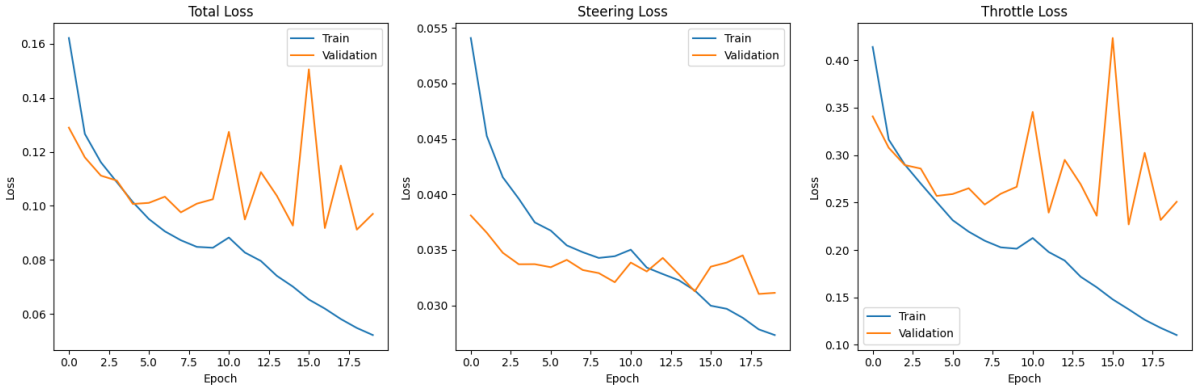

Figure 7.1: Result of CNN On AirsimNH Stimulator prediction steering and Throttle

8 Approach 5: ViT On AirsimNH (Predicting Steering Only)

This research investigates an end-to-end learning approach for predicting steering angles in the AirsimNH simulation environment using a Vision Transformer (ViT) architecture.

8.1 Detailed Layer Summary:

The following table provides a layer-wise breakdown of the network architecture, including approximate parameter counts based on the standard `vit_small_patch16_224` configuration:

Table 8.1: ViT Architecture for Steering Prediction

Layer	Input Shape	Output Shape	Parameters
Backbone (ViT)	∼21.6M		
Patch Embedding	(3, 224, 224)	(196, 384)	∼590K
Transformer Blocks (12)	(196, 384)	(196, 384)	∼21M
Pooling/Norm	(196, 384)	(384,)	∼0.8K
Head			∼132K
LayerNorm	(384,)	(384,)	768
Linear	(384,)	(256,)	98,560
GELU	(256,)	(256,)	0
Dropout (0.3)	(256,)	(256,)	0
Linear	(256,)	(128,)	32,896
GELU	(128,)	(128,)	0
Dropout (0.2)	(128,)	(128,)	0
Linear	(128,)	(1,)	129
Total Parameters			∼21.73M

8.2 Steering Smoothing:

The same steering smoothing technique was used as in the previous approach.

8.3 Steering-Dependent Throttle Reduction:

The same technique was used as in the previous approach.

8.4 Result

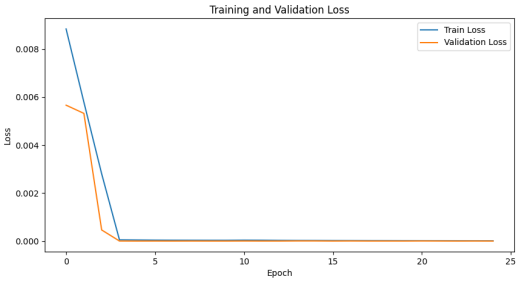

Figure 8.1: Result of ViT On AirsimNH Stimulator prediction steering

9 Approach 6: ViT on AirsimNH (Predicting Steering and Throttle)

This approach utilizes a Vision Transformer (ViT) architecture to simultaneously predict both steering and throttle values for autonomous navigation in the AirsimNH simulation environment. The model takes raw RGB images as input and outputs continuous values for both steering angle and throttle.

Table 9.1: ViT Architecture for Steering and Throttle Prediction

Layer Name	Type	Input Shape	Output Shape
backbone	Vision Transformer (ViT)	(3, 224, 224)	(384,)
shared_head[0]	LayerNorm	(384,)	(384,)
shared_head[1]	Linear	(384,)	(256,)
shared_head[2]	GELU	(256,)	(256,)
shared_head[3]	Dropout (0.3)	(256,)	(256,)
steering_head[0]	Linear	(256,)	(128,)
steering_head[1]	GELU	(128,)	(128,)
steering_head[2]	Dropout (0.2)	(128,)	(128,)
steering_head[3]	Linear	(128,)	(1,)
throttle_head[0]	Linear	(256,)	(128,)
throttle_head[1]	GELU	(128,)	(128,)
throttle_head[2]	Dropout (0.2)	(128,)	(128,)
throttle_head[3]	Linear	(128,)	(1,)
sigmoid	Sigmoid Activation	(1,)	(1,)

9.1 Steering Smoothing:

The same steering smoothing technique was used as in the previous approach.

9.2 Result

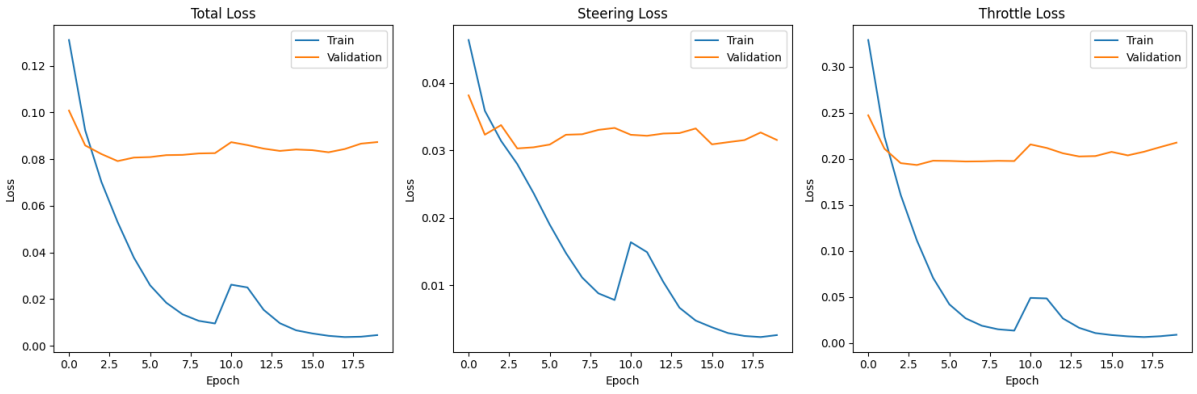

Figure 9.1: Result of ViT On AirsimNH Stimulator prediction steering and Throttle

Deployment

The trained model has been deployed as a web application to allow for interactive demonstration and testing. The application provides a user interface to visualize the model's predictions and control commands. The deployment can be accessed at the following link:
`https://requirementstxt-vhoxy8j2hvuzsqcfbesfxj.streamlit.app/`

Code Repository

The complete source code for this project, including model implementations, training scripts, and data processing utilities, is available on GitHub. The repository contains detailed documentation and instructions for reproducing the experiments and utilizing the codebase. Access the code at:
`https://github.com/vishvaspatel/Lateral-Control-for-Autonomous-Vehicles-Self-Driving-Car-`

Deployed Website Demo

`https://requirementstxt-vhoxy8j2hvuzsqcfbesfxj.streamlit.app/`

Deployment Website ScreenShot

9.3 Result

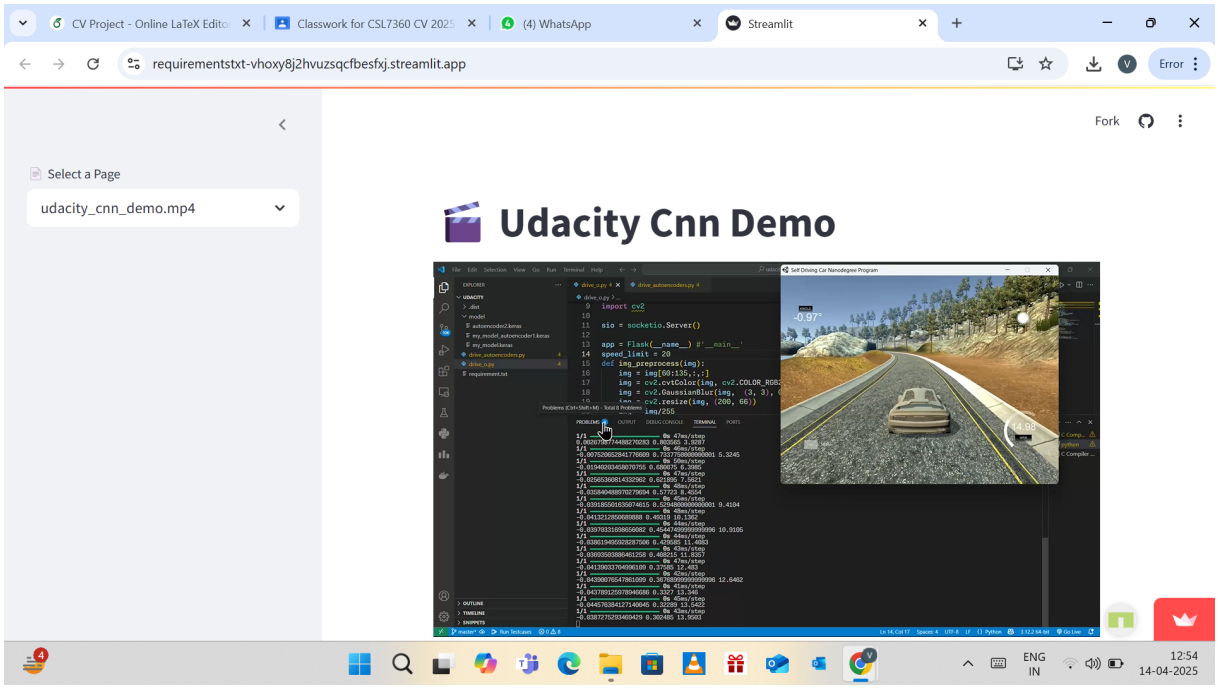

Figure 9.2: Website Demo

Video Demo

https://drive.google.com/file/d/1KY-AiZtJS-uPN8IbmP-pbl9t1usOdNmM/view?usp=drive_link

Contribution

- **Udit Kandpal (M24CSE027):** Udit contributed to the project by working on the Vision Transformer (ViT) model, including training the model and generating data within the AirsimNH simulator.
- **Om Patel (M24CSA019):** Om's contribution involved autoencoder model development for Udacity and training and deployment of the Vision Transformer (ViT) model within the AirsimNH environment.
- **Vishvaskumar Patel (M24CSE029):** Vishvaskumar was responsible for the Convolutional Neural Network (CNN) model training within the AirsimNH simulator environment and also played a key role in report writing.
- **Rahul Maurya (M24CSA025):** Rahul's work involved data generation and the training of the CNN conducted using the Udacity simulator.
- **Patil Divya Kailash (M24CSE018):** Divya focused on the Convolutional Neural Network (CNN) model training, specifically utilizing the Udacity simulator and Report writing for this project.