ADVANCED DRIVER ASSISTANT SYSTEM FOR STABILITY CONTROL

By

KALANSURIYA T.A. (E/19/176)

PERAMUNAGE V.P. (E/19/484)

PERERA W.A.H.M. (E/19/286)

ME 325-MECHANICAL ENGINEERING GROUP PROJECT

presented in partial fulfillment of the
requirements for the degree of B.Sc.
Engineering degree

University of Peradeniya

01/12/2024

Supervised by:

Prof. Asanga Ratnaweera, Department of Mechanical Engineering
Faculty of Engineering.
Dr. D.H.S. Maithripala, Department of Mechanical Engineering
Faculty of Engineering.

Declaration

We, the undersigned, hereby declare that this project report, titled "**Advanced Driver Assistance System for Stability Control**", is an original work carried out by us under the supervision of **Prof. Asanga Ratnaweera** and **Dr. D.H.S Maithripala**, Department of Mechanical Engineering, Faculty of Engineering, University of Peradeniya Sri Lanka.

This report is submitted in partial fulfillment of the requirements for the B.Sc. in Mechanical Engineering degree at the University of Peradeniya.

We confirm that,

1. The work presented in this report is original and has not been submitted previously for any degree, diploma, fellowship, or other similar titles or recognition.
2. All sources of information and references used in this project have been properly cited.
3. Any assistance received during the execution of the project has been duly acknowledged.

We take full responsibility for the content of this report and the results presented herein.

Team Member

1. Name : KALANSURIYA T.A.
Registration Number : E/19/176
Signature : 

2. Name : PERAMUNAGE V.P.
Registration Number : E/19/484
Signature : 

3. Name : PERERA W.A.H.M
Registration Number : E/19/286
Signature : 

Approval

The project report titled "**Advanced Driver Assistance System for Stability Control**" submitted by:

1. KALANSURIYA T.A. (Registration Number: E/19/176)
2. PERAMUNAGE V.P. (Registration Number: E/19/484)
3. PERERA W.A.H.M (Registration Number E/19/286)

has been reviewed and approved in partial fulfillment of the requirements for the **B.Sc. in Mechanical Engineering degree at the University of Peradeniya**.

Approved By,

1. Prof Asanga Ratnaweera,
   Department of Mechanical Engineering,
   University of Peradeniya,
   Sri Lanka.

   Signature: _____
   Date: _____

2. Dr D.H.S. Maithripala,
   Department of Mechanical Engineering,
   University of Peradeniya,
   Sri Lanka.

   Signature: _____
   Date: _____

Acknowledgement

We would like to express our sincere gratitude to all the people who have supported and guided us throughout this journey of completing this third-year undergraduate project. Their assistance, encouragement have been priceless for the successful completion of this project.

First, we like to give our deepest thanks to our project supervisors, **Prof Asanga Ratnaweera** and **Dr D.H.S Maithripala**, for the priceless support, the guidance and the feedback that they have given us for this project. Their expertise, dedication and encouragement have inspired us to strive for excellence throughout this project.

Secondly, we like to express our gratitude to all the faculty members of Faculty of Engineering for all the contribution and support that they have given to us. There mentorship, knowledge and encouragement played a significant role in our academic growth and for the completion of this project.

Furthermore, we would like to express our sincere gratitude for all our fellow batchmates and friends for the motivation and support that they have given us through out this journey. The discussions and brainstorming sessions that we had with them played a crucial role in refining our ideas and enhancing the quality of work.

Last but not least we extend our heartful thanks t our families for the unwavering love, and encouragement that they have given us throughout this journey. Their belief in our abilities and support have been a driving force behind all our accomplishments.

Each of you has been instrumental in shaping our professional journey and enriching our learning experience. Thank you for your unwavering support, encouragement, and mentorship.

Abstract

This project was conducted to improve the safety of amateur racing drivers particularly in formula student vehicle competitions by developing an Advanced Driver Assistant System (ADAS) for Stability Control. The aim of this project was to predict the vehicles future stability and assist drivers in managing oversteer and understeer, particularly in high-speed cornering scenarios. By using advanced machine learning (ML) techniques, our system tries to improve the driving experience and reduce the risks associated with vehicle instability.

In the methodology we integrate machine learning models to predict both current stability and future stability (vehicle stability after 2 seconds). For the current stability detection, we have implemented MLP Regression and SoftMax Regression models. For the future stability prediction again MLP Regression and Markov Chain Processes were employed leveraging neural network-based modeling and probabilistic state transitions, respectively. The data that is being used in the project includes over 200,000 labeled entries of the vehicle behaviors under varying cornering conditions, such as high-speed and low-speed maneuvers. These data was obtained through simulations in the CARSIM environment.

The results shows that both MLP Regression and Markov Chains are effective when predicting the future stability of the vehicle, giving an overall accuracy score of 83% and 84%, respectively. Particularly Markov Chains proved advantageous for modeling sequential dependencies, indicating their potential for predictive tasks in dynamic driving conditions. For current stability detection, both MLP Regressor and SoftMax Regressor showed robust performance, giving an accuracy score of 93% and 94%.

In conclusion, the project offers notable improvements in vehicle stability control by effectively incorporating machine learning methods into a predictive ADAS framework. By informing the driver when the vehicle will become unstable or stable in the future, the system also helps amateur racers perform better and be safer. Future work involves creating a fully automated ADAS system with predictive stability control. This will enable automatic interventions like braking or parameter adjustments, revolutionizing vehicle control and advancing applications in autonomous driving technologies.

Table of Contents

Declaration ..... **i**
Approval ..... **ii**
Acknowledgement ..... **iii**
Abstract ..... **iv**
Table of Contents ..... **v**
List of Figures ..... **vii**
List of Tables ..... **x**
List of Abbreviations ..... **xi**
Chapter 1 ..... **1**
INTRODUCTION 1
1.1 Background 1
1.2 Objectives 1
1.3 Methodology 1
1.4 Results and Future work 2
Chapter 2 ..... **3**
LITREATURE REVIEW 3
2.1 Importance of Vehicle Stability in Dynamics Control 3
2.2 Vehicle Dynamics and Active Control 3
2.3 Machine Learning in Stability Detection 3
2.4 Predictive Techniques for Vehicle Stability 4
2.5 Conclusion 4
Chapter 3 ..... **5**
DATA ACQUISITION 5
3.1 Track and Vehicle Configuration in CARSIM 5
3.2 Run the simulation model 7
Chapter 4 ..... **8**
DATA EXTRACTION 8
Chapter 5 ..... **11**
FEATURE ENGINEERING 11
Chapter 6 ..... **14**
MODEL DEVELOPMENT AND EVALUATION FOR CURRENT STABILITY DETECTION 14
6.1 Regression Model Approach 14
6.2 Classification Modal Approach 21
Chapter 7 ..... **24**

MODEL DEVELOPMENT AND EVALUATION FOR FUTURE STABILITY DETECTION 24
7.1 Future Stability Detection Using MLP Regressor 24
7.2 Markov Chain Process Approach 29
Chapter 8 ..... **32**
MODEL VALIDATION 32
8.1 MLP Regressor Model Validation 32
8.2 SoftMax and Markov Chain Process Approach 34
Chapter 9 ..... **35**
CONCLUSION AND RECOMENDATIONS 35
9.1 CONCLUSION 35
9.2 Recommendations 35
Chapter 10 ..... **36**
FUTURE WORK 36
References **37**
Appendices **38**

List of Figures

Figure 1 CARSIM SOFTWARE USER INTERFACE 5

Figure 2 BASIC PARAMETERS OF THE VEHICLE 6

Figure 3 SETTING UP OF SUSPENSION PARAMETERS 6

Figure 4 TYRE DATA PARAMETERS 6

Figure 5 AERODYNAMIC PARAMETR SETTINGS 7

Figure 6 SIMULATION DATA GATHERING AND PLOTTING 7

Figure 7DATA THAT HAS BEEN OBTAINED TO THE EXCEL SHEET. 8

Figure 8 CODE THAT WAS USED TO GENERATE THE FINAL DATA FILE 9

Figure 9 FINAL EXCEL THAT IS USED TO TRAIN THE MODEL 10

Figure 10 INITIAL SPREAD OF THE DATA 12

Figure 11 SPREAD OF THE DATA AFTER SCALING 13

Figure 12 MODELING USING LINEAR REGRESION 15

Figure 13 EVALUATING TRAINING SCORE AND VALIDATION SCORE 18

Figure 14 LEARNING CURVE SHOWING R^2 SCORE FOR TRAINING AND VALIDATION SETS AS A FUNCTION OF TRAINING SET SIZE 19

Figure 15 LEARNING CURVE FOR TRAINING AND TEST RMSE AS A FUNCTION OF TRAINING SET SIZE. 20

Figure 16 ACTUAL VS PREDICTED SLIP ANGLE VALUES 21

Figure 17 CONFUSION MATRIX FOR SOFTMAX REGRESSION 23

Figure 18 LEARNING CURVE SHOWING R^2 SCORE FOR TRAINING AND VALIDATION SETS AS A FUNCTION OF TRAINING SET SIZE FOR FUTURE DETECTION 27

Figure 19 CONFUSION MATRIX FOR ACTUAL VS PREDICTED CURRENT STABILITY STATES 32

Figure 20 CONFUSION MATRIX TO ACTUAL VS PREDICTED FUTURE STABILITY STATES 33

Figure 21 IMPORTING DATA TO A DATAFRAME 38

Figure 22 STABILITY COLUMNS DEFINITION 38

Figure 23 NEW FEATURE ROTATIONAL SPEED LEFT AND ROTATIONAL SPEED RIGHT 38

Figure 24 MIN MAX VALUE DETECT CODE 38

Figure 25 MIN MAX VALUE OF FEATURES BEFOR SCALING 39

Figure 26 SCALLING OF FEAATURES EXCEPT REACTION FORCES 39

Figure 27 SCALING OF REACTION FORCES 39

Figure 28 TRAIN TEST SPLIT OF THE DATA 40

Figure 29 MLP REGRESSOR TRAINING PARAMETER GRID 40

Figure 30 MLP REGRESSION GRID SEARCH IMPLEMENTATION 40

Figure 31 SCORE OF MLP REGRESSOR 40

Figure 32 CODE TO GET LEARNING CURVE SHOWING R^2 SCORE FOR TRAINING AND VALIDATION SETS AS A FUNCTION OF TRAINING SET SIZE 41

Figure 33 R^2 AND RMSE VALUES FOR TRAIN AND TEST SET 41

Figure 34 CODE FOR LEARNING CURVE FOR TRAINING AND TEST RMSE AS A FUNCTION OF TRAINING SET SIZE 42

Figure 35 CLASSIFYING STABILITY INTO THREE DIFFERENT CLASSES 42

Figure 36 TRAIN TEST SPLIT FOR SOFTMAX REGRESSION 43

Figure 37 SOFTMAX REGRESSION MODEL IMPLEMENTATION USING TENSORFLOW` 43

Figure 38 FINAL SOFTMAX MODEL 43

Figure 39 CODE TO GET SOFTMAX MODEL ACCURACY ON TEST DATASET 44

Figure 40 CODE FOR CONFUSION MATRIX OF SOFTMAX REGRESSION 44

Figure 41 EVALUATION OF SOFTMAX REGRESSION MODEL 44

Figure 42 CODE TO GET SLIP ANGLE DIFFERENCE AFTER TWO SECONDS FEATURE 45

Figure 43 PARAMETER GRID FOR FUTURE STABIITY DETECTION 45

Figure 44 GRID SEARCH FOR FUTURE STABILITY DETECTION 45

Figure 45 FUTURE STABILITY MODEL SCORE 46

Figure 46 CODE TO GET TRAINING SCORE AND CROSS VALIDATION SCORE 46

Figure 47 CODE FOR LEARNING CURVE SHOWING R^2 SCORE FOR TRAINING AND VALIDATION SETS AS A FUNCTION OF TRAINING SET SIZE FOR FUTURE DETECTION.....46

Figure 48 CODE FOR CODE FOR LEARNING CURVE FOR TRAINING AND TEST RMSE AS A FUNCTION OF TRAINING SET SIZE FOR FUTURE STABILITY DETECTION.....47

Figure 49 TRANSITION MATRIX AND FUTURE IMPLEMENTAION FUNCTION FOR MARKOV CHAIN 47

Figure 50 MARKOV CHAIN IMPLEMENTATION.....48

Figure 51 CURRENT SLIP ANGLE PREDICTION FOR MODEL VALIDATION..48

Figure 52 CONVERSION FROM SLIP ANGLE DIFFERENCE TO STABILITY STATES.....49

Figure 53 CODE TO GET CONFUSION MATRIX FOR ACTUAL VS PREDICTED STABILITY STATES 49

Figure 54 CODE CONFUSION MATRIX FOR ACTUAL VS PREDICTED FUTURE STABILITY STATES 50

Figure 55 CURRENT STABILITY PREDITION USING SOFTMAX REGRESSION FOR MODEL VALIDATION.....50

List of Tables

Table 1 MIN MAX VALUES OF COLUMNS BEFORE SCALING 12

List of Abbreviations

MLP	Multi-Layer Perceptron
CARSIM	Carsim Software
ML	Machine Learning
ADAS	Advanced Driver Assistant System
PID	Proportional Integral Derivative

Chapter 1
INTRODUCTION

Vehicle Stability is an important factor when considering the safety and performance in automotive applications such as competitive competitions like Formula Student racing. When cornering at high speed, oversteer and understeer pose a significant challenge to the drivers affecting the controlling and stability of the vehicle. These issues not only reduce the safety but also reduce the performance during the race. To address these issues, a system is required that can detect and predict both the current and future stability of vehicles, enabling drivers to take corrective actions in a timely manner.

The main goal of this project is to develop an Advanced Driver Assistance System (ADAS) for Stability Control that can predict both current and future stability of the vehicle in real time. The system is designed to help amateur racing drivers particularly in formula student vehicle competitions by providing real-time feedback to overcome oversteer and understeer. By integrating machine learning models into the control system, the project aims to enhance safety and performance, particularly in scenarios that need precise vehicle handling.

1.1 Background

The development in the automotive industry it has led to the increase use of intelligent system to improve the vehicle safety and control. However, the instability caused by oversteer ad understeer still remains a significant challenge during high performance and high-speed driving scenarios. Traditional vehicle stability control systems, such as PID controllers are reliable in simpler scenarios, but struggle with the complexity of dynamic driving scenarios. To answer this problem this project adopts ML based approach to overcome these limitations.

1.2 Objectives

When considering the objectives of this project there are two main objectives. First objective is, for a given track when vehicle is speeding or cornering to predict whether current stability state of the vehicle. Which are neutral steer, over steer or under steer. The second objective of this project is for a instance predict whether the vehicle is going to understeer or oversteer (change of stability state) within the next two seconds. These objectives aim to provide real-time insights and proactive measures to enhance vehicle stability and safety.

1.3 Methodology

The project methodology was consisted with several key stages, first the data acquisition. Here vehicle dynamics data were collected from simulated driving scenarios, including high-speed and low-speed cornering using CARSIM software. Next, model design and development focused on choosing a correct machine learning model that will fit the data correctly. Here several machine learning models has been tested and the MLP Regressor and SoftMax regression has been chosen as the best performing model. These models were then subjected to testing within controlled environments to evaluate their performance in detecting and predicting vehicle stability. Finally, evaluation and refinement processes assessed model performance

based on metrics like prediction accuracy and confusion metrics to enhance the system's overall reliability.

1.4 Results and Future work

The results of this project shows that the effectiveness of the proposed machine learning-based model for vehicle stability control. The system successfully predicts the current stability state of the vehicle with high accuracy and provides reliable predictions for future stability changes, such as transitions to understeer or oversteer, within a two-second timeframe. The use of models like Markov Chains and MLP Regression highlights the potential of predictive analytics in enhancing vehicle safety and performance. Looking ahead, the future scope of this project involves the development of a fully automated ADAS system that integrates predictive stability control into the vehicle's control system. This system would enable real-time, autonomous interventions, such as brake application or parameter adjustments, paving the way for advancements in autonomous driving technologies.

Chapter 2

LITREATURE REVIEW

2.1 Importance of Vehicle Stability in Dynamics Control

The stability of a vehicle, especially in high-performance or adverse driving conditions, is a cornerstone of automotive engineering. Key conditions such as understeer, oversteer, and neutral steer play a pivotal role in determining handling characteristics, safety, and driver experience. Understeer occurs when the vehicle turns less sharply than intended due to the front tires losing grip, while oversteer refers to excessive turning caused by rear tire slip. Neutral steer, the ideal condition, maintains a balance between front and rear tire slip angles.

In Formula Student scenarios, where precision driving is critical, understanding and predicting these states is essential. The use of our machine learning (ML) model in dynamic stability management represents an evolution from traditional rule-based systems to intelligent, adaptive technologies

2.2 Vehicle Dynamics and Active Control

Dynamic Understeer Control Using Active Rear Toe Mark Kaufman's thesis on dynamic understeer control emphasizes the challenges in controlling vehicle dynamics, particularly understeer. The thesis introduces an Active Rear Toe (ART) mechanism that adjusts the rear wheel angles dynamically, allowing better control over the vehicle's lateral acceleration and turning radius. The study implements sliding mode control (SMC), which adapts steering inputs to maintain stability within the vehicle's limits of grip.

The ART system demonstrated effectiveness in controlling understeer across various conditions but was limited by tire grip at extreme lateral forces. Kaufman suggested integrating ART with braking systems for enhanced control. This study underscores the importance of predicting lateral accelerations and turning radii, as achieved in our model through MLP regression

2.3 Machine Learning in Stability Detection

BMW's Oversteer Detection System BMW's approach to detecting oversteer using supervised machine learning serves as a benchmark for ML-based stability systems. BMW utilized data from real-world tests to train models that detected oversteer conditions with over 98% accuracy. Key input features included forward acceleration, lateral acceleration, steering angle, and yaw rate. Notably, the study highlighted the challenge of determining optimal thresholds for different conditions, such as icy vs. dry roads, and addressed this variability using ML classifiers like decision trees, k-nearest neighbors (KNN), and support vector machines (SVMs).

The decision tree model, with its high true-negative rate (95.86%), was deployed in a prototype electronic control unit (ECU) for real-time tests. Despite variations in vehicles, drivers, and tracks, the system achieved a real-time accuracy rate of about 95%, demonstrating the robustness of ML algorithms in dynamic and uncertain environments

Our model, which uses a combination of SoftMax regression and Markov chains, builds on this concept, offering an alternative probabilistic approach to predict stability states. The integration of deterministic MLP regression and probabilistic Markov models reflects a robust strategy to handle the inherent uncertainties in dynamic vehicle behavior

2.4 Predictive Techniques for Vehicle Stability

MLP Regression for Predicting Slip Angle Differences The Multilayer Perceptron (MLP) regression in our model is designed to predict the slip angle differences for up to 2 seconds ahead, enabling early detection of understeer, oversteer, or neutral steer states. This aligns with Kaufman's sliding mode control model, which calculates lateral acceleration and uses it to adjust vehicle dynamics proactively. The predictive aspect of MLP regression, however, extends beyond real-time control to preemptive stability management. Such foresight is crucial in competitive environments like Formula Student racing.

Softmax Regression with Markov Chains for Future States The hybrid Softmax regression-Markov chain model in our system introduces a state-dependent prediction mechanism. This method considers the probabilistic transition between stability states (neutral, understeer, and oversteer) based on historical and current data. Markov models are particularly advantageous in capturing sequential dependencies and handling variability in vehicle dynamics, such as changes in road conditions, tire wear, or driver input.

In contrast to purely deterministic models, the Markov chain-based approach aligns with BMW's use of decision trees and other classifiers to address variations in conditions. The probabilistic nature ensures adaptability, making our system robust for both racing and real-world applications

2.5 Conclusion

The intersection of control theory and machine learning in our model highlights a cutting-edge approach to vehicle stability. By combining deterministic predictions with probabilistic modeling, our system aligns with advancements in automotive engineering while addressing the unique demands of Formula Student racing. The integration of these models into a cohesive framework offers a robust, adaptable, and predictive stability management system, setting a precedent for future innovations in dynamic vehicle control.

Chapter 3

DATA ACQUISITION

After establishing the theoretical basis, the next step was to identify a simulation platform capable of generating accurate and comprehensive datasets, including slip angle values. The evaluation led to the selection of CARSIM, a widely recognized software in the automotive industry for its reliability in simulating dynamic vehicle behaviors. CARSIM was chosen due to its ability to replicate real-world scenarios and generate high-quality data necessary for training the machine learning model

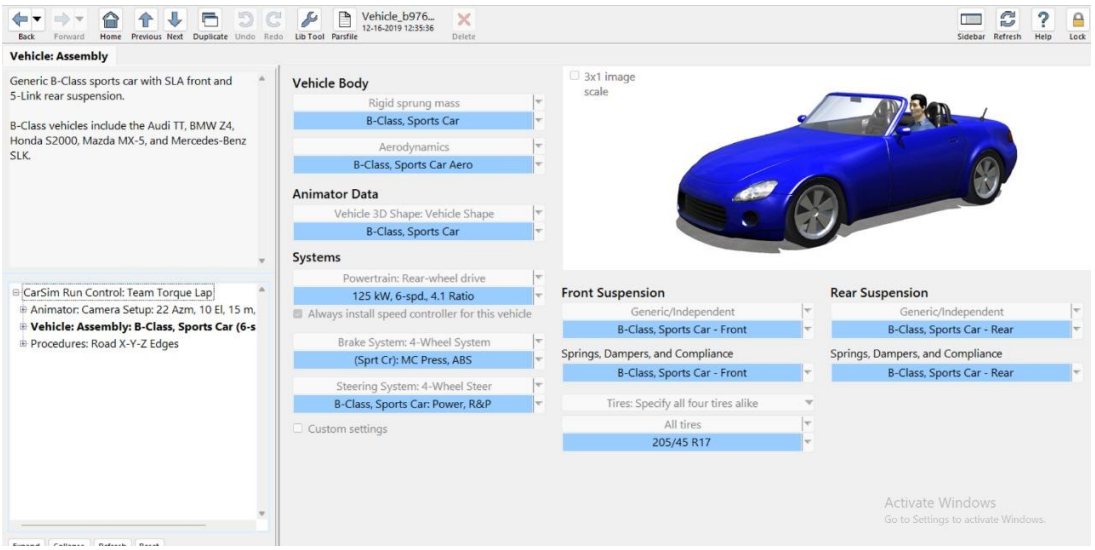


Figure 1 CARSIM SOFTWARE USER INTERFACE

3.1 Track and Vehicle Configuration in CARSIM

Using CARSIM, a custom track was created by specifying precise coordinates. The track design incorporated numerous corners to thoroughly test vehicle stability during cornering, as this is where understeer and oversteer conditions are most likely to occur. A vehicle model was then configured in CARSIM to match the specifications of a sports-type car, aligning with the project's focus on Formula Student vehicles. The parameters were adjusted to simulate the dynamic behavior of such vehicles accurately. Parameters of the vehicle simulation were set as follows,

- First the basic control parameters of the vehicle were set as follows,

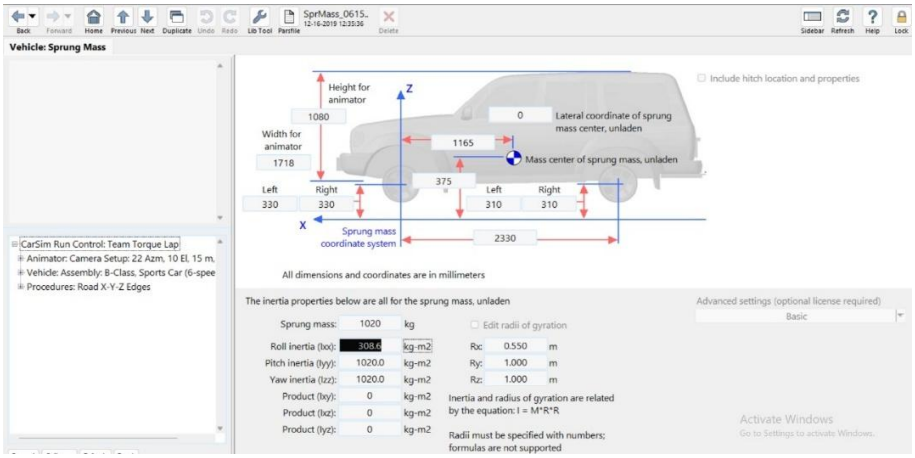


Figure 2 BASIC PARAMETERS OF THE VEHICLE

- Next the suspension parameters were set as follwos,

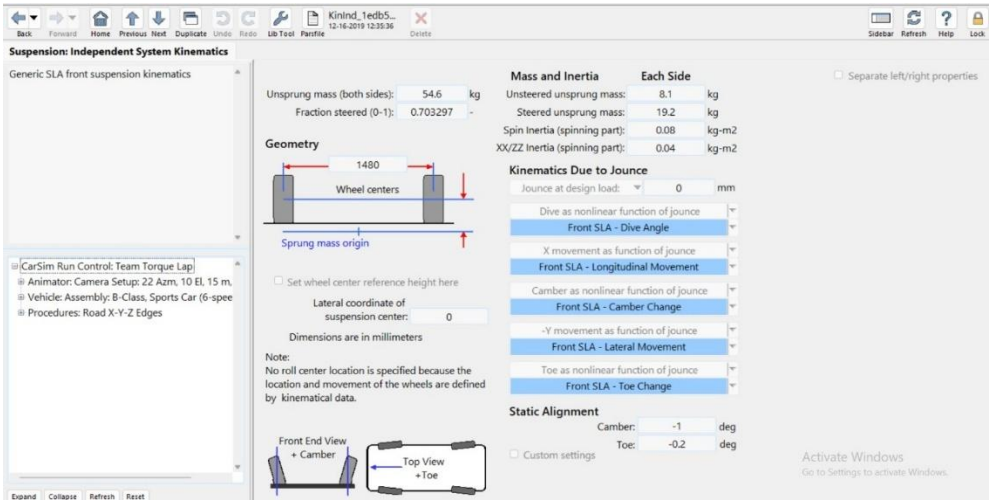


Figure 3 SETTING UP OF SUSPENSION PARAMETERS

- Next the parameters regarding tire data as follows,

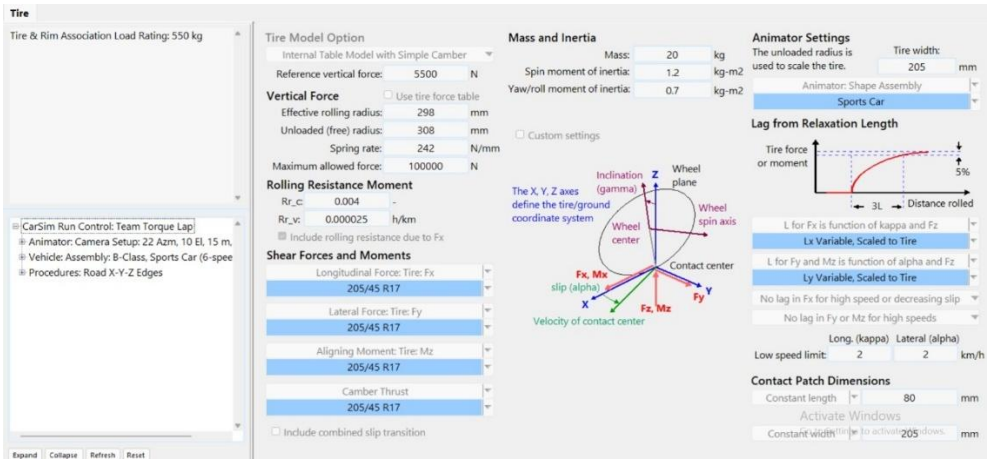


Figure 4 TYRE DATA PARAMETERS

- Finally the parameters regarding the aerodynamics of the vehicle were set as follows.

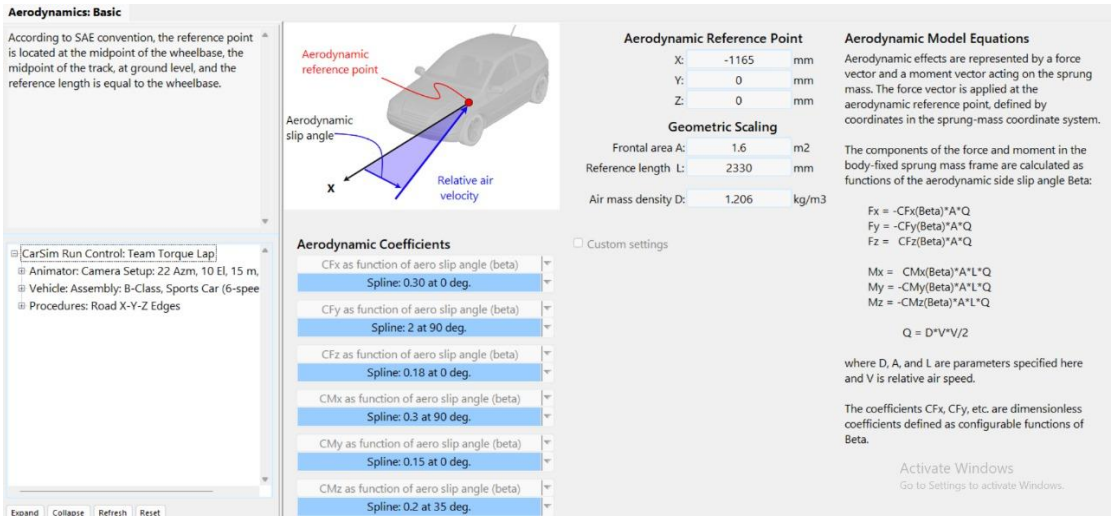


Figure 5 AERODYNAMIC PARAMETR SETTINGS

The above parameters were chosen to ensure they are suitable for a Formula Student vehicle.

3.2 Run the simulation model

To gather datasets for the machine learning model, simulations were run at discrete vehicle speeds, varying in small increments (e.g., 30 km/h, 31 km/h, etc.). This approach ensured a granular understanding of the vehicle's stability across a wide range of conditions. The datasets collected to excel sheets including key parameters such as slip angles, lateral acceleration, and steering angles. These simulations provided data for neutral steer, understeer, and oversteer conditions, forming the foundation for training the predictive model.

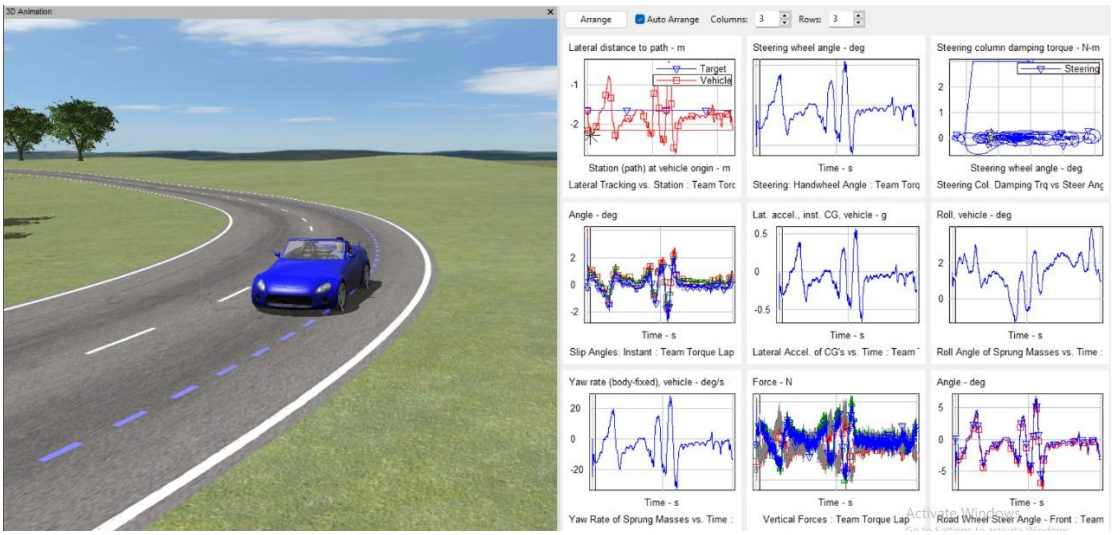


Figure 6 SIMULATION DATA GATHERING AND PLOTTING

Chapter 4
DATA EXTRACTION

As mentioned in the previous chapter, when collecting data from the CARSIM software, it provided an Excel sheet containing various columns of data. Therefore, it was necessary to extract the correct columns containing the required features to be used for training our machine learning models.

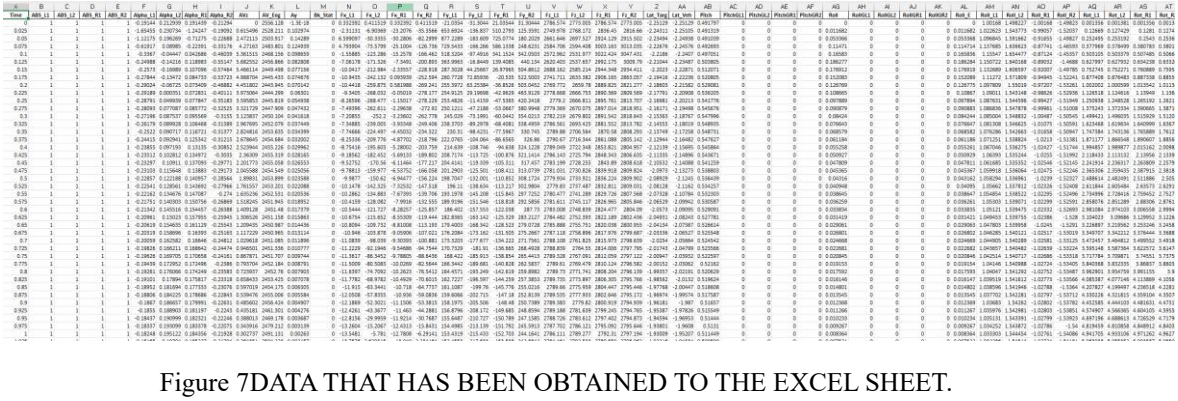


Figure 7DATA THAT HAS BEEN OBTAINED TO THE EXCEL SHEET.

From those couple of features, we have chosen the 15 different variables to develop our machine learning model which are,

- Time
- Left wheel steer
- Right wheel steer
- Driving steer input
- Lateral acceleration
- Yaw rate
- Rotational speed left front
- Rotational speed left rear
- Rotational speed right front
- Rotational speed right rear
- Reaction force left front
- Reaction force left rear
- Reaction force right front
- Reaction force right rear
- Slip angle Difference

The primary condition for selecting these variables was to ensure that they are easily measurable and reconstruct able, making them suitable for practical application. The selected variables can be conveniently measured using sensor readings, providing accurate and reliable data for training the machine learning models.

Since there were multiple Excel files containing different datasets obtained for various target speeds of the vehicle, we extracted the relevant columns from these files and merged them into a single Excel file. This process was accomplished using the following Python code.

```python
from openpyxl import load_workbook
def extract_data(source_file, target_file, target_speed):
    wb1 = load_workbook(source_file)
    wb2 = load_workbook(target_file)
    ws1 = wb1.active
    ws2 = wb2.active
    columns_to_extract = [
        ('A', 1),  # Time
        ('AC', 3),  # Left wheel steer
        ('AD', 4),  # Right wheel steer
        ('AE', 5),  # Driving steer input
        ('G', 6),  # Lateral acceleration
        ('F', 7),  # Yaw rate
        ('X', 8),  # Rotational speed left front
        ('Y', 9),  # Rotational speed left rear
        ('Z', 10),  # Rotational speed right front
        ('AA', 11),  # Rotational speed right rear
        ('I', 12),  # Reaction force left front
        ('J', 13),  # Reaction force left rear
        ('K', 14),  # Reaction force right front
        ('L', 15)  # Reaction force right rear
    ]
    check = ws2.max_row + 1
    for source_col, target_col in columns_to_extract:
        current_row = check
        for cell in ws1[source_col][1:]:
            ws2.cell(row=current_row, column=target_col, value=cell.value)
            current_row += 1
    for i in range(check, check + ws1.max_row - 1):
        ws2.cell(row=i, column=2, value=target_speed)
    for i in range(1, ws1.max_row):
        front_average = (ws1['B'][i].value + ws1['D'][i].value) / 2
        rear_average = (ws1['C'][i].value + ws1['E'][i].value) / 2
        result = abs(front_average) - abs(rear_average)
        ws2.cell(row=check + i - 1, column=16, value=result)
    print(target_speed)
    wb2.save(target_file)
```

Figure 8 CODE THAT WAS USED TO GENERATE THE FINAL DATA FILE

The slip angle was calculated by taking the average of the front slip angle and the rear slip angle, and then finding the difference between them.

The final Excel file, which was used to train the model, looks as follows

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
Time	Target Spe	Left whe	Right wh	Driver st	Lateral A	Yaw rate	Rotation s	Rotation s	Rotation s	Rotation s	Reaction f	Reaction f	Reaction f	Reaction Force rig	Slip angle difference	
0	78	0.20003	-0.20003	-0.6	-0.02345	0	0.00156	0.001613	0.001588	0.001584	2910.53	2869.831	2465.431	2262.31002	-0.00025738	
0.025	78	-1.37097	-1.79035	-30	-0.15979	-0.64328	0.291544	0.290849	0.290199	0.290777	3127.298	2766.307	3043.064	2944.759466	1.550079549	
0.05	78	-3.21383	-3.76096	-60	-0.39826	-4.21185	0.581707	0.580273	0.578119	0.579627	3369.267	3119.763	2934.925	2824.57336	3.070983664	
0.075	78	-4.96433	-5.6788	-90	-0.51435	-10.1183	0.872107	0.870348	0.864932	0.867592	3745.579	3569.367	2943.269	2619.469479	4.258027969	
0.1	78	-6.70816	-7.67999	-120	-0.54746	-16.1103	1.162886	1.161317	1.150188	1.154359	4125.957	4078.631	2236.725	2528.069326	5.474054219	
0.125	78	-8.39086	-9.6891	-150	-0.63079	-21.3224	1.453762	1.453072	1.433719	1.439962	4290.504	4268.674	2351.787	2333.997687	6.735267927	
0.15	78	-10.0752	-11.7779	-180	-0.65864	-25.3068	1.744364	1.745347	1.715348	1.724524	4221.525	4541.644	1920.949	1867.106415	8.169851308	
0.175	78	-11.2544	-13.2778	-199.856	-0.75469	-28.0084	2.033935	2.037855	1.99521	2.008317	4514.306	4697.058	2018.438	1962.184304	9.145189331	
0.2	78	-11.2148	-13.2304	-199.438	-0.77693	-29.7405	2.322254	2.330402	2.273784	2.291472	4326.957	4818.246	1801.536	1767.846225	8.893752202	
0.225	78	-11.1783	-13.1743	-198.655	-0.82928	-30.7678	2.610233	2.622815	2.551075	2.574219	4402.646	4730.69	1907.504	1865.287415	8.724514626	
0.25	78	-11.1221	-13.0969	-197.649	-0.83739	-31.683	2.897874	2.915163	2.82734	2.856801	4328.254	4293.209	2138.897	1791.051928	8.545848839	
0.275	78	-11.0537	-13.0108	-196.362	-0.82372	-32.7064	3.185192	3.207732	3.102748	3.139858	4103.354	4009.214	2119.677	1780.254884	8.361040557	
0.3	78	-10.9811	-12.9142	-194.877	-0.81318	-33.2358	3.472185	3.500784	3.377444	3.423788	4078.835	4089.673	1876.095	1606.598382	8.216434052	
0.325	78	-10.8992	-12.7998	-193.31	-0.82159	-33.6902	3.758834	3.794396	3.651523	3.70924	4022.516	4220.995	1809.62	1585.805725	8.06979442	
0.35	78	-10.8354	-12.7087	-191.908	-0.84608	-33.3394	4.045128	4.088504	3.925079	3.996159	3956.006	4385.426	1648.822	1936.159651	8.033963442	
0.375	78	-10.7652	-12.6132	-190.936	-0.87496	-32.8272	4.330967	4.382982	4.198167	4.283882	4025.245	4457.955	1883.323	1909.892252	8.010129314	
0.4	78	-10.7413	-12.5796	-190.383	-0.85282	-32.1059	4.616378	4.677829	4.470884	4.572718	4016.215	4476.078	1681.557	1760.489314	8.053656713	
0.425	78	-10.7346	-12.5644	-190.254	-0.8555	-31.5751	4.901328	4.972953	4.743245	4.86316	4022.803	4377.162	1778.167	1780.444337	8.095322984	
0.45	78	-10.7435	-12.5738	-190.446	-0.8326	-31.2406	5.185833	5.268352	5.015294	5.155006	4084.018	4292.741	1700.735	1547.660733	8.136133408	
0.475	78	-10.7478	-12.5837	-190.821	-0.8902	-31.1629	5.469924	5.56399	5.286989	5.448559	4097.442	4442.926	2039.309	1876.643416	8.154757004	
0.5	78	-10.7786	-12.621	-191.403	-0.85356	-30.7698	5.753653	5.859774	5.558374	5.742207	4171.861	4459.433	1656.266	1622.022169	8.23524889	
0.525	78	-10.8216	-12.6759	-192.346	-0.85792	-30.193	6.036994	6.155602	5.829404	6.036817	3979.789	4427.184	1765.938	1801.363957	8.348193298	
0.55	78	-10.8884	-12.7619	-193.693	-0.86327	-29.556	6.319883	6.451504	6.100105	6.331635	3984.343	4501.811	1781.709	1770.787996	8.492645358	
0.575	78	-10.9938	-12.8948	-195.485	-0.87447	-28.6094	6.602313	6.747362	6.370475	6.626885	3913.363	4652.813	1667.233	1979.083067	8.720944772	
0.6	78	-11.1138	-13.0465	-197.871	-0.88619	-27.5941	6.884191	7.043178	6.640488	6.921613	4051.949	4453.886	1978.704	1891.131185	8.958420109	
0.625	78	-11.284	-13.2646	-200.616	-0.85028	-27.0003	7.165537	7.338895	6.910183	7.217208	4158.2	4435.328	1617.845	1695.615763	9.221250232	
0.65	78	-11.4404	-13.4721	-203.642	-0.86715	-26.63	7.446324	7.634697	7.179485	7.514289	4217.262	4474.199	1762.418	1730.7649	9.448519814	
0.675	78	-11.6234	-13.7099	-206.919	-0.84842	-26.3831	7.726611	7.93063	7.448381	7.812223	4262.448	4299.626	1781.798	1606.208559	9.67956861	

Figure 9 FINAL EXCEL THAT IS USED TO TRAIN THE MODEL

Chapter 5

FEATURE ENGINEERING

After getting the Final excel file first the feature engineering of the features are performed. The data has been imported to a ipynb file to models development [**Error! R eference source not found.**]. First a new feature called stability has been formed using the following condition. [Figure 22]

- If slip angle difference >4.5; Stability =1
- If Slip angle difference <-4.5; Stability=-1
- Else; Stability=0

Here the +1 represents the understeer state, -1 represents the oversteer state, 0 represents the neutral steer state.

After the setting up the new feature the our total features looks as follows.

	Time	Target Speed	Left wheel steer	Right wheel steer	Driver steering input	Lateral Acceleration	Yaw rate	Rotation speed left front	Rotation speed left rear	Rotation speed right front	Rotation speed right rear	Reaction force left front	Reaction force left rear	Reaction force right front	Reaction Force right rear	Slip angle difference	Stability
0	0.000	78	0.200030	-0.200030	-0.6	-0.023454	0.000000	0.001560	0.001613	0.001588	0.001584	2910.529759	2869.830717	2465.431322	2262.310020	-0.000257	0
1	0.025	78	-1.370966	-1.790352	-30.0	-0.159788	-0.643283	0.291544	0.290849	0.290199	0.290777	3127.298144	2766.306577	3043.063712	2944.759466	1.550080	0
2	0.050	78	-3.213833	-3.760956	-60.0	-0.398263	-4.211852	0.581707	0.580273	0.578119	0.579627	3369.266892	3119.762517	2934.924940	2824.573360	3.070984	0
3	0.075	78	-4.964329	-5.678796	-90.0	-0.514346	-10.118336	0.872107	0.870348	0.864932	0.867592	3745.579489	3569.366642	2943.268624	2619.469479	4.258028	0
4	0.100	78	-6.708164	-7.679995	-120.0	-0.547463	-16.110322	1.162886	1.161317	1.150188	1.154359	4125.956512	4078.630679	2236.725399	2528.069326	5.474054	1

Reducing the number of features used in developing the machine learning model can significantly enhance the training process by improving computational efficiency, minimizing the risk of overfitting, and focusing on the most relevant data for better predictive accuracy. To minimize the number of features the following decisions were taken [Figure 23].

- Make a new feature average rotations speed left by getting the mean value of rotational speed left front and rotational speed left rear.
- Make a new feature average rotational speed right by getting the mean value of rotational speed right front and rotational speed right rear.
- Drop the Rotation speed left front, Rotation speed right front ,Rotation speed left rear , Rotation speed right rear.

After creating the new feature columns and removing the old feature columns, a histogram was plotted to visualize the distribution and spread of the data, providing insights into its underlying patterns and variability.

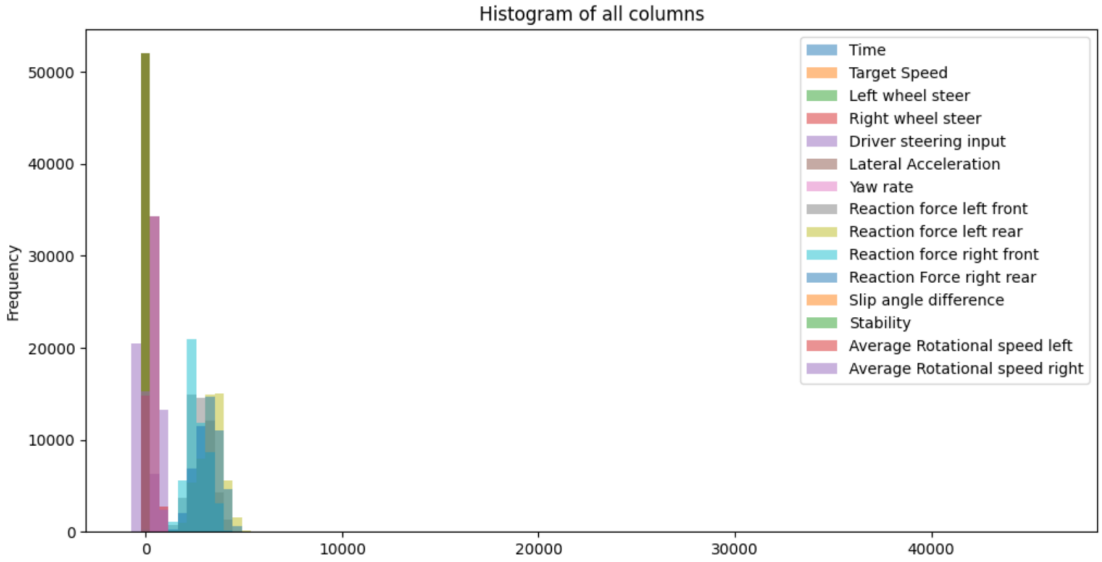


Figure 10 INITIAL SPREAD OF THE DATA

Although the histogram shows a fairly wide distribution of the data, this wide-spread nature is undesirable, since it can lead to a drop in accuracy and performance of the machine learning model. When this is relevant, scaling the data is important to avoid this problem. Scaling ensures that the features are normalized, having all features in the same scale-Aid in making model training faster, unbiased and better prediction overall.

To determine a proper scaling function for the corresponding features the min and max values of each feature has been considered [Figure 25]. The min max value of each feature are as follows,

Table 1 MIN MAX VALUES OF COLUMNS BEFORE SCALING

Column	Min	Max
Left wheel steer	-39.99	60.79
Right wheel steer	-60.78	39.97
Driving steer input	-720	720
Lateral acceleration	-3.96	2.26
Yaw rate	-229.60	210.21
Average rotational speed left	0.00158	904.65
Average rotational speed right	0.0015	899.86
Reaction force left front	0	32609.21
Reaction force left rear	0	46112.90
Reaction force right front	0	31930.71
Reaction force right rear	0	23252.67

Here for the scaling process Time, Target Speed, slip angle difference was not considered. Because,

- Slip angle difference is a target variable (can be detect the stability of the vehicle using slip angle difference value)
- Target speed is being used to shuffle the data set.

As observed in the table, some data values range from negative to positive, while others vary from zero to a finite positive value. To ensure consistency and improve model performance, the following scaling decisions were made:

1. Data with negative to positive range will be scaled to between -1 to +1
2. Data with zero to finite positive range will be scaled to between 0 to 1

This approach ensures all features are appropriately scaled for optimal model training [Figure 26,Figure 27].

After scaling the data, a histogram was plotted again to observe the changes in the distribution. This step was undertaken to verify the effectiveness of the scaling process and ensure the data is now properly normalized, with reduced variability and improved consistency across features, facilitating better model training and performance.

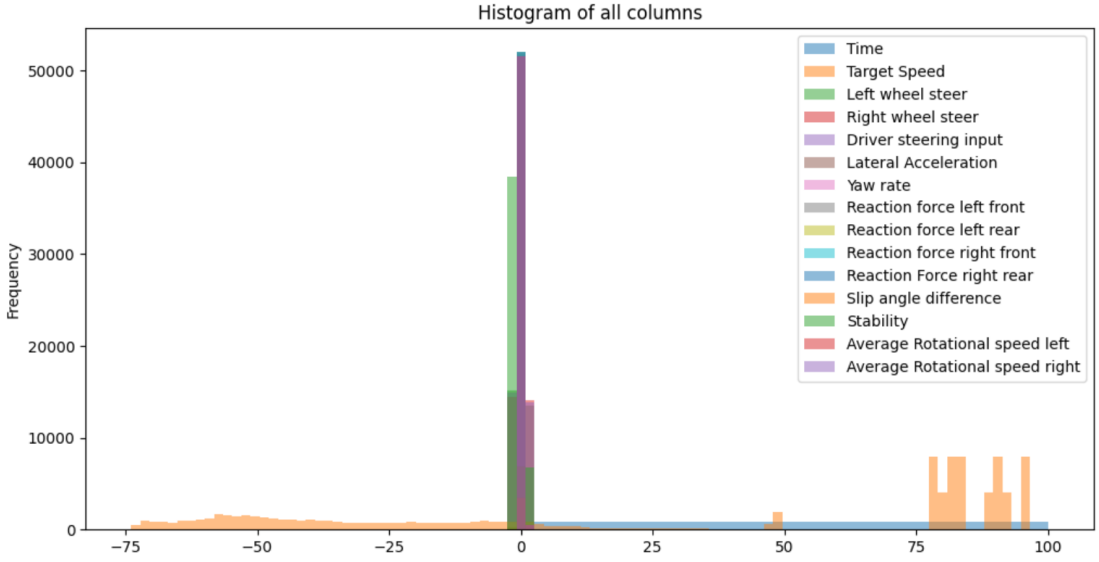


Figure 11 SPREAD OF THE DATA AFTER SCALING

As observed in the histogram, the data now falls within a specific and well-defined range after scaling. This indicates that the scaling process was effective, resulting in a more uniform distribution. Such normalization is beneficial for the machine learning model, as it ensures consistent feature representation and enhances the accuracy and efficiency of the training process.

Chapter 6

MODEL DEVELOPMENT AND EVALUATION FOR CURRENT STABILITY DETECTION

To assess the current stability of the vehicle, two primary approaches were employed. The first approach model the problem as a regression problem because, by predicting the slip angle difference, which directly correlates with the vehicle's stability, providing precise numerical insights. The second approach treated the problem as a multi-class classification task, categorizing the data into three stability states: stable, understeer, and oversteer. This classification method offers clear, actionable insights into the vehicle's handling dynamics. Both approaches complement each other, with the regression model focusing on precise stability metrics and the classification model providing categorical stability assessments.

6.1 Regression Model Approach

Regression is used here to predict the slip angle difference, a key indicator of the vehicle's stability. By providing continuous numerical outputs, regression enables precise monitoring and detailed analysis of stability dynamics. This approach is particularly useful for understanding subtle variations and ensuring accurate assessments of the vehicle's handling performance under different conditions.

To implement this approach, the dataset was first split into training and testing sets to ensure robust model evaluation. For the input features (**X variables**), the following were selected based on their relevance to vehicle stability:

I. Right wheel steer
II. Left wheel steer
III. Driving steer input
IV. Lateral acceleration
V. Yaw rate
VI. Reaction force left front
VII. Reaction force left rear
VIII. Reaction force right front
IX. Reaction force right rear
X. Average rotational speed left
XI. Average rotational speed right

For the output variable (**Y variable**), the **slip angle difference** was chosen as it serves as a critical indicator of vehicle stability. This setup ensures that the model is trained on key features directly influencing the stability dynamics, enabling precise and reliable predictions [Figure 28].

We stratified data with the Target Speed variable while we did the train-test split. This guarantees that the target speed values are consistently distributed in the training and test set. Through stratification, we ensure all target speed categories are equally represented, which is paramount to preclude potential bias that may develop through a distribution maldistribution.

At first, a linear regression model was used in our approach to predict the slip angle difference. But the performance of the model was not very well as it got an accuracy score of only 0.21. The low accuracy shows us that a linear regression model doesn't work enough to learn those highly nonlinear relationships between input features and slip angle difference. Therefore, the following steps prior to training involved introducing alternative machine learning models capable of better capturing nonlinearities and dynamic interactions among input features.

```
from sklearn.linear_model import LinearRegression
lr=LinearRegression()
lr.fit(X_train,y_train)
lr.score(X_test,y_test)
```

✓ 0.2s

```
0.21503544625856874
```

Figure 12 MODELING USING LINEAR REGRESION

Then we moved to an MLP Regressor model, which is a neural network-based model and is able to capture complex non-linear relationships in data. We will use an MLP Regressor (Multilayer Perceptron Regressor) for such prediction as which makes it an ideal candidate since it can learn complex nonlinear mappings from the input variables to the output predictions very well. While linear regression assumes a straightforward relationship, the MLP Regressor utilizes multiple interconnected neuron layers to model complex patterns and relationships between input features and the target variable. Due to this flexibility, it seems especially promising for the prediction of vehicle stability, where the dynamics are highly nonlinear and have to deal with many interacting factors. Moreover, its flexibility with different datasets and ability to manage high dimension input data also favors its application on this project.

Here in this MLP regressor modeling. First we have set up a grid search matrix to find the best. For the parameter grid search matrix following parameters were set[Figure 29],

- Hidden layer sizes= (10,), (50,), (100,), (10, 10), (50, 50), (50,50,50)
- Activation function= relu, sigmoid, TanH
- Solver= sgd
- Learning rate = constant, adaptive, invers scaling
- Learning rate_init= 0.001

Ther reasons for those decisions are as follows,

- **Hidden layer sizes -** The selected hidden layer sizes, such as (10,), (50,), (100,), and multi-layer configurations like (50, 50) and (50, 50, 50), allow the model to explore a range of architectures. Smaller hidden layers are suited for simpler problems or datasets, while larger and deeper layers can capture complex patterns and relationships. By testing various sizes, the grid search ensures the network structure aligns with the complexity of the data.

- **Activation Function-** Activation functions, including ReLU, sigmoid, and tanh, are tested to determine the best fit for capturing non-linear relationships in the data. ReLU is popular for its computational efficiency and avoidance of vanishing gradient issues. Sigmoid and tanh are also included, as they may perform better in certain scenarios, with tanh offering centered outputs for faster convergence.

- **Solver -** Simply, the sgd (Stochastic Gradient Descent) solver was selected as it is more flexible when integrated with learning rate strategies. Other solvers such as adam are usually utilized, but since the focus of the grid search performed is on how different learning rates and activation functions affect the model performance, sgd will be used. This assists in identifying computations that must be made in order to balance between computational efficiency and accuracy.

- **Learning rate -** Several forms of learning rate are incorporated in this experiment such as constant, adaptive and invscaling to analyse the behaviour of the model under various conditions. In constant learning rate, updates are made with fixed amount. In adaptive, updates are adjusted when the loss stops improving. Finally, learning rate invscaling is helpful since it lowers the learning rate with time which helps the model become more intricate as the training goes on. These techniques allow more room for the training to be optimized.

- **Learning rate initialization-** A learning rate of 0.001 has been chosen at the start because it is a default value that seems to provide a good rate of convergence. Hence it may be a good initial value for many training cases as an initial parameter that may be too high would increase the chances of missing out on the best solution. This allows the model to hold a smooth trajectory towards reducing the loss function during training.

After setting up the parameter grid we have done the grid search to find the best fitted model. The parameters of the grid search was set as follows[Figure 30],

- Learning model- MLP Regressor with max iterations 500, random state 42 , early stopping=true, validation fraction=0.2
- n_jobs=-1
- cv=5
- scoring – r2, neg_mean_squared_error
- refit = r2
- verbose= 4

the reasons for those decisions are as follows,

- **learning model**- The MLP Regressor was employed due to its nonlinear relationships in the data making it worthy to model complex stability dynamics. The parameter max_iter=500 makes sure that the model does not take overly large amounts of time while converging. The parameter random_state=42 makes sure that the results can be reproduced since the chances in initializing weights and splitting the data are fixed. Early stopping (early_stopping=True) allows tracking of the validation score of the model in training and stops further training in case there is no improvement in validation score, therefore reducing the possibility of overfitting and wasting computer resources. The parameter validation_fraction=0.2 sets aside 20% of training data for validation purposes during early stopping so that the model can generalize.
- **n_jobs**- Setting the n_jobs parameter to -1 is useful in this situation since it tells the grid search to use all possible CPU cores. Effectively lowers the time required for tuning hyperparameters, especially in cases when big parameter grids are searched or when the models are computationally heavy like neural networks. All parameters are split evenly across all the processors, and therefore the grid search can provide results in the reasonable time frame even for large data sets.
- **cv**- The application of 5-fold cross-validation ensures that the performance of the model is rigorously assessed through mechanisms, which in this case was to divide the dataset into five distinct parts. The aimed part was the validation set consisting of one subset while other four subsets were used to train the model so that a variety of data splits are available for testing purposes. This reduces the chances of overfitting and is able to make a broad evaluation of the model's performance with respect to the different data distributions. Evaluating the model with a 5-fold strategy appears to be a compromise between time efficiency and accuracy of performance evaluation
- **scoring** - R^2 (coefficient of determination), Measures how well the model explains the variance in the target variable, serving as a general indicator of performance. Negative Mean Squared Error (MSE), Quantifies the prediction error by penalizing larger deviations between predicted and actual values. By using both metrics, the model's accuracy is assessed from different perspectives, ensuring a thorough evaluation of its predictive capability.
- **Refit**- The refit='r2' argument means that the model will refit using the parameter combination with the best R^2. This verifies the final model is alpha maximum in explained variance for overall data. R^2 is the perfect refit metric in this case since it directly represent how well the model is able to capture the relationships in the dataset and then explain them.
- **Verbose**- Setting verbose=4 ensures detailed logging of the grid search process. This verbosity level provides insights into the progress of hyperparameter evaluations, including the combination being tested and the corresponding scores.

After running the grid search model the best parameters that will fit our model were found as follows,

- **Activation function**- relu
- **Hidden layer size**-50,50,50
- **Learning rate** – adaptive
- **Solver** – sgd

Thus, upon showcasing the top model, its performance on the test data was finally inspected. It scored 0.93 on the test, showing high accuracy and good model prediction capabilities [Figure 31]. But it is very important to check whether you overfitting the model or not. Overfitting is when the model learns the training data too well, performing it excellently, but does not generalize as well to unseen data.

To check whether the model is overfitting first we have check the training score and the cross validation scores of the model. For the training score we got a score of 0.94 and for the validation score we got a score of 0.92. Since the scores are very close, this indicates that the model generalizes well and is not overfitting. The minimal difference between the training and validation scores suggests that the model performs consistently across both the training data and unseen validation data, demonstrating a good balance between bias and variance.

```
train_score = grid_search.score(X_train, y_train)

val_score = grid_search.best_score_

print(f"Training score (R²): {train_score}")
print(f"Cross-validated validation score (R²): {val_score}")
```

```
Training score (R²): 0.9454593185539981
Cross-validated validation score (R²): 0.9230996556140088
```

Figure 13 EVALUATING TRAINING SCORE AND VALIDATION SCORE

Furthermore, we have plot the learning curve to for the **R^2 score** as a function of the training set size for both the training score and validation score. This visualization provides valuable insights into the model's learning behavior, generalization capability, and performance stability. By examining the relationship between training size and performance, we can assess how well the model is utilizing the data and identify potential issues such as overfitting or underfitting[Figure 32].

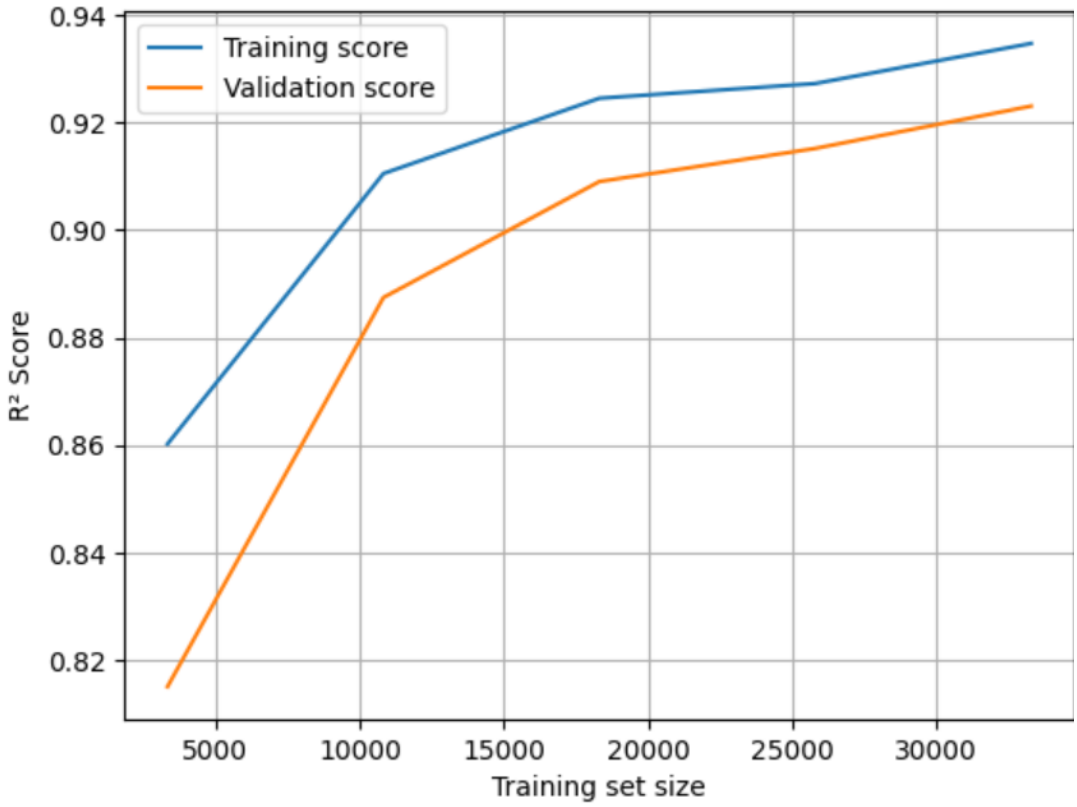


Figure 14 LEARNING CURVE SHOWING R^2 SCORE FOR TRAINING AND VALIDATION SETS AS A FUNCTION OF TRAINING SET SIZE

Based on the above graph we can come to the following conclusions.

- Both the training and validation scores are relatively high, with the training score around 0.94 and the validation score reaching approximately 0.92. This indicates that the model is performing well and generalizing effectively to unseen data.
- From the small gap between the training and validation curves we can say that the model has not overfit the training data. It performs consistently across both the training set and validation set.
- The validation score plateaus after a specific number of samples (about 20,000–30,000 examples), indicating that the model is performing at its best with the features and parameters in place. Performance is unlikely to be considerably enhanced by adding more training data.
- The validation score is close to the training score and relatively high, which rules out underfitting.

Next, we calculated the Root Mean Squared Error (RMSE) and R^2 score for both the training set and the test set to evaluate the model's performance. The RMSE measures the average magnitude of prediction errors, with lower values indicating better accuracy. The R^2 score, on the other hand, evaluates the proportion of variance in the target variable that the model successfully explains, providing insight into its predictive strength[Figure 33].

- Training RMSE: 7.355763
- Training R^2: 0.945459
- Test RMSE: 7.980042
- Test R^2: 0.9362889

With a test R^2 of 0.93628 and a training R^2 of 0.9454, the model performs well and explains the majority of the variance in the target variable. The test's RMSE score of 7.9800 and training's 7.3557 are close, indicating reliable accuracy and strong generalization to new data. The model is stable and dependable for forecasting vehicle stability in practical applications because of the narrow discrepancy between training and test measures, which verifies that it is neither overfitting nor underfitting.

Next we have plotted the Learning Curve for Training and Test RMSE as a Function of Training Set Size[Figure 34].

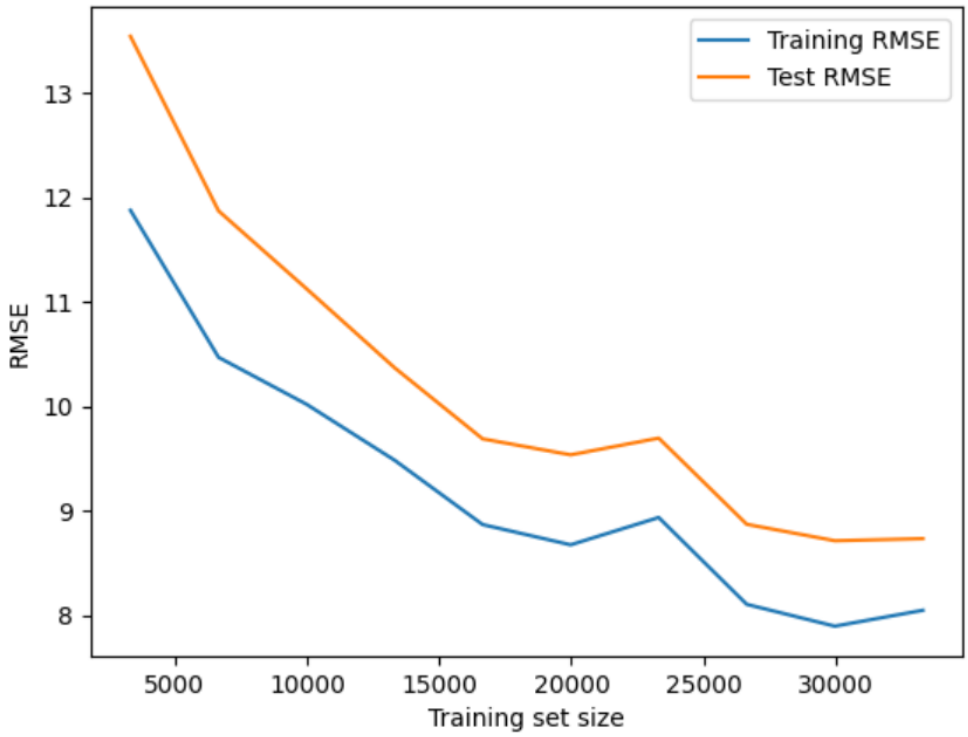


Figure 15 LEARNING CURVE FOR TRAINING AND TEST RMSE AS A FUNCTION OF TRAINING SET SIZE.

The learning curve shows that both training and test RMSE decrease with more data, converging as the training set size increases, this indicates a good generalization and minimal overfitting. The RMSE stabilizes beyond 25,000 samples. This suggests that sufficient training data for optimal model performance.

Finally, we have plotted the actual vs predicted slip angle values for 100 data points. To visualize our model with real values.

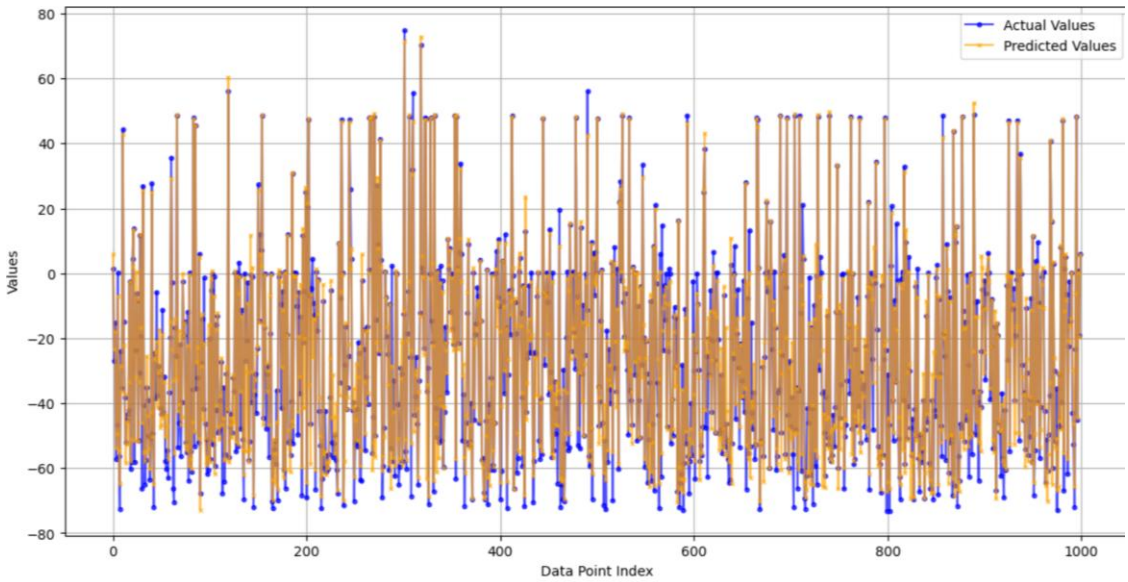


Figure 16 ACTUAL VS PREDICTED SLIP ANGLE VALUES

From all the above observations, we can conclude that the MLP Regressor model generalizes well and is pretty stable. Both the R^2 score and RMSE learning curves are showing a minimal gap between training and validation/test metrics, which means very slight overfitting. The model is steadily improving with more training data, eventually stabilizing at high performance, and generally yields reliable predictions on unseen data. These observations confirm that MLP Regressor suits the purpose and models real complicated nonlinear relations in this dataset.

6.2 Classification Modal Approach

In the second approach we have considered our problem statement as classification model problem and we have use SoftMax regression to model to build a machine learning model that can be used predict the current stability of the vehicle. In this method have classified the stability into three distinct classes, which are oversteer, understeer, neutral steer [Figure 35]. By assigning probabilities to each class the SoftMax regression model will predict the current stability of the vehicle is neutral steer understeer or oversteer.

To implement this approach, the dataset was first split into training and testing sets to ensure robust model evaluation. For the input features (**X variables**), the following were selected based on their relevance to vehicle stability

I. Right wheel steer
II. Left wheel steer
III. Driving steer input
IV. Lateral acceleration
V. Yaw rate
VI. Reaction force left front
VII. Reaction force left rear
VIII. Reaction force right front
IX. Reaction force right rear
X. Average rotational speed left
XI. Average rotational speed right

We have chosen the Stability as the Y(Target) variable. Next likewise in the previous model when splitting our dataset, we have stratified out dataset according to the target speed feature. Furthermore, in this approach we have divided our dataset in to 3 subgroups [Figure 36]. which are,

- Training set
- Testing set
- Validation set

After that when modeling our function using SoftMax regression first we have implement the model using TensorFlow to classify the vehicle stability into three distinct classes [Figure 37]. when implementing the model the following decisions were taken,

- For the model architecture sequential layer was created with four layers which are,
  - Input Layer- This layer receives input features representing various parameters related to the vehicle's dynamics (e.g., speed, steering angle, yaw rate, lateral acceleration, etc.).
  - First hidden layer- consists of 11 neurons, and the relu activation function was used because it avoids vanishing gradient problem which will help the model to learn effectively
  - Second hidden layer – consists of 10 neurons. Similarly as mentioned above relu function was used as the activation function.
  - Output layer- consists of three neurons which corresponds to oversteer, understeer and neutral steer. A linear activation function is used because this is a regression task where we predict continuous or scalar values for each output neuron.
- Loss function – for the loss function Sparse Categorical Cross entropy was used because it is well suited for multi-class classification tasks where the target variable is encoded as integers. Setting from_logits=True ensures the logits are correctly converted into probabilities.

- Optimizer- for the optimizer Adam optimizer was used because it is efficient in handling sparse gradients and has adaptive learning rate capabilities, which ensures faster convergence.
- Training and Validating- The model was trained on the training dataset (X_train and y_train) for 100 epochs, with the validation dataset (X_cv and y_cv) used to monitor the model's performance during training.

After training of the model using TensorFlow neural network. The model was input to the SoftMax function to get the corresponding final results[Figure 38].

To find the models performance, Next model has been evaluated using the test dataset and got an true positive accuracy score of 94.04%.[Figure 39]. Furthermore confusion matrix was plotted to observe the true performance of the model[Figure 40].

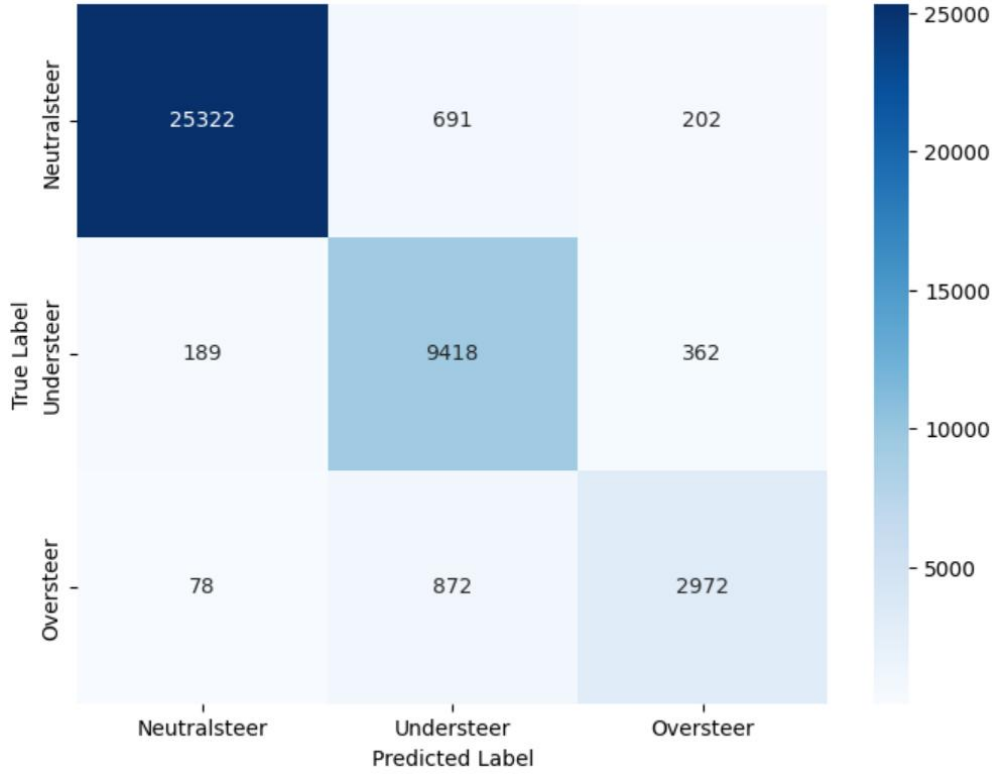


Figure 17 CONFUSION MATRIX FOR SOFTMAX REGRESSION

Next to check whether model is overfitting or underfitting the data training setloss, training set accuracy, cross validation set loss and cross validation set accuracy was found[Figure 41]. Which are;

- Training Set Loss (Cost Function): 0.1677
- Training Set Accuracy: 0.9420
- Cross-Validation Set Loss (Cost Function): 0.1717
- Cross-Validation Set Accuracy: 0.9408

The close concordance between these metrics disproves overfitting and underfitting, showing that the model generalizes well. This means the model is durable and reliable and can be used for the prediction of vehicle stability in practice.

Chapter 7

MODEL DEVELOPMENT AND EVALUATION FOR FUTURE STABILITY DETECTION

As explained in the previous chapter, we successfully developed models to predict the current stability of the vehicle using the MLP Regressor and SoftMax Regression. These models demonstrated robust performance in accurately identifying the vehicle's stability state in real time.

In this chapter, the emphasis lies on the models that were used to project the future state of the stability of a vehicle. The objective is to predict the stability state of the vehicle two seconds ahead, thereby, empowering drivers or autonomous systems with concrete information to avert possible instability. For this model implementations two methods has been considered

- Future stability prediction using MLP Regressor model.
- Future stability prediction using Markov chain process.

7.1 Future Stability Detection Using MLP Regressor

In this approach the aim is to develop a machine learning model that can predict vehicles slip angle difference before 2 second's time. Since through slip angle difference we can directly defines vehicle stability, hence we can predict vehicle stability in the future. For the development of the machine learning model following features has been chosen as the input variables (X variables).

I. Right wheel steer
II. Left wheel steer
III. Driving steer input
IV. Lateral acceleration
V. Yaw rate
VI. Reaction force left front
VII. Reaction force left rear
VIII. Reaction force right front
IX. Reaction force right rear
X. Average rotational speed left
XI. Average rotational speed right
XII. Current Slip angle difference

For the output variable of our model (Y variable) slip angle difference after 2 seconds time is being chosen. Hence to train this model a new feature called Slip angle difference after 2 seconds has being formed(for the sake of coding simplicity this feature is called as FUTA)[Figure 42].

For this MLP regressor modeling, First we have set up a grid search matrix to find the best model parameters. For the parameter grid search matrix following parameters were set[Figure 43],

- Hidden layer sizes= (50, 50), (50,50,50),(100,100,100)
- Activation function= relu, sigmoid, TanH
- Solver= sgd
- Learning rate = adaptive, invers scaling
- Learning rate_init= 0.001

The reasons for this decisions are as follows

- **Hidden layer sizes -** multi-layer configurations, such as (50, 50) , (50, 50, 50)OR (100,100,100) enable the model to experiment with different architectures. Smaller layers work well for simpler datasets, while larger, deeper layers capture more complex patterns. Grid search helps optimize the network structure to match the data's complexity.
- **Activation Function-** Various activation functions, such as ReLU, sigmoid, and tanh, are evaluated to identify the most effective for capturing non-linear relationships in the data. ReLU is widely used for its efficiency and ability to prevent vanishing gradient issues, while sigmoid and tanh are tested for their potential advantages in specific cases. Tanh, in particular, provides centered outputs, which can accelerate convergence.
- **Solver -** The Stochastic Gradient Descent (SGD) solver was chosen for its flexibility with learning rate strategies. While solvers like Adam are commonly used, the focus here is on evaluating the impact of different learning rates and activation functions on model performance. Using SGD helps balance computational efficiency and accuracy during this grid search.
- **Learning rate -** This experiment incorporates various learning rate strategies—constant, adaptive, and invscaling—to evaluate the model's performance under different conditions. With a constant learning rate, updates are made at a fixed rate. Adaptive learning adjusts the rate when the loss stops improving, while invscaling gradually reduces the learning rate over time, allowing the model to fine-tune as training progresses. These approaches enhance optimization and improve training efficiency.
- **Learning rate initialization-** An initial learning rate of 0.001 has been selected as it is a standard default value that often ensures a good rate of convergence. This value reduces the risk of overshooting the optimal solution, enabling the model to follow a smooth and steady path toward minimizing the loss function during training.

After setting up the parameter grid we have done the grid search to find the best fitted model. The parameters of the grid search was set as follows[Figure 30],

- Learning model- MLP Regressor with max iterations 500, random state 42 , early stopping=true, validation fraction=0.2
- n_jobs=-1
- cv=5
- scoring – r2, neg_mean_squared_error
- refit = r2
- verbose= 4

the reasons for those decisions are as follows,

- **learning model**- The MLP Regressor was used to model complex, non-linear stability dynamics. Key parameters include max_iter=500 to limit training time, random_state=42 for reproducibility, and early_stopping=True to halt training if validation performance stops improving, preventing overfitting and saving resources. Additionally, validation_fraction=0.2 reserves 20% of the training data for validation during early stopping to enhance generalization.
- **n_jobs**- Setting n_jobs=-1 allows the grid search to utilize all available CPU cores, significantly reducing the time required for hyperparameter tuning. This is especially beneficial for large parameter grids or computationally intensive models like neural networks, enabling faster results even with large datasets.
- **cv**- The use of 5-fold cross-validation rigorously evaluates the model by dividing the dataset into five parts, with one subset for validation and the others for training. This approach reduces overfitting, ensures performance across different data distributions, and balances efficiency and accuracy in model evaluation.
- **scoring** -R^2 evaluates how well the model explains variance, while Negative MSE measures prediction error, penalizing large deviations. Together, they provide a comprehensive assessment of the model's predictive performance..
- **Refit**- The refit='r2' argument ensures the model is refitted using the parameter combination that achieves the best R^2 score, maximizing explained variance. R^2 is an ideal refit metric as it directly reflects the model's ability to capture and explain relationships in the dataset.
- **Verbose**- Setting verbose=4 enables detailed logging of the grid search process, offering insights into hyperparameter evaluation progress, including tested combinations and corresponding scores. This transparency aids in monitoring and diagnosing the model tuning process.

After running the grid search model the best parameters that will fit our model were found as follows,

- **Activation function**- relu
- **Hidden layer size**-50,50,50
- **Learning rate** – adaptive
- **Solver** – sgd

Thus, upon showcasing the top model, its performance on the test data was finally inspected. It scored 0.84 on the test, showing high accuracy and good model prediction capabilities [Figure 45]. But it is very important to check whether you overfitting the model or not. Overfitting is when the model learns the training data too well, performing it excellently, but does not generalize as well to unseen data.

First, the training score and the cross-validation score were evaluated. The training score of the model is 0.84, while the cross-validation score is 0.81. This implies that the model performs well on the training data and generalizes effectively to unseen data. The small difference between the two scores indicates minimal overfitting, suggesting that the model captures the underlying patterns in the dataset while maintaining robust predictive performance across different data splits[Figure 46].

Additionally, we plotted the learning curve to visualize the R^2 score as a function of training set size for both the training and validation scores. This visualization offers valuable insights into the model's learning dynamics, generalization ability, and performance stability. By analyzing the relationship between training size and performance, we can evaluate how effectively the model utilizes the data and detect potential issues such as overfitting or underfitting [Figure 47].

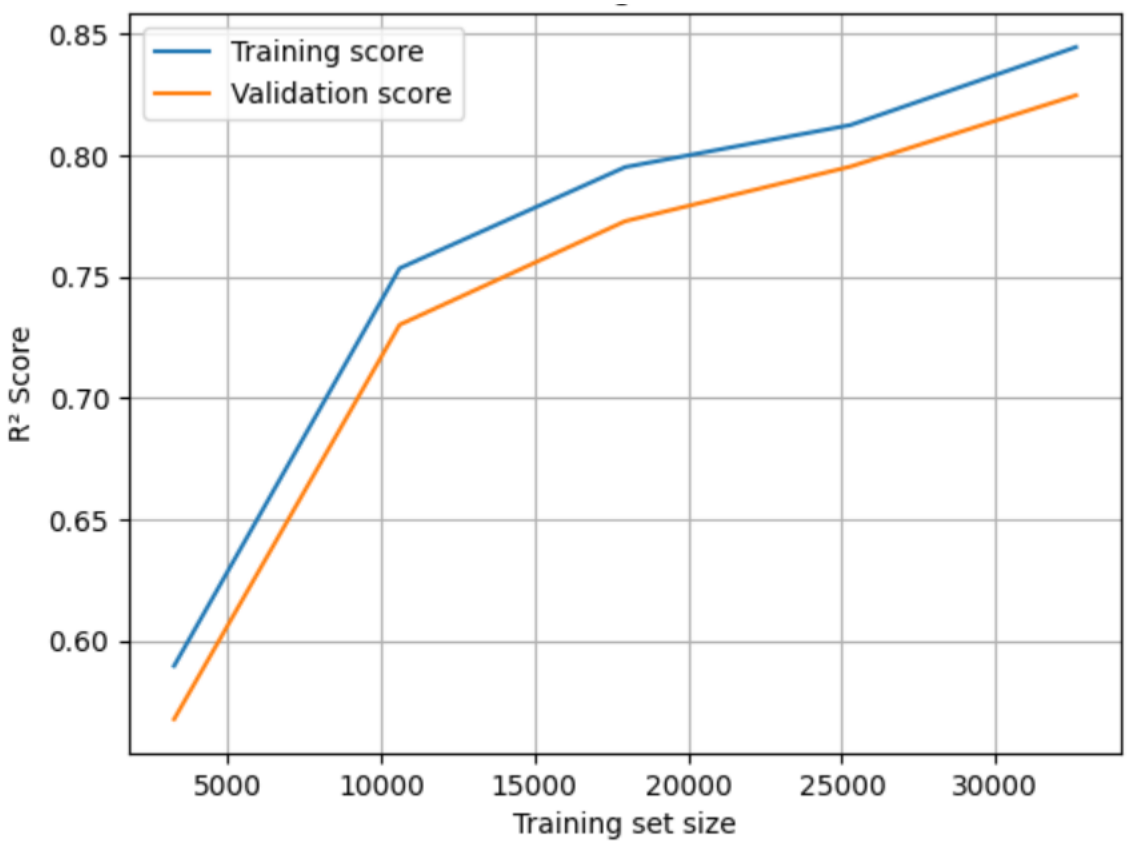


Figure 18 LEARNING CURVE SHOWING R^2 SCORE FOR TRAINING AND VALIDATION SETS AS A FUNCTION OF TRAINING SET SIZE FOR FUTURE DETECTION

Based on the graph, we can draw the following conclusions:

- Both the training and validation R^2 scores are high, with the training score reaching approximately 0.85 and the validation score around 0.82, indicating strong model performance and effective generalization to unseen data.
- The small gap between the training and validation curves suggests that the model is not overfitting. It performs consistently across both datasets, which demonstrates good balance.
- The validation score plateaus after approximately 25,000–30,000 training samples, suggesting that the model has achieved optimal performance with the current features and parameters. Adding more data is unlikely to significantly improve performance.
- The validation score being close to the training score further confirms that the model is not underfitting and is effectively capturing the underlying patterns in the data.

This analysis indicates that model can do reliable predictions

We then evaluated the model's performance by calculating the Root Mean Squared Error (RMSE) and R^2 score for both the training and test datasets. RMSE quantifies the average prediction error, where lower values indicate higher accuracy. Meanwhile, the R^2 score measures how well the model explains the variance in the target variable, offering a clear indicator of its predictive capability.

- Training RMSE: 11.9988
- Training R^2: 0.848092
- Test RMSE: 12.65928
- Test R^2: 0.83222

The model performs quite well, obtaining an R^2 score of 0.848 on the training set and 0.832 on the test set, hence it captures a good proportion of the variance of the target variable. The training RMSE 11.9988 and test RMSE 12.65928 are closely spaced values, thereby showing consistent accuracy as well as good generalization to unseen data with minimal overfitting. That said, while the model is robust and reliable, further feature engineering or hyperparameter tuning may result in even greater accuracy.

From the above all observations our mode is not overfitting and can be used to detect the future slip angle difference of the vehicle. Hence the future stability can be predicted.

7.2 Markov Chain Process Approach

A Markov chain is a mathematical model used to describe systems that transition between states in a probabilistic manner. This model is named after the Russian mathematician Andrey Markov, who first developed the concept in the early 20th century. Markov chains are widely used in various fields, including computer science, economics, physics, biology, and artificial intelligence, due to their ability to model dynamic systems and processes.

The defining property of a Markov chain is the Markov property, which states that the future state of the system depends only on the current state and not on the sequence of past states. This is also referred to as the property of memorylessness. Mathematically, if $X_n$represents the state of the system at step n, the Markov property can be expressed as:

$$P(X_{\{n+1\}} = x_{\{n+1\}} | X_n = x_n, X_{\{n-1\}} = x_{\{n-1\}}, .., X_0 = x_0) = P(X_{\{n+1\}} = x_{\{n+1\}} | X\_n = x_n)$$

This property makes Markov chains particularly simple and computationally efficient for modeling complex systems.

Components of a Markov Chain include;

1) State space (S)
   The set of all possible states the system can occupy. The state space can be:
   - **Finite**: A limited number of discrete states, e.g. $S = \{1,2,3\}$
   - **Infinite**: Countable or uncountable states, e.g., $S = N$ or $S = [0,1]$
2) Transition Probabilities $(\boldsymbol{P})$
   These are usually represented in a transition probability matrix for finite state spaces. For example:

$$P = \begin{bmatrix} P_{1,1} & P_{1,2} & \dots & P_{1,j} & \dots & P_{1,\alpha} \\ P_{2,1} & P_{2,2} & \dots & P_{2,j} & \dots & P_{2,\alpha} \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ P_{i,1} & P_{i,2} & \dots & P_{i,j} & \dots & P_{i,\alpha} \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ P_{\alpha,1} & P_{\alpha,2} & \dots & P_{\alpha,j} & \dots & P_{\alpha,\alpha} \end{bmatrix}.$$

3) Initial Distribution
   The probability distribution over the states at the start of the process. It specifies where the system begins.
4) Time Steps
   Markov chains operate in discrete or continuous time. For discrete-time chains, transitions occur at fixed intervals, while for continuous-time chains, transitions occur at irregular intervals

The neural network provides probabilistic predictions for the vehicle's stability states (e.g., understeer, oversteer, neutral stability, or unstable). These softmax probabilities are then used as inputs for a Markov chain model to forecast the sequence of stability states over the next short time frame (2 seconds). Here's how this process is structured;

Step 1: Collecting and Preparing the Dataset

1. **Input Data**:
    - The dataset consists of the vehicle's state predictions, derived from the SoftMax output of the neural network.
    - Each prediction is a probability distribution over four stability states.
2. **Time Interval**:
    - The model is applied to a data sequence of size 40, representing the current states of the vehicle.
    - The 40 data points correspond to stability predictions sampled at 20 Hz (20 samples per second), covering a 2-second time window.

Step 2: Calculating the Transition Matrix

1. **Markov Chain Fundamentals**:
    - A **Markov chain** is a stochastic process where the future state depends only on the current state, not on the sequence of states that preceded it. This property is called the **Markov property**.
    - In this case, the "states" are the four stability conditions: understeer, oversteer, neutral stability, and unstable.
2. **Transition Matrix**:
    - The transition matrix is a 4×4 matrix representing the probabilities of transitioning from one stability state to another. Each element in the matrix, $P_{(i,j)}$, indicates the probability of moving from state ii to state jj.
    - To calculate the transition matrix:
        - Analyze the sequence of 40 predicted states.
        - Count the number of transitions between states (e.g., understeer to oversteer).
        - Normalize the counts for each row to ensure the probabilities sum to 1.

Example: For a sequence like neutral → oversteer → understeer, the transitions recorded might be:

- Neutral → Oversteer
- Oversteer → Understeer

These transitions contribute to the counts in the matrix, which are then normalized.

Step 3: Predicting the Next 40 States

1. **Initial State Distribution**:
   - Start with the current state distribution from the softmax output, which provides the probabilities of the four states at the current timestep.
2. **Future State Prediction**:
   - Using the transition matrix, predict the **next 40 states** (covering 2 seconds).
   - This involves iteratively applying the transition matrix to the current state distribution: $P_t + 1 = P_t . T$ Where:
     - $P_{\{t\}}$ is the state probability distribution at time tt,
     - T is the transition matrix.
     - $P_t + 1$ becomes the input for the next timestep.

The implementation of this Markov chain process in python language is shown in Figure 49 Figure 50

Chapter 8

MODEL VALIDATION

In validating our model, we had the common problem of not having real sensor data. We solved this by creating an entirely new simulation environment that would produce new, realistic data for us to test on. We then tested our model rigorously with this newly collected data. Thus, the validation process is made reliable and robust in the absence of real-world sensor data and demonstrates the ability of the model to generalize properly over new scenarios.

8.1 MLP Regressor Model Validation

After getting the new simulation data as discussed in the Chapter 5 we have scaled the data accordingly and fed it into our Current stability predicting MLP Regressor model, and obtained the predicted current stability values [Figure 51]. Since from the MLP Regressor model the predicted value is slip angle difference it has to be converted to the stability states. The conversion from slip angle difference to stability states is done in Figure 52. Finally, a Confusion Matrix was plotted[Figure 53] to visualize the predicted current stability states which has a true positive accuracy of 85.12%.

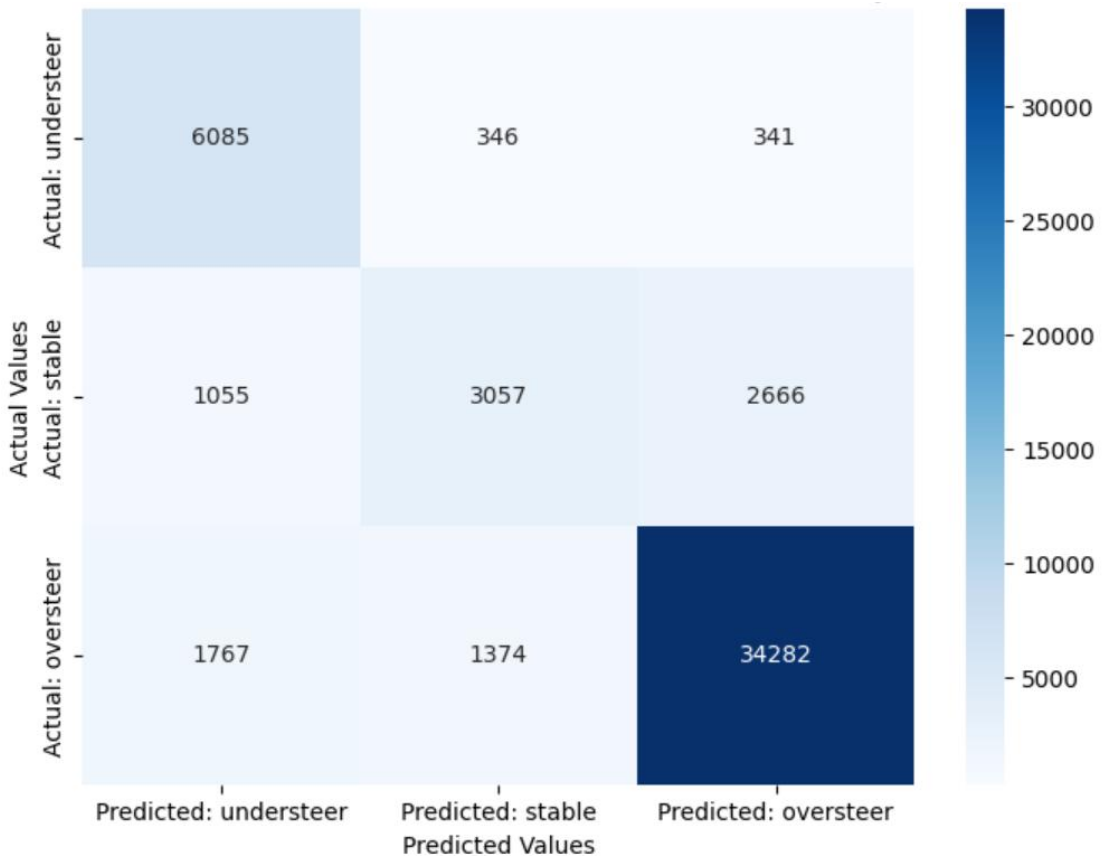


Figure 19 CONFUSION MATRIX FOR ACTUAL VS PREDICTED CURRENT STABILITY STATES

Next we have input the predicted slip angle difference data to the Future predicting MLP Regressor model and evaluate the predicted future slip angle difference values. Here also since from the MLP Regressor model the predicted value is slip angle difference it has to be converted to the stability states. The conversion from slip angle difference to stability states is done in Figure 52. Finally, a Confusion Matrix was plotted[Figure 54] to visualize the predicted current stability states which has a true positive accuracy of 84.55%.

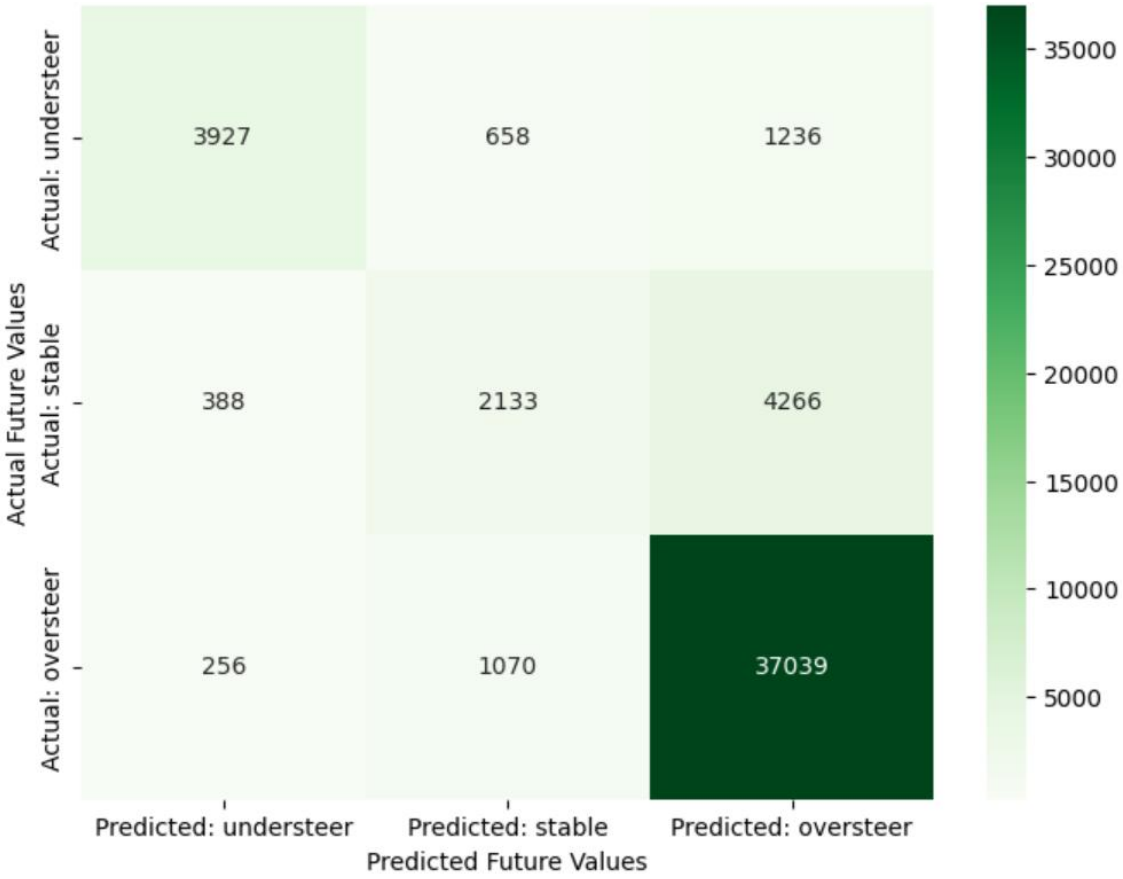
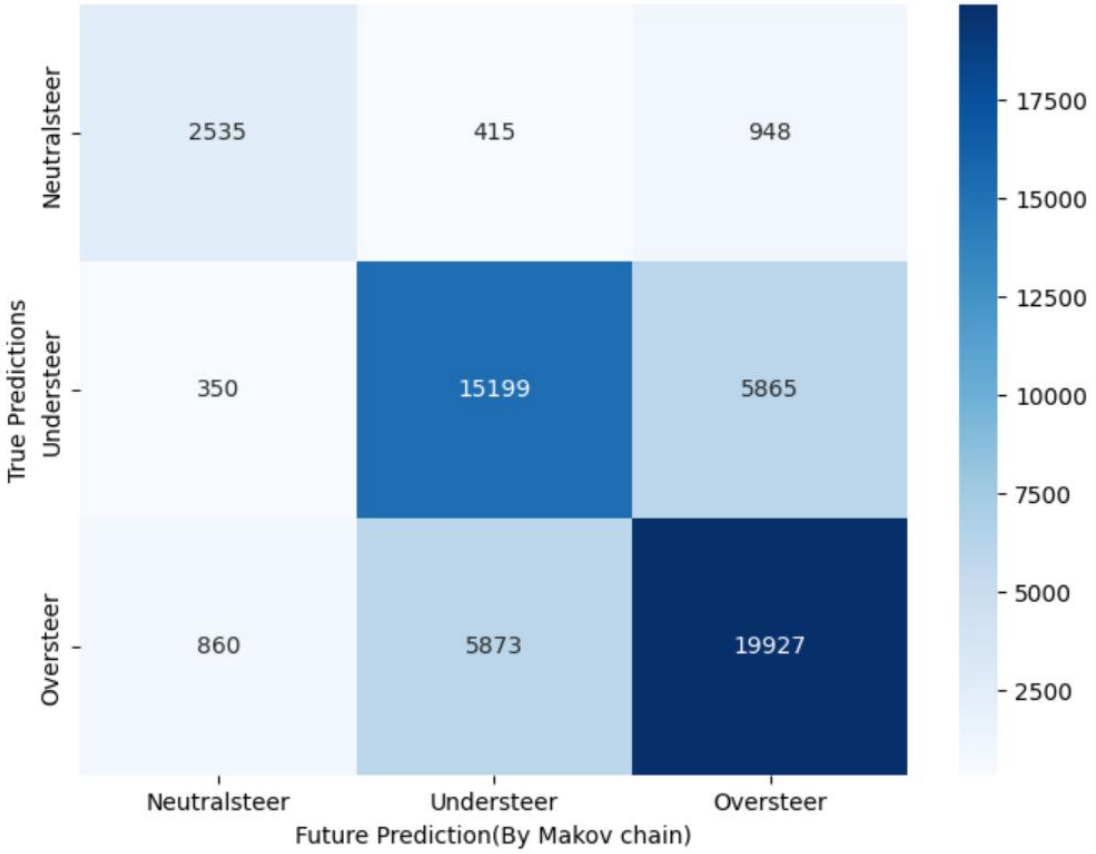


Figure 20 CONFUSION MATRIX TO ACTUAL VS PREDICTED FUTURE STABILITY STATES

8.2 SoftMax and Markov Chain Process Approach

In this approach after getting the new simulation data as discussed in the Chapter 5 we have scaled the data accordingly and fed it into our Current stability predicting SoftMax Regressor model, and obtained the predicted current stability values [**Error! Reference s ource not found.**].After that those obtained current stability values are then being input to the Markov chain process and the future stability values are being obtained. The confusion matrix was plotted to visualize the final values and this approach has an overall accuracy score of 85%.



Chapter 9

CONCLUSION AND RECOMENDATIONS

9.1 CONCLUSION

The MLP Regressor models, which predict vehicle stability, mark a significant step forward in the development of advanced driver assistance systems. These models address precisely the gap created by limited access to real sensor data in the context of a carefully crafted simulation-based approach. When it comes to predicting current and future stability states, at least one model showed robustness with true positive accuracy rates of 85.12% and 84.55%, respectively, for current and future stability predictions. Results like these truly emphasize the models' performance capabilities in generalizing at rather unsighted data: qualities that make them appropriate for any practical implementation. In addition to this, confusion matrices made it diagnose reliably - thereby providing applicable information regarding vehicle stability while being driven under varying conditions. This work sets up machine learning-based stability prediction integration into ADAS for enhanced safety along with vehicle performance.

9.2 Recommendations

- **Integration with Real Sensor Data**:
  While the simulation-based approach proved effective, integrating the models with real-world sensor data would further validate their performance and enhance reliability. Collaboration with automotive manufacturers or research institutions could facilitate access to such data.
- **Hybrid Model Development**:
  Future efforts should explore combining MLP Regressors with other machine learning models, such as Markov Chains or Long Short-Term Memory (LSTM) networks, to capture temporal dependencies and improve prediction accuracy.
- **Real-Time Implementation**:
  To maximize practical impact, the models should be optimized for deployment in real-time systems. This includes ensuring computational efficiency and testing under real-world driving conditions.
- **Expanding the Dataset**:
  Collecting additional simulation data that encompasses a broader range of driving scenarios, including adverse weather conditions and extreme maneuvers, would help improve the model's robustness and generalizability.

By addressing these recommendations, the models can evolve into a fully adaptive system, significantly contributing to vehicle safety and the advancement of autonomous driving technology.

Chapter 10

FUTURE WORK

The success of this project highlights several promising directions for future development and enhancements. While the current models demonstrate robust performance in predicting both current and future vehicle stability states, there are opportunities to further advance their accuracy, reliability, and applicability. The following areas are recommended for future work:

- **Development of a Fully Automated ADAS**:
  Future work will focus on integrating the stability prediction models into a fully automated Advanced Driver Assistance System (ADAS). This system would utilize predicted stability states to actively intervene, such as applying brakes or adjusting steering, to maintain vehicle control.
- **Real-Time Optimization:**
  To facilitate deployment in real-world vehicles, the models need to be optimized for real-time operation. This includes reducing computational complexity and ensuring minimal latency in predictions and interventions**.**
- **Testing and Validation in Real-World Conditions**
  Conducting extensive on-road tests and long-term validation studies will ensure that the models perform consistently and reliably under diverse driving scenarios and conditions**.**
- **Adapting to Autonomous Vehicles**
  The models can be tailored for integration into autonomous vehicles, where stability prediction and control play a critical role in ensuring safety and smooth operation

Following these future works, the models developed in this project can continue to scale toward a full-fledged, real-time stability control system that would significantly enhance safety and safety compliance and further propel the development of fully autonomous as well as semi-autonomous driving systems. This upgraded system will handle accurate predictions, timely intervention, and solid performance to create an environment of safer, smarter, and more effective transportation solutions.

References

1. in.mathworks.com. (n.d.). *Detecting Oversteering in BMW Automobiles with Machine Learning*. [online] Available at: https://in.mathworks.com/company/technical-articles/detecting-oversteering-in-bmw-automobiles-with-machine-learning.html.
2. Stone, R. and Ball, J.K. (2004). *Automotive Engineering Fundamentals*. SAE International.
3. Reif, K. (2014). *Fundamentals of Automotive and Engine Technology Standard Drives, Hybrid Drives, Brakes, Safety Systems*. Wiesbaden Springer Fachmedien Wiesbaden, Imprint: Springer Vieweg.
4. Zhang, Y., Amir Khajepour and Xie, X. (2015). Rollover prevention for sport utility vehicles using a pulsed active rear-steering strategy. *Proceedings of the Institution of Mechanical Engineers, Part D: Journal of Automobile Engineering*, 230(9), pp.1239–1253. doi:https://doi.org/10.1177/0954407015605696.

Appendices

Importing the excel data to a pandas data frame

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import  matplotlib.pyplot as plt


df=pd.read_excel('Final.xlsx')
```

Figure 21 IMPORTING DATA TO A DATAFRAME

Making new Stability feature

```
#to create the new stabiliuty column
# Create a new 'Stability' feature
df['Stability'] = df['Slip angle difference'].apply(
    lambda x: 1 if x > 4.5
    else -1 if x < -4.5
    else 0)
```

Figure 22 STABILITY COLUMNS DEFINITION

New feature of rotational speed left and rotational speed right.

```
df["Average Rotational speed left"]=df[['Rotation speed left front','Rotation speed left rear']].mean(axis=1)
df["Average Rotational speed right"]=df[['Rotation speed right front','Rotation speed right rear']].mean(axis=1)
#let us drop the unwanted coumns now
df.drop(['Rotation speed left front','Rotation speed right front','Rotation speed left rear','Rotation speed right rear'],axis=1,inplace=True)
```

Figure 23 NEW FEATURE ROTATIONAL SPEED LEFT AND ROTATIONAL SPEED RIGHT

Scaling of Data

```
for column in df.columns:
    min_value = df[column].min()
    max_value = df[column].max()
    print(f"Feature: {column}")
    print(f"Min: {min_value}")
    print(f"Max: {max_value}")
    print()
```

Figure 24 MIN MAX VALUE DETECT CODE

```
Feature: Time
Min: 0.0
Max: 100.0

Feature: Target Speed
Min: 78
Max: 96

Feature: Left wheel steer
Min: -39.99219965
Max: 60.79622045

Feature: Right wheel steer
Min: -60.78581964
Max: 39.97447733

Feature: Driver steering input
Min: -720.0
Max: 720.0

Feature: Lateral Acceleration
Min: -3.969002523
Max: 2.263532324

Feature: Yaw rate
Min: -229.6057972
Max: 210.2150181
```

```
Feature: Reaction force left rear
Min: 0.0
Max: 46112.90868

Feature: Reaction force right front
Min: 0.0
Max: 31930.71697

Feature: Reaction Force right rear
Min: 0.0
Max: 23252.67335

Feature: Slip angle difference
Min: -73.886315725
Max: 74.79464335649999

Feature: Stability
Min: -1
Max: 1

Feature: Average Rotational speed left
Min: 0.0015863205000000002
Max: 904.6519169000001

Feature: Average Rotational speed right
Min: 0.0015863235
Max: 899.8600243999999
```

Figure 25 MIN MAX VALUE OF FEATURES BEFOR SCALING

```
#to perform min max scaling
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler(feature_range=(-1,1))
scaler1=MinMaxScaler()
df[['Left wheel steer','Right wheel steer']] = scaler.fit_transform(df[['Left wheel steer','Right wheel steer']])
df[['Driver steering input','Yaw rate']]=scaler.fit_transform(df[['Driver steering input','Yaw rate']])
df[['Average Rotational speed left','Average Rotational speed right']]=scaler1.fit_transform(df[['Average Rotational speed left','Average Rotational speed right']])
```

Figure 26 SCALLING OF FEAATURES EXCEPT REACTION FORCES

```
df['Reaction force left rear'] = df['Reaction force left rear']/10000
df['Reaction force left front'] = df['Reaction force left front']/10000
df['Reaction force right front'] = df['Reaction force right front']/10000
df['Reaction Force right rear'] = df['Reaction Force right rear']/10000
```

Figure 27 SCALING OF REACTION FORCES

Train Test split of the data

```
from sklearn.model_selection import train_test_split

X = df.drop(columns=['Target Speed','Time','Slip angle difference','Stability'])  # Features
z = df['Target Speed']  # Target variable
y = df['Slip angle difference']  # Target variable

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, stratify=z, random_state=42)
X_train.head()
```

Figure 28 TRAIN TEST SPLIT OF THE DATA

MLP Regressor Model Training

```
from sklearn.model_selection import GridSearchCV
from sklearn.neural_network import MLPRegressor



param_grid = {
    'hidden_layer_sizes': [(10,), (50,), (100,), (10, 10), (50, 50), (50,50,50)],
    'activation': ['relu'],
    'solver': ['sgd'],
    'alpha': [0.0001,],  # Regularization parameter
    'learning_rate': ['adaptive',],
    'learning_rate_init': [0.001],  # Initial learning rate
```

Figure 29 MLP REGRESSOR TRAINING PARAMETER GRID

```
grid_search = GridSearchCV(
    MLPRegressor(max_iter=500, random_state=42, early_stopping=True, validation_fraction=0.2),
    param_grid,
    n_jobs=-1,  # Utilize all available processors
    cv=5,  # 5-fold cross-validation
    scoring={'r2': 'r2', 'neg_mean_squared_error': 'neg_mean_squared_error'},
    refit='r2',  # Use R2 as the metric to refit the best model
    verbose=4  # Higher verbosity for detailed output
)
grid_search.fit(X_train, y_train)

print("Best parameters found: ", grid_search.best_params_)
```

Figure 30 MLP REGRESSION GRID SEARCH IMPLEMENTATION

```
current=grid_search.best_estimator_
current.score(X_test,y_test)
```

```
0.9362889815939206
```

Figure 31 SCORE OF MLP REGRESSOR

```python
import matplotlib.pyplot as plt
from sklearn.model_selection import learning_curve

# Generate learning curve data
train_sizes, train_scores, val_scores = learning_curve(
    MLPRegressor(hidden_layer_sizes=grid_search.best_params_['hidden_layer_sizes'],
                 activation=grid_search.best_params_['activation'],
                 solver=grid_search.best_params_['solver'],
                 alpha=grid_search.best_params_['alpha'],
                 learning_rate=grid_search.best_params_['learning_rate'],
                 max_iter=500, random_state=42, early_stopping=True, validation_fraction=0.2),
    X_train, y_train, cv=5, scoring='r2', n_jobs=-1
)

# Compute mean and standard deviation of training/validation scores
train_scores_mean = train_scores.mean(axis=1)
val_scores_mean = val_scores.mean(axis=1)

# Plot learning curves
plt.plot(train_sizes, train_scores_mean, label="Training score")
plt.plot(train_sizes, val_scores_mean, label="Validation score")

plt.xlabel("Training set size")
plt.ylabel("R² Score")
plt.title("Learning Curves")
plt.legend(loc="best")
plt.grid()
plt.show()
```

Figure 32 CODE TO GET LEARNING CURVE SHOWING R² SCORE FOR TRAINING AND VALIDATION SETS AS A FUNCTION OF TRAINING SET SIZE

```python
from sklearn.metrics import r2_score, mean_squared_error
import numpy as np

# Evaluate model on training data
y_train_pred = grid_search.best_estimator_.predict(X_train)
train_rmse = np.sqrt(mean_squared_error(y_train, y_train_pred))
train_r2 = r2_score(y_train, y_train_pred)

# Evaluate model on test data
y_test_pred = grid_search.best_estimator_.predict(X_test)
test_rmse = np.sqrt(mean_squared_error(y_test, y_test_pred))
test_r2 = r2_score(y_test, y_test_pred)

print(f"Training RMSE: {train_rmse}")
print(f"Training R^2: {train_r2}")

print(f"Test RMSE: {test_rmse}")
print(f"Test R^2: {test_r2}")
```

```
Training RMSE: 7.355763009512477
Training R^2: 0.9454593185539981
Test RMSE: 7.980042508273912
Test R^2: 0.9362889815939206
```

Figure 33 R^2 AND RMSE VALUES FOR TRAIN AND TEST SET

```
import matplotlib.pyplot as plt
from sklearn.model_selection import learning_curve

train_sizes, train_scores, test_scores = learning_curve(
    grid_search.best_estimator_,
    X_train, y_train,
    cv=5, n_jobs=-1,
    train_sizes=np.linspace(0.1, 1.0, 10),
    scoring='neg_mean_squared_error'
)

# Calculate the mean and standard deviation of the training/test scores
train_rmse_mean = np.sqrt(-train_scores.mean(axis=1))
test_rmse_mean = np.sqrt(-test_scores.mean(axis=1))

plt.plot(train_sizes, train_rmse_mean, label='Training RMSE')
plt.plot(train_sizes, test_rmse_mean, label='Test RMSE')
plt.ylabel('RMSE')
plt.xlabel('Training set size')
plt.legend()
plt.title('Learning Curve')
plt.show()
```

Figure 34 CODE FOR LEARNING CURVE FOR TRAINING AND TEST RMSE AS A FUNCTION OF TRAINING SET SIZE

Classification Model Approach

```
#to create the new stabiliuty column
# Create a new 'Stability' feature
df['Stability'] = data['Slip angle difference'].apply(
    lambda x: 'Oversteer' if x > 4.5
    else ('Understeer' if x < -4.5
    else 'Neutralsteer')
)
df.head()
```

Figure 35 CLASSIFYING STABILITY INTO THREE DIFFERENT CLASSES

```
from sklearn.model_selection import train_test_split

# Separate features and target variable
features = df.drop(columns=["Stability","Time","Slip angle difference"])
target = df["Target Speed"]
y_target = df['Stability']

# Perform stratified split: 60% train, 20% validation, 20% test
X_train, X_temp, y_train, y_temp = train_test_split(features, y_target, test_size=0.4, stratify=target, random_state=42)
X_cv, X_test, y_cv, y_test = train_test_split(X_temp, y_temp, test_size=0.5, stratify=X_temp['Target Speed'], random_state=42)

# Sizes of each split
split_sizes = {
    "Training Set": len(X_train),
    "Validation Set": len(X_cv),
    "Test Set": len(X_test)
}

split_sizes
```

✓ 0.2s

```
{'Training Set': 120318, 'Validation Set': 40106, 'Test Set': 40106}
```

```
X_train = X_train.drop(columns = 'Target Speed')
X_cv = X_cv.drop(columns = 'Target Speed')
X_test = X_test.drop(columns = 'Target Speed')
```

✓ 0.0s

Figure 36 TRAIN TEST SPLIT FOR SOFTMAX REGRESSION

```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.losses import SparseCategoricalCrossentropy

model = Sequential([Dense(units=11,activation='relu'),
                    Dense(units = 10,activation='relu'),
                    Dense(units = 4,activation = 'linear')])

model.compile(optimizer = 'adam', loss = SparseCategoricalCrossentropy(from_logits=True))
model.fit(X_train,y_train,epochs = 100,validation_data=(X_cv,y_cv))
```

Figure 37 SOFTMAX REGRESSION MODEL IMPLEMENTATION USING TENSORFLOW`

```
logits = model(X_test)
f_x = tf.nn.softmax(logits)
```

```
import tensorflow as tf
predicted_categories = tf.argmax(f_x, axis=1)

# Convert to numpy array for better readability and display
predicted_categories_np = predicted_categories.numpy()
print("Predicted Categories:", predicted_categories_np)
```

Figure 38 FINAL SOFTMAX MODEL

```python
import numpy as np
import tensorflow as tf

# Get the logits (raw predictions) from the model on the test data
logits = model.predict(X_test)

# Convert logits to softmax probabilities
softmax_scores = tf.nn.softmax(logits).numpy()

# Get the predicted class labels (the index of the highest probability)
predicted_classes = np.argmax(softmax_scores, axis=1)

# Calculate accuracy by comparing predicted labels with true labels
accuracy = np.mean(predicted_classes == y_test)
print(f"Test Accuracy: {accuracy:.4f}")
```

Figure 39 CODE TO GET SOFTMAX MODEL ACCURACY ON TEST DATASET

```python
# Confusion Matrix Heat Map
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import confusion_matrix

# Get the predicted class labels
predicted_classes = np.argmax(model.predict(X_test), axis=1)

categories = ['Neutralsteer','Understeer', 'Oversteer']
# Generate confusion matrix
cm = confusion_matrix(y_test, predicted_classes)

# Plot the confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=categories, yticklabels=categories)
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.title('Confusion Matrix')
plt.show()
```

Figure 40 CODE FOR CONFUSION MATRIX OF SOFTMAX REGRESSION

```python
# Re-compile the model with the accuracy metric
model.compile(optimizer='adam',
              loss=SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])

# Now you can evaluate the model and obtain both loss and accuracy
train_loss, train_accuracy = model.evaluate(X_train, y_train, verbose=0)
print(f"Training Set Loss (Cost Function): {train_loss:.4f}")
print(f"Training Set Accuracy: {train_accuracy:.4f}")

# Similarly, evaluate on the cross-validation set
cv_loss, cv_accuracy = model.evaluate(X_cv, y_cv, verbose=0)
print(f"Cross-Validation Set Loss (Cost Function): {cv_loss:.4f}")
print(f"Cross-Validation Set Accuracy: {cv_accuracy:.4f}")
```

Figure 41 EVALUATION OF SOFTMAX REGRESSION MODEL

```
filtered_values = df[(df['Target Speed'] == 30) & (df['Time'] == 2 + 0.075)]['Slip angle difference']
slip_angle_values = filtered_values.tolist()
slip_angle_values
```

```
[]
```

```
import pandas as pd
import numpy as np

def malawade(df):
    for target_speed in range(30,151):
        for time in np.arange(0, 100, 0.025):  # Use np.arange for floating-point increments
            # Filter for the specific target speed and time
            filtered_values = df[(df['Target Speed'] == target_speed) & (np.isclose(df['Time'], 2 + time, atol=1e-5))]['Slip angle difference']
            # Convert filtered values to a list (it will be a single value or empty)
            slip_angle_values = filtered_values.tolist()

            # Only update if there are values found
            if slip_angle_values:
                df.loc[(np.isclose(df['Time'], time, atol=1e-5)) & (df['Target Speed'] == target_speed), 'FUTA'] = slip_angle_values[0]

    return df  # Return the updated DataFrame

df = df.dropna(subset=['FUTA'])

# Save the updated DataFrame to an Excel file
output_file_path = 'updated_data.xlsx'  # Specify the desired output file name
df.to_excel(output_file_path, index=False)

print("Updated DataFrame saved successfully as", output_file_path)
```

Figure 42 CODE TO GET SLIP ANGLE DIFFERENCE AFTER TWO SECONDS FEATURE

```
from sklearn.model_selection import GridSearchCV
from sklearn.neural_network import MLPRegressor

param_grid = {
    'hidden_layer_sizes': [(50, 50),(50,50,40)],
    'activation': ['relu','sigmoid','tanh'],
    'solver': ['adam'],
    'alpha': [0.0001,],  # Regularization parameter
    'learning_rate': ['adaptive'],
    'learning_rate_init': [0.001],  # Initial learning rate
}
```

Figure 43 PARAMETER GRID FOR FUTURE STABIITY DETECTION

```
grid_search = GridSearchCV(
    MLPRegressor(max_iter=500, random_state=42, n_iter_no_change=10, validation_fraction=0.2,early_stopping=True),
    param_grid,
    n_jobs=-1,  # Utilize all available processors
    cv=5,  # 5-fold cross-validation
    scoring={'r2': 'r2', 'neg_mean_squared_error': 'neg_mean_squared_error'},
    refit='r2',  # Use R2 as the metric to refit the best model
    verbose=4  # Higher verbosity for detailed output
)

# Ensure X_train1 and y_train1 are defined
# X_train1, y_train1 = ...

grid_search.fit(X_train1, y_train1)

print("Best parameters found: ", grid_search.best_params_)
```

✓ 2m 14.4s

Figure 44 GRID SEARCH FOR FUTURE STABILITY DETECTION

```
future.score(X_test1,y_test1)
✓ 0.0s

0.8322236514466845
```

Figure 45 FUTURE STABILITY MODEL SCORE

```
train_score = grid_search.score(X_train1, y_train1)


val_score = grid_search.best_score_

print(f"Training score (R²): {train_score}")
print(f"Cross-validated validation score (R²): {val_score}")
✓ 0.0s

Training score (R²): 0.8480926227973583
Cross-validated validation score (R²): 0.8186444636404193
```

Figure 46 CODE TO GET TRAINING SCORE AND CROSS VALIDATION SCORE

```
import matplotlib.pyplot as plt
from sklearn.model_selection import learning_curve

# Generate learning curve data
train_sizes, train_scores, val_scores = learning_curve(
    MLPRegressor(hidden_layer_sizes=grid_search.best_params_['hidden_layer_sizes'],
                 activation=grid_search.best_params_['activation'],
                 solver=grid_search.best_params_['solver'],
                 alpha=grid_search.best_params_['alpha'],
                 learning_rate=grid_search.best_params_['learning_rate'],
                 max_iter=500, random_state=42, early_stopping=True, validation_fraction=0.2),
    X_train1, y_train1, cv=5, scoring='r2', n_jobs=-1
)

# Compute mean and standard deviation of training/validation scores
train_scores_mean = train_scores.mean(axis=1)
val_scores_mean = val_scores.mean(axis=1)

# Plot learning curves
plt.plot(train_sizes, train_scores_mean, label="Training score")
plt.plot(train_sizes, val_scores_mean, label="Validation score")

plt.xlabel("Training set size")
plt.ylabel("R² Score")
plt.title("Learning Curves")
plt.legend(loc="best")
plt.grid()
plt.show()
```

Figure 47 CODE FOR LEARNING CURVE SHOWING R² SCORE FOR TRAINING AND VALIDATION SETS AS A FUNCTION OF TRAINING SET SIZE FOR FUTURE DETECTION

```
from sklearn.metrics import r2_score, mean_squared_error
import numpy as np

# Evaluate model on training data
y_train_pred = grid_search.best_estimator_.predict(X_train1)
train_rmse = np.sqrt(mean_squared_error(y_train1, y_train_pred))
train_r2 = r2_score(y_train1, y_train_pred)

# Evaluate model on test data
y_test_pred = grid_search.best_estimator_.predict(X_test1)
test_rmse = np.sqrt(mean_squared_error(y_test1, y_test_pred))
test_r2 = r2_score(y_test1, y_test_pred)

print(f"Training RMSE: {train_rmse}")
print(f"Training R^2: {train_r2}")

print(f"Test RMSE: {test_rmse}")
print(f"Test R^2: {test_r2}")
```

Figure 48 CODE FOR CODE FOR LEARNING CURVE FOR TRAINING AND TEST RMSE AS A FUNCTION OF TRAINING SET SIZE FOR FUTURE STABILITY DETECTION

```
import numpy as np
unique_states = np.unique(predicted_classes)
num_status = len(unique_states)
def trans_metrix(predicted_data):
    t_matrix = np.zeros((num_status,num_status))
    for i in range(len(predicted_data)-1):
        current_state = predicted_data[i]-1
        next_state = predicted_data[i+1]-1
        t_matrix[current_state][next_state] += 1

        row_sums = t_matrix.sum(axis=1, keepdims=True)
        t_matrix = np.divide(t_matrix, row_sums, where=row_sums != 0)

    return t_matrix

def future_Prediction(trans_matrix,predicted_data,steps):
    current_state = np.where(unique_states == predicted_data[-1])[0][0]
    predictions = []
    for i in range(steps):
        next_state = np.random.choice(
            unique_states, p=trans_matrix[current_state, :]
        )
        predictions.append(next_state)
        current_state = np.where(unique_states == next_state)[0][0]

    return predictions
```

Figure 49 TRANSITION MATRIX AND FUTURE IMPLEMENTAION FUNCTION FOR MARKOV CHAIN

```
batch_size = 40
steps = 40

result_df = pd.DataFrame()

i=0
while(i<len(predicted_cat)):
    if((i+batch_size)<len(predicted_cat)):
        t_mat = trans_metrix(predicted_cat[i:i+batch_size])
        f_predict = future_Prediction(t_mat,predicted_cat[i:i+batch_size],steps)
        batch_df = pd.DataFrame({
            'Prediction': f_predict
        })

        result_df = pd.concat([result_df, batch_df], ignore_index=True)
        i+=batch_size

    else:
        t_mat = trans_metrix(predicted_cat[i:len(predicted_cat)])
        f_predict = future_Prediction(t_mat,predicted_cat[i:len(predicted_cat)],len(predicted_cat)-i)
        batch_df = pd.DataFrame({
            'Prediction': f_predict
        })
        i = len(predicted_cat)
        result_df = pd.concat([result_df, batch_df], ignore_index=True)

result_df.to_excel("Future_Prediction.xlsx",index=False)
```

Figure 50 MARKOV CHAIN IMPLEMENTATION

```
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler(feature_range=(-1,1))
scaler1=MinMaxScaler()
check_df[['Left wheel steer','Right wheel steer']] = scaler.fit_transform(check_df[['Left wheel steer','Right wheel steer']])
check_df[['Driver steering input','Yaw rate']]=scaler.fit_transform(check_df[['Driver steering input','Yaw rate']])
check_df[['Average Rotational speed left','Average Rotational speed right']]=scaler1.fit_transform(check_df[['Average Rotational speed left','Average Rotational speed right']])
```

```
X_current = check_df.drop(columns=['Target Speed','Time','Slip angle difference'])  # Features
```

Generate + Code + Markdown

```
X_current.head()
```

	Left wheel steer	Right wheel steer	Driver steering input	Lateral Acceleration	Yaw rate	Reaction force left front	Reaction force left rear	Reaction force right front	Reaction Force right rear	Average Rotational speed left
0	-0.202443	0.202573	-0.000833	-0.023454	0.044088	0.291053	0.286983	0.246543	0.226231	0.000000
1	-0.233618	0.171006	-0.041667	-0.159788	0.041163	0.312730	0.276631	0.304306	0.294476	0.000471
2	-0.270187	0.131892	-0.083333	-0.398263	0.024935	0.336927	0.311976	0.293492	0.282457	0.000942
3	-0.304923	0.093824	-0.125000	-0.514346	-0.001923	0.374558	0.356937	0.294327	0.261947	0.001413
4	-0.339527	0.054102	-0.166667	-0.547463	-0.029171	0.412596	0.407863	0.223673	0.252807	0.001882

Now let us get the current stability using over trained model

```
check['Predicted Current Slip angle difference'] = current.predict(X_current)
check.head()
```

Figure 51 CURRENT SLIP ANGLE PREDICTION FOR MODEL VALIDATION

```
check['Actual Current Stability'] = check['Slip angle difference'].apply(
    lambda x: 'oversteer' if x > 4.5
    else 'understeer' if x < -4.5
    else 'stable')

check['Predicted Current Stability'] = check['Predicted Current Slip angle difference'].apply(
    lambda x: 'oversteer' if x > 4.5
    else 'understeer' if x < -4.5
    else 'stable')

check['Actual Future Stability'] = check['Slip angle difference after 2 seconds'].apply(
    lambda x: 'oversteer' if x > 4.5
    else 'understeer' if x < -4.5
    else 'stable')


check['Predicted Future Stability'] = check['Predicted future Slip angle difference'].apply(
    lambda x: 'oversteer' if x > 4.5
    else 'understeer' if x < -4.5
    else 'stable')
```

Figure 52 CONVERSION FROM SLIP ANGLE DIFFERENCE TO STABILITY STATES

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix

# Assuming your dataframe `check` is already populated with the columns

# Extract the actual and predicted stability values
actual = check['Actual Current Stability']
predicted = check['Predicted Current Stability']

# Generate the confusion matrix
conf_matrix = confusion_matrix(actual, predicted)

# Convert the confusion matrix into a DataFrame for better visualization
conf_matrix_df = pd.DataFrame(conf_matrix,
                              index=['Actual: understeer', 'Actual: stable', 'Actual: oversteer'],
                              columns=['Predicted: understeer', 'Predicted: stable', 'Predicted: oversteer'])

# Visualize the confusion matrix using a heatmap
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix_df, annot=True, fmt='d', cmap='Blues')

# Add labels and a title
plt.title('Confusion Matrix of Actual vs Predicted Current Stability')
plt.ylabel('Actual Values')
plt.xlabel('Predicted Values')

# Show the plot
plt.show()
```

Figure 53 CODE TO GET CONFUSION MATRIX FOR ACTUAL VS PREDICTED STABILITY STATES

```python
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix

# Assuming your dataframe `check` is already populated with the columns

# Extract the actual and predicted future stability values
actual_future = check['Actual Future Stability']
predicted_future = check['Predicted Future Stability']  # Assuming you have a column for future predictions

# Generate the confusion matrix
conf_matrix_future = confusion_matrix(actual_future, predicted_future)

# Convert the confusion matrix into a DataFrame for better visualization
conf_matrix_future_df = pd.DataFrame(conf_matrix_future,
                                     index=['Actual: understeer', 'Actual: stable', 'Actual: oversteer'],
                                     columns=['Predicted: understeer', 'Predicted: stable', 'Predicted: oversteer'])

# Visualize the confusion matrix using a heatmap
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix_future_df, annot=True, fmt='d', cmap='Greens')

# Add labels and a title
plt.title('Confusion Matrix of Actual vs Predicted Future Stability')
plt.ylabel('Actual Future Values')
plt.xlabel('Predicted Future Values')

# Show the plot
plt.show()
```

Figure 54 CODE CONFUSION MATRIX FOR ACTUAL VS PREDICTED FUTURE STABILITY STATES

```python
pred_value = model(model_input)
f_x = tf.nn.softmax(pred_value)
predicted_cat = tf.argmax(f_x, axis=1)
```

```python
predicted_cat
currentpredict_df = pd.DataFrame({'Stability':predicted_cat})
currentpredict_df.head()
```

Figure 55 CURRENT STABILITY PREDITION USING SOFTMAX REGRESSION FOR MODEL VALIDATION