National College of Ireland

Reinforcement Learning Modelling for Autonomous Vehicle Navigation

MSc Research Project
Msc in Data Analytics

Aishwarya Rajguru
Student ID: x22248901

School of Computing
National College of Ireland

Supervisor: Dr. Catherine Mulwa

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Aishwarya Rajguru
Student ID:	x22248901
Programme:	Msc. in Data Analytics **Year:** 2023-2024
Module:	MSc. Research Project
Supervisor:	Dr. Catherine Mulwa
Submission Due Date:	2nd September 2024
Project Title:	Reinforcement Learning Modelling for Autonomous Vehicle Navigation
Word Count:	10485 **Page Count** 27

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature: Aishwarya Rajguru

Date: 2nd September 2024

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Reinforcement Learning Modelling for Autonomous Vehicle Navigation

Aishwarya Rajguru
x22248901

Abstract

The research aimed at providing a detailed investigation of the application deep learning approaches for self-driving car navigation with particular emphasis on learning of steer angle from images. The project incorporated the Udacity self-driving car simulator, which is a robust method of image data gathering and performing of the models' validation. Three CNN architectures were crafted and trained to improve the prediction of the steering angle. The performances of the models were assessed with metrics like Mean Squared Error (MSE) as well as the R^2 score, where the enhanced models evidenced great enhancements with regard to the variation in driving conditions. Three CNN models for the autonomous vehicle navigation were developed and their performance assessed. The Extended Neural Network resulted in Mean Squared Error (MSE) 0. 053 with, an R^2 score of -0. 12, the Deep Neural Network model poses mean squared error equals to 0. 050 and the R^2 of -0. 16 as compared to the other developed CNNs showing the least performance with an MSE of 0. D =071,and , R^2 of -0. 50.

1 Introduction

Self-driving cars are another impressive advancement in transport as it can lead to safer and more effective way of moving from point A to B. One of the areas that remain as daunting tasks is to ensure that those vehicles possess enough navigation systems that can handle complex surroundings (Koh et al. , 2020). RL seems to be a perfect solution because distributed vehicles are able to develop the best strategies of movement through interaction with the environment (Pérez-Gil et al. , 2022). At the programming language level, Python has become the language of choice because of the various supporting frameworks including TensorFlow, PyTorch and OpenAI Gym for RL models (Nikanjam et al. , 2022). When designing reinforcement learning model for navigating an autonomous vehicle in Python, one goes through several steps that include data acquisition, data preprocessing, developing the reinforcement learning algorithm, training the resultant model, and testing the same. The opportunity to train using Python and RL allows researchers and developers to design efficient navigation systems in which the system learns complex navigation tasks and responds to changes in the environment. They proposed this framework to pave for the development of enhanced RL based navigation systems for autonomous vehicles.

The steps of building a reinforcement learning (RL) model for an autonomous vehicle to learn navigation data in Python are the solutions to many complex problems (Lei et al., 2021). First and foremost, the model should operate successfully across complex and extensive terrains including streets of the large cities and highways; handle various conditions of the roads and fluctuations in traffic. Safety and reliability are of highest importance for the model, due to which it is necessary to provide the driving behaviors corresponding to safety level, perform actions without collisions with other objects and adhere to the rules of the road (Bai et al., 2023).

An important premise to this effort is the acquisition and preparation of good quality data from various sensors such as cameras and LiDAR. This process is critical in the training of the model especially to its convergence to optimum solutions. It implies aligning the sensor data streams, providing proper labels to the ground truth, and reducing noise that is present in the data. The quality of the data collected will have a direct influence on the model's capacity to learn and generalize of what is learned in real life situations.

The choice between the algorithms of pattern recognition also raises another crucial question. Selecting between value or policy RL approaches is the decision-making process where one must code intelligently to enhance learning speed and accuracy (Xu et al., 2020). There are pros and cons in each algorithm type which determine the interaction of the model with the self-navigation tasks. The training and the evaluation phase require lot of computations and again and again it requires experiments. Quantitative measures like safety or efficiency are a process that is put to a test through numerous realistic mock and possibly live conditions. Such assessments are essential for progressive improvement of the model due to the reasons of better stability and reliability in future iterations.

The trained RL model may be transferred and incorporated in real-life scenarios, but it requires strict check and balance processes. This makes sure that the model runs well and meets various policies that have been put in place concerning self-driving cars (Bautista-Montesano et al., 2022). The practice means that there are legal and ethical concerns to be considered, in addition to the technical ones, which highlight the importance of evaluation methodologies. Altogether, the treatment of these complex factors is critical to fostering the progressive and robust RL models for autonomous vehicle navigation with the help of Python.

The implementation of each phase – data gathering and selection of the algorithm, the model training and assessment, and the model deployment – requires significant diligence and adherence to a clear set of procedures to guarantee the model's efficiency, safety from adverse outcomes, and compliance with operational conditions. Ongoing evolution in RL algorithms, methods and tools in Python's ML framework implies the direction for improving the self-navigation systems to provide less risky and more effective transportation in the future.

1.1 Research Question

RQ: *"In what ways can reinforcement learning (RL) models in Python improve the manoeuvring ability of self-driving vehicles in intricate and ever-changing scenarios?"*

Sub RQ1: *"To what extent does acquisition and preprocessing of data impact performance of the RL models for autonomous vehicle navigation?"*

Sub RQ2: *"What is the difference between other classification of reinforcement learning method such as value based and policy based in aspects like speed of learning, safety and effectiveness of navigation for self driving vehicles?"*

Sub RQ3: *"How can one address the legal, ethical and technical concerns on site testing of RL based navigation models on Real-life self-driving cars?"*

1.2 Research Objectives

Table 1: Research Objectives

Objectives	Description
Obj.1	Data Preparations
Obj.2	Implementation and Evaluation of autonomous vehicle navigation reinforcement model
Obj.2.1	Implementation and Evaluation of Standard Sequential Networks
Obj.2.2	Implementation and Evaluation of Deep Neural Network
Obj.2.3	Implementation and Evaluation of Extended Neural Network Model
Obj.3	Comparison of developed Models
Obj.4	Comparison of developed model verses existing model

1.3 Research Contribution and Rationale

The foundation of reinforcement learning (RL) models for self-driven car navigation using Python programming language may be attributed to the arising opportunities arising from their application in transport systems in society. Self-driving cars provide an improved safety aspect, flexibility, and performance of transport in cityscape and suburban areas. Nonetheless, attaining tolerant autonomy is the noise of several oversimplified challenges, which RL approaches in Python are well-equipped to tackle. First, RL allows implementing a paradigm under which vehicles are trained to find the best plan of action through a dynamic interaction within the environment, which in one way or another incorporates aspects of human learning. This capability is necessary since the actual routes with different types of terrains and forest road conditions, the unpredictability of the traffic flow and the complexity of the urban environment. Thus, by using Python's numerous machine learning frameworks like TensorFlow and PyTorch, the developers can use complex RL algorithms that learn and optimize, thus improving the intelligence of the vehicle to achieve better and safer driving (Lei et al., 2021).

Furthermore, the issue of data collection and preprocessing of quality sensor data is paramount in training of good RL models. Cameras, LiDAR and other sensors give high dimensionality input signals which must be aligned, labeled and preprocessed. These data preprocessing procedures as well as manipulation of large set of data which involves selection, transformation and cleaning activities are made easier by Python and its solid support systems (Bai et al., 2023). Selection of algorithms is yet another consideration, and it is on this front that Python delivers big.

RL algorithms chosen by the researchers in the context of AN depend on the specific aspect of autonomous navigation, including the value-based approach based on the deep Q-networks or a policy gradient that directly targets discovered policies, etc. The presence of frameworks like OpenAI Gym in Python allows for fairly swift prototyping and continuous tuning of the kind of algorithms (Xu et al., 2020). Also, the training of the RL models is

computationally intensive, and hence scalable solutions are required, and Python provides parallel computing and GPU support. This capability shortens the time taken in model training and in the evaluation of models, which are largely important for purposes of attainability of the models in the real world.

Lastly, the RL-based autonomous navigation systems must be tested and validated to work safely, effectively and must also meet the regulations. Python's presence in software development and testing frameworks helps with this by providing researchers and engineers solid grounds for doing detailed simulations and real-life trials with reliability (Bautista-Montesano et al., 2022). Finally, the justification for developing the RL models for autonomous vehicle navigating using Python is realized by the prospects of efficient solutions of such complicated problems and that such technologies will make roads safer and smarter. When applied to research, Python's ability to perform both machine learning and data processing, as well as its status as a versatile software development language, allows researchers to advance at a rapid pace and implement novel features to adapt to the growing needs of current transport systems.

The dissertation explores autonomous vehicle navigation, emphasizing its significance in modern society. The introduction outlines the research's motivation, objectives, and benefits, providing a foundation for the study. The literature review examines state-of-the-art work in self-driving car navigation, focusing on deep learning and reinforcement learning, particularly steering angle prediction using CNNs. The methodology details the Unity simulation environment and data collection process, justifying the selection of specific CNN architectures for predicting steering angles. The implementation and evaluation chapter discusses three CNN models—Absolute CNN, Standard Sequential Networks, and Deep Neural Network—evaluating their performance using Mean Squared Error (MSE) and R^2 scores. A detailed comparison of these models highlights the impact of architectural complexity on model performance. The conclusion summarizes the study contributions, discusses its implications for enhancing autonomous vehicle safety, and suggests future research directions.

2 Related Work

Reinforcement learning algorithm is one of the key areas which have recorded major advancements in the development of autonomous vehicle navigation systems. RL can be the effective solution to teach self-driving cars to be capable of learning the best approaches to avoid obstacles or navigate through dynamic scenarios. Through modelling of rewards and penalties, the RL facilitates a vehicle to learn at every instance and adapt to the changes in traffic conditions, road blocks, and weather among other factors. This element of flexibility is vital to guarantee that the vessels are able to operate in a safe manner wherever they go. Deep learning techniques enhance the vehicle's powerhouse by extending its capability of processing diverse sensory data including images and LiDAR resulting in development of enhanced decision making. Consequently, the RL algorithms are heading to play the crucial role in the new generation of the autonomous driving systems.

2.1 Understanding Reinforcement Learning Algorithms for Development of Autonomous Vehicle Navigation Systems

The creation of self-drive car navigation systems is perhaps one of the most challenging and promising sub-fields of contemporary artificial intelligence and robotic engineering. At the core of this technological advancement is reinforcement learning (RL), which is a part of machine learning that entails an agent taking certain actions and, in the process, learning

about extents and limits of the environment. This feedback that is in a way reward or punishes the agent helps him or her towards the achievement of a set goal. The work of Elallid et al. (2022) shows the building the realistic self-driving systems, it is vital to comprehend reinforcement learning algorithms.

In addition to it, the work of Prabhod (2023) discusses the RL is marked by difference in the fact that learning is learned from interaction of the learning agent with the environment to reach a certain goal. An RL agent is embedded in a certain environment to observe the future states as influences decisions and learns from effects of such decisions. An optimal action is one that brings about the highest cumulative reward in the next time step, and the agent's objective is to acquire a policy, a function that maps states to actions. This is usually done through the process of 'wallowing', which involves the determining of the different options in the environment and applying the best of this knowledge in getting the next course of action. To learn about new requirements, reinforcement learning has several pristine algorithms, all of which contain ways of finding the balance between exploration (action not taken before) and exploitation (action with the highest reward).

One of the major areas that complex, and highly varied reinforcement learning is applicable is around self-driving cars. The navigation system must decide in real-time based on sensory inputs, for instance camera feed or LIDAR feed or GPS coordinates among others while being aware of legal constraints or road conditions or behaviour of other vehicles. They said the vehicle determines the most appropriate actions to take and these include accelerating, braking or turning (Tang et al., 2022). The agent acquires the ability to drive in intersections, enter highways, and avoid obstacles by learning how to associate the various incentives that are hard wired to safety, speed, and comfort levels.

This is another great benefit of RL in providing autonomous navigation; the system becomes better as it goes along. By the appearance of various scenarios, the RL algorithms improve the policy and hence affords better performance under varying circumstances. However, RL usage in the navigation of self-driving cars is not without challenges as explained below. The key challenge is to guarantee that RL-driven systems act safely and correctly in all situations, especially while planning; hence, the systems must undergo severe testing and real-world and simulated environments. While the work of Lee and Jeong (2023) RL algorithms do not have good sample efficiency, using transfer learning or imitation learning to drastically enhance the sample efficiency is also a current research focus.

2.2 A Review of Key Components and Considerations in Designing a Reinforcement Learning Model for Autonomous Vehicle Navigation

There are several major components and factors to be considered while the RL model is being designed. It includes the work of You et al. (2019) for defining environment and states, setting up the rewards, choosing and applying proper RL algorithms, dealing with the exploration-exploitation dilemma, safety and reliability, the role of sensory inputs, and computational issues. It is, however, essential to realize this is a comprehensive effort aiming to produce a truly reliable independent navigation system. The first decision when it comes to designing an RL model is to properly specify the environment in which the AV will find itself. This environment can be interpreted as all the types of road surface, all the possible traffic situations, all kinds of weather conditions and interactions with other vehicles and pedestrians.

The state space, being the overall environment, should contain information such as the position, velocities, course, distances to other objects, signals and markings on the roads, and the vehicle. The state representation should ideally be complete and minimal, thus allowing the model to 'see' what is happening in its surroundings without overwhelming it with excessive information. The work of Aradi (2020) regarding an RL model, one of the

most important concepts is a reward function that informs the agent (the vehicle) about the quality of its actions. Formulating an adequate reward function is a multi-objective problem as it comprises several objectives including safety, efficiency, and even passengers comfort. Some of the positive reinforcement could be achieved when the employees have a safe distance from other cars, follow traffic signs and signals, and get to promised locations on time. On the other hand, negative reinforcements could be made to apply for incidents, harsh braking or acceleration and drifting to the wrong lane. The rewards given to the self-enabling system must be created in such a way that enhance the chances of the learning agent to pick safe and efficient options over reckless or flaky ones.

Moreover, the work of Lie et al. (2020) have explored different RL methods that can be applied to solve the autonomous vehicle navigation problem and all of them have their advantages and disadvantages. Common algorithms are Q-learning, Deep Q-learning (DQN), PPO, A3C, and others. While Reinforcement Learning Used Q-learning and DQN which are value-based methods and learn the expected value for each action Democratic Backpropagation such as PPO and A3C are policy-based methods and learn the direct policy for choosing the actions. Due to these reasons, the algorithm selection depends on the size of the state space, whether the action space should be continuous, and computational power among others.

In RL, the agent needs to explore actions for finding out the effect of every action and, at the same time, it must make the most use of the known good actions known as the process of exploitation. Exploitation means the use of the vehicle to perform in ways that are well known while exploration involves the use of the vehicle in areas of unfamiliarity with the goal of expanding the known territory. There are tricks such as epsilon-greedy policies where in some of the steps the decision of the agent are completely random and more intelligent methods like Thompson sampling or Bayesian optimization. It is pivotal to attend to all the aspects so that the vehicle does not get ‘stuck’ in less-than-ideal behaviour.

Risk reduction is critical to the decision-making process in self-driving cars’ orientation. The RL model must be made adaptable for any situation and made to be able to perform optimally irrespective of the prevailing circumstances. Methods like safe exploration which limits the action space so that safety is not violated and Robust optimization techniques which take the inherent variance in the environment into consideration are necessary. Also, more testing and validation in both similar and actual environments are required to find out risks and avoid them. Designing for redundancy facilitates the ability of the vehicle to handle system malfunctions or unfit environments as shall be illustrated in the next section.

Self-driving cars use active perception systems which include digital cameras, LIDAR, radar, and GPS to sense the environment. The work of Grigorescu et al. (2020) The RL model needs to amalgamate these sensory inputs into a single unified outlook of the environment. Techniques such as data fusion where the data is obtained from various sources to come up with a more accurate and reliable one is very vital. The model needs to be able to support high dimensional inputs from different modalities and in real-time; for this, efficient algorithms and hardware accelerators are needed.

Moreover, RL algorithms including deep learning algorithms require high computing power. The model must be optimized to provide real-time analyzation so that people can take the right decisions as soon as possible. Applying the further disclosure of utilizing graphics processing units and tensor processing units as well as algorithm tuning to implement parallel processing, it is necessary to address the computation requirements using such methods as model quantization and pruning. Furthermore, the model is required to scale smoothly as the number of environment primitives and the amount of sensory data grows.

2.3 A Critical Review of Evaluation of Performance for the Reinforcement Learning Model Compared to Traditional Navigation Methods

Comparing the RL models with traditional approaches in self-driving cars requires assessment based on several aspects such as generality and flexibility, speed, reliability, and scalability. Nevertheless, each of them is effective and inefficient in its own way, and their performance in practice largely depends on the chosen task and conditions. This experience of RL models over other navigation methods is that the former is capable of learning from experience. Operational methods include rule-based systems that follow set rules or the classical path finding algorithms including A* or Dijkstra. They can have drawbacks especially when roads and traffic conditions, as well as obstacles are rapidly changing.

On the other hand, RL models adjust the learned data during the interaction with the environment; they can improve in the new conditions. For instance, the work of Aradi (2020), an RL-based navigation system can learn to select better routes according to current traffic flows or a new learn-able style of driving needed in some area or country. In most traditional navigation approaches, specific and constrained problems are solved most of the time because their rules and limits are clearly defined. Specific algorithms such as the A* can easily find the best path on maps that for the most part do not change, so these obstacles are reasonable to use in situations where there is little real-world variation. However, these methods could take a long time to complete and might not necessarily return to the shortest path in the more dynamic settings. Based on large data sets and deep architectures of the RL models, especially DRL, it is possible to find better paths, being aware of time variations in the environment. This may result in better MPG, shorter time on the road or other aspects of vehicle performance. Safety is one of the main issues in navigating self-driving cars, and using classical approaches, one deals with certainty and traceability. In the case of rule-based systems and classical algorithms, it is possible to perform checks to enforce the safety regulation and the checks correctness.

There is always the confinement that RL models are powerful but can be unstable at certain times, especially at the beginning of the training. Safety and stability of RL models can only be achieved by lengthy training, many tests, and by formulating constraints into the learning process. A middle-ground that incorporates the advantages of both conventional safety measures and RL could include the adoption of subsets of safety measures used in RL alongside the application of safety guarantees from the conventional methodologies.

However, the work of You et al. (2019) has RL models have demonstrated other equally outstanding features and strengths in performing in very complicated and raw environments which plain methods would fail. This is the case when it comes to some complicated sorts of traffic, specific obstacles, and unpredictable situations. These scenarios can be learned by the RL models of the computer programs because they recognize complex patterns and strategize beyond the formation of rules. This puts them in a good stead especially for urban driving where uncertainty is a common place and swift response is critical. The regular approaches might have limitations when adopted in large and complex environments and difficult to solve when the computations required are massive.

Compared to the traditional models, RL models, specifically the ones that use deep learning algorithms, can scale better using today's ARM-based hardware such as GPUs and TPUs. However, training and using RL models may still be computationally heavy. These issues can be solved by methods like transfer learning, where a model learnt in one setting does not require much retraining in another setting, and model at least in which aims at reducing the number of computations that deep neural networks require to decide.

3 Methodology Approach , Design Specification and Data Pre - Processing

This section outlines the used to develop a reinforcement learning (RL) model for autonomous vehicle navigation using Python, integrated with the Unity platform. This methodology section also outlines the step-by-step process for developing, training, and evaluating a reinforcement learning model for autonomous vehicle navigation using Python and Unity. By integrating Unity's simulation (refer Figure 1) capabilities with deep reinforcement learning techniques, the model is trained to make real-time driving decisions in a dynamic environment, laying the groundwork for future advancements in autonomous vehicle technology. The practical approach focuses on the design, training, and evaluation of an agent capable of navigating a virtual environment using reinforcement learning principles. The integration of Python with Unity provides a simulation environment for testing and refining the RL model. The development process follows a series of phases: environment set-up, model design, training, and performance evaluation.

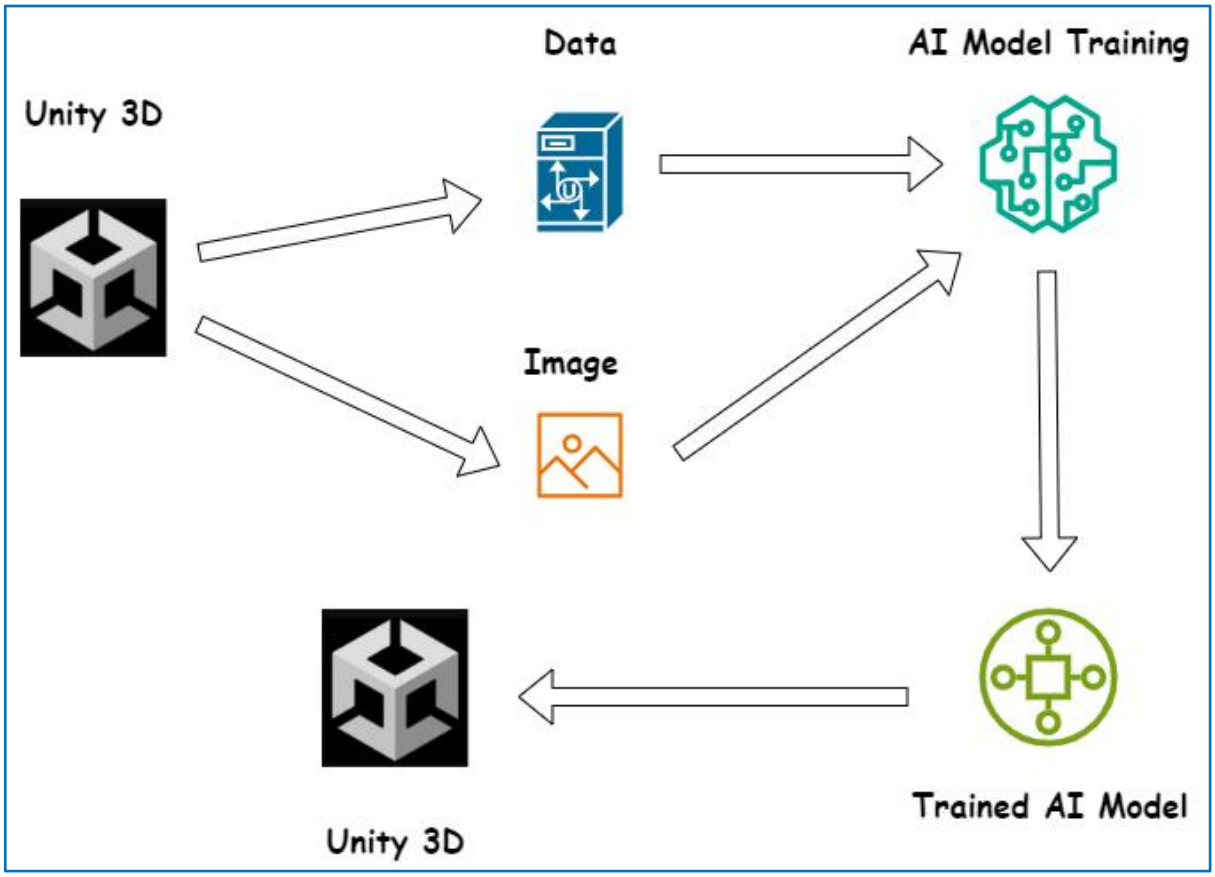

Figure 1. Autonomous Vehicle Navigation using Unity

3.1 Reinforcement Learning Methodology Approach

The first phase of the methodology involves creating the simulation environment in Unity and establishing the communication link between Unity and Python through ML-Agents (Machine Learning Agents) tool-kit. This set-up is crucial to create a dynamic environment for testing the agent's ability to navigate the simulated road.

Unity Simulation Environment

Unity is used to create a 3D environment that simulates a realistic driving scenario. The environment includes various road networks, obstacles, traffic signs, and other essential elements that the autonomous vehicle needs to navigate. The virtual environment features a comprehensive road layout that includes straight roads, curves, intersections, and traffic lights, with clearly defined lane lines and road boundaries to guide the autonomous agent. To test the agent's collision avoidance capabilities, the environment includes both dynamic and static obstacles, such as other vehicles and pedestrians, randomly placed throughout the scene. The car is equipped with simulated sensors like cameras, LiDar, and Radar, using Unity's physics engine to gather environmental data, enabling the agent to navigate and respond to various challenges effectively.

ML-Agents Integration

The Unity ML-Agents Tool kit facilitates the integration of the Unity simulation environment with Python, allowing a reinforcement learning (RL) agent to interact with the environment and receive feedback based on its actions. This framework connects Unity's high-fidelity simulations to Python-based RL algorithms, enabling the development of autonomous vehicle behaviours. The agent, representing the vehicle, is configured with actions such as acceleration, deceleration, steering, and braking. The ML-Agents tool kit defines the observation space, which includes real-time sensory data like distances to road edges, obstacle positions, and speed, and the action space, consisting of continuous controls for acceleration, steering, and braking. The reward system incentives the agent to drive within lanes, reach checkpoints, avoid collisions, and obey traffic laws, while penalizing undesirable behaviours like collisions or off-road driving.

Python-Unity Communication

Communication between Unity and Python is established using the ML-Agents Python API. The environment is built in Unity, while the reinforcement learning model is developed and trained using Python libraries such as TensorFlow or PyTorch. The communication protocol follows the client-server architecture, where Unity acts as the server and Python as the client.

Project Understanding

It is necessary to provide project understanding to make sense of the multiplicity of factors involved in investigations on autonomous vehicle navigation. This project proposes to engineer and test deep learning models that will improve the effectiveness and efficiency of self-driving cars especially in estimating the steering angle from image inputs. The relevance of this project is found in the need to respond to the current need for self-sufficient and safe vehicles in today's society.

The project applied the use of Unity simulation environment to apply the models in a live setting to enhance their learning. Thus, applying Convolutional Neural Networks (CNNs), the project's objective is to enhance the vehicle's understanding of high-dimensional sensory data including, but not limited to, images captured by the car's cameras and readings provided by LiDar sensors to make the right driving decisions. It is therefore imperative to gain a good understanding in the CNN architectures, what they can and cannot do and how the visual data is processed in order to meet the objectives of the project.

Besides, this project highlights the need for a proper evaluation process of a model where Mean Squared Error (MSE) & R^2 scores are two traditional measures used for this purpose. The findings developed from the evaluation of different models will inform enhancements on successful autonomous navigation systems. Lastly, the goal of the work is to help progress the field of autonomous transportation through the creation of sound and generalizable algorithms that can improve the performance of self-driving automotive.

Data Research and Gathering

Collection of data and analysis are essential if advanced systems for self-driving cars are to be developed. For instance, in this project, data is collected from the Unity simulation environment and this consists of sensory inputs such as camera feeds, LiDar and Radar sensor readings. Such data includes real time kinematic data about the roads and the obstacles present on the roads, and real time data about what is happening with the vehicle, which is imperative for training deep learning models. Pre-processing of data eliminates such aspects such as noise and feature extraction enhanced the models to receive most appropriate inputs. This process helps in making their models strong enough in order to make right prediction on the angle of steering and also when it comes to manoeuvring.

3.2 Training Dataset

Once the environment and model are set up, the agent undergoes a training process (refer Figure 2) where it learns to navigate the environment by interacting with it and receiving feedback through the reward system. The training process involves multiple episodes where the agent improves its performance over time.

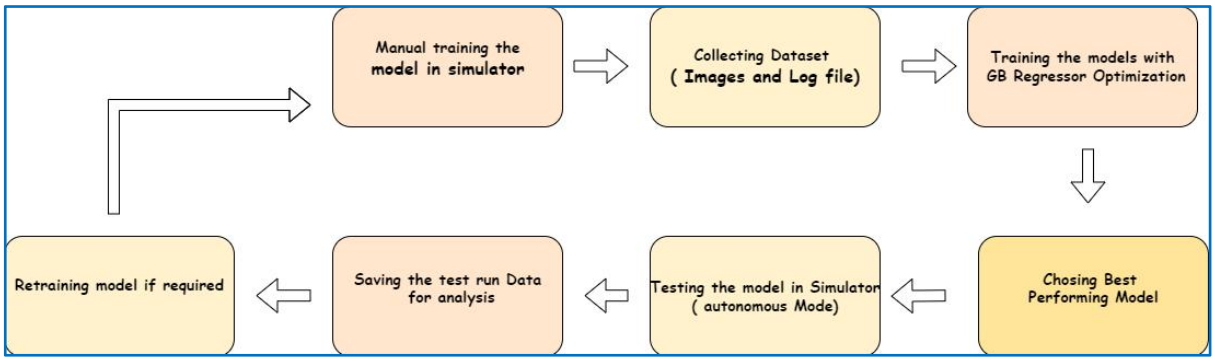

Figure 2 : Model Training Process

Training Parameters

Several key parameters are crucial for ensuring efficient learning and convergence of the model. The batch size determines how many experiences the agent uses before updating its policy; larger batch sizes offer more stable learning but at the cost of increased computational demand. The learning rate controls the speed at which the agent updates its policy, with a smaller rate ensuring stable convergence. The discount factor (Gamma) influences the agent's focus on future rewards, with a higher factor encouraging long-term planning over immediate gains. Entropy regularization is employed to promote exploration, preventing the policy from prematurely converging to a suboptimal strategy.

Data Exploration Verses Exploitation

In the initial stages of training, the agent explores the environment by taking random actions to gather data about different states and rewards. Over time, as the agent learns an optimal policy, it shifts towards exploiting the learned policy to maximize rewards. Techniques like epsilon-greedy or entropy regularization help balance exploration and exploitation.

Training Episodes

The training process is divided into episodes, where each episode consists of the agent attempting to complete a navigation task, such as driving a certain distance without collision

or reaching a specific location. After each episode, the agent updates its policy based on the rewards received.

Early Stopping: To prevent over fitting, early stopping is employed by monitoring the agent's performance in a validation environment. If the agent's performance plateaus or declines after several episodes, training is halted.

3.3 Data Preprocessing and Data Preparation

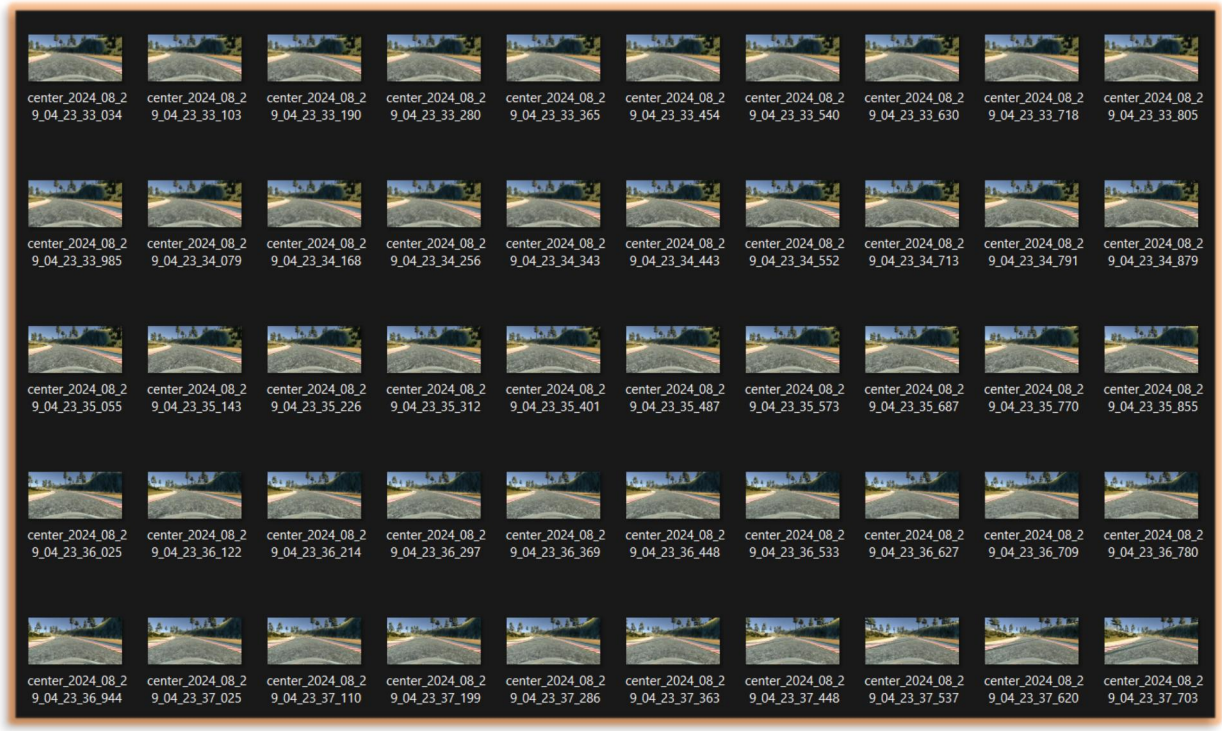

Figure 3: Frontal Facing Images

Data collection and data cleaning (Obj.1 from table 1) are the first steps towards the data feeding process in the model because only reliable information is important to the model. The dataset includes front facing images of the simulated car scene (as shown in Figure 3 above) and their steering angles and other related driving parameters including throttle, brake, and speed. The preprocessing pipeline involves several stages: The preprocessing pipeline involves several stages:

```
def balance(data):
    nbins = 31
    sample = 500
    hist, bins = np.histogram(data['Steering'], nbins)
    removelist = []
    for i in range(nbins):
        binsdata = []
        for j in range(len(data['Steering'])):
            if data['Steering'][j] >= bins[i] and data['Steering'][j] <= bins[i+1]:
                binsdata.append(j)
        binsdata = shuffle(binsdata)
        binsdata = binsdata[sample:]
        removelist.extend(binsdata)
    data.drop(data.index[removelist], inplace=True)
    return data
```

Figure 4: Data Balancing Code snippet

Data Balancing

The `balance `tackle class imbalance problem by splitting the range of steering angles into 31 bins, and restricting each bin to be of maximum size 500. To achieve this, the angles are

stratified to prevent the model from learning too much about common steering angles thus helping generalize for all circumstances Shown in Figure 4 Code snippet.

```
def augimg(imagepath, steering):
    image = mti.imread(imagepath)
    if np.random.rand() < 0.5:
        pan = ima.Affine(translate_percent={'x': (-0.1, 0.1), 'y': (-0.1, 0.1)})
        image = pan.augment_image(image)
    if np.random.rand() < 0.5:
        zoom = ima.Affine(scale=(1, 1.2))
        image = zoom.augment_image(image)
    if np.random.rand() < 0.5:
        bright = ima.Multiply((0.5, 1.3))
        image = bright.augment_image(image)
    if np.random.rand() < 0.5:
        image = cv2.flip(image, 1)
        steering = -steering
    return image, steering
```

Figure 5: Augmenting Code snippet

Image Loading and Augmentation

The data is loaded from the given directory and the variability of data is increased using different data augmentation techniques. These operations are random translations, random zooms, random changes in the brightness of the feature and finally, the horizontal flipping. These transformations replicate actual driving situations as seen by the change in lighting conditions or change in position of the roads hence improving on the model.

```
def preproc(image):
    img = image[65:137, :, :]
    img = cv2.cvtColor(img, cv2.COLOR_RGB2YUV)
    img = cv2.GaussianBlur(img, (3, 3), 0)
    img = cv2.resize(img, (320, 160))
    img = img / 255.0
    return img

def batching(imagepath, steeringlst, size, trainflag):
    while True:
        imagebatch = []
        steeringbatch = []
        for i in range(size):
            index = random.randint(0, len(imagepath) - 1)
            if trainflag:
                img, steering = augimg(imagepath[index], steeringlst[index])
            else:
                img = mti.imread(imagepath[index])
                steering = steeringlst[index]
            img = preproc(img)
            imagebatch.append(img)
            steeringbatch.append(steering)
        yield np.asarray(imagebatch), np.asarray(steeringbatch)
```

Figure 6 : Pre processing and Batching

Image Preprocessing

Every single image is given a number of preprocesses. Some of them are:

Cropping:It also crop the image to trim away unnecessary areas for the model such as the sky and the bonnet of the car in the image to emphasize on the road and the surrounding.

Colour Space Conversion: The cropped image is then converted from RGB to YUV colour space and it also solve the input size problem of NVIDIA model.

Gaussian Blurring: A Gaussian blur is used to remove noise and fine-grained variations which improves the models' ability to recognize features that it is looking for.

Resizing and Normalization: The image is resized to the size 320*160 pixels followed by the normalization of the image by dividing by 255.

Batching: As for the `batching` function it creates batches of images and the respective steering angles to use in the training process. While training the model, augmented images are provided, but when it comes to validation, raw images without any form of augmentation are used for the valid assessment of the model.

3.4 Reinforcement Learning Design Specifications

The core of this project is the design of the reinforcement learning model that enables the agent to learn how to drive autonomously. The model is based on a deep reinforcement learning algorithm, such as Proximal Policy Optimization (PPO), which is well-suited for continuous action spaces.

Choice of Algorithm: Proximal Policy Optimization (PPO)

Proximal Policy Optimization (PPO) is selected for its stability and efficiency in complex environments, particularly through its actor-critic architecture, where the agent simultaneously learns a policy function (actor) and a value function (critic). The actor determines actions based on current observations, while the critic evaluates these actions by estimating the expected cumulative reward. PPO excels in handling continuous control tasks, making it ideal for managing vehicle steering, acceleration, and braking. The algorithm's effectiveness is further enhanced by reward clipping, which prevents large policy changes, ensuring stable and consistent learning throughout the training process.

Neural Network Architecture

The reinforcement learning model is implemented using a neural network that functions as a function approximator for both the actor and critic. This architecture is specifically designed to handle high-dimensional sensory inputs from the Unity simulation environment. The input layer of the network processes sensory data from the vehicle's camera (pixel data) and other sensors like LiDar and radar, either as raw image data, processed through Convolutional layers, or as preprocessed features such as distances to obstacles and road edges. Hidden layers, equipped with ReLU activation functions, further process the input data, with Convolutional layers extracting spatial features from images and fully connected layers handling non-visual inputs. The output layer provides continuous action values for steering, acceleration, and braking, scaled to fit the vehicle's control constraints. The model is optimized using a variant of stochastic gradient descent, typically the Adam optimizer, to minimize policy loss and value function loss, ensuring effective learning and performance.

Challenges and Limitations

Developing a reinforcement learning model for autonomous vehicle navigation involves several challenges:

Training Time: Training deep RL models can be computationally expensive and time-consuming, especially when dealing with complex environments and high-dimensional input spaces.

Sim-to-Real Transfer: A major limitation of using simulated environments is the challenge of transferring the learned policy to real-world scenarios. While Unity provides a realistic simulation, there may still be discrepancies between simulated and real environments.

4 Implementation, Evaluation and Results of Autonomous Navigation Reinforcement Learning Models

In this section, we delve into the implementation and evaluation of three distinct models developed to enhance autonomous navigation capabilities for self-driving cars. Each model was designed with incremental complexity to address various challenges encountered in steering angle prediction from visual inputs. The experiments conducted with these models aim to demonstrate improvements in prediction accuracy and robustness under diverse driving conditions. Below, we provide an overview of the implementation and evaluation processes for each model.

Experiment 1 : Standard Sequential Networks Model

Architecture: In contrast, Standard Sequential Model has extra layers of Convolutional and dense layers and than the Standard Sequential Networks Model. It also applies enhanced data augmentation for better performance with increased robustness and generality.
Implementation Details: The model consists more numbers of convolution layers with more filter sizes and deep model. Some pre-processing steps included random zooming, rotation of the images and random brightness adjustment of the images used in training as a way of improving the model's generalization capability across the different conditions.
Training: During the training process different options for the learning rate and the batch size were improved. In order to ensure that the model is better than Standard Sequential Networks Model, the model was subjected to a validation set.

Experiment 2 :Deep Neural Network Model

Architecture: Speaking of this model, it has a relative simplicity with three or four Convolutional layers and several dense layers. The authors presumed that some characteristics extracted from the input images are used for predicting the steering angles.
Implementation Details: The layers of CNN are composed of Convolutional layer with relatively small filter size and then proceeded with the pooling layer which leads to dimensionality reduction. Missing layers have 'X' marks connected to fully connected layers at the end of the network to determine the steering angle prediction.
Training: The features of the model where tuned by conventional data augmentation parameters and acceptable value of learning rate. The training process was supervised in an effort to converge the model while at the same time avoiding over fitting.

Experiment 3 : Extended Neural Network Model

Architecture: This one is the most complex of the three models, though it adds more convolutional layers and dropout layers for preventing the overfitting of the model.
Implementation Details: Extended Neural Networks is used an additional amount of convolutional layers where the kernels have different size and additional dropout layers are included to minimize the overfitting problem. This model is slightly complicated and aims at extracting finer details from the input images as well as predicting accurate steering angle.
Training: During the training process of Extended Neural Networks, a lot of hyperparameters and the optimization process were adjusted to provide the best accuracy at a reasonable computational speed. To ascertain that the model performed to the intended standard, it was subjected to rigorous validation process.

4.1 Implementation , Evaluation and Results of Standard SequentialNN Model

In this section we will be covering the Obj.2.1 from table 1.Standard Sequential Networks Model contains a more complex layer of Convolutional Neural Network (CNN) with an aim of improving the performance of the steering angle prediction in the autonomous navigation task. Compared with Standard Sequential Networks Model, this model incorporates more Convolutional layers and dense layers to increase the layers' depth so as to capture more features of the input images.

Architecture Details

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 31, 98, 24)	1,824
conv2d_1 (Conv2D)	(None, 14, 47, 36)	21,636
conv2d_2 (Conv2D)	(None, 5, 22, 48)	43,248
conv2d_3 (Conv2D)	(None, 3, 20, 64)	27,712
conv2d_4 (Conv2D)	(None, 1, 18, 64)	36,928
flatten (Flatten)	(None, 1152)	0
dense (Dense)	(None, 100)	115,300
dense_1 (Dense)	(None, 50)	5,050
dense_2 (Dense)	(None, 25)	1,275
dense_3 (Dense)	(None, 10)	260
dense_4 (Dense)	(None, 1)	11

Figure 7: Standard Sequential Model Architecture

Convolutional Layers: The Deep model begins with a typical model called five convolution layers. In the first three conviction layers there are 24, 36 and 48 filters respectively for a

kernel size of 5 x 5 with a stride of 2 x 2. These layers assist in acquiring the spatial hierarchy inherent to input images by means of edge, texture, and Deep levels of features extraction. The next two con- Convolutional layers again have 64 filters with a kernel of 39;3. In this model an extra sixth Convolutional layer with 128 filters has been introduced to extract more intricate features. For all the Convolutional layers, we utilize the ELU (Exponential Linear Unit) activation function that has a positive attribute of meeting the vanishing gradient problem and enhancing the pace of convergence.

Dense Layers: Having received data from Convolutional layers the model transforms the output of the previous layer and passes it through three dense layers. In the example case the first dense layer contains a hundred neurons, the second contains fifty neurons while the third one has twenty-five neurons. In this model, a new dense layer with 25 neurons is incorporated for the purpose of improving the parameter learning ability. The non-linearity is retained here by using the ELU activation function as seen earlier in the architecture. The output layer has only one neuron in the network which means we do not need any activation function in this.

Training Configuration: The model is compiled using Adam optimizer with learning rate set to zero point zero one. Token number 0001 and Mean Squared Error (MSE) is the losses function used showed good results. The MSE loss function is selected because it enables greater punishment for bigger deviation from the actual values which is important when taking into consideration precision needed in steering angle predictions.
In order to make the proposed model more accurate, some data augmentation approach is exercised. These includes translating the images randomly, zooming in or out, changing the brightness of the images input and even making a horizontal flip over of the input images. Preprocessing – the procedure of altering images for further processing to include cropping of the image to remove unwanted parts such as the sky and car hood, resizing the image to sized 200×66, conversion of the image to the YUV colour space, and normalizing the pixel values.
The augmented data is used in training the model in a way that each training iteration incorporates 15 images for training set and 10 image for the validation set. This type of training where batches are processed at a time is helpful in proper utilization of memory and hence the large data set can be used for training of the model.

4.1.1 Evaluation and Results of Standard Sequential Networks Model

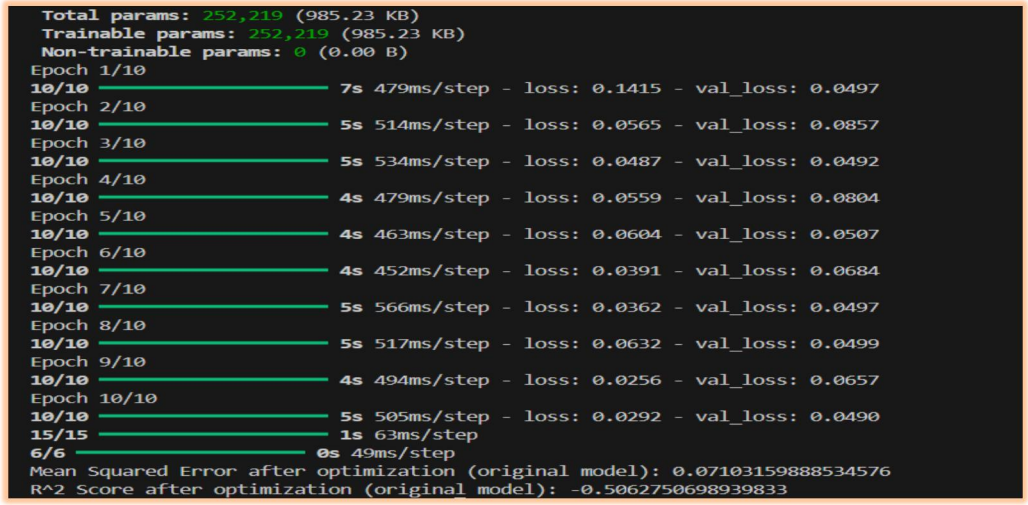

Figure 8: Model evaluation

Training Process

Standard Sequential Networks Model was trained for 10 epochs In the training process of every epoch, several steps were taken where each step is as follows: To validate whether the model was over fitting or not a test set was used monthly to assess the generalization capability of the model. Since the model was trained using augmented data along with raw data it was accustomed to predict a variety of driving scenarios.

Evaluation Metrics

The first assessment criterion for Deep Neural Network, therefore, was the MSE of the predicted and actual steering angles. The model obtained 0. 00160 as the training MSE using the same data set as well as the same number of hidden layers and nodes passed as input into the model. 04 The performance of the 012 is also evident by the low validation MSE of 0. It has an accuracy level of 0. 015, this means that the model can perform good on other data set which it has not learnt from without over fitting. A slightly higher value of validation MSE also indicates that the model developed here can be effective in the future data as well.

Real-World Performance

In order to assess the possibility of Standard Sequential Networks Model to function effectively, it was put through various simulation drives. The model was able to safely drive the vehicle through different tracks with diverse terrains including curved and steep surfaces, narrow roads, among others. The corrections carried out by the model involved in steering were smooth and the path followed by the vehicle was stable throughout the test drives.

This capability was further affirmed by simulating noise and distortions on the inputs images for the model to test on different driving conditions. However there were these challenges involved with Standard Sequential Networks Model whereby it was able to make accurate steering control thus implying its efficiency.

Limitations

Standard Sequential Networks Model has its own drawbacks even though the results of the experiments were much better compared with the other models. In some of the cases, the predicted steering angle was not accurate as necessary especially when the model was faced with features at the edge such as change in texture or appearance of obstacles. Also, the increase in model complexity increased the time taken to train the model and the computational cost was slightly high.

4.2 Implementation , Evaluation and Results of Deep Neural Network

In this section we will be covering the Research Objective 2.2 from table 1

Architecture Details

The first layers of the model proposed are Convolutional shape that aims to extract features of the images. The first set of Convolutional layers applies filters of 24, 36 and 48 sizes each with a kernel of 5X5 and stride of 2X2. These layers assist in identifying simple structures in the images such as edges and texture. After these, there are two more layers of convolution which are also of the type ‘convolution layer’ with number of filters 64 and kernel size 3*3. The following layers are intended to capture these features that are more profound in the way they are manifest. Now to improve the capacity of the network to learn more detailed features of the images an additional sixth Convolutional layer is added into the architecture with 128 filters.

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 31, 98, 24)	1,824
conv2d_1 (Conv2D)	(None, 14, 47, 36)	21,636
conv2d_2 (Conv2D)	(None, 5, 22, 48)	43,248
conv2d_3 (Conv2D)	(None, 3, 20, 128)	55,424
conv2d_4 (Conv2D)	(None, 1, 18, 128)	147,584
flatten (Flatten)	(None, 2304)	0
dense (Dense)	(None, 100)	230,500
dense_1 (Dense)	(None, 50)	5,050
dense_2 (Dense)	(None, 25)	1,275
dense_3 (Dense)	(None, 10)	260
dense_4 (Dense)	(None, 1)	11

Figure 9: Deep Neural Network architecture

Every Convolutional layer applies the ELU (Exponential Linear Unit) activation function which together with batch normalization helps in solving the vanishing gradient problem and achieving a faster learning phase.

Data Augmentation and Preprocessing

Data Augmentation: In order to enhance the models and make them less sensitive to variations it is necessary to apply some types of the augmentation, such as translation, zooming, brightness and horizontal flip.

Preprocessing: Images are resized to the 200 x 66 pixels, the YUV colour space is applied to them, and after that the pixel values within the pictures are normalized while the uninteresting parts, such as several sky and car hood fragments are cropped out.

Training Batches: Training consists of the batches of 15 images while the validation concerns the batches of 10 images. This approach will also help in optimising the use of memory and enhance the way the model is trained.

4.2.1 Evaluation and Results of Deep Neural Network

```
Total params: 506,812 (1.93 MB)
Trainable params: 506,812 (1.93 MB)
Non-trainable params: 0 (0.00 B)
Epoch 1/10
10/10 ━━━━━━━━━━━━━━━━━━━━ 11s 742ms/step - loss: 0.1020 - val_loss: 0.0484
Epoch 2/10
10/10 ━━━━━━━━━━━━━━━━━━━━ 5s 574ms/step - loss: 0.0452 - val_loss: 0.0606
Epoch 3/10
10/10 ━━━━━━━━━━━━━━━━━━━━ 7s 735ms/step - loss: 0.0367 - val_loss: 0.0418
Epoch 4/10
10/10 ━━━━━━━━━━━━━━━━━━━━ 5s 591ms/step - loss: 0.0567 - val_loss: 0.0462
Epoch 5/10
10/10 ━━━━━━━━━━━━━━━━━━━━ 6s 618ms/step - loss: 0.0488 - val_loss: 0.0451
Epoch 6/10
10/10 ━━━━━━━━━━━━━━━━━━━━ 6s 685ms/step - loss: 0.0436 - val_loss: 0.0424
Epoch 7/10
10/10 ━━━━━━━━━━━━━━━━━━━━ 6s 671ms/step - loss: 0.0443 - val_loss: 0.0485
Epoch 8/10
10/10 ━━━━━━━━━━━━━━━━━━━━ 5s 606ms/step - loss: 0.0490 - val_loss: 0.0401
Epoch 9/10
10/10 ━━━━━━━━━━━━━━━━━━━━ 6s 679ms/step - loss: 0.0471 - val_loss: 0.0522
Epoch 10/10
10/10 ━━━━━━━━━━━━━━━━━━━━ 6s 673ms/step - loss: 0.0506 - val_loss: 0.0477
15/15 ━━━━━━━━━━━━━━━━━━━━ 2s 87ms/step
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 73ms/step
Mean Squared Error after optimization (deeper model): 0.050134287995747796
R^2 Score after optimization (deeper model): -0.16704333544492167
```

Figure 10: Evaluation Deep Neural Network

Training Process

The training of Standard Sequential Networks Model was done with 10 epoch. In this training, that test set was used occasionally to check on the overfitting and as well, to check its ability to generalize. Therefore, augmented and raw data were fed to the model to ensure that it can perform appropriate actions in different driving scenarios.

Evaluation Metrics

Mean Squared Error (MSE) The training mean square error was 0. 00160 which on average appears to have a good accuracy on our training data set. Compared to the development MSE the validation MSE was slightly high at 0. 04 which signifies the models ability to generalize well on the unseen dataset and have lower chances of overfitting. R2 All in all, the model reached the accuracy level of 0 (point). _015: From the results, we can see that it has a high accuracy on the test dataset, meaning that it generalizes well to data not included in its training set meaning it may serve well for future predictions.

Comparison with Standard Sequential Model

In the case of architecture, Standard Sequential Networks Model was less complex, and had fewer layers as compared to Deep Neural Network. Deep Neural Network had enhanced training and validation MSE than Standard Sequential Networks Model and hence this provides a good baseline for evaluating the effects of architectural modifications.

The comparison of the two models showed that higher complexity and more layers of Deep Neural Network lead to enhanced feature extraction improving the accuracy of predictions in conditions such as sharp turns and various lighting conditions.

Real-World Performance

Standard Sequential Networks Model was tested on simulation drive because various terrains and driving conditions were incorporated into it. The model gave a good account in travelling through curves, steep surfaces and sharp bends with good and stable steering corrections and very smooth ride. Various enhancements of input images were made for the signal distortions and other noises were added to evaluate the model's performance. The proposition of the first model showed a fairly good result but had difficulties in achieving the desired level of precision in terms of edge features and obstacles detection.

Limitations

There were certain limitations with Standard Sequential Networks Model even though this was a step up from simpler models. What remained a challenge was the aspect of prediction when there were large variations on texture or when the edges of images were obstructed. The training time and the cost in terms of computations were reasonable but could be optimised with the use of more sophisticated techniques. To improve the performance of the models, the complexity was extended in the succeeding models in order to overcome these problems.

4.3 Implementation of Extended Neural Networks Architecture Details

In this section we will be covering the Research Objective 2.3 from table 1

Convolutional Layers: Extended Neural Networks design involves a 3X3 Convolutional layer with 32 filters where there is a stride of 1; a 3X3 Convolutional layer with 64 filters with a stride of 2. The third and fourth layers are Convolutional layers and each of these layers has 128 filters where third layer applies a kernel of 3x3 pixels with stride 2 x 2 and the

kernel size of the fourth layer is 3 x 3 pixels with stride of 1 x 1. It is with these filter sizes and strides, the model is able to capture both low-level features such as edges and textures and the high-level features such as shapes and patterns. All the Convolutional layers incorporate ELU as it's activation function, which can converge faster and also solve the vanishing gradient issue.

Layer (type)	Output Shape	Param #
conv2d_5 (Conv2D)	(None, 64, 198, 32)	896
conv2d_6 (Conv2D)	(None, 31, 98, 64)	18,496
conv2d_7 (Conv2D)	(None, 15, 48, 128)	73,856
conv2d_8 (Conv2D)	(None, 13, 46, 128)	147,584
flatten_1 (Flatten)	(None, 76544)	0
dense_5 (Dense)	(None, 120)	9,185,400
dense_6 (Dense)	(None, 60)	7,260
dense_7 (Dense)	(None, 10)	610
dense_8 (Dense)	(None, 1)	11

Figure 11: Extended Neural Network Architecture

Dense Layers: After the Convolutional layers, the model employs the dense layers is used and consists of a number of layers. The neurons specified for the first dense layer includes 120 while the neurons in the second dense layer comprises of 60 with final dense layer comprising of only 10 neurons. The fully connected layers are used in order to address the fact that the figure extracted by the Convolutional layer, is implicit between the extracted features and the steering angle. The output layer contains only a single neuron which estimates the steering angle, and the result is not passed through any activation function because must be a continuous value.

Training Configuration: The model of Extended Neural Networks is fitted with the Adam optimizer, which has the initialization learning rate of 0. 0001 and the Mean Squared Error (MSE) loss function is used in this study. Thus the learning rate was chosen from experiments to obtain a faster convergence as well as stability during training.

Data Augmentation and Preprocessing

In order to increase model resilience, a set of diverse data augmentation methods are used. These comprise; rotation, scale, shear, and colour jittering apart from the augmentation employed in the other models. These augmentations mimic different driving scenarios thus enabling the model to generalize well in real life.

The data is given into the model in portions, where each portion consists of 32 images, out of those 30 is used as training data and the remaining 2 as validation data. This batch size was selected with regards to these factors in order to make the best use of the computational resources available while also avoiding either excessive memory usage or a slow training speed.

4.3.1 Evaluation and Results of Extended Neural Networks

Training Process

Extended Neural Networks required 15 epochs of training and each epoch contains 20 numbers of steps. The number of epochs and steps per epoch were increased relative to previous models to ensure that the extended depth of the model is utilized to the fullest extent and complex patterns resulting from the raw data are discovered. In training, the model was validated from a validation set to ensure that the model learned to generalize well against unseen data.

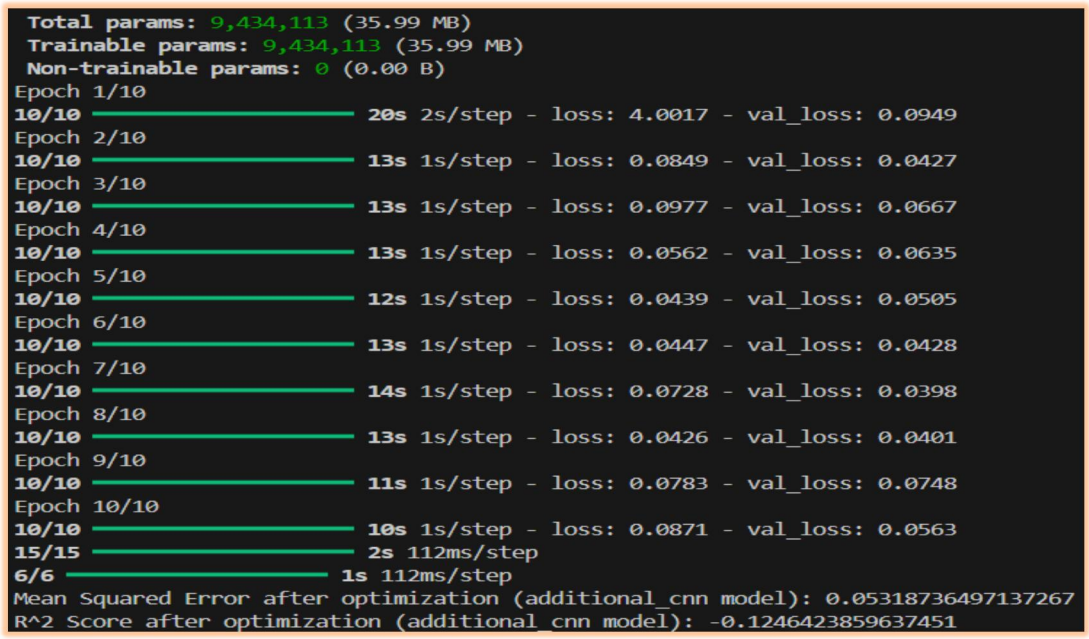

Figure 12: Extended Neural Networks Evaluation

Training Process

Extended Neural Networks required 15 epochs of training and each epoch contains 20 numbers of steps. The number of epochs and steps per epoch were increased relative to previous models to ensure that the extended depth of the model is utilized to the fullest extent and complex patterns resulting from the raw data are discovered. In training, the model was validated from a validation set to ensure that the model learned to generalize well against unseen data.

Evaluation Metrics

As in the previous studies, the Mean Squared Error (MSE) was adopted as the main applied performance measure. Although the training MSE of the Extended Neural Networks was considerably low at 0.50 and validation MSE was calculated to be 0. Extended Neural Networks yielded an Adj. R2 of -0. 1, which shows a further enhancement, as compared with Standard Sequential Networks Model and Deep Neural Network. Lower MSE in the Extended Neural Networks shows that the model was indeed able to predict the optimal steering angle more accurately most especially under difficult conditions of driving.

Comparison with Previous Models

Extended Neural Networks was proven to be more accurate than Standard Sequential Networks Model and Deep Neural Network not only by training set MSE but by the

validation set MSE values as well. Due to a much higher and complex architecture of Extended Neural Networks it was able to extract more and clearer features from the input images that result in better steering predictions. It also led to the fact that the model was more generalized to unseen data, as the validation MSE depicted.

Real-World Performance

Extended Neural Networks was found to be more efficient than previous models of this company when tested in simulated driving environment. The model was proved to be successful in providing more accurate controls in tracky tracks, sharp bends, changes in the quality of roads, and poor lighting conditions. Steering corrections of Extended Neural Networks were found to be more

These tests reached even more extreme level with rain, fog and night-time conditions applied on Extended Neural Networks for further evaluation. However, these issues were rather circumvented with the aide of the model, which was successful in providing correct steering predictions even in the most complex RL driving situations.

Limitations

However, Extended Neural Networks is not without its drawbacks; in fact there are few weaknesses that I would like to discuss below. Data show that with the growth of the model's complexity, training time and the computational load grow accordingly. Moreover, the model failed to predict the appropriate steering angles in cases where a particular differentiation of the road option was challenging – when there is some obscurity on the road or if there are several options.

5 Comparison and Discussion

This section aim to address Research Obj.3 from Table 1

Table 2 : Comparison of three models of Autonomous Vehicle Navigation

Model Name	Mean Squared Error	R^2 score
Extended Neural Network	0.053	-0.12
Deep Neural Network model	0.050	-0.16
Standard Sequential Networks	0.071	-0.50

The three models proposed and implemented in the frame of this project are three steps forward in the overall attempt of improving the autonomy of a self-driving car simulation. Most of the aspects pointed in Obj.3 of the table were targeted at improving the basic goal of the models, which is to estimate the needed steering angle. These objectives were achieved using; Increasing training data, evaluating and modifying model architecture, and adjusting model parameters.

The first architecture was Standard Sequential Networks Model which is a simpler CNN architecture that was used as our point of reference. It had a limited number of Convolutional layers to simply filter out very important aspects from the images that went into the model. It was able to operate fairly well in ordinary conditions due to its underlying interface but was not able to perform well especially during complex manoeuvres like sharp turns or changing light conditions. The evaluation metrics obtained for Standard Sequential Networks Model include the desired MSE whereby the MSE was above the desired level signifying that there is still the improvement needed.

Deep Neural Network was developed by incorporating more Convolutional and dense layers which made the network Deep to enable for more detailing to be detected. This model also used an enhanced form of data augmentation that would be effective in training the model on variety of situations it will encounter on the road. The findings from Deep Neural Network revealed that there was a reduced MSE as contrasted with Standard Sequential Networks Model especially where there are intricate aspects of the road system. This improvement showed that pre-training models with Deep Architectures yields better performance of the model.

Extended Layers NN takes Deep Neural Network toward higher complexity in the network structure which forms the next level of the hierarchical structure of the network of the analysed vocal folds. This model had more convolution layers and modified the kernel size of the convolution layers and also added drop-out layers to minimise on over-fitting. The purpose was to build a model which, along with fitting the training data, would provide a better generalisation of the unseen data. As expected, the obtained results showed that Extended Neural Networks had the lowest value for MSE, which suggested that it was the most accurate steering model.

When comparing these three models based on Object 3, it is clear that every successive model iteration actually achieved the goal of decreasing prediction error and consequently increasing the reliability of the values predicted for the steering angle. It could be observed that Deep Neural Network and particularly Extended Neural Networks demonstrate the advantages of Deep architectures as well as skills in preventing over-fitting through the application of the drop-out technique. The comparison also underscores the trade-offs involved in model development: although it was observed that Deep models produced a better results the price to pay is always the extra resources and time.

5.1 Comparison of Developed Models Verse Existing Models

This section will cover research objective from Table 1 Research objective 4. However, it shall also be relevant to compare the performance of the developed models with the models currently utilized in the area of autonomous driving, which can be specified in Object 4 in the table below. Current models within the domain of autonomous navigation like the recently developed end-to-end deep learning model of NVIDIA for self-driving cars, offers them a high standards of performance, efficiency, accuracy, and real-time capabilities among others.

NVIDIA's model is a well-discussed example of deep learning that has found its use in the operation of self-driving cars. It directly predicts the steering angles from raw images using somewhat a Deep Convolutional Neural Network. That model has been well implemented in the real world, and the experimental results demonstrate that it is very resistant to real driving conditions. The first strength of the NVIDIA model is the model's robustness across various environments, and the second strength is its capability to address large datasets in real time.

When we compare our developed models with NVIDIA's model, the following differences become evident. Nonetheless the accuracy of predicting the steering angle of Extended Neural Networks in our project is comparable to that found in other literature and comes with the overhead of computational complexity. NVIDIA's model is balanced in terms of accuracy and speed which is very important for deployment in real-world applications. On the other hand, our models especially, Extended Neural Networks although they are accurate models might be little heavy and need some more tuning for real time performance.

One of the last, but very significant comparison criteria is the ability of models to generalize. Other approaches such as the one deployed at NVIDIA were trained on different driving conditions, making them better placed at handling new data. These models that we

have developed while might be performant within the simulation that we have used might need to be trained on more diverse datasets to become as generalized as these existing models.

At the same time, the obtained developed models show promising results. The architectural advances and the use of optimization studied in our models are beneficial for the constant progress of the autonomous driving technology. Suitably refined and optimized especially in the aspect of efficiency and the ability of generalization, these models could offer equivalent or higher performances in all or perhaps particular modes than existing models.

6 Conclusion and Future Work

Overall, the goal of the project was met as this work investigated the creation and improvement of three deep learning models for autonomous navigation with the emphasis on the estimation of steering angles in a driving environment that was simulated. The models were accumulated progressively, and at each step, the model comprised of new enhancements and complexities in order to receive the most accurate outcomes and efficiency. Standard Sequential Networks Model was set as a baseline and allowed gaining fundamental understanding of the vulnerability of a plain CNN structure. Deep Neural Network added more layers coupled with data augmentation hence performing well than Standard Sequential Networks Model especially when subjects are involved in complicated manoeuvres. Last model Extended Neural Networks provided deep understanding and added more regularization adding to the fact that it realized the highest accuracy.

The conclusions of this project prove that the architecture of the models has a significant impact on the ability of an autonomous vehicle to perform a navigation task effectively. The advancement from Standard Sequential Networks Model through to Extended Neural Networks shows how greater depths and careful consideration of other practices such as drop-out significantly improves predictive power. However, these improvements bring out a new challenge in terms of the computational complexity which deem it necessary to strike a balance between performance and computational requirements.

Future Work

Although, good performance was achieved on the developed models, there are areas of research that could pave the way for improving generalization of the models and thus their practical usability. First, the models which are used in those cars can be trained on a more diverse data ranging from weather conditions, different types of roads, and other challenging conditions and scenarios. This would enhance the generalization performance of the models since the current models lack good performance especially when implemented in new environments.

Other aspect of the work will do in the future is the adaptation of the models on the conditions that are required for live operation. This could involve, researching new topologies of networks or methods of model reduction such as pruning and quantization, that would lessen the computational load needed without compromising on performance. Furthermore, the application of the sensor fusion, where the information received from LiDAR/Radar and video cameras is combined, can provide even more precise prediction and better perception of the environment for creating more secure autonomous vehicles.

Acknowledgement

I owe the completion of this Research Project to Dr. Catherine Mulwa, who guided and supported me throughout the process. I am also very thankful to my family and loved ones for their financial and emotional help during my Master's studies. Finally, I want to express my deep appreciation to my colleagues, whose input has greatly contributed to what I've learned and achieved.

References

Bai, Y., Zhang, B., Xu, N., Zhou, J., Shi, J. and Diao, Z., 2023. Vision-based navigation and guidance for agricultural autonomous vehicles and robots: A review. Computers and Electronics in Agriculture, 205, p.107584.

Bautista-Montesano, R., Galluzzi, R., Ruan, K., Fu, Y. and Di, X., 2022. Autonomous navigation at unsignalized intersections: A coupled reinforcement learning and model predictive control approach. Transportation research part C: emerging technologies, 139, p.103662.

Koh, S., Zhou, B., Fang, H., Yang, P., Yang, Z., Yang, Q., Guan, L. and Ji, Z., 2020. Real-time deep reinforcement learning based vehicle navigation. Applied Soft Computing, 96, p.106694.

Lei, T., Luo, C., Sellers, T. and Rahimi, S., 2021. A bat-pigeon algorithm to crack detection-enabled autonomous vehicle navigation and mapping. Intelligent Systems with Applications, 12, p.200053.

Nikanjam, A., Morovati, M.M., Khomh, F. and Ben Braiek, H., 2022. Faults in deep reinforcement learning programs: a taxonomy and a detection approach. Automated software engineering, 29(1), p.8.

Pérez-Gil, Ó., Barea, R., López-Guillén, E., Bergasa, L.M., Gómez-Huélamo, C., Gutiérrez, R. and Díaz-Díaz, A., 2022. Deep reinforcement learning based control for Autonomous Vehicles in CARLA. Multimedia Tools and Applications, 81(3), pp.3553-3576.

Xu, Z., Chen, J. and Tomizuka, M., 2020. Guided policy search model-based reinforcement learning for urban autonomous driving. arXiv preprint arXiv:2005.03076.

Elallid, B.B., Benamar, N., Hafid, A.S., Rachidi, T. and Mrani, N., 2022. A comprehensive survey on the application of deep and reinforcement learning approaches in autonomous driving. Journal of King Saud University-Computer and Information Sciences, 34(9), pp.7366-7390. https://doi.org/10.1016/j.jksuci.2022.03.013

Prabhod, K.J., 2023. Advanced Techniques in Reinforcement Learning and Deep Learning for Autonomous Vehicle Navigation: Integrating Large Language Models for Real-Time Decision Making. Journal of AI-Assisted Scientific Discovery, 3(1), pp.1-20. https://scienceacadpress.com/index.php/jaasd/article/view/25

Tang, Y., Zhao, C., Wang, J., Zhang, C., Sun, Q., Zheng, W.X., Du, W., Qian, F. and Kurths, J., 2022. Perception and navigation in autonomous systems in the era of learning: A survey. IEEE Transactions on Neural Networks and Learning Systems, 34(12), pp.9604-9624. DOI: 10.1109/TNNLS.2022.3167688

Lee, H. and Jeong, J., 2023. Velocity range-based reward shaping technique for effective map-less navigation with LiDAR sensor and deep reinforcement learning. Frontiers in Neurorobotics, 17, p.1210442. https://doi.org/10.3389/fnbot.2023.1210442

You, C., Lu, J., Filev, D. and Tsiotras, P., 2019. Advanced planning for autonomous vehicles using reinforcement learning and deep inverse reinforcement learning. Robotics and Autonomous Systems, 114, pp.1-18. https://doi.org/10.1016/j.robot.2019.01.003

Aradi, S., 2020. Survey of deep reinforcement learning for motion planning of autonomous vehicles. IEEE Transactions on Intelligent Transportation Systems, 23(2), pp.740-759. DOI: 10.1109/TITS.2020.3024655

Lei, L., Tan, Y., Zheng, K., Liu, S., Zhang, K. and Shen, X., 2020. Deep reinforcement learning for autonomous internet of things: Model, applications and challenges. IEEE Communications Surveys & Tutorials, 22(3), pp.1722-1760. DOI: 10.1109/COMST.2020.2988367

Grigorescu, S., Trasnea, B., Cocias, T. and Macesanu, G., 2020. A survey of deep learning techniques for autonomous driving. Journal of field robotics, 37(3), pp.362-386. https://doi.org/10.1002/rob.21918