**REINFORCEMENT LEARNING FOR SELF DRIVING CARS IN CARLA SIMULATOR**

A thesis submitted in partial fulfilment of the requirements for the award of the degree of

**B.Tech.**

**in**

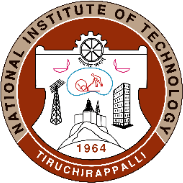
**Instrumentation and Control Engineering**

By

**Pedanayuni Venkata Sai (110119076)**

Completed at the

**National Institute of Technology, Tiruchirappalli, India**



**INSTRUMENTATION AND CONTROL ENGINEERING**

**NATIONAL INSTITUTE OF TECHNOLOGY TIRUCHIRAPALLI-620015**

**MAY 2023**

**BONAFIDE CERTIFICATE**

This is to certify that the project titled **ELECTRICITY LOAD FORECASTING IN ENGINEERING INSTITUTE** is a bonafide record of the work done by

**Pedanayuni Venkata Sai (110119076)**

in partial fulfilment of the requirements for the award of the degree of **Bachelor of Technology in Instrumentation and Control Engineering of the NATIONAL INSTITUTE OF TECHNOLOGY, TIRUCHIRAPPALLI**, during the year **2022-2023.**

|  |  |
| --- | --- |
| **Dr. B. Vasuki** | **Dr. K. Dhanalakshmi** |
| Internal Guide | Head of Department |
| Professor | Professor |
| NIT Trichy | NIT Trichy |

# ABSTRACT

The aim of this project is to perform load forecasting using Machine Learning and Deep Learning models. To analyse various trends and insights of power consumption in different areas of NIT Trichy. To visualize load usage with respect to seasons, weather conditions, such as Temperature, Humidity, etc. To examine trends based on weekdays, weekends, holidays, fests, etc. Analysis is performed using Python libraries such as seaborn, matplotlib and plotly.

*Keywords*: Machine Learning; Deep Learning; Python; seaborn; matplotlib; plotly.

# ACKNOWLEDGEMENTS

I wish to express my sincere thanks to my project guide, Dr. B. Vasuki for providing me with the expertise and knowledge for the implementation of the project work and for having faith in our aptitude through this entire period, without which the project would never see completion. I would like to thank Head of the CSG Department, Dr. N. Sivakumaran for helping us to acquire data for the project. I would also like to thank Assistant Engineer (AE) Mr.C.Nallathambi and Junior Engineer (JE/Electrical) Mr. SR Indrajith of EMC department for providing required data for the project. I would also like to extend my gratitude to the Project Review Committee consisting of Dr. A. Ramakalyan, Dr. D. Ezhilarasi, and Dr. C. Geetha for ensuring a seamless progression of the project. Finally, I wish to thank the Head of Department, Dr. K. Dhanalakshmi for her support.

**TABLE OF CONTENTS**

[ABSTRACT 3](#_Toc133320135)

[ACKNOWLEDGEMENTS 4](#_Toc133320136)

[LIST OF FIGURES 6](#_Toc133320137)

[LIST OF TABLES 7](#_Toc133320138)

[CHAPTER 1 – INTRODUCTION 8](#_Toc133320139)

[1.1 Introduction 8](#_Toc133320140)

[1.2 Necessity of Load Forecasting 9](#_Toc133320143)

[CHAPTER 2 – LITERATURE REVIEW 10](#_Toc133320144)

[CHAPTER 3 – Data Collection 11](#_Toc133320145)

[3.1 Data Collection 11](#_Toc133320146)

[3.2 Categorizing the Data 12](#_Toc133320147)

[CHAPTER 4 – Outlier Handling 14](#_Toc133320152)

[4.1 Outliers 14](#_Toc133320153)

[4.2 Outliers Identification 14](#_Toc133320154)

[4.2.1 Mean 15](#_Toc133320155)

[4.2.2 Non-Outlier Maximum 16](#_Toc133320156)

[4.3 Handling Outliers 16](#_Toc133320158)

[CHAPTER 5 – Data Imputation 18](#_Toc133320159)

[5.1 Missing Values Identification 18](#_Toc133320160)

[5.2 Filling Missing Values 18](#_Toc133320161)

[CHAPTER 6 – Model Results and Discussions 20](#_Toc133320162)

[6.1 Models Used 20](#_Toc133320163)

[6.2 Models Results 22](#_Toc133320163)

[6.3 Models Performance 24](#_Toc133320163)

[6.4 Discussions and Improvements 26](#_Toc133320163)

[CHAPTER 7 – CONCLUSIONS 27](#_Toc133320167)

[ABBREVIATIONS 28](#_Toc133320168)

[BIBLIOGRAPHY 29](#_Toc133320169)

# 

# LIST OF FIGURES

|  |  |  |
| --- | --- | --- |
| **Figure No.** | **Figure Name** | **Page No.** |
| 1.1 | Use cases of Load Forecasting | 9 |
| 3.1 | Data Classification | 11 |
| 3.2 | Data Categorization | 13 |
| 3.3 | Load usage vs Temperature relation | 13 |
| 4.1 | Maximum KWH usage by each dept in February | 14 |
| 4.2 | KWH usage of ICE dept on every Thursday | 15 |
| 4.3 | KWH usage of ICE dept after outlier handling on every Thursday | 17 |
| 5.1 | ICE KWH usage in November (before imputation) | 19 |
| 5.2 | ICE KWH usage in November (after imputation) | 19 |
| 6.1 | Deep Neural Network Architecture | 20 |
| 6.2 | Long Short-Term Memory Model | 21 |
| 6.3 | DNN model for training | 22 |
| 6.4 | DNN model results | 23 |
| 6.5 | LSTM model for training | 23 |
| 6.6 | LSTM model results | 24 |
| 6.7 | Mean Square error loss for DNN | 24 |
| 6.8 | Mean Square error loss for LSTM | 25 |
|  |  |  |

# LIST OF TABLES

|  |  |  |
| --- | --- | --- |
| **Table No.** | **Table Name** | **Page No.** |
| 3.1 | ICE Dept KWH usage on 02/03/2023 | 12 |

# CHAPTER 1 – INTRODUCTION

## 1.1 Introduction

Artificial Intelligence has become extremely popular in today’s world. It is the reproduction of regular knowledge in machines that are customized to impersonate the activities of people. These machines can learn and execute human-like tasks. As these technologies continue to grow, they will significantly affect our quality of life. One of the technologies that have emerged with this growth is autonomous driving. An autonomous car is a vehicle which senses its environment and operates without any human involvement.

The main aim of this thesis is to set a base on the topic of autonomous driving with the help of the open-source software CARLA in which we will be able to set up an environment and simulate the learning of an intelligent car.

Terms and concepts

The key concepts of this work are defined in the next section as the reader must be familiarized with the following concepts and understand them correctly as well as the topics related to autonomous driving that can generate more confusion.

Machine learning

Machine learning is a method of data analysis that automates analytical model building. It is a branch of artificial intelligence based on the idea that systems can learn from data, identify patterns, and make decisions with minimal human intervention.

Reinforcement learning

Reinforcement learning is a machine learning training method based on rewarding desired behaviors and/or punishing undesired ones. In general, a reinforcement learning agent can perceive and interpret its environment, take actions and learn through trial and error.

DQN

The Deep Q-Network algorithm was developed by DeepMind in 2015. It could solve a wide range of Atari games (some to superhuman level) by combining reinforcement learning and deep neural networks at scale. The algorithm was developed by enhancing a classic RL algorithm called Q-Learning with deep neural networks and a technique called experience replay.

ANNs

Neural networks, also known as artificial neural networks (ANNs) are a subset of machine learning and are at the heart of deep learning algorithms. Their name and structure are inspired by the human brain, mimicking the way that biological neurons signal to one another.

Autonomous Car

A self-driving car, also known as an autonomous vehicle (AV) or driverless car, is a car incorporating vehicular automation, that is, a ground vehicle that can sense its environment and move safely with little or no human input.

CARLA

CARLA is the open-source simulator for autonomous driving research. It has been developed from the ground up to support development, training, and validation of autonomous driving systems. The simulation platform supports flexible specification of sensor suites, environmental conditions, full control of all static and dynamic actors, map generation and much more.

Problem to be resolved.

The project consists of using the CARLA Simulator to obtain autonomous driving. It has been divided into two sections to facilitate and ensure the correct completion of the work. The first part consists of setting up the environment that we will be using for the execution of the learning process. This includes the prerequisites, software installation, library dependencies and any other requisite. The second part of the project is the proper learning of the car. To accomplish this a circuit will be defined with an origin, a destination, and a series of waypoints where the car must pass through. Once the waypoint system has been developed, we can start with the learning process of the car. Thanks to reinforcement learning, the car will perceive and interpret its environment, take actions, and learn through trial and error.

# CHAPTER 2 – LITERATURE REVIEW

Vehicle control can correct errors coming from motion planning and decision-making tasks. In recent years, great efforts have been made to improve vehicle control for AVs, particularly using DL. Sharma et al. [1] collected data include images labeled by vehicle speed, steering angle, and throttle using the TORCS simulator (TORCS). This data is fed into a CNN to train their model to estimate vehicle speed and steering angle at the same time. However, their approach is limited only by CNN. Thus, TORCS simulator is not realistic to validate the performance of their model.

Mujica et al. [2] presented a model to steer AVs through several scenarios with lane changes and traffic lights. They trained their proposal with CNN and RNN to extract features from the data collected by the Udacity simulator. Their results are somewhat better compared to previous.

Xing et al. [3] suggested method to effectuate lane change on a freeway. The model uses both CNN and RNN to train their model. Their results are not tested in a simulator to validate their performance.

In [7], authors presented RL method to generate the steering angle of an AV to avoid sharp turns in the road. Their model showed significant stability during learning, but it excluded other road users in the environment.

The previous approaches, one of them being trained with DL techniques using data collected from simulators such as TORCS or Udacity, do not provide realistic scenarios like in the real world. Moreover, training a model with single data is inefficient because each environment has its own characteristics. For these reasons, we propose an RL model based on DQN method that allows AV to learn from the environment to reach its destination and also avoid collisions with other vehicles and pedestrians. Moreover, we test and validate our approach in the latest version of the CARLA simulator, which is more realistic and contains buildings, pedestrians, etc.

# CHAPTER 3 – CARLA SETUP

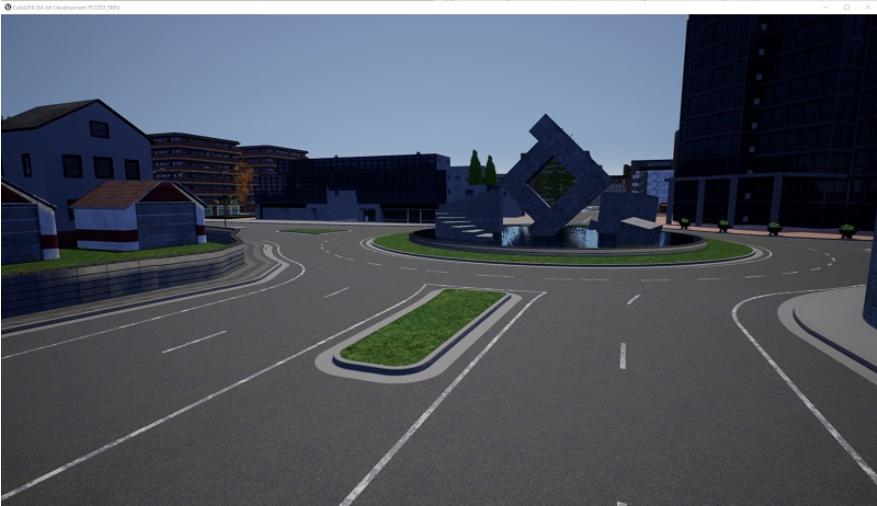
Our first order of business is to get Carla. It's simple. Head over to CARLA official website and then scroll down a bit to the [latest release](https://github.com/carla-simulator/carla/blob/master/Docs/download.md).

From here, at least at the time of my writing this, the latest version to have windows support is CARLA 0.9.8. We are using Windows machine for this project; we need to grab CARLA 0.9.8.zip.

Later, you might be able to get more up-to-date versions. Feel free to do that, just know that things will change over time, including the code syntax. It might be best for you to learn on the same versions that I am using, then later upgrade. Do whatever you want though!

Once you have the compressed info, extract it, and you will have what you need to run. In the main directory of what you just extracted; you will have a CarlaUE4.exe on Windows.

To run this on Windows, just double click the .exe.



This will run the server. You should eventually see the map, which you can navigate with the WASD keys and your mouse. Of course, nothing is yet here. This is just our environment. The GET STARTED section shows some examples of what we can do, which make use of the examples that we can find in the PythonAPI directory.

Go ahead and navigate from the main Carla directory to the examples: PythonAPI/examples. Here, you will have some files like manual\_control.py, dynamic\_weather.py, and spawn\_npc.py to name a few. We can check some of these out. Open terminal/cmd and run one like manual\_control.py:

**py -3.7 manual\_control.py**

This will create another window. What you're seeing is an example of what we can do with the Python API. Here, you can control the car with WASD keys, and Q will change into and out of reverse.



There are python API’s that allow users to control every aspect of the simulation. Our view is to have everything available from Python; loading maps, adding sensors, adding static obstacles, controlling vehicles and pedestrians, and what not.

Before moving forward here are the 3 important concepts:

* **Actor:** Actor is anything that plays a role in the simulation and can be moved around, examples of actors are vehicles, pedestrians, and sensors.
* **Blueprint:** Before spawning an actor you need to specify its attributes, and that’s what blueprints are for. We provide a blueprint library with the definitions of all the actors available for you to choose.
* **World:** The world represents the currently loaded map and contains the functions for converting a blueprint into a living actor moving around in the simulation.

**Sensors Used**

**Cameras** are the first main type of sensors that we can use in vehicles. They take a shot of the world from their point of view a can retrieve data in every simulation step.

* RGB camera provides clear vision of the surroundings. Looks like a normal photo of the scene.

The next group of sensors are detectors, and they retrieve data when the object they are attached to registers a specific event.

* Collision detector retrieves collisions between its parent and other actors.

# CHAPTER 4 – REINFORCEMENT LEARNING (DQN)

Reinforcement Learning is all about learning the optimal behavior in an environment to obtain maximum reward with an agent.

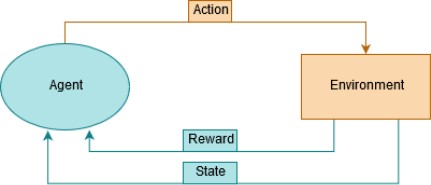
1. Introduction

In RL we have two main actors: the agent and the environment, in this case CARLA, in which an agent is trained. To make this training possible we have three main interactions between them. These are the following:

- State: it describes the status of the agent. This could be the current position of the car, the speed or even the orientation of the vehicle.

- Action: it describes what can an agent do. Moving forward, pressing the brake, or rotating the car are different actions that can be performed.

- Reward: it is feedback on the action previously taken on a state. This could be a positive or negative reward.



Maximizing the reward obtained by performing a series of actions is the main objective of the agent as we previously mentioned. Therefore, a proper reward system is very important as choosing one action or the other will be affected by it.

During the training process we will see two main stages: exploration and exploitation. The exploration stage has a specific purpose, to visit states that you have not yet visited or to take actions you have not yet taken. When you start the first episodes you want the agent to explore and do some trial and error to find the actions that will return a positive reward. On the other hand, the exploitation stage is when you decide to use the knowledge and memory from the previous episode to maximize the rewards. This stage usually comes at the end of the training session or when you have a trained agent.

The agent will require a policy to learn. The policy defines how the agent will behave at a given time and map from perceived states of the environment to actions to be taken when in those states. To store all this data the policy uses a huge table with all the information saved for each state. At the beginning, if the agent has not been trained before this table will be empty, in other words, it will have no experience. This in when the exploration stage becomes useful. After the agent has performed a series of steps, it will start leaning by adjusting its predicted rewards for specific pairs of state-actions towards the received rewards. These predicted values are known as Q-values and are used as feedback on how good a state is.

The parameter that determines the probability of the agent performing a random action is the epsilon (∈). It is a real value that exists in [0, 1]. This variable introduces randomness into our training algorithm forcing the agent to try different actions. A high epsilon, for example, ∈ = 1 will result in all actions are random which means we are in the exploration stage. On the other hand, a lower epsilon such as ∈ = 0 will result in no random actions which means we are in the exploitation stage. Implementing an epsilon decay formula is very important during the training process. This is because we want to travel from the exploration stage to the exploitation to maximize learning. The formula we will be using will be the following:



We will need to set an ∈𝑑𝑒𝑐𝑎𝑦 that, when multiplied by the current epsilon, will give us the next epsilon value. To calculate the epsilon value in a certain number of episodes we will use the following formula:



Where 𝑛 is the number of episodes, ∈𝑑𝑒𝑐𝑎𝑦 the epsilon decay per episode and ∈𝑖𝑛𝑖𝑡𝑖𝑎𝑙 the epsilon used at the beginning.

Another important parameter is the discount factor (𝛾). It is a real value that exists in [0, 1] and determines how the agent should care about the rewards achieved in the past, present and future. If 𝛾=0 the agent will focus on actions that produce an immediate reward whereas if 𝛾=1 the agent cares on actions that will produce a high future reward due to a sequence of actions.

4.2 DQN Agent

Deep Q-Network is a reinforcement learning algorithm that combines Q-Learning with deep neural networks to let RL work for complex, high-dimensional environments, like video games, or robotics.

DQN uses the Bellman’s equation to solve the problem of being unable to see future rewards. Otherwise, only immediate rewards would be shown. The formulation of the equations is as follows:

A picture containing text, font, white, typography

Description automatically generated

Where:

* Q is the accumulated reward value
* t is the current state
* s is a specific state
* a is a specific action
* ∈ is the epsilon, also known as learning rate.
* r is the reward associated to a specific state
* 𝛾 is the discount factor.

The agent can use this equation to link up all the information learned resulting in a batch of memory that can be used for improving the efficiency of its learning.

4.3 ANN

For this project we used a CNN.

It consists of:

* 3 convolutional layers with 64 neurons each + ReLU activation function which is used for filtering information.
* 3 pooling layers performed with Average pooling. They are used to reduce the dimensions of the feature maps, in this case 3x3.
* 3 dropouts layers following the convolutional networks to avoid overfitting.
* 1 flatten layer to convert the vector to 1D.
* 1 dense layer with 64 neurons.
* 1 dense layer with 3 neurons (3 actions) with linear activation function. It is used to determine the output of neural network like yes or no.

# CHAPTER 4 – IMPLEMENTATION

Here's a brief outline of the steps involved in implementing DQN on Carla:

1. Set up Carla and the Python API: Follow the installation instructions for Carla and the Python API. You will need to have Python and TensorFlow installed on your system.
2. Define the Carla environment: In the Carla environment, the state space consists of the current vehicle position, velocity, and orientation, as well as the positions and orientations of nearby vehicles and obstacles. The action space consists of discrete steering and throttle values. The reward function should be designed to encourage safe and efficient driving behaviour.
3. Implement the DQN algorithm: The DQN algorithm involves using a neural network to learn a Q-function, which maps states to expected rewards for each action. The neural network is trained using a replay buffer to store experience tuples and a target network to stabilize the learning process. The TensorFlow library provides a convenient way to implement the neural network and train it using gradient descent.
4. Train the DQN agent: To train the DQN agent, you will need to run the Carla simulator and interact with the environment to collect experience tuples. The agent then uses these tuples to update its neural network and improve its Q-function estimates. Training can take many episodes, and the agent's performance should be periodically evaluated to ensure it is improving.

Top of Form

# CHAPTER 5 – MODEL RESULTS AND DISCUSSIONS

The main important parameters that are needed to get tuned or the whole process of Deep Q learning depends on Q values that depends on the rewards that agent is getting for a particular action. In our project we trained the model 3 times with varying reward function and ANN used.

Training 1:

The first training is done in “Town1” where the rewards and actions are described below. This test is the first approach with CARLA and all its environment setup. Obviously, good results are not going to be expected in the iteration.

The goal primarily here is to avoid collisions. The agent will be taking actions based on Q values associated with each action.

Training conditions:

|  |  |
| --- | --- |
| Total Training Time | 2 hrs |
| No.of Episodes | 100 |
| Actions | Action 0: Turn left  Action 1: Accelerate the car  Action 2: Turn right |
| Rewards | Reward 1: Collision: -200 points  Reward 2: Velocity > 50 km/h: +1 point  Reward 3: Velocity < 50 km/h: -1 point |
| Sensors Used | RGB camera  Collision detection sensor |
| Model | Xception |

Results:

|  |  |
| --- | --- |
| Accuracy    Episodes | Epsilon    Episodes |
| Loss    Episodes | Reward\_Average    Episodes |
| Reward\_maximum    Episodes | Reward\_minimum    Episodes |

The first thing we discovered was that both loss and q values were basically exploding:

This was seemingly, and now obviously, due to the monumental size of the crash penalty in comparison to all else. Also possibly the bounds being out of range. For example, we might have had more success by doing something more like:

* +0.005 (1/200) for each frame driving > 50KMH
* -0.005 (-1/200) for each frame driving < 50KMH
* -1 (-200/200) for a collision and episode is over

But instead we just went with:

* +1 for each frame driving > 50KMH
* -1 for each frame driving < 50KMH
* -10 for a collision and episode is over

And also we were finding that our accuracy for the model was quite high (like 80-95%), so this tells me that we were almost certainly just overfitting every time.

Training 2:

To avoid above mentioned problems we went with other model the simple CNN and with the new reward function.

Training conditions:

|  |  |
| --- | --- |
| Total Training Time | 20 hrs |
| No.of Episodes | 100,000 |
| Actions | Action 0: Turn left  Action 1: Accelerate the car  Action 2: Turn right |
| Rewards | Reward 1: Collision: -10 points  Reward 2: Velocity > 50 km/h: +1 point  Reward 3: Velocity < 50 km/h: -1 point |
| Sensors Used | RGB camera  Collision detection sensor |
| Model | CNN |

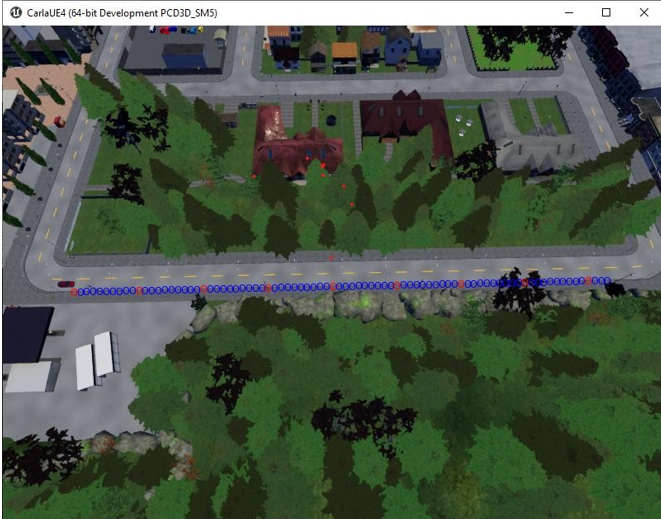
Results:

|  |  |
| --- | --- |
| Accuracy    Episodes | Epsilon    Episodes |
| Loss    Episodes | Reward\_Average    Episodes |
| Reward\_maximum    Episodes | Reward\_minimum    Episodes |

The graphs show that during the training session, the maximum and average reward had an upward trend. This is a good sign as it shows that the car is learning something and trying to maximize this reward. The problem is that when we look at the learning episodes themselves, we realize that the car is driving but is not aware that it must get to a particular place. Moreover, it does not consider the directions of the lane, the car simply decides to stay on the road and avoids collisions.

Training 3:

For this training it was decided to implement a new reward system as the previous one did not consider anything about reaching a desired destination. In addition, a new waypoint system was created which consists of dividing a trip from point A to point B into small one-meter sections. The actions remained the same.



Training conditions:

|  |  |
| --- | --- |
| Total Training Time | 4 hrs |
| No.of Episodes | 1400 |
| Actions | Action 0: Turn left  Action 1: Accelerate the car  Action 2: Turn right |
| Rewards | Reward 1: Collision: -10 points  Reward 2: Arrived at destination: +20 points  Reward 3: If distance to destination decrease: +5  Reward 4: If distance to destination increases: -5  Reward 5: short training episode: -10 |
| Sensors Used | RGB camera  Collision detection sensor  Waypoint system |
| Model | CNN |

Results:

|  |  |
| --- | --- |
| Accuracy    Episodes | Epsilon    Episodes |
| Loss    Episodes | Reward\_Average    Episodes |
| Reward\_maximum    Episodes | Reward\_minimum    Episodes |

There is a slight upward trend that can be seen in average, maximum and minimum reward graphs that indicates the car is learning something, but the trend is not that high as we have limited its training to a smaller number of episodes. The agent is avoiding collisions and was able to move from source to destination, but the movement is not that smooth there is change in action for each frame. Also, it is not following the lane changes which need to handle.

# CHAPTER 6 – CONCLUSIONS

Conclusions

* Using DQN we are succ

# BIBLIOGRAPHY

[1] S. Sharma, G. Tewolde, and J. Kwon, “Lateral and longitudinal motion control of autonomous vehicles using deep learning,” in 2019 IEEE International Conference on Electro Information Technology (EIT). IEEE, 2019, pp. 1–5.

[2] D. Mújica-Vargas, A. Luna-Álvarez, J. de Jesús Rubio, and B. CarvajalGámez, “Noise gradient strategy for an enhanced hybrid convolutionalrecurrent deep network to control a self-driving vehicle,” Applied Soft Computing, vol. 92, p. 106258, 2020.

[3] Y. Xing, C. Lv, H. Wang, D. Cao, and E. Velenis, “An ensemble deep learning approach for driver lane change intention inference,” Transportation Research Part C: Emerging Technologies, vol. 115, p. 102615, 2020

[4] F. Hui, C. Wei, W. ShangGuan, R. Ando, and S. Fang, “Deep encoder– decoder-nn: A deep learning-based autonomous vehicle trajectory prediction and correction model,” Physica A: Statistical Mechanics and its Applications, vol. 593, p. 126869, 2022.

[4] X. Nie, C. Min, Y. Pan, K. Li, and Z. Li, “Deep-neural-network-based modelling of longitudinal-lateral dynamics to predict the vehicle states for autonomous driving,” Sensors, vol. 22, no. 5, p. 2013, 2022.

[5] L. Chen, X. Hu, B. Tang, and Y. Cheng, “Conditional dqn-based motion planning with fuzzy logic for autonomous driving,” IEEE Transactions on Intelligent Transportation Systems, 2020.

[6] J. Chen, C. Zhang, J. Luo, J. Xie, and Y. Wan, “Driving maneuvers prediction based autonomous driving control by deep monte carlo tree search,” IEEE Transactions on Vehicular Technology, 2020.

[7] Ó. Pérez-Gil, R. Barea, E. López-Guillén, L. M. Bergasa, C. GómezHuélamo, R. Gutiérrez, and A. Díaz-Díaz, “Deep reinforcement learning based control for autonomous vehicles in carla,” Multimedia Tools and Applications, vol. 81, no. 3, pp. 3553–3576, 2022.