Trinity College Dublin
Coláiste na Tríonóide, Baile Átha Cliath
The University of Dublin

School of Computer Science and Statistics

A DRL-based Adaptive Cruise Control model for safe and efficient car following

MSc Computer Science - Data Science

August 19, 2022

A dissertation submitted in partial fulfilment
of the requirements for the degree of
MSc Computer Science - Data Science

Declaration

Abstract

Health and Safety is the main concern for car companies in the modern world as they ensure to keep the passengers comfortable and safe while driving or during a collision by maintaining standards and requirements. A study conducted in 2012 reveals that fatal road traffic occurs every 50 seconds on average and there are 1.35 million deaths caused every year due to road accidents according to the WHO. The inability of drivers to control a vehicle at its friction limits is the main cause of accidents. As a result, this study aims to maintain a safe distance between two vehicles contributing to traffic safety, traffic capacity, and efficiency.

The main objective of this research is to design, implement and evaluate Machine Learning based Adaptive Cruise Control (ACC) for collision avoidance of autonomous vehicles. The research aims to develop a scalable solution for Adaptive Cruise Control using Deep Reinforcement Learning in Traffic simulation. With the recent development of Autonomous vehicles, extensive research are made in Adaptive Cruise Control (ACC) technology. Autonomous Vehicles (AV) is an interesting topic that plays a lead role in future transportation and has potential benefits to society by reducing traffic congestion and reducing the number of fatalities. With Reinforcement Learning techniques traffic optimizations are done by learning optimal policy in order to solve decision-making problems without any human interactions. Deep Reinforcement learning has been used to optimize vehicle speed control and avoid collisions. Therefore, this thesis is focused on using the Deep Reinforcement Learning method with Double Deep Q Learning (DDQN) by training two agents with different scenarios. The CARLA traffic simulator is used for these simulations where key metrics such as environment, number of agents, number of collisions, etc are initialized and detected. Throughout the simulation, three sets of actions are analyzed and tested. Double Deep Q Learning (DDQN) Method is implemented to model the ACC which takes input from RADAR and LiDAR sensors in order to calculate the accurate position and distance between two vehicles. A single-agent is trained with respect to a leading vehicle and after training, agents are able to avoid collision by controlling the speed and maintaining the minimum distance set by the user.

Acknowledgements

First and Foremost I would like to thank my project supervisor Melanie Bouroche for her constant guidance and encouragement, expertise, and willingness to answer all my questions without which this thesis work would not be possible. Thank you

Secondly, I would like to thank my family, my mum, dad, and uncle for their continued support. Lastly to all my professors and student friends who helped and supported me throughout this course. Thank you.

Contents

1 Introduction **1**

1.1 Motivation 1

1.2 Goals and Research Question 3

1.3 Thesis Contribution 4

1.4 Thesis Structure 5

2 Background **6**

2.1 Cruise Control 6

2.1.1 Adaptive Cruise Control 7

2.1.2 Longitudinal Control Algorithm 7

2.2 Autonomous Vehicles 9

2.2.1 Connected Autonomous Vehicles 9

2.3 Artificial Intelligence 11

2.3.1 Machine Learning 12

2.3.2 Q-Learning 15

2.3.3 Double Q Learning 16

2.4 Deep Learning 17

2.4.1 Backpropagation 18

2.4.2 Optimization Algorithms 18

2.5 Double Deep Q Learning 20

2.6 Sensors 21

2.6.1 RADAR 22

2.6.2 LIDAR 22

2.6.3 CAMERA 24

2.7 Summary 24

3 State of Art 25

3.1 Recent Advancements in Self Driving Vehicles 26

3.1.1 Advantages of Autonomous Cars 26

3.1.2 Drawbacks of Self Driving Cars 27

3.2 Vehicle Control Schemes for Automation 28

3.2.1 Classical Control Approach without Machine Learning 28

3.2.2 Classical Control Approach using Reinforcement Learning 29

3.2.3 Controllers using DDQN in Autonomous Vehicles 33

3.3 Simulation and Training of DQN 35

3.4 Summary 37

4 Design and Implementation 39

4.1 Proposed Method for Adaptive Cruise Controller 39

4.1.1 Implementation of RADAR and LiDAR Based System 40

4.1.2 LIDAR Structure 41

4.1.3 RADAR Structure 42

4.2 MDP Formulation for Adaptive Cruise Control 42

4.2.1 State Representation 43

4.2.2 Action Space Representation 44

4.2.3 Reward Function 44

4.3 CARLA Simulation 45

4.3.1 CARLA Simulator for Autonomous Vehicles 46

4.3.2 Configuration of Actors 48

4.3.3 Retrieving Simulation 49

4.3.4 Simulation Recording and Replaying 51

4.4 Hardware Requirements 51

4.5 Adaptive Cruise Control Problem using CARLA 52

4.5.1 Agent Training using DDQN 52

4.5.2 Neural Network Architecture 53

4.6 Training and Results of the Model 54

4.7 Difficulties 56

4.8 Summary 57

5 Evaluation **59**

5.1 Objective 59

5.2 Metrics 60

5.3 Evaluation Scenarios 60

5.4 Setup 61

5.4.1 Weather Parameters 61

5.4.2 Spawning more vehicles - High traffic 62

5.4.3 Evaluation of Action sets 63

5.5 Comparison with DQN Model 64

5.6 Evaluation Summary 65

6 Conclusion **66**

6.1 Thesis Contribution 66

6.2 Future Work 67

List of Figures

1.1 **CARLA Simulation using Reinforcement Learning** 2

2.1 **Level of Automation in Present Vehicles** 10

2.2 **Reinforcement Learning** 13

3.1 **Model Predictive Controller [9]** 29

3.2 **Architecture of Double DQN system [34]** 34

4.1 **Ego and Leading Vehicle in CARLA Simulation in Town 1** 40

4.2 **CARLA Simulation using Reinforcement Learning** 47

4.3 **CARLA Simulation Loop - Autonomous Vehicles** 48

4.4 **Q Learning Method** 54

4.5 **Deep Neural Network Architecture** 54

4.6 Model Output 58

5.1 Dynamic weather patterns in CARLA 62

5.2 **Average Reward vs Episode** 63

5.3 **Number of Collisions vs Episode bn** 63

5.4 **Number of Collisions - DQN vs DDQN** 64

5.5 **Average Reward - DQN vs DDQN** 65

List of Tables

2.1 Different specifications of RADAR technology 22

3.1 Existing Research in Deep Reinforcement Learning for Adaptive cruise control 35

3.2 Previous work on Autonomous Vehicles using Reinforcement Learning and the respective Simulation Environment used 38

4.1 Hardware Specification 52

4.2 Training Parameters 55

5.1 CARLA Weather conditions 62

Nomenclature

A	Area of the wing	m^2
B		
C	Roman letters first, with capitals...	
a	then lower case.	
b		
c		
Γ	Followed by Greek capitals...	
α	then lower case greek symbols.	
β		
ϵ		
TLA	Finally, three letter acronyms and other abbreviations arranged alphabetically	

If a parameter has a typical unit that is used throughout your report, then it should be included here on the right hand side.

If you have a very mathematical report, then you may wish to divide the nomenclature list into functions and variables, and then sub- and super-scripts.

Note that Roman mathematical symbols are typically in a serif font in italics.

1 Introduction

The dissertation explains the traffic safety issue in Self Driving cars and studies the Adaptive Cruise Control (ACC) using Deep Reinforcement Learning (DRL) method so that to maintain a set distance as a safety parameter between vehicles to avoid collisions. In order to obtain an efficient self-driving techniques even in complex situations, Deep Neural Networks are introduced to train the Adaptive Cruise Control. Therefore, this thesis will use the Double Deep Q Learning method (DDQN) which allows the agent to learn independently based on inputs from LiDAR and RADAR. For evaluation we use already developed vision-based DQN model and analyze how the learning of an agent is affected in terms of collision avoidance and maintaining a safe distance. This chapter explains about the motivation, goals, research question, project contribution and its structure.

1.1 Motivation

A self-driving car is a vehicle that can drive itself without any human intervention. The possibility to drive a car autonomously using software has been contemplated for years, but the technology to develop such a complex system has never been powerful enough. Artificial Intelligence is becoming really targeted and is now being used in automotive industries which would contribute to making the public roads safer and avoid accidents due to human error. Artificial Intelligence is contributing to big improvements in autonomous vehicle navigation (including lane changing, control methodologies, etc) and is accelerating more and more over years. In this project, we study widely a branch of AI using the reinforcement learning approach (Q Learning) applied to ACC in autonomous driving which is an end-to-end model based on

sensor data. Simulation has become a key technology for virtual testing in a highly realistic environment that can mimic weather conditions, lighting, road conditions, etc and is safer and potentially faster than real world testing. *Cruise Control* is a well established Driver Assistance System where the velocity is set by the driver and this velocity is maintained by controlling the throttle even during the influence of bad weather, road slope and changing vehicle parameters. *Adaptive Cruise Control* is the further development of cruise control where the traffic is also considered and supports driving by increasing the comfort of driving as well as traffic safety. Environmental sensors are used in Advanced Driver Assistance Systems (ADAS) to control the vehicles during traffic in order to increase the safety and comfort by reacting to potentially dangerous traffic situations.
ACC system is shown in Figure 1.1 where the vehicle on left is the host vehicle equipped with ACC and the vehicle on right is the target vehicle. The relative distance between both the vehicles is given by

$$x_r = x_t - x_h \tag{1}$$

where x_t is the target vehicle position and x_h is the host vehicle position. The velocity difference between host vehicle (v_h) and the target vehicle (v_t) which is the relative velocity is given by

$$v_r = v_t - v_h \tag{2}$$

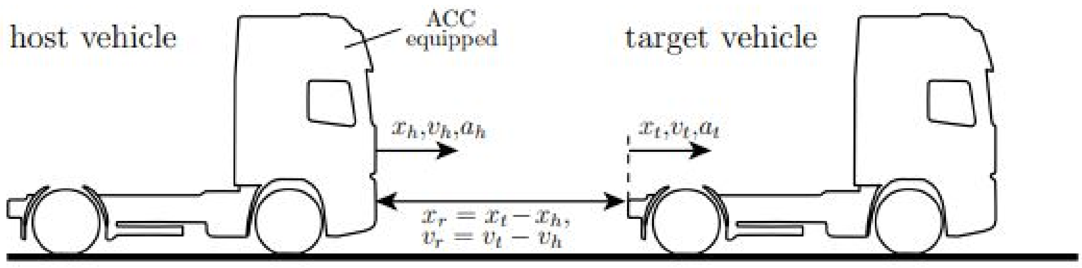


Figure 1.1: **CARLA Simulation using Reinforcement Learning**

In recent years, Deep Reinforcement learning (DRL) has shown exciting results in various other fields. AlphaGO, a computer program is trained with DRL during the year 2016 and it became the first computer program to win the professional player at Go which is a challenging game for computers to learn and play as it has a high number of possible moves and the difficulty lies in finding the best position. Using Supervised learning the program was trained with the help of expert games initially and is then improved to next stage with the use of RL method where the games are played against itself. AlphaZero outperformed AlphaGo in the following year, which was developed by the same team from DeepMind which was fully trained from the beginning without any human skills but only using DRL method. The only information given to the program was the rules of the game. AlphaZero is a common reinforcement learning algorithm which is trained to play chess and shogi. End-to-end autonomous driving model using Deep Reinforcement Learning is an advantage as it consumes less manpower, human knowledge, and skills when compared to other approaches that are used in driving methods today. With the use of simulation environment, the models can be trained and tested easily in a cost-effective manner and are also closing the reality gap between software simulation and a real road environment in order to deploy the trained model on a road infrastructure.

1.2 Goals and Research Question

Human error causes the majority of collisions due to several reasons such as bad road conditions, lack of driving skills, drunk and drive and so on. Even if the speed is reduced by the driver there may be accidents because of the failure of surrounding vehicles which has a considerable amount of effect on humans, the economy, traffic flow and capacity, and so on. There are number of studies based on Autonomous driving navigation, Adaptive Cruise Control, and Cooperative Adaptive Cruise Control which mainly concentrate on increasing the safety by avoiding collisions and maintaining traffic flow. Classical Controllers like PI and PID controllers have disadvantages in terms of accuracy and usability, which is overcome by the use of Machine Learning concepts with increased traffic safety, comfort, minimum fuel consumption and many more advantages. A controller-based model for autonomous vehicles may destabilize the vehicle system which is dangerous to the safety of autonomous vehicles and the surrounded traffic. Robust control stabilizes the vehicle system in case of uncertainties or errors but still the transient response of the system is delayed. In recent advancements of self-driving cars with reinforcement learning, vision-based technology is used which also has its disadvantages during bad weather, poor lighting conditions, etc. and neural networks should be powerful enough to process multiple camera images at the same time. Luminar at CES 2022 tested the low lighting conditions in the presence of a pedestrian and compared it with the Tesla vehicle which was unable to detect and avoid the pedestrian. In order to overcome these technical obstacles, a Deep Neural Network based optimal control model is developed for connected autonomous vehicles based on RADAR and LiDAR. Both these sensors map out the surroundings where LiDAR scans the object using infrared light and a 3D object of the environment is created in order to track movement and direction with more accuracy and precision. It can be used in ACC to find where the leading vehicle is moving towards. But LiDAR is not more accurate when it comes to poor weather conditions. In order to overcome this flaw, Radar is also incorporated which has many advantages like avoiding collisions, introducing parking assistance, and powering cruise control systems. Radio waves are used in RADAR to measure the distance between objects and also detect the velocity and angle of obstacles. RADAR sensor technology is used in many automotive companies, including BMW, Volkswagen, and Mercedes-Benz for autonomous driving. Since both the sensors have some flaws, a combination of the sensors is used in order to achieve true autonomy which can predict the environment, control autonomously with emergency braking at times of collision, make the right decisions, as well as share information with other vehicles and road infrastructure with an AI based algorithm.

1.3 Thesis Contribution

In this Dissertation, we are planning to build a DNN Based Adaptive Cruise Control for autonomous vehicles for efficient car following. We use Deep Reinforcement Learning (DRL) obtain autonomous braking control with the right decision-making strategy in emergency scenarios. In order to reduce the complexity of learning in very difficult environments, Double

Q Learning algorithm based on autonomous control architecture is used allowing the agent to find optimal policies in order to take an appropriate action. Multi objective reward is introduced to make the vehicles come out of danger quickly by avoiding collisions by maintaining a safe distance and also passenger comfort is considered with the idea of satisfied reward function. The idea developed is validated with the help of extensive simulations. Results from simulation test the performance and efficiency of the developed algorithm in terms of optimal learning, accuracy in decision making and safety of driving by avoiding collisions. Since the obstacle's state can change at any time, the collision risk varies accordingly, we implement our model for different traffic scenarios using a CARLA Simulation which models vehicle dynamics in real-time. A realistic-looking simulation environment also offers advantages of simulating virtual sensors like LiDAR, RADAR and cameras and also apply machine learning algorithms into the simulator. In our case, the CARLA simulator is utilized to prepare, train and assess models which gives an adaptable and safe climate for rapidly creating independent driving frameworks.Finally, validation is done on the basis of collision history and average rewards of training episodes. Average rewards and collision history is compared with the DQN model from the previously implemented studies.

1.4 Thesis Structure

The structure of the thesis is as follows. The background of this project is discussed in Chapter 2, where the basic concepts of Adaptive Cruise Control, Autonomous Vehicles, Artificial Intelligence, Reinforcement Learning Algorithms, Sensors used in self driving cars are discussed in detail. Chapter 3 discusses the related work in the areas of Autonomous vehicles, Adaptive Cruise controller without machine learning and with machine learning, and the simulation platforms used in various projects. The Design and Implementation are discussed in Chapter 4 where the key approach such as state space, action space and reward functions for the project and the implementation of the stated design using CARLA platform is discussed in detail.Chapter 5 concludes the overall thesis concepts and explains about the future work that can be done to improve the performance of ACC in Autonomous vehicles.

2 Background

This section covers the concepts of this dissertation project. Section 2.1 describes the concept of Cruise Control, Adaptive Cruise Control and Longitudinal Control Algorithm. Section 2.2 describes about Autonomous Vehicles and the theory relevant to Connected autonomous vehicles. Artificial Intelligence and its subdomains are described in next section 2.3. Deep Learning and its applications in autonomous vehicles are discussed in Section 2.4. Double DQN method and its architecture are explained in Section 2.5 The sensors used in most of the autonomous vehicles at present is explained in the section 2.6.

2.1 Cruise Control

Cruise Control (Speed Control) allows the driver to set the target speed and mostly used in long journeys where the control is mounted on steering wheels for most of the vehicles. Modern vehicles includes computer control systems, position sensor, input signals and vacuum actuator. Input from steering and speed signals are taken as input to computers and the computer again sends these to vacuum actuators through vacuum valve. Throttle valve is connected to actuator using a cable which has a position sensor mounted on it. The position sensor again sends signals back to the computer so that forming a loop so that the computer ensures of maintaining correct position for the target speed set by the driver. Most of the cruise control systems uses PID (Proportional Integral Derivative)control to set the throttle position. One of the main disadvantage of the CC is that it can be used only in very light traffic situations.

2.1.1 Adaptive Cruise Control

Adaptive Cruise Control is an enhancement of Cruise Control which reduces driver workload and fatigue . ACC uses the radar technology and is installed behind the grill of a vehicle to measure the headway and velocity in order to monitor other vehicles on road. Here, the target speed is set by the driver and also ACC adjusts is speed automatically to maintain an appropriate distance between vehicles on the same path. Using a digital signal processor, ACC system runs its radar signal to find the distance of the nearest moving vehicles and longitudinal controller is used to maintain safe following distance from the vehicle by applying brakes using brake control module. Engine control module is used to regulate the engine throttle using the data received from ACC module sensor. Cruise control systems and ACC will get disabled when brakes are applied manually. ACC can be used in heavy traffic conditions and ACC comes with pre crashed warning systems which notifies the driver to apply brakes if there is a sudden change in speed of the nearby vehicle.

2.1.2 Longitudinal Control Algorithm

Longitudinal control is the basic function of ACC system which is used to achieve consistent speed of the vehicle and avoiding the lead vehicle by maintaining a safe distance between them. It is also used for tracking and identifying the curve of the leading vehicle with automatic braking and other functions in order to increase the safety and comfort of the ACC systems.

Longitudinal controller is used to control the longitudinal motion of the vehicle i.e., longitudinal velocity, longitudinal acceleration and longitudinal distance from the leading vehicle travelling in the same lane. Actuators like throttles and brakes are mainly used to implement longitudinal control. The efficiency of the longitudinal control impacts the safety and comfort of the ACC system. Since ACC systems has a longer safe distance due to its speed range where radar technology is used as the sensors in present ACC systems. The idea of creating automated vehicles has been studied for decades and many control hypothetical arrangements were presented, planning to determine this issue. As an initial step, longitudinal speed and distance control issue was read up particularly for ACC frameworks according to the host vehicle perspective. Then, at that point, the execution of such frameworks was researched during a non-uniform road condition, for example, cornering lane change maneuvering . In such exploration studies, the vehicle behaviour is changed all together not to cause distress for the driver and passengers. The planning of such frameworks is to impersonate the reactions of a normal driver at the point when it operates.

A longitudinal direction driver assistance system use sensor modules measures the relative speed and distance between host vehicle on which ACC is installed and the leading vehicle in order to operate brakes and accelerators. The main disadvantage of ACC is its string stability problem of a platoon vehicle which is studied since late 1970s. The term "String stability" is referred as stable upstream propagation of a vehicle speed perturbation through a string of vehicles. The ACC control scheme consists of two hierarchial structures – Firstly, ACC control is used to compute the required acceleration or deceleration of the vehicle according to the control algorithm and it is required to provide control input for the brakes or throttle actuators for the purpose of safety and comfort. The sensor inputs (relative speed and relative distance) are directly given to the control blocks of the controller. These control inputs are then transferred to the actuators and the control issues are acknowledged by the low-level regulators guaranteeing the activation of the right results in view of the desired output. When observing the effect of ACC systems in a traffic flow different control approaches are considered as below:

Constant Space Headway Control - The distance between the leading and the host vehicle is assumed to be constant

Constant Time Headway Control – Here the control input is a function of a time gap and when it is multiplied by vehicle speed it gives controllers desired distance. The string stability of a system is guaranteed when there is a smooth acceleration/deceleration.

Variable Time Headway Control – Desired distance is controlled by the relative velocity of the vehicle where the "Time gap" value is changing continuously.

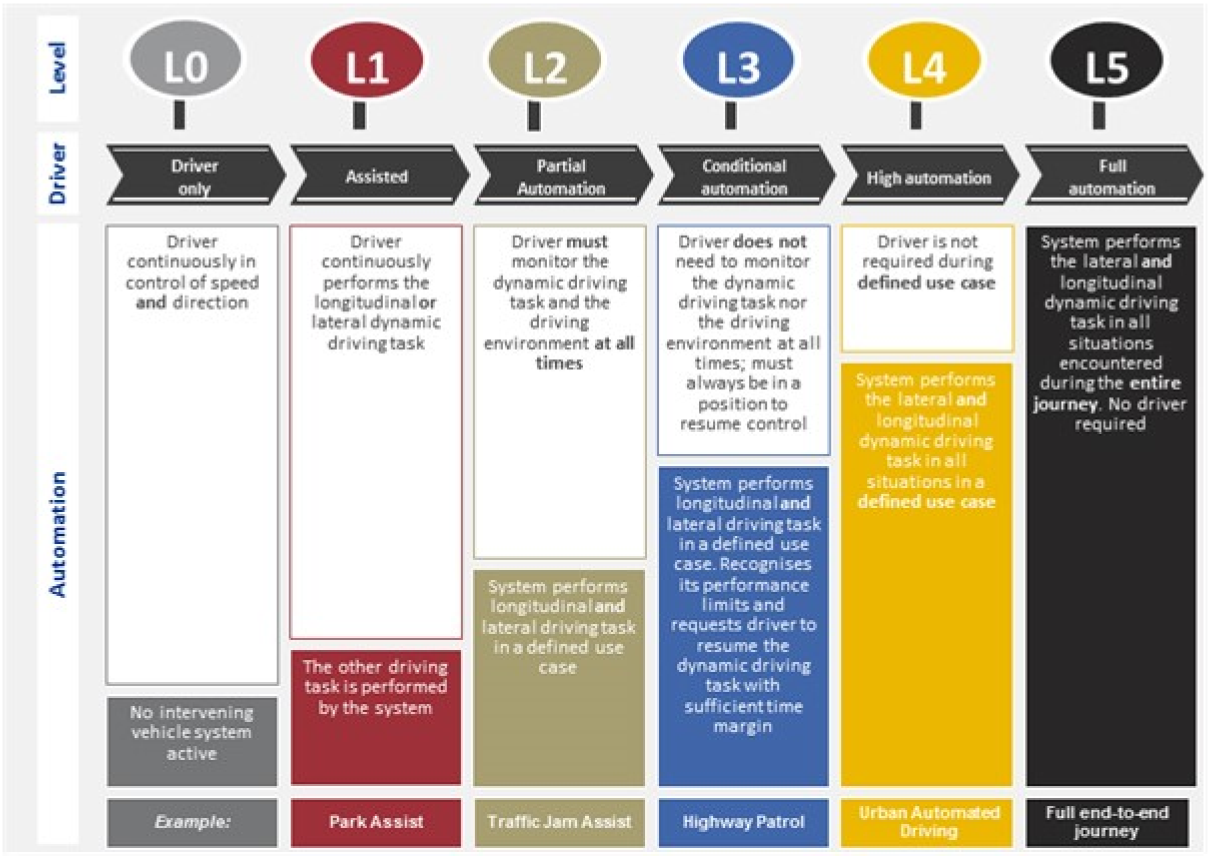
By forcing the relative distance and speed towards the plane origin the control problem can be summarized. This problem is solved in Hatipoglu et al [14] by introducing a "decision region" where different control actions are solved in different regions of the phase plane called decision region in order to predict the driver behaviour and the aim is to design a hybrid control system for the vehicle's longitudinal control in highways which switches between different control actions and the hybrid control switches between smooth-constant acceleration/deceleration and linear control region. The most researched areas of ACC is about the control methodology where longitudinal and lateral control algorithms are developed for the radar based convoying problem by Haskara et al [13] assuming that the convoy consists of only two trucks.

2.2 Autonomous Vehicles

Most of the vehicles now are currently operated by humans. However, advancements in technology like sensors, computing power, data processing and connectivity paved the way for autonomous vehicles where the vehicle is able to perform a transportation task without any human control. In recent years, autonomous driving helps to reduce the number of crashes on our roads and there are many research projects taking place based on AV's. According to World Health Organization (WHO), the death caused by accidents each year are around 1.35 million. Autonomous vehicle driving in a safe manner will help reduce the number of deaths due to accidents. Saving time is another benefit where people with long commutes can start their work while travelling which increases the spare time as well as productivity. Self Driving vehicles are more productive for traffic stream and furthermore there is a possibility of independent cabs and ride sharing which would reduce the requirement for people to purchase their own vehicles.

2.2.1 Connected Autonomous Vehicles

CAV's are vehicles that replace drivers for the driving tasks and brakes, accelerators an steering control are all automatic under the supervision of the driver. CAV's collects and transmits sensor data with different vehicles and roadside infrastructure where enormous number of information is made built upon a digital ecosystem. CAV's can drastically reduce accidents, enhance quality of lives, improving efficiency of transportation systems and creates demand services, new business models and a shared economy. CAV's uses sensors and wireless networks to obtain traffic and other important information whereas the driving control is regulated by any one of the six automation levels. There are six levels of automation used by industry which ranges from zero (complete human driving) to five (full automated navigation without human interaction) as shown in Figure 2.1:



Level
Driver
Automation
L0
L1
L2
L3
L4
L5
Driver only
Assisted
Partial Automation
Conditional automation
High automation
Full automation
Driver continuously in control of speed and direction
Driver continuously performs the longitudinal or lateral dynamic driving task
Driver must monitor the dynamic driving task and the driving environment at all times
Driver does not need to monitor the dynamic driving task nor the driving environment at all times; must always be in a position to resume control
Driver is not required during defined use case
System performs the lateral and longitudinal dynamic driving task in all situations encountered during the entire journey. No driver required
No intervening vehicle system active
The other driving task is performed by the system
System performs longitudinal and lateral driving task in a defined use case
System performs longitudinal and lateral driving task in a defined use case. Recognises its performance limits and requests driver to resume the dynamic driving task with sufficient time margin
System performs the lateral and longitudinal dynamic driving task in all situations in a defined use case
Example:
Park Assist
Traffic Jam Assist
Highway Patrol
Urban Automated Driving
Full end-to-end journey

Figure 2.1: **Level of Automation in Present Vehicles**

The levels are described in detail:

- **Level 0**: Autonomous driving features can provide warnings or limited assistance to driver. When features are engaged, the human driver is responsible for driving actions. Example: lane departure warning
- **Level 1**: Autonomous driving features control steering or acceleration/brake actions to support the driver. When the features are engaged, human driver is responsible for driving actions.

- **Level 2**: autonomous driving features control steering and brake/acceleration actions to support the driver. The driving actions are the responsibility of human driver when the features are engaged. Examples: lane centering and adaptive cruise control.
- **Level 3**: Driving features of autonomous vehicle works only under limited condition and human drivers should take control when required. When the features are engaged, the driving actions are taken by the autonomous system. Example: traffic jam chauffeur
- **Level 4**: autonomous driving features can control the vehicle under limited condition. When the features are engaged, the driving actions are taken by the autonomous system. Example:local driver-less taxi.
- **Level 5**: autonomous driving features can control the vehicle in all conditions. When the features are engaged, the driving actions are taken totally by the autonomous system. Example: drive without a human driver or occupants

The current level of automation found in present vehicles is Level 2 which is Partial automation consisting of adaptive braking and acceleration which adjusts speed without driver's assistance. Road Safety and Traffic management are important aspects of CAVs which reduces congestion. However, CAVs also poses challenges in relation to technology, compatibility, data protection and cyber-security.

2.3 Artificial Intelligence

Artificial Intelligence(AI), which is otherwise called as Machine Intelligence is the insights and knowledge shown by machines. In contrast to natural intelligence, shown by humans and animals, AI is the capability of machines like PCs, robots and other advanced machines which mimics the mental capabilities like learning and solving complex problems. The branch of software engineering, connected with AI, intends to concentrate on rational agents, which is used to analyze and perceive the environment using predefined rules, search algorithms or pattern recognition. Based on the analysis, the agents makes moves and takes decisions which maximize the chances of effectively accomplishing the objectives.

2.3.1 Machine Learning

Machine Learning (ML) is a method of analyzing data that computes and automates analytical model building. ML is a subset of AI which gives the frameworks the capacity to advance naturally and further develop through the experience acquired with less human assistance. Machine Learning algorithms which is constructed numerically based on sample information which are utilized as training data and the algorithms is used for calculation to identify the patterns and make future predictions, based on the information given. Generally, Machine Learning techniques are classified as:

- Supervised Learning learns the task from the labelled data and its main objective is to generalize.
- Unsupervised Learning learns from the unlabeled data and the main goal is to compress.
- Reinforcement Learning, which learns through trial and error method and the main goal is to take an action.

Reinforcement Learning There are variety of scenarios needed to be considered for autonomous vehicle driving and also it is important to ensure that the agent has learned all possible scenarios and safety measures in every situation. Reinforcement learning is used in this kind of situation where the agent collects environmental information in order switch between states based on the defined policy in order to maximize rewards. RL is a type of machine learning where the environment is stated in the form of Markov Decision Process (MDPs) and is applied in many disciplines with its main purpose is to make the agent learn the optimal policy that maximizes the reward function. Mathematical representation of RL using MDP is given in [31] where current state is $s_k \in S$ where k is the discrete time step and S is the set of valid states in an environment and action is defined as $u_k \in A$ where A is the set of all available actions. Reward $R_{k+1} \in R$ is assigned according to the action and state taken by the agent.

Optimal actions are learnt using Reinforcement learning from specific situations where the agent discovers the action with maximum reward signal by the trial and error method and these actions may affect future situations and the reward signals. RL is formulated on the basis of Markov Decision Process(MDP) as given below:

- Time step t
- States set $s \in S$
- Actions set $a \in A$
- Transition function $T(s_t, a_t, s_{t+1})$ which is the probability of an action that leads to state s_{t+1} from s_t, i.e., $P(s_{t+1}|s_t, a_t)$
- Reward function $R(s_t, a_t, s_{t+1})$

The transitions and reward functions are unknown and are learnt during the training process and the decisions are made when agent interacts with the environment in order to achieve goals regardless of uncertainty. Beyond the agent and the environment there are other sub components in RL - Policy, Reward signal, value function, and the environment model. At a given time, the policy explains the behaviour of an agent and maps the observed state of environment to actions. Agents tries to learn the optimal policy (π) in RL which corresponds to the best action for every possible state $\pi^* : S \rightarrow A$. In each time step, reward is sent to an agent by an environment during the transition from one state to another. The working of an Reinforcement Learning model is shown in Figure 2.2

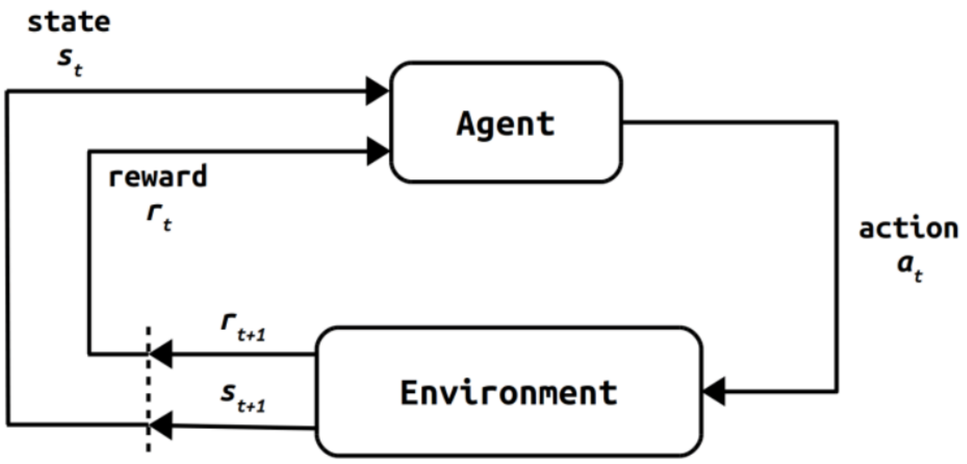


Figure 2.2: **Reinforcement Learning**

RL is a framework communicating with environment where on every time step t, the agent will be on state s_t in state space S from where action a_t is selected from action space A with the help of policy π. When there is a transition from state s_t to s_{t+1} a reward r_t is received by an agent and the reward function according to the environment is represented as $R(s_t, a_t, s_{t+1})$ and transition function as $T(s_t, a_t, s_{t+1})$ and a discount factor of $\gamma \in [0,1]$ is applied to the total reward function which is represented in equation 1

$$R_t = \sum_{t=0}^{\infty} \gamma^t R(s_t) \tag{1}$$

In order to find the policy, value function is used which is given below in equation 2

$$V^{\pi}(S) = R(s, \pi(s)) + E\left[\sum_{t=1}^{\infty} \gamma^t R(s_t, \pi(s_t))\right] \tag{2}$$

Policy π is obtained by deriving the value function in equation 2 and represented below in equation 3

$$\pi(s) = \arg\max_{a \in A} \left[R(s, a) + \gamma \sum_{s' \in S} p(s|s, a) V(s^{'})\right] (3)$$

The Bellman equation is given in equation 4 as follows:

$$V^*(s) = \max_{a \in A} \left[R(s, a) + \gamma \sum_{s' \in S} p(s|s, a) V^*(s^{'})\right] (4)$$

Solving the above non-linear equation for every state s gives an optimized value function from which optimized policy π^* can be calculated

Model-based (vs. Model-free) On/Off Policy methods

Due to safety and cost, interacting with real environment is difficult and learning a model for environment dynamics reduces the amount of interactions with the environment in real time. Agents attempt to learn the reward and transition functions in model based learning which can be used during selection of actions and keeping model approximation of the environment and storing the knowledge dynamics may result in less interactions and with costly environment. Whereas in model free approaches there is no requirement for such knowledge and the learners sample the MDP directly and gains knowledge about the model that is unknown and this can be done in the form of value function estimates as well.

For hyper parameter optimization, comparing the reinforcement models is often infeasible in a practical way. Using on-policy interactions with the target environment will help to get insights of the policy that the agent is implementing from which the performance of these models are evaluated. In Q learning the optimal policy is learnt by the agent using the greedy policy and is called off policy as the updated policy and the behaviour policy is different. Here the future rewards are estimated and the value is added to the new state without following greedy policy. On policy reinforcement learning estimates the policy value that is being followed using SARSA (State-action-reward-state-action). In this algorithm, the agent finds the optimal policy and uses the same for making actions.

2.3.2 Q-Learning

Since there is no prior knowledge of the model in RL, the transitions and reward functions are unknown and the RL agents learn Q values and the policy can be represented using Q values as shown below:

$$\pi(s) = \arg\max_{a \in A} Q(s, a) \tag{5}$$

Q Learning is a model free calculation used to learn the value of an action when it is in a specific state. For any MDP, Q value is used to find the optimal policy by maximizing the total rewards in the successive steps from the current state. Q values are computed for the state action using the below formula:

$$Q(s', a) \leftarrow (1 - \alpha) \cdot Q(s, a) + \alpha \cdot (r + \gamma \cdot \max_{a'} Q(s', a')) \tag{6}$$

Where 'r' is the reward received when moving from state s to s' and reward is given at any particular time step or only at the end of the time step. Learning rate is defined by 'α' for which the values lies between 0 to 1. 'γ' is the discount factor which is used to balance the immediate and future reward. Max function is used to take the maximum of future rewards and is applied to current state reward. Temporal difference method is used in Q Learning where the agent learns from environment through episodes without any prior knowledge on environment.

2.3.3 Double Q Learning

In some stochastic environments Q learning performs very poorly because of overestimation of action values. Since Q learning uses maximum action value as an approximation factor for the future rewards, a positive bias is introduced which results in this overestimation. Another alternative approach is introduced to approximate the highest expected value for a set of random variables and the double estimator method obtained is used to underestimate the maximum expected value than overestimate. To build Double Q learning method, a double estimator to Q learning is used which is a new off policy reinforcement learning algorithm and thus performs well when compared to Q learning where it performs poorly due to overestimation. Two separate Q value estimators are used to update each other and unbiased estimator is obtained where one Q value estimator is used to find the maximizing action and update the other Q estimator and vice versa.

The formula for Double Q learning is shown as below:

$$Q_a(s,a) \leftarrow (1-\alpha) \cdot Q_a(s,a) + \alpha \cdot (r + \gamma \cdot Q_b(s', a*) - Q_a(s,a)) \tag{7}$$

$$Q_b(s,a) \leftarrow (1-\alpha) \cdot Q_b(s,a) + \alpha \cdot (r + \gamma \cdot Q_a(s', b*) - Q_b(s,a)) \tag{8}$$

Pseudocode of Double Q learning is shown below:

Algorithm: Double DQN Learning

```
1:Initialize Q values $Q^A, Q^B, s$
```

```
2:repeat
3:  Choose an action a, from $Q^A(s, .)$ and $Q^B(s, .)$, observe r, $s^{'}$
4:  Choose an update randomly either UPDATE(A) or UPDATE(B)
5:  if chosen UPDATE(A) then
6:    Define $a^* = \arg\max_a Q^A(s^{',a)}$
7:    $Q^A(s, a) \leftarrow Q^A(s, a) + \alpha(s, a)(r + \gamma Q^B(s^{'}, a^*) - Q^A(s, a))$
8:  else if chosen UPDATE(B) then
9:    Define $b^* = \arg\max_a Q^B(s^{',a)}$
10:    $Q^B(s, a) \leftarrow Q^B(s, a) + \alpha(s, a)(r + \gamma Q^A(s^{'}, b^*) - Q^B(s, a))$
11:  end if
12:  s $\leftarrow$ s$^{'}$
13:until end
```

2.4 Deep Learning

Deep Learning is implemented using Machine Learning and is based on artificial neural networks, applied math, statistics and mimics human brain in the biological neural networks. In the last few years, there is a huge growth on the applications of Deep Learning which helped in the development of more powerful computers, used for large number of datasets and training deep neural networks. In order to introduce systems that learn and act in a same way as human brains "Deep Learning" is introduced. Neurons are the fundamental building blocks of the human brain whereas deep learning architecture contains a computation unit "perceptron" which allows modelling of non linear function and it aims to understand the represented data. Multiple processing layers are used in the computation unit in order to understand the patterns of data by using multiple levels of abstraction. These artificial neural networks (layers) are utilized in Deep Learning method which is obtained by non linear modules and these layers are not built by human but learnt directly from the data using learning mechanism.A DL architecture consists of simple modules in a multilayer stack which is subject to learning and computation of non linear input-output mappings. DL is used in large datasets in order to find complex structures by using backpropogation algorithm where internal parameters of layers are changed based on the previous layers. Image, video, audio and speech recognition are done using Deep Convolutional Networks in order to find patterns of the data by processing the structured array of data (audio, video, etc).

2.4.1 Backpropagation

Backpropagation is used to calculate derivatives in the feedforward deep neural networks and calculates the gradient of the error function. It can be used as a function of neural networks for training the artificial neural network using gradient descent algorithms. Main features of Backpropogation are iterative, recursive, and efficient method to calculate the weight of the neural networks to improve to improve the training task. As long as the modules are smooth functions of the inputs and thir internal weights are computed using backpropagation procedures which is a practical application of the chain rule of derivatives. The function derivative with respect to input can be computed using the output gradient in a backward manner with respect to the module output or subsequent module's input. In order to spread the gradients to all modules, backpropagation method is applied continuously. Feedforward neural networks can learn to map a fixed size to fixed output. Weighted sum of inputs from the previous layer is computed by a set of units from the previous layer and the result is passed through the non linear function to the next layer. The most commonly used non-linear activation function in neural networks are Rectified Linear Unit (ReLU). Hidden units are referred to those which are not in both input and output units and the hidden layer deceives the input in a non linear manner so that the categories become linearly separable by the last layer.

2.4.2 Optimization Algorithms

The most commonly used algorithms for neural networks is the gradient descent approach which is used to minimize the cost function J(θ) where the parameters are updated in the opposite direction of the cost function $\nabla\theta J(\theta)$. The commonly used optimization algorithms in Deep Learning are Momentum, Adagrad, ADAM, Adadelta and RMSPRop.

Adagrad

In [11] Adagrad is proposed as a method which adaptively updates parameters based on squared sum of gradients per parameter. Learning rate is normalized before the update of each parameter i using the below formula:

$$G_i^t = G_i^{t-1} + \Big(\frac{\delta J(\theta)}{\delta \sigma j^{t-1}}\Big)^2 (9)$$

$$\sigma j^t = \sigma j^{t-1} - \frac{\alpha}{G_i^t + \epsilon} + \frac{\delta J(\theta)\sigma}{\delta \sigma j^{t-1}} (10)$$

where epsilon ϵ is used to avoid division by zero. Based on the past updates the learning rate for each parameter is assigned. Adagrad is used to tune the lear remove the extensive learning rate tuning by dividing the learning rate by the sum of past square gradients.

RMSProp

RMSProp and Adadelta has been developed in an independent manner to solve the diminishing learning rates of Adagrad. RMSProp is developed to solve the problem by using an exponential decaying average of squared gradients.

$$G_i^t = \gamma G_i^{t-1} + (1-\gamma)\Big(\frac{\delta J(\theta)}{\delta \sigma j^{t-1}}\Big)^2 (11)$$

where the recommended value of γ is 0.9.

ADAM

In [17] Adam algorithm was developed by combining RMSProp and Adagrad with a new Momentum implementation. Adam uses a decaying average of squared gradients and past gradients as shown below:

$$m^t = \beta_1 . m^(t-1) + (1-\beta_1).\nabla J(\theta) \tag{12}$$

$$v^t = \beta_2.v^(t-1) + (1-\beta_2).\nabla J(\theta)^2 \tag{13}$$

where m^t and v^t are the mean and variance of the gradients and β_1 and β_2 are the hyperparameters. m^t and v^t are initialized to zeroes so this causes a biased towards zero during the initial time steps and when the hyperparameters are close to 1 causing lower decay rates and this issue is solved by updating the value of θ_t

2.5 Double Deep Q Learning

As seen in chapter , the main goal of the Double Q method is to eliminate or reduce overestimation by deteriorating the max operation in the target for action selection and evaluation. A double Q learning method implemented with Deep Learning is known as Double DQN. Double DQN has two neural networks one with target network and another with Deep Q Network. Deep Q Network is used to obtain the best action which has the highest Q value of the future state and target network is used to calculate the Q value estimated with the action selected above.

Step 1 : Action 'a' selected by Deep Q Network is given as

$$a = \max_a Q_{qnet}(s_{t+1}, a) \tag{14}$$

Step 2: Target network used to calculate the Q value estimated with action 'a' selected from the Deep Q Network is given as

$$q_{estimated} = Q_{tnet}(s_{t+1}, a) \tag{15}$$

Step 3: Based on the estimated Q value from the above target network the Deep Q Network's Q value is updated as given below:

$$Q_{qnet}(s_t, a_t) \leftarrow R_{t+1} + \gamma Q_{tnet}(s_{t+1}, a) \tag{16}$$

Step 4: The target network's parameters are updated based on Deep Q Network parameters per several iterations.

$$Q_{tnet}(s, a) = Q_{qnet}(s, a) \tag{17}$$

Step 5: The Deep Q Network parameters are updated using Adam optimizer

2.6 Sensors

As discussed in previous section, autonomous driving system depends mainly on three practical blocks:

- Perception: Surrouding environment is sensed and the representation of the environment is defined.
- Decision Making: The environment is analyzed and an appropriate action is chosen.
- Actuation: Setup the action in place

Perception involves discussing about sensors which is mainly of two forms:

- Proprioceptive Sensors which is responsible for sensing the vehicle's state like wheel encoders, inertial measurement unit etc.)
- Exteroceptive Sensors which is reponsible for sensing ambient surrounding like cameras, LiDAR, RADARs, ultrasonic, etc.)

Extroceptive sensors are the main in independent driving since it is utilized to explain an environment in a detailed manner sufficient permit right choice in the following stages. The principal sensors associated with our independent driving is described in the following sections.

2.6.1 RADAR

The principal of operation for both RADAR and LiDAR are almost same where LiDAR uses light waves and RADAR uses radio waves. Distance, velocity, and angle of the obstacle in the surrounding of the agent is calculated using the time taken by the radio waves returning from the obstacle to the device. RADAR is of low cost, small size and less power consumption. Measurements from RADAR are produced even in bad weather conditions like fog, rain, poor light conditions and even at night. The disadvantage of RADAR is it cannot provide the shape of the object and its classification due to less resolution and clutter.

RADAR in Autonomous vehicles

In autonomous vehicles RADAR technology operates with millimeter waves and offers millimeter precision. RADAR ensures high resolution in detecting obstacle and movement with the help of millimeter waves in autonomous vehicles. When compared to other sensors, RADAR work in low visibility conditions as well such as cloudy weather, snow, rain and fog in a reliable manner. The use of RADAR is common in the field of automotive sector mainly for adaptive cruise control, highway assist and other automation levels in driving. Table 2.1 shows range of sensors used in autonomous vehicles with its applications.

Range	**Frequency(GHz)**	**Bandwidth(GHz)**	**Application**
Short	24	5	Blind-spot detection
	79	4	Stop and Go Assist
Mid	24	0.2	Collision avoidance
	77	1	Adaptive Cruise control
	79	4	Stop and Go Assist
Long	77	1	Adaptive Cruise Control

Table 2.1: **Different specifications of RADAR technology**

2.6.2 LIDAR

To gain information on surrounding environment it is necessary to detect the presence of obstacles like vehicles, pedestrians, buildings, vegetation, etc. Sensors are used to capture the data and acts as a perception system in autonomous vehicles and is of two types namely active and passive sensors. LiDAR's emit lasers and illuminate the surroundings and are active sensors. By processing the received laser from the reflecting surfaces, ranges are measured.

LIDAR in Autonomous vehicles

Since there is good progression with camera based sensors from which image processing are done to calculate distances and much more, it encountered difficulties when distances are calculated for cross traffic entities. So, LiDAR's overcome this issue and currently it is used in many high level autonomous vehicles despite its moving parts and high costs. In Autonomous vehicles, LiDAR's are mainly used for localization and perception. A perception system provides representation of the environment around the vehicle which is can be interpreted by machine. Three levels of information can be obtained from the output of the perception system which is given below:

- **Physical description**
- **Semantic description**
- **Intention Prediction**

Thus LiDAR outputs are mainly used in object detection, tracking, classification,etc and the ranging accuracy provides physical information which is highly reliable. LiDAR requires mapping of the environment which contains the visualization of all the objects near the vehicle and the density computed by 2D and 3D point cloud histograms are used to represent the map.It provides many advantages over camera sensor in areas of AV safety and mapping the environment. When scanning using LiDAR, there are two ways of viewing the collected data
(1) Bird's eyes view which is an elevated view from certain height and can be encoded using height, intensity and density.
(2) Front view which resembles the perspective of human eye.

2.6.3 CAMERA

The cameras just like human being's eyes will help the robot to see the world. It provides images and have various size and resolutions for different purposes. In autonomous vehicles, it is mostly used to train neural networks to find obstacles.The main limitation of cameras are it does not provide useful information when the weather condition is snow or rain and also when the lighting is poor during bright and dark lighting. Some self driving vehicles use camera only for navigation and obstacle detection because camera's does not provide accurate distance information as LiDAR's and RADARS do.

2.7 Summary

Considering the Autonomous vehicle architecture, the vehicle control and navigation functionality is implemented by the control layer. CARLA simulator is chosen as it provides the user the actual odometry of the vehicle and the ground truth of the vehicle which makes easier to evaluate the proposal performance. As a fundamental autonomous driving requirement, LiDAR is used for 3D object detection and depth estimation with more accuracy.

Cruise control is a driver assistance system which takes over the vehicle's throttle to maintain the speed set by the driver in an automated way but an ACC technique automatically adjusts the longitudinal vehicle speed in order to maintain the safe distance from the preceding vehicle. Three major components of an ACC system are: Perception, Planning and Actuation. Here RADAR and LiDAR will be used in the ACC system to find the depth of the environment. The longitudinal control algorithm is defined in the planning phase and it calculates the reference acceleration profile in accordance with the perception data. By converting this acceleration into throttle and brake, the low level controller actuates the actions for driving

3 State of Art

Engineers, computer scientists, and applied mathematicians believe that the last two decades belonged to the world of internet and communication and current decade would belong to artificial intelligence (AI) [35]. Machine Learning (ML) and Deep Learning (DL) which is a subset of AI has also opened large areas of unprecedented research. In order to understand the rationale behind developments made on "Adaptive Cruise Control using Deep Learning", it is vital to have an initial foundation of the empowering influences that have moved around the technological improvements tracked down today in current vehicles and progressed models.

The primary motivation which moved the realisation of this thesis is the need of assessing the potential of sensor technology approaches applied to the adaptive cruise control of the autonomous vehicle.

From previous works it can be seen that Deep Reinforcement learning with Q learning approach is used for modelling Adaptive Cruise Control for safety, comfort and energy consumption purposes. In the first section we discuss the concept of Reinforcement learning with its essential for complex problems and the most important techniques, such as Q Learning and Double Q learning used in previous work. In the second section, the Simulation techniques used for training and the advantages and disadvantages of using CARLA simulator is discussed. The next section describes the proposed method Double DQN using CARLA simulator with RADAR and LiDAR sensors.

3.1 Recent Advancements in Self Driving Vehicles

In order to develop an autonomous vehicle, the analysis of the behaviour and strategies which is frequently employed by human drivers in a safe manner is very essential [5]. There are number of researches taking place to extract the strategies of a human driver and to model those strategies and behaviour without any human intervention in different situations [5]. In order to model the driving behaviour most of the ML algorithms used in the previous decade are predictive control, fuzzy logic, adaptive control and hybrid dynamic models and Markov chain models. The Model Predictive Control successfully predicted the driving behaviours of human and also reproduced it under constrained environments [21]. This method mostly relied on data collection and representation with prior knowledge of rule base. The Markov Chain model accuracy mainly relies upon the earlier information about the state change likelihood between each state pairs [8]. With the emergence of Deep Learning which tries to learn human driving behaviour from the driving data has gained importance even without any prior knowledge of the conditions of driving [36].

3.1.1 Advantages of Autonomous Cars

Information and Communication Technology (ICT) keeps on tracking applications in transportation to avoid collisions, decrease pollution and increase public transportation with new developments with resource sharing [32]. Due to drunk and drive, distracted and drowsy driving there are around 1.3 million deaths due to accidents every year [24]. In order to avoid the human driving accidents, AI systems can be used to save lives with the following advantages which motivates the self driving research:

- The autonomous cars can be programmed to drive safely with less fuel consumption, less carbon dioxide emissions, enhanced user productivity. It is also used to find blind spots, and to follow speed limits [32]
- Enhanced capacity of roadways, less road casualties, and less on-road driving accidents with the use of self driving cars

- Less accidents are benefits for student and elder community people by making them feel more comfortable towards autonomous vehicles. It also supports greener mode of transport with less noise pollution and increased mobility [24]
- In the current driving environment, cars are parked for longer time but with self driving cars parking lots can also be transformed to parks [30]
- Autonomous cars can be equipped with improved way of scheduling, routing, and to provide best routes to improve travel times with lower cost of travelling from one point to other [3]
- Shared access can be wide spread with the use of self driving cars even if the ownership of the car is eliminated. Transportation can be personalized, efficient and reliable [35]

3.1.2 Drawbacks of Self Driving Cars

One of the widespread and readily available mode of transportation are Cars. Driving is a safer undertaking even if technology has developed safer cars [5]. There are some disadvantages of self driving cars which is mentioned below:

- Due to self driving cars, job loss to many people who are employed in transportation sector may increase [32]
- Acceptance of self driving cars is a fundamental research problem in psychological and cognitive science and is also argued that the person will not die or suffer injuries even if there is a failure in AI systems [24]
- In unmapped areas, self driving cars cannot be used because it uses Global Positioning Systems (GPS) for localization [23]
- Self driving cars are susceptible to hacking as the car is online all the times and the safety and privacy is not guaranteed as their moves can be tracked [12]
- Driving at convergences without traffic signals, breaking down traffic signals, uncontrolled convergences, occupied crossing points, places with humans in proximity are a challenge for self-driving vehicles [3]

3.2 Vehicle Control Schemes for Automation

The most researched area in autonomous driving is the control methodology. Once if the required information is gathered to understand the vehicle's state with respect to the road and other vehicles, a control scheme is required to assist the driver to control the vehicle or to autonomously control the vehicle by itself. In order to determine the required kinematics of the vehicle, a higher level controller is required where in driver assist vehicles, the kinematics required would be compared with the performance of the driver and a warning is given if necessary. When it comes to more "autonomous" systems, the desired motion of vehicle to control the engine, brakes and steering are determined by higher level controllers. So, the higher level controller needs a good understanding about the vehicle environment and a suitable model of the vehicle is required by lower-level controllers. Lots of studies are focused on longitudinal vehicle control which is the base for ACC, car platooning, forward collision, etc. Raza and Ioannou [28] developed a well organized control plan for vehicle longitudinal control in various various operation modes. This supervisory controller gathers inputs from the driver, road infrastructure, vehicles, and the locally available sensors and processes them, and sends the control signals to the brake and choke. ACC and platooning are both similar vehicle following modes where in car platoons the main goal is to maintain close space between the vehicles whereas the main objective in ACC is to maintain a safe distance to get the driver out from spacing adjustments. The vehicle's acceleration in a platoon is determined to ensure the string stability of the vehicle that closely follow each other so that the spacing error between vehicles do not grow towards the end of platoon [29].

Developing an effective reinforcement learning techniques for the longitudinal control has become the major goal in many recent researches.

3.2.1 Classical Control Approach without Machine Learning

Proportional Derivative(PD)and Proportianal Integral Derivative(PID) controllers are the most commonly used approach in Adaptive Cruise Control systems in earlier. [16] discusses the use of PID controllers in engine or brake to obtain the speed and acceleration reference of the follower vehicle with a reference distance to the leading vehicle. The two non-model based approaches in this paper are compared in simulation for stop and go application where efficiency is high when using PID controller in terms of performance and computation cost. Single PID controller for accelerator/brake is used in [2] to control the car that is following another vehicle and a better performance is seen with extensive simulations. It takes input as position error which is the difference between the safety distance and the distance of the car ahead. Model Predictive Controller (MPC) is developed for an autonomous vehicle steering and ACC control and is compared to classical feedback controller with PID in [9]. The MPC framework is given below in Figure 3.1. It is seen that PID controller operates well when there is a single input and single output but MPC is capable of handling high level models with multi input and outputs. The overall performance of MPC controller is better than that of PID in terms of accuracy, and comfort by avoiding jerks.

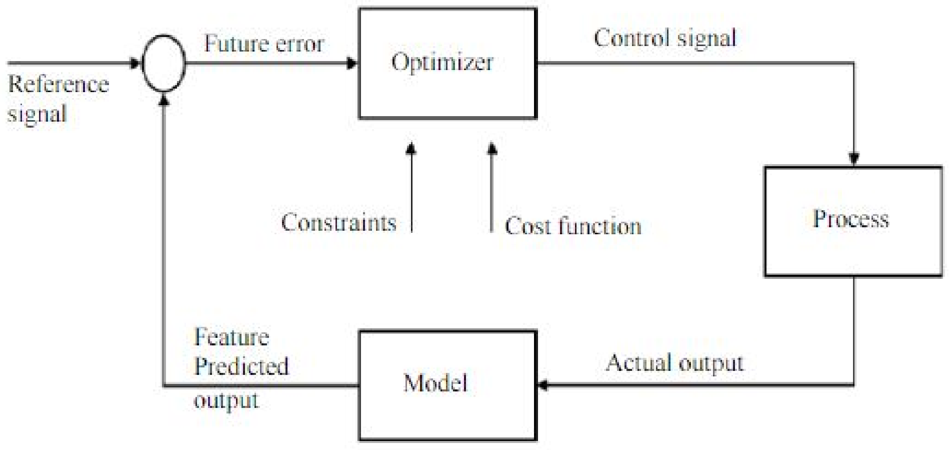


Figure 3.1: **Model Predictive Controller [9]**

3.2.2 Classical Control Approach using Reinforcement Learning

Interest in Machine learning has grown up in recent years with the growth of parallel computing technology and enormous amount of data. Researchers investigate the applications of machine learning and deep neural networks in autonomous vehicles by applying deep learning methods to autonomous driving from vision based models and an end-to-end approach is studied of how the vehicles learns the mapping and applying control actions by sensing the environment.

As DNN is adopted, Reinforcement learning has also been improved significantly and the technique is called as Deep Reinforcement Learning which is known to perform well in robotics and control problems. Deep Q network was proposed where Q value is approximated using neural networks. It is demonstrated that DQN performs better compared to human experts in different Atari games and this technique is recently used in Autonomous vehicles [15], [10]. In [10] the author concentrates on implementing vehicle to vehicle communication to improve ACC system so that the current speed and acceleration can be transmitted to the following vehicle by inter vehicle communication. Machine learning techniques are used to design the autonomous vehicle controllers for the secured longitudinal following of a front vehicle and gradient descent method is used to optimise the performance by directly modifying the control policy. For an autonomous system, information is provided to the vehicle control system by sensors of the vehicle and for cooperative systems require communications with other vehicles or with transportation infrastructure is required.

Most of the projects in CACC used classical control theory to develop autonomous controllers and one of the first research to use machine learning to control autonomous vehicles was Pomerleau's autonomous land vehicle in a neural network (ALVINN) [26] which consists of computer vision system based on neural network.

DRL approach is used in [10] controls CACC using Reinforcement Learning (RL), in RL the agent (vehicle) is a learner which observes the state S (S = finite set of states) and it takes action a depending on the state assuming the action set of an agent to be finite. In these conditions after taking an action the agent receives reinforcement signal under a real value form depending on the action taken and the successor state. Import aspects of the policy gradient method is discussed and this approach is shown to be efficient for the resolution of autonomous longitudinal vehicle control problem.

Autonomous Braking system using Deep Reinforcement learning is introduced in [7] where the system decides automatically decides to apply brake or not when a collision event is noticed with the help of sensors. The designing of a brake control problem is defined as finding out an optimal policy in Markov decision process (MDP) where the obstacle's position, speed of the vehicle is used to obtain the state and action space is obtained by a set of braking actions. Here, the policy used for control actions is learnt through simulations with Deep Q learning approach and reward function is proposed to overcome the obstacle and the another reward is given when the agent runs out of risk in order to obtain efficient braking policy. The DRL braking system model is evaluated for different positive and negative scenarios by controlling the vehicle velocity when there is a pedestrian crossing the road, and no action taken when there is an event of collision. In [22], Learning Based Cooperative Adaptive Cruise control is implemented to avoid traffic congestion and accidents. When compared to classical controllers, it may be difficult to guarantee model free DRL's robustness but the advantages include: - As DRL agent can directly learn from a real system or using simulation there is no need to create a mathematical model of a system

- All the relevant information is fed into the agent's state vector and the systems constraints are learned by agent's neural network

- Deployment of trained neural network of an agent is computationally efficient

In [6], authors investigated and compared DRL to that of MPC for Cruise Control for predictive velocity tracking and was shown to achieve similar performance but with less (70-fold lower) computational effort and in [18] authors compared DRL to MPC for ACC model and found DRL model is performing better than MPC model. Also, DRL provided lower costs with combined penalized error, control signal amplitude, and vehicle jerk when compared to MPC due to its generalization capabilities. A new ACC concept which always guarantee safety and comfort is implemented in [20] assuming autonomous vehicles take over the driving duties from humans completely when the leading vehicle suddenly decelerates. The main contribution is to design a nominal controller which is supervised by emergency controller which is used to guarantee safety and comfort.

Designing a Reward Function

The reward function is used to describe how the agent (vehicle) is "ought" to behave and is basically a feedback from the environment which measures the success or failure of an agent's action. Choosing the reward function is an important task in reinforcement learning where the good reward function maximizes the desired behaviour intended. Reward function is a mapping from state and action to a scalar value which is called "reward". Higher the value is the better. The agent learns the "policy" which is mapping from state to action. For every given state, the policy learns what is the best action to take in the given state in order to maximize the reward. Episode is a sequence of (SARS - State, Action, Reward, new State) transitions and the episode ends if it reaches any terminal condition. The agent learns to alter its behaviour according to the reward received due to its actions. In [7] desirable reward function is found to obtain the balance between the punishment given to the agent when there is an accident and the reward gained when the agent overcomes the collision as soon as possible. Here the agent gets the value of states from the environment and the reward value is obtained for the action took where the main purpose of the proposed system is to increase the future reward values. Early braking of an agent is prevented by reward function using a penalty value that is directly proportional to square of distance between the vehicles and obstacle so that it helps the agent to avoid deceleration if the pedestrian is far from the agent and there is a penalty the agent receives when there is a collision event. In [22] where string stability constraint is considered, and the main control purpose is of maintaining a safe distance while reducing air drag is fed into ACC environment through the use of reward function and is defined in such a way that it was always negative. Reward function consists of weighted and normalized error, power consumption and change of action when reward is created and these terms contribute to the reward achieved when the episode is not aborted. Due to error and safety constraints several conditions led to the end of an episode. When reward function is maximized the desired behaviour is obtained with minimum error and power consumption and enhancing the riding comfort by providing smooth acceleration signal. Here two sets of reward function is defined for error minimizing and absolute power consumption aimed at reducing the energy consumed by performing less acceleration and deceleration at the end of an episode. A non-quadratic reward function is used in [6] instead of a quadratic function in order to reduce the steady state error of the resulting policy where the main concept of the paper is to introduce model free learning approach using Deep Reinforcement learning in automatic vehicles replacing Model Predictive approaches.

In [31], reward function appropriately designed by system engineer in ACC in order to ensure its reliability. Here the reward function is developed in such a way that balances two conflicting objectives – collision should be always avoided and the vehicle should maintain a high speed on highways. The reward function consists of terms including incremental travel distance along the path of the ego vehicle and the reward of the action taken at time t which encourages the vehicle to drive as fast as possible within the speed limit to achieve higher reward and to penalize frequent speed changes. Weight parameters are used to control the trade off between two objectives. The reward function defines in which way the agent should proceed. For example, bigger reward is defined if the car is in the middle of the road than the in the side of the road.

3.2.3 Controllers using DDQN in Autonomous Vehicles

Double Deep Q learning method is used in [4] for autonomous vehicle navigation and obstacle avoidance based on RNN in an unknown dynamic environment. The preparation strategies of Double Deep Q training, a reinforcement learning approach empowers the specialist to gain proficiency with the route choices successfully and the model is tested in gaming environment which mimics the real world environment. A few dynamic techniques have been performed utilizing Deep Q learning approach but to stay away from overestimations Double Q learning is presented here. The framework for DDQN doesn't reduce the overoptimistic assessment, but gives favored execution when contrasted with DQN strategy on a virtual accustom. DDQN shown in Figure 3.2 also gives accurate results even in the case of randomness. Here, four decisions or actions are considered for different scenarios. Hence, the reaction and choice of an agent are classified into accelerate, decelerate, lane change, and stop modes. If the agent does not predict any obstacle at some distance, then it will be in acceleration mode. If there is no hindrance on near the agent on the left or right side, then lane change option will be available. If obstacles are found on both sides of the agent, then deceleration mode is initiated. Finally, exploration is promptly halted in the event of any collisions. In [34], vehicle's speed is controlled using Double Deep Q learning method. Data is obtained from naturalistic driving where high dimensional video data are processed into meaningful data and the Double DQN calculates Q value estimates for speed control actions. One major characteristic of naturalistic driving data is that it is collected in natural state of normal driving condition without interference and experimenters.

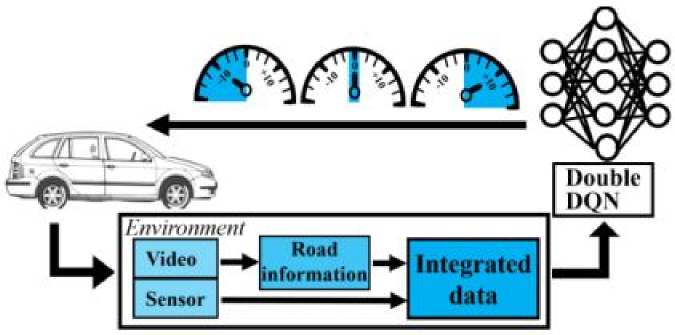


Figure 3.2: **Architecture of Double DQN system [34]**

Under environment constructed by naturalistic driving data, reinforcement learning algorithm could learn human's decision-making and judging ability. Human-like behavior appears more promising when dealing with changeable real environment.The data is trained by letting the vehicle agent drive on the data recorded in real environment using three different optimization algorithms - gradient descent, Adam and RMSProp. Double DQN method appeared to be more robust to this more challenging evaluation and had improvements both in terms of value, accuracy and policy quality. In [33] DDQN method is used where reward function considering the motion characteristics of the preceding car is implemented as the adaptive cruise control system fails to follow the preceding car with extreme motion in most of the cases. In [25] two methods are compared for end to end autonomous driving - DQN and Duelling Deep Double DQN method. The main contributions of this thesis is as follows:

1. An end-to-end autonomous driving method through Dueling Double Deep Q-Network on TORCS. The state space, action space and reward function and implement RL algorithm is designed to realize end-to-end autonomous driving.

2. A mixed state input that comprises both camera image and a vector of ego vehicle speed. The corresponding dueling network architecture is designed, which includes both CNN and fully connected layers.

3. Visualization of neural network for end-to-end autonomous driving. The visualization map of the learned neural network shows that the vehicle drives by observing the lane lines.

Table 3.1 explains the existing research methods for Adaptive Cruise Control using Reinforcement Learning. Algorithm used, the neural network architecture, and the type of state, action and reward function used in various research projects.

Research	Algorithm	Architecture	State	Action	Reward
[10]	Policy Gradient	Backpropogation neural network	Headway, headway derivative, front vehicle acceleration	Braking, Gas action, no-op action	Safe inter-vehicle distance
[7]	DRL	Fully connected feed forward network	Relative position of obstacle, vehicle speed	Braking actions - null, mid, strong	Balancing the damage imposed to obstacle when accident occurs
[6]	DDPG	DNN	Vehicle speed	taking actions from different policy	Non quadratic reward function to reduce steady state error of policy
[31]	DQN	DNN	Relative position velocity of the vehicle	Heavy, slight acceleration, keep speed and slight, heavy deceleration	Avoiding Collision, Maintain high speed on highways
[34]	DQN	DNN	Vehicle Speed	Acceleration, Deceleration	Reward network to map each state to scalar

Table 3.1: **Existing Research in Deep Reinforcement Learning for Adaptive cruise control**

3.3 Simulation and Training of DQN

To validate a proposed controller, real data from a driving scenario is taken as a reference to feed a simulation. In [7], PreScan software is used for simulation to model vehicle dynamics in real time which is used to train the DQN agent by randomly simulating the pedestrian behaviour. Noise is added to the system to make it more practical. During the training either of the two scenarios (cross the road or stay at initial position) is selected with equal probability. Here the neural network used for DQN comprises of fully connected layers with five hidden layers and a learning rate of 0.0005 is used to minimize the loss with the use of RMSProp algorithm. In [22], each CACC environment was initialized with processed NGSIM (Next Generation Simulation) velocity and acceleration trajectory and the trajectories were sampled from NGSIM training dataset. In NGSIM dataset, position of vehicles were measured using cameras at two to three specific times in a day which helped to portray different amount of traffic and different vehicle trajectories during the build-up congestion and transition between congested and uncongested traffic. The DRL environment here is created based on python based DRL framework using Open AI Gym and the Modelica model of the vehicle was exported as Functional Mockup Unit (FMU). PySimulator was used to connect to FMU and to create a python based interface. In [31], OpenAI-Gym and ROS/Gazebo is used in order to model the highway environment in real time which helps to train the DQN by simulating the behaviour of vehicles on highway. RMSProp algorithm is used here as well to minimize the loss with learning rate = 0.0005 and replay memory size to 10,000 and replay batch size to 32. After training the agents performed good enough and is able to stay in the lane adjusting its speed and choosing a good time to lane changes. The most difficult part in [31] is mentioned that it was extremely hard to stabilize reinforcement learning with non linear function approximators and tuning of hyper parameters required plenty of work. In order to validate the proposed controller in [6], real data from driving scenario was taken as a reference to feed simulation and the longitudinal vehicle dynamics model is implemented using OpenAI gym environment in Python. The learning performance was evaluated after training a DDPG agent with reference and advance knowledge information from the real world dataset. Learning performance is evaluated by training a DDPG agent and performed training runs on APRBS sequences and finally they were able to demonstrate that Deep Reinforcement learning has a potential to be used for predictive tracking control which incorporates advance knowledge of disturbances. Also, once DRL based controller is trained it has low computing time when compared to

NMPC (Non Linear Model Predictive Controller) achieving close to optimal performance. Main challenges faced are high variance between different training runs, performing time consuming evaluation runs, choosing best policy. In [10], microscopic vehicle simulator is built in order to design an autonomous longitudinal vehicle controller using RL. The three primary necessities to construct a testing simulator system are behaviour of vehicles ought to be essentially as close as real vehicles, adaptability to insert RL motor, run quicker than real time in order to execute enormous number of episodes in simulation. In [33] longitudinal control of vehicle using Double Deep Q learning is implemented. The training is done with the images taken by the rear car camera, rear car speed acceleration, distance between rear car and front car and the simulation is done using unity environment.In [25] TORCS(The Open Racing Car Simulator) is used for autonomous driving research where it provides users with vehicle dynamic model, various tracks and graphics which allows them to train and test the autonomous driving network on an end to end basis. TORCS also has an inbuilt engine with inputs such as steering angle, velocity and acceleration along with outputs such as camera image, vehicle position and velocity. Table 3.2 summarizes the different applications of Deep Reinforcement learning used in previous research and the corresponding simulation environment used.

3.4 Summary

This chapter explains the main techniques used in previous studies related to Autonomous vehicles and Adaptive Cruise Controllers. State of art methods using classical controllers and Deep Reinforcement Learning and its comparison with respect to performance, comfort and efficiency are stated. From the above research papers, we can see the implementation of ACC most commonly using vision based sensors and combination of all sensors (camera, RADAR and LiDAR) using DQN or DDPG methods. Thus, we formulate the proposal with the use of RADAR and LiDAR based Adaptive Cruise Control using Double DQN algorithm. The chapter 4 explains the design and implementation of the proposed idea and its key components.

Research	Applications	Simulated Environment
[31]	Autonomous Driving system for adjusting the speed and changing the lane	OpenAI-Gym and ROS / Gazebo
[7]	Designing Brake Control using DRL – Deceleration as the vehicle gets close to the obstacle	PreScan – Modelling vehicle dynamics in real time
[22]	Application of DRL to CACC with continuous action and state space - maintaining safe distance, close enough to preceding vehicle and reducing air drag -Reducing vehicle jerk and Drive in string stable way	Python-based DRL framework OpenAI Gym
[20]	DRL for low level longitudinal vehicle control and Predictive RL controller to incorporate information about reference trajectories and future disturbances	Data is collected using Next Generation SIMulation (NGSIM)
[20]	ACC with different time gap policies – CTG and VTG	SUMO
[25]	Double Q learning used to control the speed of the vehicle based on the naturalistic driving data environment	TORCS is used for training Convnets and recurrent neural networks
[19]	Double Q learning used to improve the performance of autonomous driving	CARLA
[27]	DRL using DQN and DDPG to train control layer of autonomous vechicles	CARLA

Table 3.2: **Previous work on Autonomous Vehicles using Reinforcement Learning and the respective Simulation Environment used**

4 Design and Implementation

In this paper, we use DDQN as our working DRL algorithm to obtain the Adaptive Cruise Control technique. This section explains the DDQN algorithm applied to Adaptive Cruise Control and includes the design of DDQN and the techniques applied to train the agents in order to achieve the collaborative environment. It also includes state representation, action space and reward functions and besides that, it presents and justifies the design decisions taken when selecting DDQN techniques and RL components.

4.1 Proposed Method for Adaptive Cruise Controller

Deep Q learning method is extensively used in training adaptive cruise control and in applications of autonomous vehicles. DQN network introduces two separate networks - one parameter is used to make decision while interacting which is the Q network and another network is used to form the target when training and is called the target network. The two networks working together are proved to make learning more stable. In order to neglect the overestimation created by max argument in reinforcement learning as discussed above, Double Deep Q learning method is used to train the adaptive cruise control.

So far, in the previous studies, Double DQN method is used to train autonomous vehicles with the use of camera images and mixed state space with vehicle speed vectors and camera images. To evaluate the performance of the Double DQN method for Adaptive cruise control using RADAR and LiDAR based sensors when compared to vision-based sensors the first step is to do model design which are explained in below sections. Figure 4.1 represents the leading and ego vehicles in CARLA simulation which will be used for training ACC.

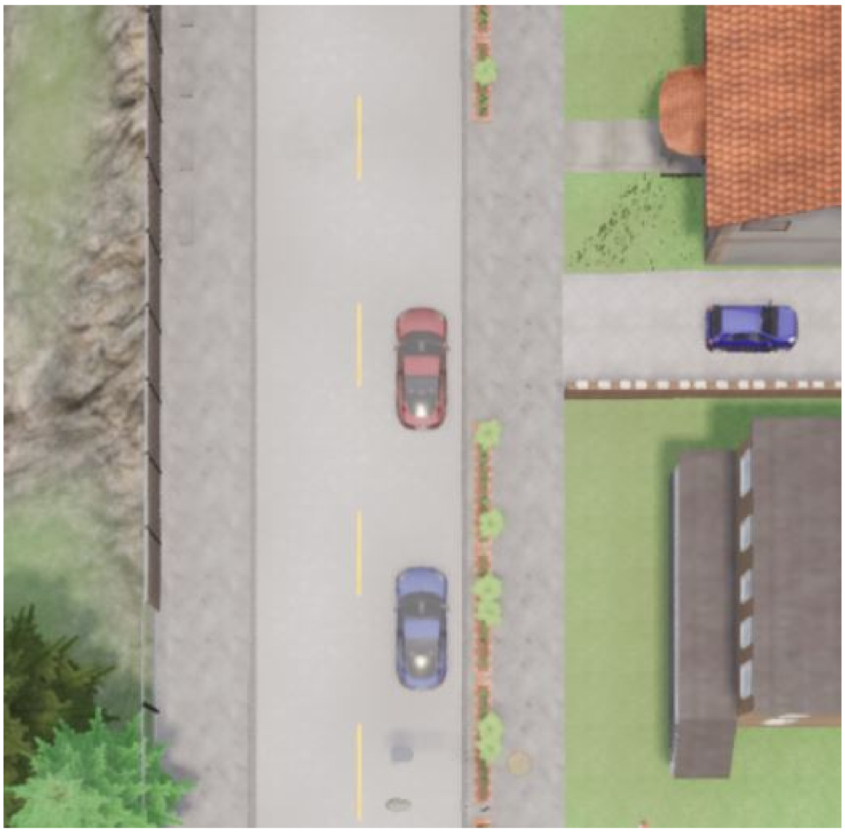


Figure 4.1: **Ego and Leading Vehicle in CARLA Simulation in Town 1**

4.1.1 Implementation of RADAR and LiDAR Based System

However, in order to deploy a fully autonomous vehicle in recent trends, it requires a large amount of information from the sensor modules attached to the vehicles. Once the target is detected, the host vehicle needs to address the following queries with respect to the target object:

How far is the object? How quick is the object approaching or departing? Is the objective article is in left, right or straight ahead? Is the object in right or left pr straight ahead? What is this target object? Is it a person or vehicle.

RADAR and LiDAR innovation helps the vehicle and the installed perception algorithms to respond these questions with the help of "five-dimensional" datasets defined as Range, Azimuthal direction of arrival (DoA), Elevation direction of arrival (DoA), Doppler, and Micro-Doppler

As vehicles move from level 1 with full manual control to level 5 with full freedom, auto

radar advancement will be used for emergency breaking and versatile journey control with continuously extending reliability and accuracy demands. Hardware simulation is important during the stage of sensor design and software simulation becomes more important when testing and validating radar and LiDAR sensor execution in the reality. In particular, RADAR specialists ought to develop RADAR sensors that precisely sense the environment and gives nonstop data to vehicle's perception model. Failure in the sensor configuration can seriously influence the security of completely independent vehicles. Moreover, engineers ought to test the sensor execution with all the corner cases that might demonstrate exceptionally hazardous or of significant expense for actual testing. It is additionally assessed that 8.8 billion traveling miles should be finished before the independent vehicles arrive at clients. It is also estimated that 8.8 billion driving miles must be completed before the autonomous vehicles reach customers. In order to drive this much miles, simulation is the only practical way for testing and reaching the goal by driving a billion miles virtually along with safety testing and evaluating the performance in corner cases.
LiDAR (Light Detection and Ranging) is the only sensor which gives resolutions at range and is used to get fine and accurate detection of objects in space. Sonar is used in LiDAR where pulse laser waves are used to map the distance of the surrounding objects upto 60 metres. In order to navigate environment, it is used in large number of autonomous vehicles. LiDAR is used to create 3D cloud of points which is better at judging distances by collecting and recording the external surfaces of objects and scenes. Thus, the processing of sensor inputs are the main area we have to focus while designing ACC.

4.1.2 LIDAR Structure

During the operation of LiDAR system, it rotates the laser emitter in horizontal direction and in some LiDAR's vertical alignment is also created. The frequency of horizontal resolution is referred as Refresh rates and the frame rates are referred as FPS (Frames per second) which is denoted by the following equation.

$$FPS = \frac{\alpha_{FoV}}{\Delta\alpha} \times f_{rot} \tag{1}$$

where α_{FoV} is horizontal Field of View, $\Delta\alpha$ is horizontal resolution and f_{rot} is refresh rate. From the above equation, points per second is is obtained by:

$$PPS = N_{channel} \times FPS \tag{2}$$

where $N_{channel}$ denotes the number of channels which denotes the number of laser emitters on the device. LiDAR output data obtained in CARLA is the 3D point cloud which has many formats from various standards and organizations like Polygon or Stanford triangle format (PLY) and Point Cloud Data (PCD).

4.1.3 RADAR Structure

RADAR plays an important role in autonomous vehicles because of their robustness against poor weather conditions and CARLA provides a RADAR sensor that gives distance, angular position, and the relative speed of the obstacle. In RADAR, transmitter is used to emit electromagnetic waves which is reflected back on the surface of nearby objects. The distance between the object and the sensor are calculated using the waves reflected by the below formula:

$$D = c * T/2 \tag{3}$$

where D is the distance to the obstacle, c is the speed of electromagnetic waves and T is the time between emission and reflection of waves. Output from the RADAR sensor is a two-dimensional data format.

4.2 MDP Formulation for Adaptive Cruise Control

In DRL, the agent interacts with the environment consisting of roads, vehicles and sensors. The agent can control its speed and maintain a safe distance to that of the leading vehicle. Considering the formulation of MDP in previous sections, autonomous controller task is solved using MDP which comprises of an agent that notices the state (s_t) of the ego vehicle and then action (a_t) is generated. Now the vehicle will lead to a new state s_{t+1} and a new reward

$r_t = R(s_t, a_t)$ is produced based on the exploration. Markov Decision process is generally defined as $(S, A, P_a, R_a,)$ where the primary objective is to find the good "policy" which is used to decide which action to choose when the agent is in a specific state s_t

a) **State space (S)** : In each algorithm step, the data is received from the environment. In our case, we use RADAR and LiDAR from which the estimated speed, distance to the leading vehicle and other driving features that are extracted from the sensors.

b) **Action Space (A):** In order to interact with the vehicle in the simulator, the commands used for throttle, steering and brake ought to be given constantly. Throttle and brake range is [0,1] and steering range is [-1,1]. Therefore, at each step an action $a_t = (acc_t, steer_t, brake_t)$ is produced by the agent with the commands into their ranges.

c) **State Transition Function (P_a)** : Likelihood of an action a in state s at time t will lead to state (s_{t+1}) at time t+1. $P_a = P_r(s_{t+1}|s_t, a_t)$

d) **Reward function R:** The immediate reward of transition of an agent from state s_t to s_{t+1} is generated by the reward function. The goal of MDP is to find a good policy that will choose an action a_t in a given state. This function is used to maximize the expectation of average future rewards.

Each of these sections are discussed in detail in below steps:

4.2.1 State Representation

The state vector depends on the data input used for the DDQN algorithm. Here for ACC scenario, we use the distance and speed information of ego vehicle and social vehicles for state representation. In autonomous vehicles, considering the collision scenario, the system design is done in such a way that the major challenge of the intersection scenario comes from the interaction between vehicle. The state vector of ego vehicle is defined as a 3D representation of the simulation at a given step. The location of the vehicle is in binary form matrix and velocity matrix of the vehicle is also represented at a given time step. The two matrices are then combined to form a 3d matrix. Here, LiDAR and RADAR sensor are responsible for getting the state of the agent at each time. The output from the RADAR is a conic view of 2D point elements in sight and speed regarding the sensor from which the direction and movement of the leading vehicle is evaluated.

4.2.2 Action Space Representation

Once the agent has observed the given state, s_t, an action is chosen from the given set of possible actions, a_t in A. Each action changes the values of vehicles which are present in state space. CARLA gives five types of control commands - steer, throttle, brake, hand brake, and reverse gear.

Control Commands (Actions): In CARLA, control commands for steering is [-1,1] and for throttle is [0,1].

It is difficult to select an appropriate action space due to number of possible actions, so an action space is chosen where each action alters the value of vehicles that are present in the state space. Initially, two sets of action pairs are selected and then compared with each other in order to measure the performance difference. The action set contains three different actions A = 0,1,2. Here, Action 0 forces the ego vehicle to decrease their speed, 1 used to maintain, 2 forces two vehicles to increase their speed and acceleration and decrease the minimum distance between two vehicles and 3, Different scenarios of action space are mentioned below in detail:

- **Action 0:** Low Acceleration
- **Action 1:** Medium Acceleration
- **Action 2:** High Acceleration

In each time step, the vehicle chooses any of the above actions in order to adjust the speed according to the leading vehicle.

4.2.3 Reward Function

One of the most challenging parts in Reinforcement learning problems is defining the reward function in order to find out the policy that is optimal for the given problem. The reward is a scalar value r and is returned when an agent performs a given action in the environment. Here, the global reward function is added and the aim is to keep the global reward as high as possible. Here we chose a reward function with values -200 if collision occurs and a value of -1 reward if the distance between two vehicles are not maintaining the certain distance and a reward value of +1 if there is no collision and the vehicles are driving with a set distance between them.

$$R = \begin{cases} -200, & \text{if Collision} \\ -1, & \text{if No safety distance} \\ 1, & \text{if No Collision} \end{cases} \tag{4}$$

4.3 CARLA Simulation

Car Learning to Act (CARLA) is an open source simulator built for autonomous driving research and is based on Unreal engine 4. OpenDRIVE standard is used in CARLA in order to define roads and urban settings. The simulator consists of scalable client-server architecture where the server is responsible for simulation like sensor rendering, world-state and actor updates, computation of physics, etc. The client comprises of all the client modules used to control the actors and setting up the environment conditions which is developed by CARLA API using python/C++. There are many features that can be used with CARLA some of which are listed below:

Traffic Manager: A built-in framework which takes the control of the actors and acts as a guide given by CARLA to reproduce urban environments with realistic conditions.

Sensors: In CARLA, sensors are actors attached to vehicles, and the information received can be retrieved and stored to reduce the process complexity. Currently, CARLA has different types of sensors - cameras, RADAR, LiDAR, etc.

ROS bridge and Autoware implementation: CARLA is used to integrate the simulator with other learning environments as well using ROS bridge and Autoware.

Open Assets: CARLA consists of various maps with metropolitan settings for command over weather patterns and the components can be modified.

Scenario runner: A series of routes are provided by CARLA in order to make the learning process easy which describes different situations to iterate on.
When compared with other simulators CARLA has the open source options and also provides existing towns and benchmarks and is easier to use. Creating Autonomous vehicles and test drive in preexisting towns is made easy using CARLA which can provide feed from specific sensors which can be selected according to our usecase. There are numerous examples and tutorials on CARLA on how to use the software and the main disadvantage of using CARLA is that the simulation is time consuming due to heavy computational software.

4.3.1 CARLA Simulator for Autonomous Vehicles

CARLA (Car Learning to Act) has been created to support advancement, training, and validation of independent driving frameworks. The architecture consists of client and server where the server is the simulated environment and client is the interface to the simulator by controlling the weather, velocity of the vehicle, etc. CARLA simulator is used in [19] in order to achieve autonomous driving and compared the performance using Q learning and Double Q learning method ending up with Double Q learning method which has faster convergence method and good performance. In [27] Deep Learning method is used in the control layer of the autonomous vehicles using Deep Deterministic Policy Gradient (DDPG) and Deep Q-Network (DQN) algorithms in order to train the vehicle to navigate and follow the determined route efficiently. CARLA simulator is used in testing the agent with deep learning algorithms and Figure

CARLA is being used currently and one of the most powerful simulators in developing and testing Autonomous vehicles which is an open source based on Unreal engine and for the control layer CARLA provides the actual vehicle odometry to the user which makes easier to evaluate the performance of the proposed models.

Highlighted Features of CARLA simulation are stated below:

- **Scalability:** CARLA supports multiple clients in the same or different nodes which is

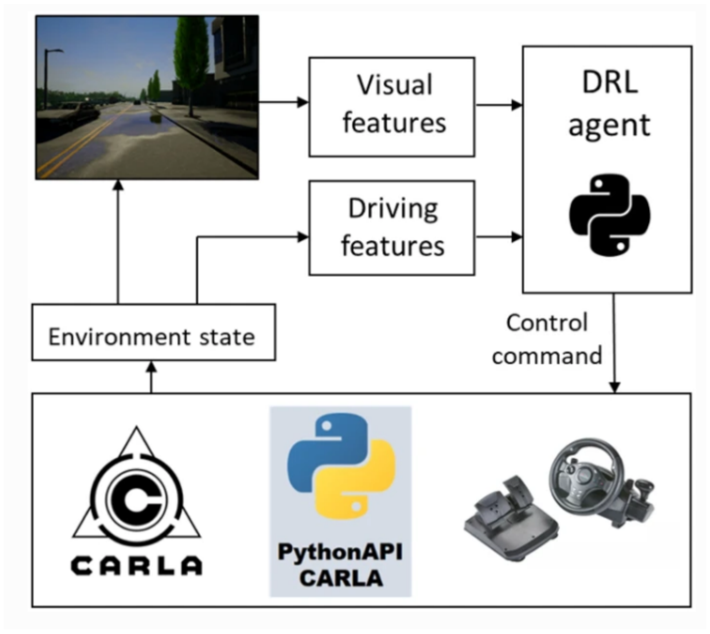


Figure 4.2: **CARLA Simulation using Reinforcement Learning**

used to control different actors

- **Flexible API:** CARLA has a powerful API which allows user to control the simulation aspects including traffic generation, weather,sensor, pedestrian behaviour, etc.

- **Autonomous Driving Sensor Suite:** Sensors can be configured including LiDAR, cameras, RADAR's, depth sensors, GPS, etc.

- **Maps:** Users are able to create their own maps using OpenDRIVE standard tools like RoadRunner

- **Traffic Scenarios:** ScenarioRunner engine allows the users to specify different traffic scenarios based on modular behaviours

- **Autonomous Driving:** In CARLA, Autonomous baselines are used as agents which includes AutoWare agent and conditional Imitation Learning agent.

4.3.2 Configuration of Actors

In order to set the client, Client class is used which returns an object that can be used to access the information of the running simulation. The client configuration is shown below:

client = Carla.Client('localhost',2000) client.set_timeout(2.0)

After setting the client, server side is initiated in order to connect to the World instance which includes methods such as blueprint library, maps, actors and weather conditions. Below command is used in order to retrieve the World that is running currently

world = client.get_world()

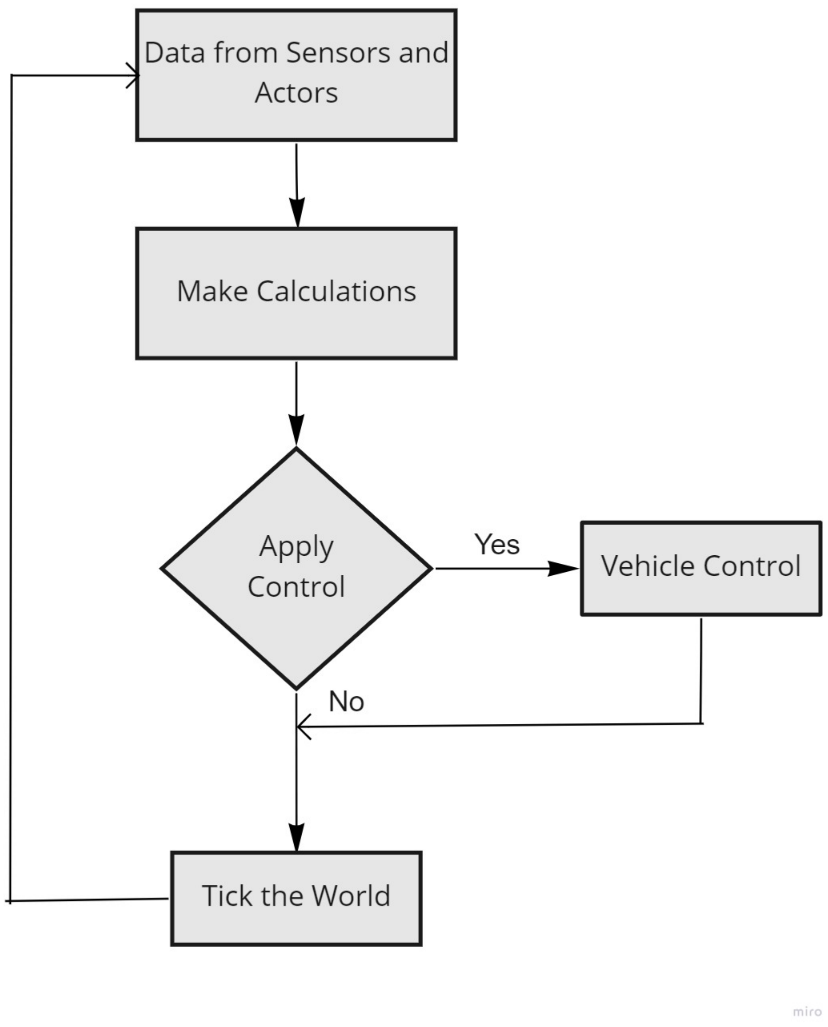


Figure 4.3: **CARLA Simulation Loop - Autonomous Vehicles**

The above Figure 4.3 shows the simulation loop of autonomous vehicles using CARLA. Object 'world' gives access to the blueprint library which contains a list of ActorBlueprint elements. ID is used to identify the actors by blueprint.

```
blueprint_library = world.get_blueprint_library() bp =
random.choice(blueprint_library.filter('vehicle))
bp.set_attribute(_attr_name_,_attr_value_)
```

An initial random transform to the vehicle is given from the list of spawn points recommended in the map

```
transform = random.choice(world.get_map().get_spawn_points())
```

Spawning the vehicle in the world

```
vehicle = world.spawn_actor(bp,transform)
```

Enabling Autopilot mode

```
vehicle.set_autopilot(True)
```

Destroying Actor - In order to exit the actors from the simulation after quitting the python script

```
actor.destroy()
```

4.3.3 Retrieving Simulation

The client is initialized in the first step which gets the pointer to the world object hosted by the server in order to make an interaction. During Simulation, the client can get the information on vehicles using Vehicle class at anytime. Here we use Town 1 in our design for training the agent. An actor is spawned once the client is started and the actors are not destroyed until the destructors are explicitly called even though the python script of client is terminated. In order to spawn an actor, we must need a transform object and blueprint. Location and orientation are obtained using transform object. The other actor's location and orientation can set to be a reference and by default the reference point is the world coordinate system origin. In CARLA, the actor's LiDAR coordinate system consists of three axis x (roll),y(pitch) and z(yaw) where x points forward, y points right and z points up. Using the LiDAR point cloud the local camera coordinates can be formed as shown below:

$$(x_{cam}, y_{cam}, z_{cam}) = [x_{pc}, y_{pc}, z_{pc}] \times \begin{bmatrix} 0 & -1 & 0 \\ 0 & 0 & -1 \\ 1 & 0 & 0 \end{bmatrix} \tag{5}$$

In order to implement the point cloud CARLA uses ray casting emissions which detects mesh models in Unreal Engine. The output is a 32-bit array format which is fed into the neural network. The simulation is started with customized traffic settings and an ego vehicle is set to wander around the city with fundamental sensors. The recorded simulation is used to query and to find highlights. New sensors can be added to recover predictable information and the weather patterns can likewise be adjusted. Recording simulation is used to test specific scenarios with different outputs. CARLA provides a visualization script for LiDAR using Open3D. In CARLA, LiDAR is developed with ray-casting technology and the 3D points are calculated by adding a laser for every channel which is developed in a vertical field of view. Rotation is simulated by utilizing the horizontal angle that the LiDAR rotated in a frame the point cloud is determined by doing a beam cast for each layer in each step. LiDAR measurement contains a package with all the points that is created during 1/FPS interval where the materials are not refreshed during this interval so that all the points in measurement reflects the similar "constant picture" of the scene. The output LiDAR information is encoded as 4D points where the first three points are the space coordinates (xyz) and the last point is the loss of intensity during the movement which is computed by $I/I_o = e^{-a.d}$ where a is the attenuation coefficient which is depends on sensor wavelength and atmospheric conditions and d is the distance from the hit point to the sensor. RADAR is similar to LiDAR and creates conic view and is transformed to 2D elements in sight and their speed is calculated with respect to the the sensor which helps in shaping elements and evaluation of its direction and movement. Here the points will be concentrated around the center of view due to the use of polar coordinates. The raw information given by RADAR can be changed over completely to a usable format with the help of numpy.

4.3.4 Simulation Recording and Replaying

All the events that happened during the simulation of autonomous vehicles are recorded and stored for future purposes, allowing for reproduction while testing the autonomous systems. The data stored in the binary file is written on the server side and the recorder is managed by the client using Python API. The data is stored on every tick of simulation where the information consists of the vehicle's position id, steering, throttle, brake, id, and speed. Information on maps, traffic lights, date and time of the simulation was also saved. Recording is started using the command below

```
client.start_recorder("/home/carla/recording01.log")
```

To stop the recording, at any time from the client script the call is also straight forward:

```
client.stop_recorder()
```

CARLA provides an additional feature to trace and study the events happened in simulation such as collision and blocked actors for a specific amount of time.

4.4 Hardware Requirements

In any machine learning or artificial intelligence problems, hardware requirement is the most important to focus on. Good result is obtained when there are large number of computational resources. In our case, the experiment is done based on CPU memory and without any GPU support. The hardware specification is found in below table 4.1:

Hardware Elements	**Specification**
CPU Memory	8 GB
Operating System	Windows 10

Table 4.1: **Hardware Specification**

4.5 Adaptive Cruise Control Problem using CARLA

The functionality of cruise controlled systems is extended in Adaptive Cruise Control systems which can be used to adjust the velocity of the host vehicle in order to maintain a safe distance from the leading vehicle by controlling the throttle and brake. Here we use RADAR and LiDAR sensor to measure the distance from the front vehicle. Actors are defined in CARLA which plays the role of simulation which includes pedestrian, vehicles, sensors, and traffic signs. Spawning of actors are done in the simulation using carla.World and carla.ActorBlue

There are various types of objects in CARLA simulator which are mentioned below:

- **World:** Spawning of actors are done in the simulation using carla.World. World is the overall environment where we have the actors like cars, sensors, pedestrian, etc.
- **Blueprint Library:** The Blueprint library is the summary for all actor blueprints and its attributes available to the user
- **Actors:** The world object is used to spawn the actors and keep track of it. Spawning requires blueprint, and a carla.Transform which is used to describe the location and rotation for the actor.

Here we have two actors where the distance between them are measured using RADAR and LiDAR sensor and a safety distance of 10metres is set manually between two vehicles.

4.5.1 Agent Training using DDQN

A Markov Decision process is utilized to take care of the control problem by finding the ideal actions in each time step. The training of the agent is done using DDQN method where the observation state is taken from simulation and is passed to Deep Neural Network and Q value is calculated for each possible action.

- One Q network is used for predicting the Q function (Online network)
- One network is used for computing the targets (Target network)
- In each step the agent trains the online network
- Methods to update the target network weights

Here, we used step() function to modify the agent and its action where we adjust the throttle, brake, and steering as required and these are the actions that will be considered by the agent during training. If a collision occurs, the episode ends and an agent will be destroyed with a negative reward.

4.5.2 Neural Network Architecture

Here we build a Deep Neural Network to approximate the Q value function and the input is the current state of the agent and the output is the Q Values of all possible actions generated. Memory is used to store past experiences in the model and the maximum value of Q is used to determine the next action. Below are the steps involved in developing DDQN model:

- Gather the RADAR and LiDAR sensor data from CARLA Simulator and then feed it into DDQN Sequential model which returns the possible set of actions (Steer, Throttle, Brake)
- An action is selected by taking the highest Q value or in a random manner in order to implement exploration of the agent
- The Agent performs the transition from the current state to a new state and receives the reward. The new state is the preprocessed points detected by the Radar and the LiDAR sensor (current state, action, reward, new state)
- Random sample transition batches are picked up from the memory and calculate the loss

- The loss value is calculated as the squared difference between target Q and predicted Q
- Loss function is minimized using Gradient-Descent algorithm
- After every 'n' number of steps or episodes, weights and target models are updated

Figure 4.4 shows how the highest Q value is selected from Q table and Figure 4.5 shows the architecture of the DRL agent used in the work.

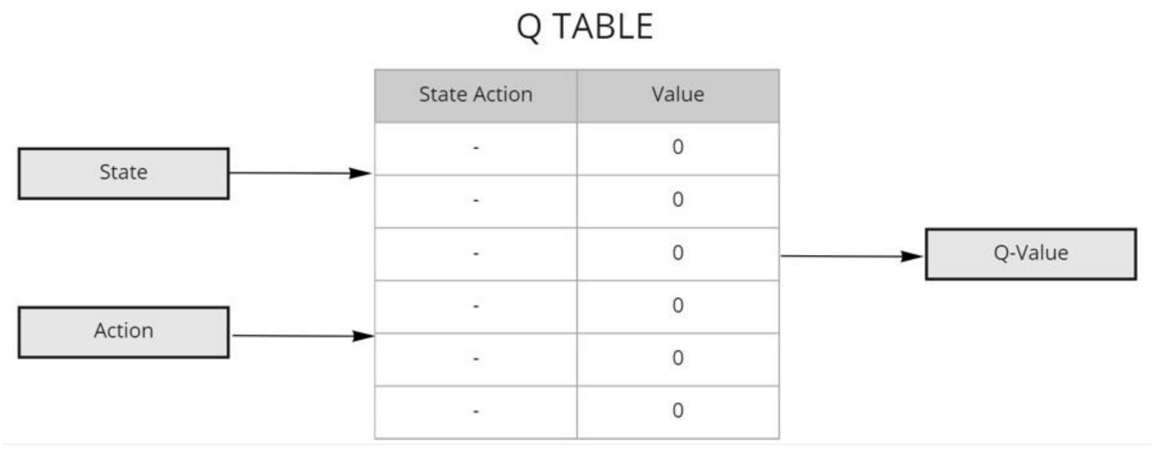


Figure 4.4: **Q Learning Method**

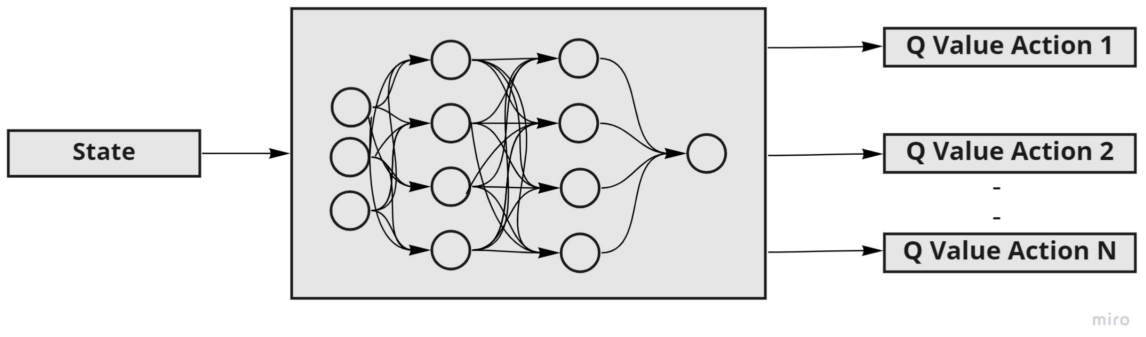


Figure 4.5: **Deep Neural Network Architecture**

4.6 Training and Results of the Model

The training of the model took 2 days and 3 hours. Training steps and the model results are explained in below steps:

In order to choose a different set of actions, we use step() function which is passed to the agent and action. If there is a collision, the agent is stopped and is destroyed with a maximum negative reward. If the minimum distance between two vehicles is not maintained, then a negative reward is applied. A positive reward is given if the vehicle maintains a safe distance and there is no collision. The algorithm should find the right action among the three actions using the highest Q value for the given input. In the first iteration, agent chooses a random action for a given state. The model takes the sensor data as input and gives three different outputs (action space) that is used to drive the car. This randomness factor is chosen by the parameter epsilon which is decayed as the number of episodes increases which means as the agent train the actions are less random. In order to train the model, the samples are randomly generated and training is started in a batch. The future q values are picked up once the batch is created. The transition in DDQN method is defined in the format (current state, action, reward, new state). Using this transition, the inputs and outputs for the DDQN model, X and y are created. Before training, random inputs are taken from the and then the continuous training of the model (inside infinite loop) is done using model.fit which uses the actual values from the sensor input. During training, the environment and the DDQN agent is initialized and then the metrics are reported. The training hyper parameters used for the executions are shown in Table 4.2.

Parameter	**Value**
Episodes	3000
Seconds per Episode	10
Replay Memory Size	5000
Minimum Replay Memory Size	1000
Minibatch Size	16
Training Batch Size	4
Minimum Reward	-200
Discount	0.99
Epsilon	1
Epsilon Decay	0.95
Minimum Epsilon	0.001
Safety distance	10 metres

Table 4.2: **Training Parameters**

Figure 4.6 show the epsilon, average reward value, minimum reward value, maximum reward, and the loss value over the number of episodes while training the agent. As the number of episodes increases, the value of epsilon decay decreases to 0.01 gradually which means that the learning rate of the agent increases in a higher rate till 2500 episodes and then increases slowly after 2500 episodes. The average reward value is very low at the start of the training and then increases at a higher rate till 500 episodes and then increases slowly. The average reward curve decreases after some point which points that the agent still explores a number of situations and then again increases slowly after 2500 episodes. Higher the average reward means higher agent exploration and higher will be the training accuracy. Minimum reward graph seems to increase at initial period of training as the agent is first penalized for wrong control actions and then decreases slowly after 2000 episodes as the weights are adjusted to prevent the extreme control values. When the agent is penalized for collisions, then it learns how to drive in order to not hit the other vehicles. It can be seen that the maximum reward increases as the episode goes on. At times, it decreased slightly but not for longer time. The number of collisions shown in loss graph decreased at higher rate initially and then increases slightly. Decrease in the number of collisions directly states that the agent is learning and performing better during training. From the above graphs, we can see that overall the model improved during training in complex environmental settings.

4.7 Difficulties

The main difficulty faced during training is the hardware requirement and the storage limits. The training for 3000 episodes took two days when using Intel core i5 processor and the use of advanced hardware systems with GPU would have reduced the time of training drastically. During simulation, the experiences are stored in RAM by the experience replay technique which increased the memory of the RAM gradually. The RAM storage got full during training where it removed the older entries and replaced them with recent ones which continuously gave an error. In order to tackle this issue, the replay memory is reinitialized to empty after 1000 time steps which fixed the previous occurring error.

4.8 Summary

In this chapter, we discussed the implementation and training if an agent using CARLA simulation. The training results from the tensorboard are visualized and interpreted. DDQN implementation, hardware setup, training hyperparameters and the difficulties while implementing along with the solution is also discussed in this chapter. The tensorboard graph of the reinforcement model are discussed and the results are interpreted.

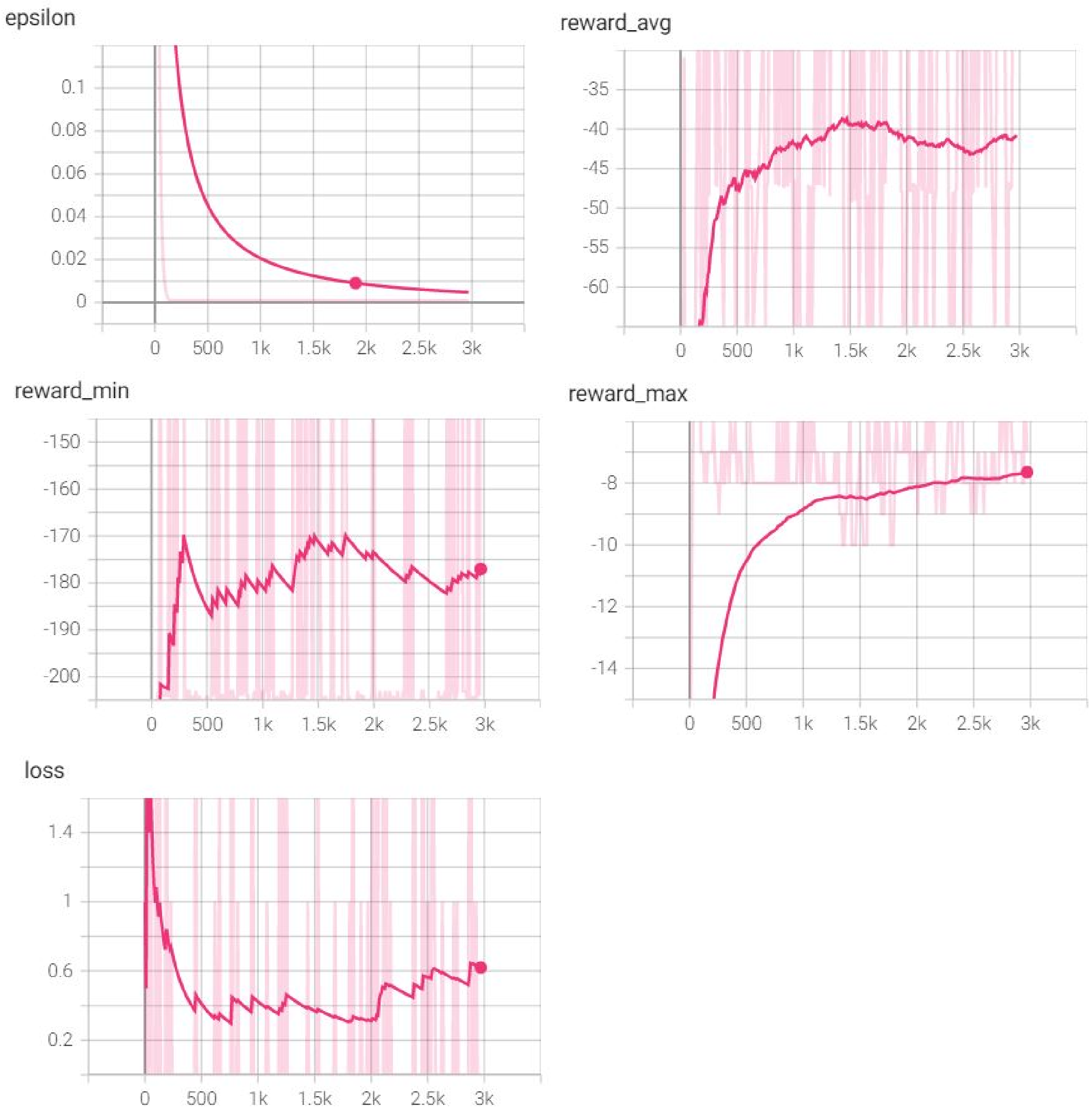


Figure 4.6: Model Output

5 Evaluation

In this section, we implement a DDQN solution for training an agent in different scenarios. We discuss the objective of the evaluation along with the hyperparameters used in the experiment. Metrics used for measuring the performance of the model are also detailed. The experiments used for evaluation are described and the outcomes are analyzed

5.1 Objective

The goal of the project is to develop a DDQN agent for Adaptive cruise control using RADAR and LiDAR sensors in order to avoid collisions and maintain a safe distance. In previous chapter 4, we discussed the Design and Implementation which addresses the requirements of an ACC system and satisfies in maintaining a safe distance and avoiding collisions. The developed DDQN model is successful in training the agent if it meets the following requirements:

- Number of collisions is decreased over time
- Average reward value is increased over time
- Epsilon Decay is converged over time

The model is compared with DQN model in terms of the above mentioned parameters. Since DQN model previously developed uses camera sensor for autonomous navigation, we refined the code for adaptive cruise control with camera sensor. The results of comparison are stated in section 5.5.

5.2 Metrics

The metrics used to validate the performance of the agents are explained below in detail:

- **Number of collisions**: Loss value defines the number of collisions over time which helps to determine the performance of the algorithm over time. This is the important metric that shows the traffic safety where every time step collision is checked in order to ensure that safety braking has been applied or not during simulation. During the end of every episode, the sum of total collisions are calculated and reset to 0 at the start of every episode.
- **Average Reward**: It is most commonly used metrics in any Deep Reinforcement Learning experiments which directly gives the performance by identifying how much reward the agent receives over time. During simulation, the agent collects as much as rewards as possible and higher the reward higher the performance. The average reward graph is shown in the results.
- **Epsilon**: The epsilon-greedy methodology chooses the action with the highest reward. It is used to maintain a balance between exploration and exploitation. We used decay epsilon greedy policy method which determines the exploration and exploitation of an agent throughout the episodes. During the initial period of training, there will be a higher proportion of exploration samples and the function decreases as the number of episodes increases. We specify the initial and final values of epsilon which is 0.95 and 0.001 respectively. If the function converges to the specified final value then optimal learning has been performed by the agent which is shown in the results.

5.3 Evaluation Scenarios

We first train the agent with different two different set of actions and from that best action is chosen which is used to test the agent for number of training episodes.

- **Action Space:** As discussed in 4.2.2, we define actions with low, medium and high acceleration. Here we choose two action sets with acceleration magnitude [0.3,0.5,0.8] and [0.2,0.7,0.9] respectively. As the throttle input value ranges from [0,1] with 0 being the lowest acceleration and 1 being the highest we select three values which is low, medium and high between 0 and 1 in a random manner according to the CARLA documentation in the official website. During simulation, we trained the agent with the two action sets and compare the performance in terms of collisions and average reward function. It is seen that the action set 1 performs better than action set 2 in different type of weather conditions and when more vehicles are spawned which is explained further in below section.

- **Changing weather parameters**: CARLA simulations enables users to change the weather parameters. The agent was trained in different weather conditions in order to check if the sensors are able to detect the leading vehicle's position and speed.
- **Spawning of more vehicles**: We spawn more vehicles nearby in the same lane as well as in different lanes in order to increase the traffic in simulation and train the agent for less collisions.

Each of these points a detailed in the following section.

5.4 Setup

This section explains the environment used in experiments as well as the different parameters chosen for evaluation.

5.4.1 Weather Parameters

Here we customize the weather parameters as rainy, sunny, day and night in order to develop a controller that works under all climatic conditions. In CARLA, we define the climate conditions using combinations of 4 parameters (Sun, Cloud, Rain and Wind).

The changing of different weather parameters in CARLA simulation is shown below in 5.1.

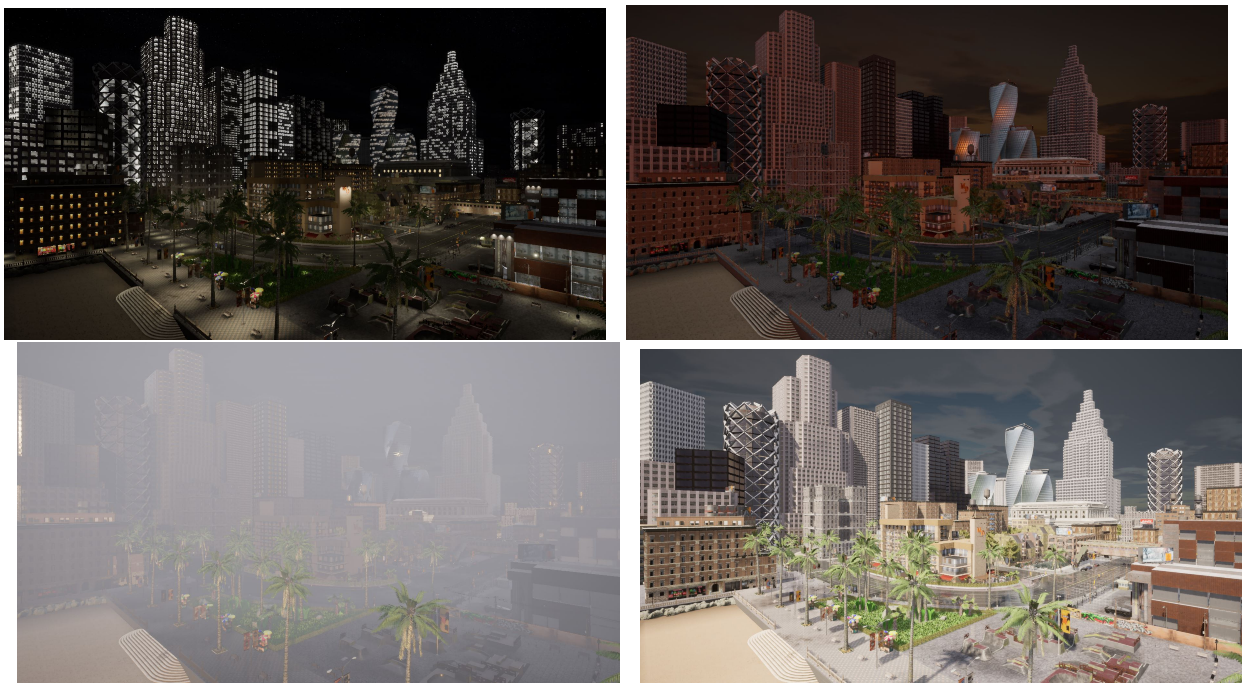


Figure 5.1: Dynamic weather patterns in CARLA

The combination of weather parameter values chosen to train the agent are given in table 5.1. These values are changed during the start of every episode randomly and are trained for 180 episodes and is seen that the average reward value is increasing with less collisions.

S.No	Sun	Cloud	Rain	Wind
1	0%	32%	0%	0%
2	20%	10%	90%	5%
3	0%	20%	90%	80%
4	50%	13%	0%	5%

Table 5.1: **CARLA Weather conditions**

5.4.2 Spawning more vehicles - High traffic

An increase in the traffic level results in higher chances of collisions, and trains the agent to learn to take actions in various states. In order to increase the traffic load, we spawned 1000 vehicles and 100 pedestrians and the collision rate is monitored.

5.4.3 Evaluation of Action sets

The two action sets are compared with random weather conditions and with more traffic load. It is seen that action set one performs better than action set 2 which can be clearly seen in below reward function graph.

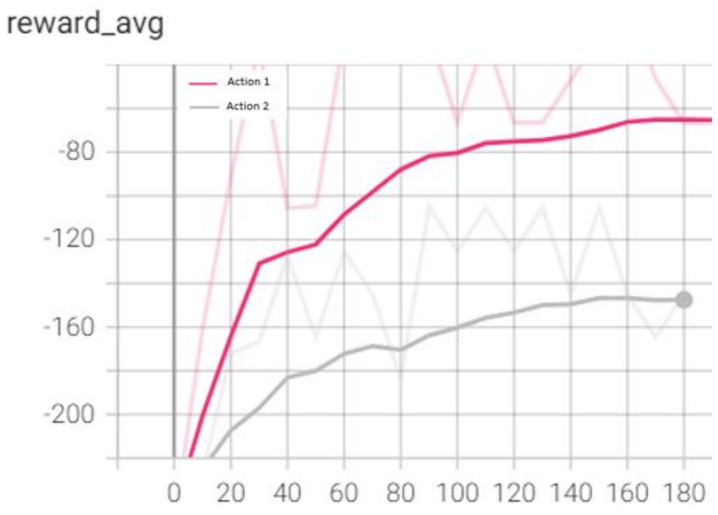


Figure 5.2: **Average Reward vs Episode**

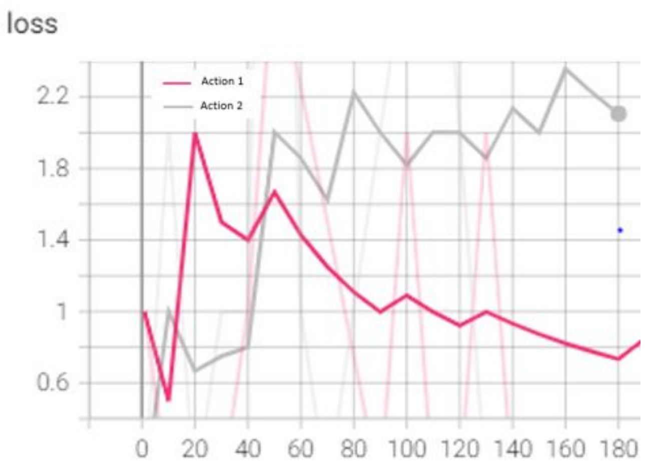


Figure 5.3: **Number of Collisions vs Episode bn**

With the above-mentioned scenarios, an agent is trained for 180 episodes and the average rewards and collision rate are monitored which is shown below in 5.2 and 5.3 respectively. It is clearly seen from Figure 5.2 that the average reward function increases during the simulation with action set 1 when compared to action set 2 interpreting that the agent learns in the right directions by taking correct actions during different climatic conditions and also in high traffic when using action set 1. Number of collisions is high with action set 2 as the number of episode increases as shown in Figure 5.3. Thus, we choose action set 1 to train our main model in different weather conditions and in high traffic.

5.5 Comparison with DQN Model

The proposed DDQN model with selected action set is evaluated against DQN model for which is a baseline model in our case. The DQN method is developed for Adaptive cruise control with the help of an existing research paper based on CARLA Autonomous vehicles for Deep Q Learning [1]. The model uses camera as the main sensor and is trained for 150 episodes. The DQN model is compared against the DDQN method for number of collisions and average reward value as shown below in Tensorboard Figure 5.4 and Figure 5.5.

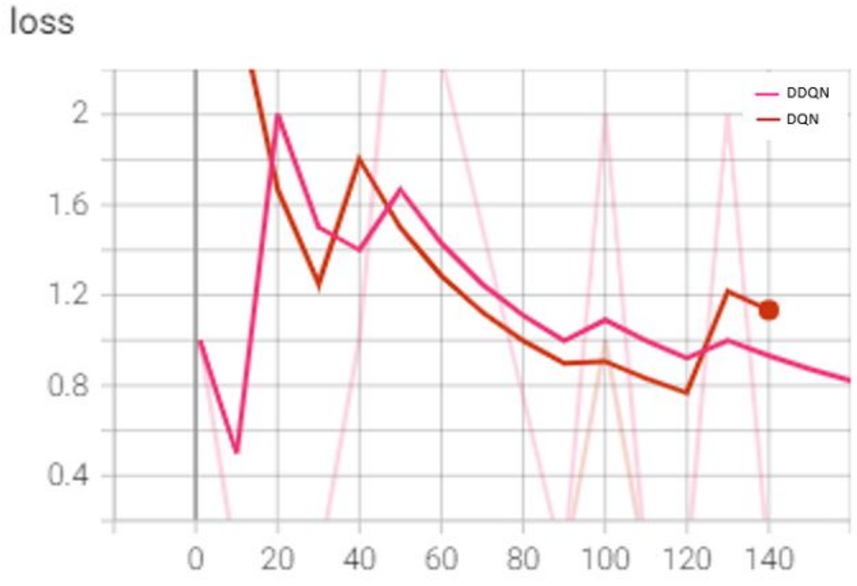


Figure 5.4: **Number of Collisions - DQN vs DDQN**

From the above figure, we can see that the number of collision is initially high when using DQN model and then decreases. When it comes to reward function, DQN model performs better than that of DDQN model for 150 episodes. Though there is not much difference between DQN (Camera) and DDQN (RADAR and LiDAR) model performance, DDQN learns gradually and DQN method first overestimates which tends to high number of collisions and then learns gradually.

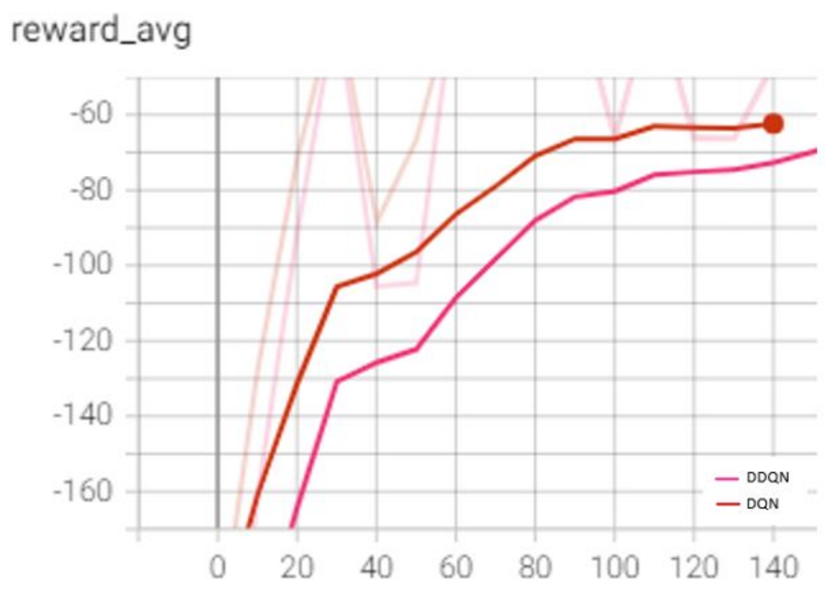


Figure 5.5: **Average Reward - DQN vs DDQN**

5.6 Evaluation Summary

In this chapter, we presented the details of the evaluation of the DQN and DDQN models for the ACC problem. We also discussed the objective of the evaluation, its metrics, the evaluation scenarios with the environmental setup and comparison with baseline. We evaluated action sets 1 and 2 in different weather conditions and high traffic. From the results, we can clearly see that the action set 1 outperformed action set 2. Action set 1 is therefore used to train the main agent with the DDQN method. The developed model with RADAR and LiDAR sensor input is then compared with a baseline DQN model with vision based model. From the comparison, we concluded that the DDQN model and DQN model performs almost equally by reducing collisions and increasing average reward, but DDQN method avoids overestimation when compared to the DQN method.

6 Conclusion

In this chapter, we summarize the thesis concept of thesis and highlight the most important achievements of the work. We further discuss the future work that can be improved from the current thesis work done.

6.1 Thesis Contribution

In this thesis, we developed a recent work for training an Adaptive Cruise Control based on RADAR and LiDAR sensors using the Reinforcement learning approach.
Chapter 1 describes the motivation behind Adaptive Cruise Control in Autonomous vehicles. We discussed the thesis goals and research question, and also outlined the thesis contributions and its structure.
Chapter 2 gave the background material used to develop the ACC model and it covers the concept of autonomous vehicles and reinforcement learning. We then cover the concepts of Deep Learning, Q learning and Double Q learning and then the proposed method for ACC.The different types of sensor modules used in current autonomous vehicles are discussed.
Chapter 3 describes the state of art which gives the classical approaches and more recent work related to control methodologies used in Autonomous vehicles. Using previous work, one can understand about the state, action and reward functions used in different scenarios with different algorithms. Simulation environment used in different research projects.
Chapter 4 presents the design and implementation details of an autonomous agent trained with respect to the leading vehicle. This includes a brief overview of the code, simulator used, and its environment. The hardware requirements for training and the training output are also

discussed.

Chapter 5 explains the evaluation method of different action sets in dynamic weather and heavy traffic conditions using CARLA simulation. The selected action set is then used in DQN and DDQN model and then the performance of the agent is evaluated and compared. The comparison results states that both DQN and DDQN perform almost equally with DDQN having its advantages without any over estimations whereas DQN has some overestimation errors. This concludes that the algorithm developed is fit for autonomous driving in order to show better performance of the agent.

6.2 Future Work

The field of RL has been advancing fast in recent years.

- There are a few new and old techniques that I would like to try: asynchronous RL, prioritized experience replay, and asynchronous Actor-Critic Agents (A3C)
- Comparing the developed models with RNN (recurrent neural network) and LSTM (Long short-term memory) models
- Test on real Autonomous Cars. It is not necessarily to test the real cars on highways in the next stage but a model car equipped with autonomous systems would be good enough to gather data close to reality which would then contribute to the modification of the simulator setting in both the system and the reward function
- An underlying Safety System. The underlying safety system has been mentioned in the Chapter 3 but can be detailed in the future, especially how to coordinate with the DQN planner
- Training for more episodes can help to find insights of the model performance and accuracy over time
- Coordinate the energy model into DRL reward function for an Eco-ACC framework

Bibliography

[1] M. Ahmed, C. P. Lim, and S. Nahavandi. A deep q-network reinforcement learning-based model for autonomous driving. In *2021 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, pp. 739–744, 2021. doi: 10.1109/SMC52423.2021.9658892

[2] L. Alonso, J. Pérez-Oria, B. Al-Hadithi, and A. Jimenez. Self-tuning pid controller for autonomous car tracking in urban traffic. 10 2013. doi: 10.1109/ICSTCC.2013.6688929

[3] J. Barkenbus. Self-driving cars: How soon is soon enough? *Issues in Science and Technology*, 34(4).

[4] R. Bin Issa, M. Das, M. S. Rahman, M. Barua, M. K. Rhaman, K. S. N. Ripon, and M. G. R. Alam. Double deep q-learning and faster r-cnn-based autonomous vehicle navigation and obstacle avoidance in dynamic environment. *Sensors*, 21(4), 2021. doi: 10.3390/s21041468

[5] D. Birnbacher and W. Birnbacher. Fully autonomous driving: Where technology and ethics meet. *IEEE Intelligent Systems*, 32(5):3–4, 2017. doi: 10.1109/MIS.2017.3711644

[6] M. Buechel and A. Knoll. Deep reinforcement learning for predictive longitudinal control of automated vehicles. In *2018 21st International Confe rence on Intelligent Transportation Systems (ITSC)*, pp. 2391–2397, 2018. doi: 10.1109/ITSC.2018.8569977

[7] H. Chae, C. M. Kang, B. Kim, J. Kim, C. C. Chung, and J. W. Choi. Autonomous braking system via deep reinforcement learning. In *2017 IEEE 20th International Conference on Intelligent Transportation Systems (ITSC)*, pp. 1–6, 2017. doi: 10.1109/ITSC.2017.8317839

[8] E. G. Debada and D. Gillet. Virtual vehicle-based cooperative maneuver planning for connected automated vehicles at single-lane roundabouts. *IEEE Intelligent Transportation Systems Magazine*, 10(4):35–46, 2018. doi: 10.1109/MITS.2018.2867529

[9] C. Dekkata and S. Yi. Improved steering and adaptive cruise control for autonomous vehicles using model predictive control. *Journal of Mechatronics and Robotics*, 3:378–388, 01 2019. doi: 10.3844/jmrsp.2019.378.388

[10] C. Desjardins and B. Chaib-draa. Cooperative adaptive cruise control: A reinforcement learning approach. *IEEE Transactions on Intelligent Transportation Systems*, 12(4):1248–1260, 2011. doi: 10.1109/TITS.2011.2157145

[11] J. Duchi, E. Hazan, and Y. Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12:2121–2159, 07 2011.

[12] S. Grigorescu, B. Trasnea, T. Cocias, and G. Macesanu. A survey of deep learning techniques for autonomous driving. *Journal of Field Robotics*, 37(3):362–386, apr 2020. doi: 10.1002/rob.21918

[13] I. Haskara, C. Hatipoglu, and U. Ozguner. Combined decentralized longitudinal and lateral controller design for truck convoys. In *Proceedings of Conference on Intelligent Transportation Systems*, pp. 123–128, 1997. doi: 10.1109/ITSC.1997.660462

[14] C. Hatipoglu, U. Ozguner, and M. Sommerville. Longitudinal headway control of autonomous vehicles. In *Proceeding of the 1996 IEEE International Conference on Control Applications IEEE International Conference on Control Applications held together with IEEE International Symposium on Intelligent Contro*, pp. 721–726, 1996. doi: 10.1109/CCA.1996.558954

[15] J. S. J. Koutnik and F. Gomez. Evolving deep unsupervised convolutional networks for vision-based reinforcement learning. *Pro ceedings of the 2014 Annual Conference on Genetic and Evolutionary Computation, ACM*, pp. 541–548, 2014.

[16] J. P. R. C. G. Jorge Villagra, Vicente Milanés. Model-free control techniques for stop go systems. *2010 13th International IEEE Annual Conference on Intelligent Transportation Systems*, 2010.

[17] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization, 2014. doi: 10.48550/ARXIV.1412.6980

[18] Y. Lin, J. McPhee, and N. L. Azad. Comparison of deep reinforcement learning and model predictive control for adaptive cruise control. *IEEE Transactions on Intelligent Vehicles*, 6(2):221–231, 2021. doi: 10.1109/TIV.2020.3012947

[19] Z. Liu, J. Hu, T. Song, and Z. Huang. A methodology based on deep reinforcement learning to autonomous driving with double q-learning. In *2021 7th International Conference on Computer and Communications (ICCC)*, pp. 1266–1271, 2021. doi: 10.1109/ICCC54389.2021.9674600

[20] S. Magdici and M. Althoff. Adaptive cruise control with safety guarantees for autonomous vehicles. *IFAC-PapersOnLine*, 50(1):5774–5781, 2017. 20th IFAC World Congress. doi: 10.1016/j.ifacol.2017.08.418

[21] R. L. E. M. J. P. Mauricio Marcano, José A. Matute. Low speed longitudinal control algorithms for automated vehicles in simulation and real platforms. *Complexity*, p. 12, 2018. doi: doi.org/10.1155/2018/7615123

[22] J. Mirwald, J. Ultsch, R. de Castro, and J. Brembeck. Learning-based cooperative adaptive cruise control. *Actuators*, 10(11), 2021.

[23] G. Nardini, A. Virdis, C. Campolo, A. Molinaro, and G. Stea. Cellular-v2x communications for platooning: Design and evaluation. *Sensors*, 18(5), 2018. doi: 10.3390/s18051527

[24] J. F. P. M. O. Garcia, G. Vitor and A. de Miranda Neto. The vilma intelligent vehicle: an architectural design for cooperative control between driver and automated system. *Journal of Modern Transportation*, 61(3):220–229, 2018. doi: 10.1007/s11431-017-9338-1

[25] S. Q. L. S. e. a. Peng, B. End-to-end autonomous driving through dueling double deep q-network. In *Automot. Innov*, vol. 4, pp. 273–275, 2021. doi: 10.1007/s42154-021-00151-3

[26] D. Pomerleau. Neural network vision for robot driving. In M. A. Arbib, ed., *Handbook of Brain Theory and Neural Networks*, pp. 161–181. MIT Press, 1995.

[27] B. R. L.-G. E. e. a. Pérez-Gil, Ó. Deep reinforcement learning based control for autonomous vehicles in carla. In *Multimed Tools Appl*, vol. 81, p. 3553–3576, 2022. doi: 10.1007/s11042-021-11437-3

[28] H. Raza and P. Ioannou. Vehicle following control design for automated highway systems [25 years ago]. *IEEE Control Systems Magazine*, 41(6):13–15, 2021. doi: 10.1109/MCS.2021.3107755

[29] D. Swaroop and R. Huandra. Intelligent Cruise Control System Design Based on a Traffic Flow Specification. Institute of Transportation Studies, Research Reports, Working Papers, Proceedings qt0941c5gg, Institute of Transportation Studies, UC Berkeley, Feb. 1999.

[30] H. Y. T. Miki, T. Ohya and N. Umeda. The overview of the 4th generation mobile communication system. *The Fifth International Conference on Information, Communications and Signal Processing*, pp. 1551–1555, 2005.

[31] P. Xu. A learning based adaptive cruise and lane control system. *Master's thesis, Case Western Reserve University, 2018*, 2018.

[32] J. K. Z. D. e. a. Yang, D. Intelligent and connected vehicles: Current status and future perspectives. *Sci. China Technol. Sci*, 61(4):1446–1471, 2018. doi: 10.1007/s11431-017-9338-1

[33] C. Zhang, X. Zhang, P. Ma, S. Dai, Y. Lu, and L. Jiang. Vehicle driving longitudinal control based on double deep q network. In *2022 14th International Conference on Measuring Technology and Mechatronics Automation (ICMTMA)*, pp. 273–275, 2022. doi: 10.1109/ICMTMA54903.2022.00059

[34] Y. Zhang, P. Sun, Y. Yin, L. Lin, and X. Wang. Human-like autonomous vehicle speed control by deep reinforcement learning with double q-learning. In *2018 IEEE Intelligent Vehicles Symposium (IV)*, pp. 1251–1256, 2018. doi: 10.1109/IVS.2018.8500630

[35] D. Zhao, D. Liu, F. L. Lewis, J. C. Principe, and S. Squartini. Special issue on deep reinforcement learning and adaptive dynamic programming. *IEEE Transactions on Neural Networks and Learning Systems*, 29(6):2038–2041, 2018. doi: 10.1109/TNNLS.2018.2818878

[36] M. Zhu, X. Wang, and Y. Wang. Human-like autonomous car-following model with deep reinforcement learning. *CoRR*, abs/1901.00569, 2019.