LOW SPEED CAR PARKING MANOEUVRING USING DEEP REINFORCEMENT LEARNING

KHIN KHIN KYI

A THESIS REPORT SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF
MASTER OF ENGINEERING IN AUTOMOTIVE ENGINEERING
SCHOOL OF ENGINEERING
KING MONGKUT'S INSTITUTE OF TECHNOLOGY LADKRABANG
YEAR 2024

KMITL-2024-EN-M- 277-201

THESIS TITLE	Low Speed Car Parking Manoeuvring using Deep Reinforcement Learning
STUDENT	Ms. Khin Khin Kyi
STUDENT ID	64601177
DEGREE	Master of Engineering
PROGRAM	Automotive and Advanced Transportation Engineering systems
THESIS ADVISOR	Assoc. Prof. Dr. Supat Kittiratsatcha
THESIS CO - ADVISOR	Prof. Dr. Masaki Yamakita

ABSTRACT

Due to the advancement of control systems, numerous research on autonomous driving has been conducted in different driving scenarios and environments intended to change to fully autonomous mode driving systems. Research on autonomous driving has been explored using various driving algorithms in diverse driving conditions. In recent years, Machine Learning and Deep Learning algorithms have been employed in automotive research field as a result of their astonishing performance in different industries. This study focuses on the parking scenario for a four-wheel vehicle in the Carla 3D Simulation Environment, using Deep Reinforcement Learning. Carla simulator is a good 3D simulator workspace as it provides realistic driving experience including a variety of infrastructures and non-player characters. The agent vehicle undergoes simultaneous learning and training in the environment, rewards and penalties are used as feedback results to evaluate the behavior of each action in every episode. The inputs to the DQN network are the state conditions of the agent in the environment and DQN network model generates the most suitable action for the given state based on the trained experiences. The collision sensor is used to detect collision accidents and three test scenarios are conducted with different spawn conditions and parking destination to evaluate the agent performance in each scenario. The agent successfully accomplished the parking process in different testing conditions and earned high rewards in each training episode by parking the car at the designated location without any collisions in a short amount of time.

K**eywords:** Deep Q Network (DQN), autonomous driving system, car parking simulation, Deep Reinforcement Learning, Carla Simulator

ACKNOWLEDGEMENTS

I would like to extend my sincere gratitude to each individual who supported and contributed to the completion of this thesis. Without their generous and positive encouragement, it would be difficult to complete this long academic journey.

First and foremost, I would like to express my deep gratitude to my co-supervisor, Dr. Jartuwat Rajruangrabin from the National Science and Technology Development Agency, for his continuous support and guidance the research journey. His expertise, insightful and invaluable advice have been instrumental in completion of this thesis research. I am truly honoured to work under his mentorship.

I would also like to extend my sincere appreciation to my supervisor, Dr. Supat Kittiratsatcha from King Mongkut's Institute of Technology, Ladkrabang and co-advisor, Dr. Masaki Yamakita from School of Engineering, Tokyo Institute of Technology. Their academic advice and insightful feedback have significantly contributed to overall improvement of my work.

Moreover, I would like to express my great appreciation to my senior, Wasinee Terapaptommakol , who gave new insights and valuable feedbacks and mental support throughout my research journey. Her intellectual sharing and advice inspired me for new ideas when I lost comprehension in the work. I am truly grateful for her contributions and guidance which improve the overall quality of thesis work.

I would like to extend my gratitude to Dr. Pramote Koowattanasuchat, whose invaluable advice and insightful ideas played a pivotal role in guiding my work in the right direction.

I would like to acknowledge the support of the mentors and staff members from King Mongkut's Institute of Technology, Ladkrabang. In addition, I would like to thank National Science and Technology Development Agency, for allowing me to use all necessary equipment required for the research work. Moreover, I am greatly thankful to TAIST-Tokyo Tech organization for the scholarship support though the Master academic years.

Last but not least, I am truly grateful to my family, friends, fellow classmates and to each individual person who encourage and support me physically and mentally. Their contributions played a significant role to pursuit my academic career and motivation to complete this research work.

TABLE OF CONTENTS

ABSTRACT I
ACKNOWLEDGEMENTS II
TABLE OF CONTENTS III
LIST OF TABLES VI
LIST OF FIGURES VII
CHAPTER 1 9
INTRODUCTION 9
1.1 Research Background 9
1.2 Objectives 12
1.3 Scope of the Work 12
CHAPTER 2 13
LITERATURE REVIEW 13
2.1 Carla Simulation Environment 13
2.1.1 Maps in Carla Environment 13
2.1.2 Actors in Carla Environment 14
2.2 Reinforcement Learning 15
2.2.1 The Agent-Environment Interface 16
2.3 Deep Reinforcement Learning 18
2.3.1 Deep Neutral Network Architecture 19
2.3.2 How the agent is trained in a DQN Network 20
2.3.3 Weight Initialization 21
2.3.4 Loss Function 21
2.3.5 Activation Functions 22
2.3.6 Rectifier Linear Unit Activation Function 23
2.3.7 Forward Propagation 24
2.3.8 Back Propagation 24
2.3.9 Target Network 25
2.3.10 Experience Replay 26
2.3.11 Epsilon Greedy Strategy 26
2.3.12 Rewards 27
CHAPTER 3 29
METHODOLOGY 29
3.1 Design Scenario 29
3.2 Carla Simulator 31
3.2.1 Parking Area Town 5 32
3.2.1.1 Assigning the Parking Area and boundary distance 32

3.2.2 Initialization of the vehicle and sensors 33
3.2.3 Vehicle Control 34
3.3 Neural Network Architecture 34
3.4 Model Setting 36
3.4.1 Input and Output Hyperparameters Setting 37
3.4.2 Reward Setting 37
3.4.3 Network Hyperparameter Setting 38
CHAPTER 4 39
RESULT AND DISCUSSION 39
4.1 Preparation of Simulation Environment 39
4.1.1 Carla Simulation Environment Setup 39
4.1.2 Parameters Tuning for the Parking Process 40
4.1.2.1 Network Model, Input and Output Parameter Setting 40
4.1.2.2 Network Hyperparameter Setting 41
4.1.2.3 Reward Setting 41
4.2 Training the agent 46
4.3 FIRST PARKING CASE SCENARIO 47
4.3.1 Neural Network Architecture 48
4.3.2 Reward and Epsilon Setting for 1st Scenario 49
4.3.3 Result and Discussion for 1st Scenario 50
4.3.3.1 Discussion for Distance Result 50
4.3.3.2 Discussion for Reward Result 51
4.4 SECOND PARKING CASE SCENARIO 52
4.4.1 Environment Modification and Explanation of Different Training process 52
4.4.2 Neural Network Architecture for 5000 and 7000 episodes 54
4.4.3 Training scenario for 5000 episodes without parking goal constraints 54
4.4.3.1 Reward Settings for the training 5000 episodes 54
4.4.3.2 Result and Discussion for training 5000 episodes 55
4.4.4 Training scenario for 7000 episodes with parking goal constraints 58
4.4.4.1 Reward Settings for the training 7000 episodes 58
4.4.4.2 Result and Discussion for training 7000 episodes 59
4.4.5 Results Comparison for different training conditions in 2nd Scenario 62
4.5 THIRD PARKING CASE SCENARIO 64
4.5.1 Neural Network Architecture 66
4.5.2 Reward and Epsilon Setting for 3rd Scenario 66

4.5.3 Result and Discussion for 3rd Scenario 69
4.5.3.1 Discussion for Distance Result 69
4.5.3.2 Discussion for Reward Result 70
4.5.4 Discussion why agent could not park successfully for 3rd Scenario 72
CHAPTER 5 74
CONCLUSION 74
CHAPTER 6 76
FUTURE WORK 76
6.1 ADDITIONAL SCENARIOS 76
6.2 Scenario 1: Parking of the car between two parked cars 77
6.2.1 Environmental Boundary Conditions 78
6.2.2 Proposed Neural Network Architecture 78
6.2.3 Reward Function 79
6.2.4 Discussion 81
6.3 Scenario 2: Parking the car while avoiding an obstacle 82
6.3.1 Environmental Boundary Condition 82
6.3.2 Perception of the Environment 83
6.3.3 Proposed Neural Network Architecture 84
6.3.4 Reward Functions 84
6.4 Omitting white lanes 86
6.5 Summary 86
REFERENCES 87
APPENDIX A: 90
CONFERENCE PARTICIPATION 90
AUTHOR BIOGRAPHY 91

LIST OF TABLES

Table 2-1 Conventional Q table Structure.....17
Table 4-1 Reward Parameters applied in training the Agent for 7000 episodes.....44
Table 4-2 List of Hyperparameters and Values for Second Scenario.....53
Table 4-3 Comparison for 5000 and 7000 Training Episodes in 2nd Scenario.....62
Table 4-4 Comparison of 5000 and 3500 Training Episodes in 3rd Scenario.....68
Table 6-1 List of Hyperparameters and Values for Additional Scenario.....79

LIST OF FIGURES

Figure 2-1 Architecture of Server-Client in Carla Simulator 13
Figure 2-2 Diverse Urban Layout supported in Carla Simulator.....14
Figure 2-3 Reinforcement Learning Diagram.....16
Figure 2-4 Conventional Q table Vs Neural Network 18
Figure 2-5 Components in a Neuron.....19
Figure 2-6 A Neuron Network Architecture.....20
Figure 2-7 Rectified Linear Unit Activation Function.....23
Figure 2-8 Forward Propagation and Backpropagation.....24
Figure 3-1 Design Model to interact Simulation Environment and DQN Network ...29
Figure 3-2 Demonstration of Scenario 1.....30
Figure 3-3 Demonstration of Scenario 2 (with different spawn angle and location)..30
Figure 3-4 Demonstration of Scenario 3 (with different spawn angle and location)..31
Figure 3-5 Basic Structure of Carla Simulator.....32
Figure 3-6 Urban Design of Town 5.....32
Figure 3-7 Parking Zone Destination and Parking Boundary Area.....33
Figure 3-8 Tesla Model 3 in Carla Simulator Environment 34
Figure 3-9 Wheel Angle variation for Tesla Model 3.....34
Figure 3-10 Q-Learning Table and Deep Q Network Architecture 35
Figure 4-1 Neural Network Architecture for the training.....41
Figure 4-2 Turning Angle of an Agent 43
Figure 4-3 Flow Diagram of the training the agent 46
Figure 4-4 Training Demonstration for the 1st Parking Scenario.....48
Figure 4-5 Distance Result Comparison for First Scenario 50
Figure 4-6 Average Episodic Reward Comparison for First Scenario 51
Figure 4-7 Agent Demonstration of Second scenario at around 450 episodes 52
Figure 4-8 Last Distance Result for training 5000 episodes in Second Scenario 55
Figure 4-9 Total Reward Earned for training 5000 episodes in Second Scenario 56
Figure 4-10 Average Reward Earned for training 5000 episodes in Second Scenario 57
Figure 4-11 Successful Parking Performance without Parking Angle Constraints 57
Figure 4-12 Successful Parking Performance without Parking Angle Constraints 58
Figure 4-13 Last Distance Result for training 7000 episodes in Second Scenario 59

Figure 4-14 Total reward for training 7000 episodes in Second Scenario.....60
Figure 4-15 Average reward for training 7000 episodes in Second Scenario 60
Figure 4-16 Successful Parking Performance I of Agent at around 2450 episodes....61
Figure 4-17 Successful Parking Performance II of agent at around 3650 episodes ...62
Figure 4-18 Comparison of Last Distance Result for training 5000 and 7000 episodes 63
Figure 4-19 Comparison of Epsilon Result for training 5000 and 7000 episodes.....63
Figure 4-20 Comparison of Loss Function Result for training 5000 and 7000 episodes 63
Figure 4-21 Comparison of Episodic Reward for 5000 and 7000 episodes 64
Figure 4-22 Comparison of Average Episodic Reward for 5000 and 7000 episodes.64
Figure 4-23 Agent Spawning Area for 3rd Scenario 65
Figure 4-24 Agent Spawning Location for the Third Scenario 66
Figure 4-25 Agent Parking performance for 3rd scenario spawned at different location 68
Figure 4-26 Agent Parking performance for 3rd Scenario.....68
Figure 4-27 Distance Result Comparison for Third Scenario.....69
Figure 4-28 Episodic Reward Comparison for Third Scenario 71
Figure 4-29 Average Episodic Reward Comparison for Third Scenario.....71
Figure 6-1 Parking Scenario for parking for the car between two parked cars 77
Figure 6-2 Boundary Conditions for Parking between two cars.....78
Figure 6-3 Boundary Conditions for Parking with an obstacle 82
Figure 6-4 Perception of an agent car using Lidar Sensor.....83

CHAPTER 1

INTRODUCTION

1.1 Research Background

The majority of individuals worldwide rely on vehicles for transportation, either for the purpose of commuting or logistics. People daily spend some part of their time on vehicles, therefore, vehicle comfort and safety should be emphasized and prioritized to secure the lives of the passengers. According to the World Health Organization[1], about 1.19 million people experience from casualties and fatal injuries due to road traffic accidents, primarily caused by human errors. Similar to other automated industries, the conventional vehicle sector has shifted the vehicle technologies towards autonomous driving systems, aiming to reduce the risks associated with human actions. With advancing technologies, the concept of fully autonomous vehicles on roads has become a realistic possibility as top-tier major automotive companies extended their presence in the autonomous vehicle sector [2], [3], [4]. As the autonomous driving scenario is dynamic and unpredictable, numerous challenges should be overcome to upgrade the vehicle technology to fully autonomous driving mode, such as enabling them to make accurate driving decisions while adhering to traffic rules without any human influence.

With the continuous technology development in automotive industry, the driver assistant technologies such as Adaptive Cruise Control, Lane Invasion Notification systems are supported in modern cars to provide easy driving. However, new technologies are required to fully automate the vehicle.

Autonomous driving is a challenging task as the car needs to (1) perceive and locate itself in the diverse environment within infrastructures, static and dynamic objects, pedestrians, and traffic signs while sensing the environment, (2) make calculations of decide the vehicle actions and (3) execute the vehicle control action plans in practice. Autonomous driving is conducted in a variety of driving environments such as urban driving[5] and [6] , driving on highways[7] and off-road driving[8].

Machine Learning algorithms have been widely used as a human brain to generate driving decisions based on the motor sensors data and current environment conditions. Rule based control strategies use long lists of conditional statements to guide the behavior of the vehicles in the traffic. The Intelligent Driver Model (IDM) with stochasticity [9] and Nonlinear car following models [10] are some notable

research projects. As the driving scenario is diverse and unforeseen, the written rule-based conditions could not cover all the driving possibilities. Therefore, the vehicle will be out of control when it faces an unfamiliar new situation.

In imitation or behavior cloning method, the vehicle learns the strategy and rules from the driving demonstration of a human using deep neural networks. Using vision-based inputs such as videos and RGB images, conducting the training using Convolutional Neural Networks is proposed in [11]. However, those methods have significant drawbacks compared to new deep algorithms, in terms of limitation in data handling and generalization in the data, overfitting and biases problems as the agent is trained with limited amount of driving demonstrations.

Model predictive control (MPC method) with machine learning approaches [12], an analysis of online and batch end-to-end imitation learning system for agile, high-speed autonomous driving with low-cost on-board observations using Deep Neural Network was published by [13]. Nonlinear control law for autonomous geometric path tracking algorithms in real time with an obstacle avoiding between hazards and on curvy road is discussed in [8]. Various past research approaches focused on different aspects of autonomous driving. A lot of research is conducted using different control algorithms, which are generally categorized into classic controllers and AI based controllers.

In the last decade, another machine learning algorithm called Reinforcement Learning has gained popularity as it could be used in environments where the input state condition to the neural network is variable with respect to the agent action. The performance of this algorithm is recognized as success in virtual games platforms, especially in a chess game or finding the exit in a maze, where the agent interacts within the grid environment and its behaviors are optimized through iterative trial and error method which does not depend on the input dataset. After the success in Atari games with Deep Q Network method [14], those algorithms are widely used in various industries including Autonomous vehicle industries. Deep Learning algorithms such as Deep Q Network (DQN), DDPG and PPO models are started to be used as a controller in autonomous driving cars.

Reinforcement learning in comparison with the classic controllers like linear quadratic regulator (LQR) controller and model predictive control (MPC) controller in a virtual open space racing car simulator (TORCS) was researched by[15]. Two different approaches of using DQN and Deep Deterministic Policy Gradient (DDPG)

in navigating the agent and drive along the waypoint trajectories is discussed in paper[16]. That study focused on following a predetermined route in a short time while avoiding the other possible static and dynamic objects.

In autonomous driving scenarios, most of the research is conducted on real-time simulated driving platforms to be cost-effective while the test performance can be evaluated which resembles the real scenario. Autonomous parking simulation framework with moderate complex parking scenario using Proximal Policy Optimization (PPO) for deep reinforcement learning via the Unity game engine is discussed in paper[17]. This paper [18] represents the idea of how the small agent car with multiple sensors learns the policy of DDPG reinforcement algorithm in a close route path. In this research, autonomous driving approach with classic modular pipeline, end-to-end imitation learning and reinforcement learning training are conducted in a city view with different weather conditions simulated in CARLA driving environment.

The car following model using DDPG algorithm method simulated in Carla Environment is discussed in [19], the agent learns the good behavior from the human driver initially then explore the environment creating a more effective training. An Automatic Parking Model Based on Deep Reinforcement Learning [20] presents about the parking process using the kinematic car model while focusing on the detail parking angles for the reverse parking process using the DDPG algorithm framework which includes the actor and critic to optimize the process. This study [21] discusses the long-range valet parking (LAVP) based on different pick-up points and parking points using the calculated distance matrix to get the shortest path. This parking process for smart cities is optimized with DQN model, which is deployed on a small, scaled size environment. This autonomous car parking system using DRL[22] presents the parking training scenario in the Unity game engine. This study[23] presents design system of an autonomous car avoiding over external obstacles using DQN network.

Among different driving tasks, car parking task can be time consuming and laborious especially at the peak hours in a huge parking area. To alleviate the workload for the individuals, it would be more convenient if they could leave their cars at someplace within the parking lot and the car itself, undertakes the parking process to the designated location without colliding the external objects.

This research work fills the gap of the analysis of the performance of the agent for the parking process in the Carla simulation environment using DQN algorithm. The Carla environment supports real-time simulation and real-world driving experience

while providing necessary sensors for the agent vehicle. DQN model observes the state environment and returns the action which maximizes the total reward possible. Rewards shape the behavior of an agent by giving feedback information of the action. The data to the DQN model may be enormous and need large computational power. After hours of iterative training, performance analysis of the agent car is proceeded to see whether it could drive from the spawned location to the determined location in a short time while avoiding the obstacles.

1.2 Objectives

This research aimed to achieve the following objectives.

- To construct a suitable Carla simulation environment for parking the agent.
- To assess the factors that influence the behavior of the agent in each scenario.
- To analyze the performance of the parking process across three scenarios.
- To develop a control strategy to swiftly accomplish the parking procedure within specified parking constraints while avoiding external obstacles through simulation.

1.3 Scope of the Work

- Construct the parking case scenario in Carla Simulator while spawning the agent with the necessary sensors from the blueprint library.
- Assign a desired parking coordinate and load it into the environment.
- Design the Deep Q Network algorithm with hyperparameters to train the agent which can optimize the agent's performance while completing the task.
- Build the training interface for communication between the agent and server.
- Train and test the performance of the agent and evaluate the training efficiency.
- Iterative tuning of the different hyperparameters is proceeded until the optimum training results are achieved.
- Evaluate the result performance quality of the training for each parking scenario.

CHAPTER 2
LITERATURE REVIEW

2.1 Carla Simulation Environment

Carla (Car Learning to Act) Simulator[24] is an open-source autonomous driving simulator which is built on Unreal Engine 4 (UE4) and operated in real-time scenarios. It is a versatile tool for simulating a realistic and immersive driving experience and realistic interactions between vehicles and pedestrians in the environment. The creative digital work in the simulator provides a wide range of actors, sensors, terrain layout, driving lanes, infrastructures, and the weather conditions. The engine is designed to behave as a sever-client system. The sever side handles and operates the simulation and visualizes the scene while the client side controls the agent's operation, training while acquiring the sensor signals and sends the new changes back to the server side. The client API is implemented in Python Language. The server updates its environment conditions according to the client's commands such as handling of the agent car, accelerating, or holding a beak. The client and the server work together to fulfil the task of the agent. Moreover, the client can make changes to the server side by changing the simulation environment to a new terrain, altering the sensor suite, or reinitializing the environment.

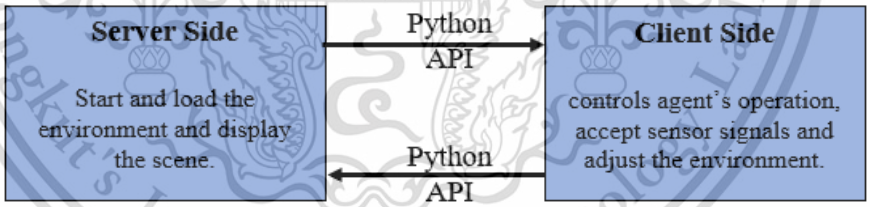


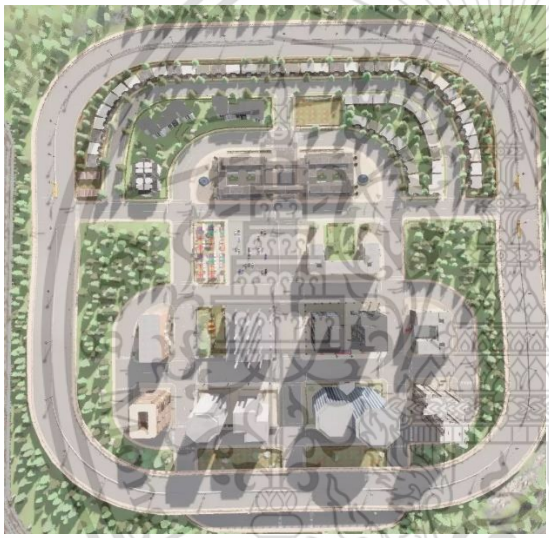
Figure 2-1 Architecture of Server-Client in Carla Simulator

2.1.1 Maps in Carla Environment

In addition to statics models like greenery, buildings, traffic signals and urban infrastructure, Carla Simulation environment is decorated with dynamics agents such as human players and motor vehicles. Currently, it supports 10 samples of maps which are intended for different driving experiences. Some are built into city urban areas with small lanes, some lanes are in deserted areas with large width suitable for high-speed driving training. Some include roundabouts and some include continuous circling lanes therefore, the user can choose freely which suits the best for specific driving purposes.

Town 01

Town 03



Town 05

Town 10

Figure 2-2 Diverse Urban Layout supported in Carla Simulator

The above is the overall view of several towns in Carla. The infrastructures and non-character players in each town can be altered and modified.

2.1.2 Actors in Carla Environment

The agent vehicle, all types of sensors and non-player character (NPC) vehicles and pedestrians are essential to create a realistic driving environment and they are the actors of this simulation environment. NPC characters can be programmed in auto-pilot mode so that their movement can resemble the realistic environment and keep a distance from each other to avoid collisions.

The vehicles are usually spawned from the CARLA Blueprint Library and their sizes are scaled to the actual cars as reference. Once an actor is selected and spawned from the library, the spawn location and its features are assigned by the user. As there

is a variety of choice for the vehicle type ranging from a bicycle to 4-wheel drive to container trucks, the user can find the most suitable one for the simulation. The agent is usually spawned by the user however, the NPC players can be spawned automatically by the system.

Sensors are also actors which can be spawned from the Carla Blueprint Library, and like the actor vehicles, its spawned location and features can be customized. They are used to locate the agent within the environment and detect the surroundings. Usually, sensors are attached to the parent vehicle body and receive sensor signal information from the environment and manage the data according to respective sensor function. Multiple sensors can be mounted onboard, and camera is one of the commonly used sensors as it supports RGB data which provides ground-truth depth and semantic segmentation. Radar and LiDAR systems are also widely used, and LiDAR systems emits light waves to the surroundings and sense back how long does the light take to make a return to the sensor to give out the distance of the obstacle form the sensor. Therefore, the agent can imagine and visualize its surroundings including the static and dynamic obstacles.

In addition to camera, radar and LiDAR sensors, the information of the agent's state such as the position of the vehicle, the angular velocity, angle, and orientation with respect to the coordinate system is open to know because of the usage of the speed sensor, accelerometer, gyroscope, and GPS sensors on board. After all the training processes, all types of actors including the vehicles, pedestrians and sensors are destroyed.

2.2 Reinforcement Learning

Reinforcement Learning [25] is one of the machine learning methods which aims to maximize the reward returned by an agent in an environment. The training process can be divided into Markov Decision Process (MDP), which is a formal way to reconstruct the problem statement into different small groups which fits into reinforcement learning algorithm. Markov Decision Process (MDP) comprises state variables (S), actions (A), transitions between state variables and rewards (R) which assists the agent to learn in the right way.

The learner who makes decisions is called an agent. State variables are the instantaneous conditions of the reinforcement learning environment. An action is a movement or step changes of the agent. In our case, action is some position movement

that agent makes at time ‘t’ that causes changes in the prior state variables to the new state conditions which is the transition period. As the agent moved to the new state variable, the positive and negative rewards will be provided according to the new conditions which may be good or bad for the agent. The mission goal of the agent is to learn the policy that will keep it survive in the environment till the end while accumulating the highest reward possible which can benefit the overall result of the process.

2.2.1 The Agent-Environment Interface

The flow chart depicts the general procedure of how the agent works in an environment in reinforcement learning. According to MDP, reinforcement learning contains a set of states S, a set of actions A and a set of rewards R. At each time step t = 0,1,2, 3…, n the agent will choose either a random or best action A_t , where $A_t \in A$, based on the current state, S_t, where $S_t \in S$. This results in changing the agent position to the next state, S_{t+1} and the agent will receive positive or negative reward, R_{t+1} based on the bad or good consequence of the current chosen action on new state. Then, as the car got into the new state, S_{t+1}, it became the current state of the agent car, and the earlier process iterated [26].

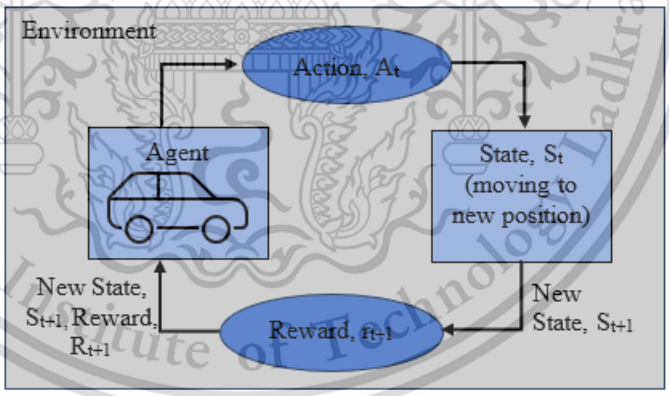


Figure 2-3 Reinforcement Learning Diagram

The goal of agent in reinforcement learning is to maximize the reward not in an episode but for overall training. Therefore, only the optimal action mode, that considers not only the immediate effect but also the future possible conditions, can return the maximum reward return. The policy of choosing the optimal action changes over time as the agent learns from the experiences. The policy, π: S ⟶ A, shows that the policy will give out

the optimal action by taking into considering all the environmental states of the current condition., that is, π (S_t) = A_t.

The reward of an agent is multiplied by a discount factor (γ) to determine whether to focus on the immediate return or the long run, and to scale down the reward to bound the total sum. The discount factor lies between 0 and 1, with a value of '0' indicating that the agent will be completely myopic and will focus only on immediate rewards over the future potential rewards. A discount value of '1' indicates that the agent foresees the future rewards and prioritizes them over immediate rewards by considering the future possibilities. The discounted current reward by considering the future rewards can be described as follows.

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots = \sum_{k=0}^{T-1} \gamma^k R_{t=k+1} \quad (2.1)$$

Q-function or action-value function is a strategy which the agent will follow to select any action by following the policy π. Q return for each state action pair is the estimated value for the particular action during the current state 's' at time 't' by following the strategy 'π'. Q return or Q value for any given state action pair can be calculated as follows.

$$q_\pi(s, a) = E_\pi[G_t | S_{t=s}, A_{t=a}] \quad (2.2)$$

Table 2-1 Conventional Q table Structure

state / **actions**	A_0	A_1	A_{2-}	…	$A_{n....}$
S_0	$Q(S_0,A_0)$	$Q(S_0,A_1)$	$Q(S_0,A_2)$	…	$Q(S_0,A_n)$
S_1	$Q(S_1,A_0)$	$Q(S_1,A_1)$	$Q(S_1,A_2)$	…	$Q(S_1,A_n)$
S_2	$Q(S_2,A_0)$	$Q(S_2,A_1)$	$Q(S_2,A_2)$	…	$Q(S_2,A_{2n})$
$S_{....}$	…	…	…	…	…

Each Q value for a state action pair is put into tubular form and the action value which returns highest Q value, max $q_\pi(s, a)$ in that given state is chosen to act in the environment. This Q-table values are updated based on the feedback rewards and the agent will keep finding the policy which will return the maximum total reward. When the agent gets into a new state, the new action for that specific state is chosen based on the updated Q values. The number of Q will be equal to the number of actions of the

agent. The action resulting the maximum estimated return is chosen as the optimal action for the current state and its respective policy is known as optimal policy, π^* . When it is represented in bell man equation,

$$q_*(s,a) = E[R_{t+1} + \gamma max_{\acute{a}}\, q_*(\acute{s},\, \acute{a})|s,a] \tag{2.3}$$

2.3 Deep Reinforcement Learning

In the reinforcement learning, Q-value is calculated by the Bellman equation and sort out in a Q-table in which all the states and possible actions are listed and matched to compute the highest possible action in a specific state. Therefore, the number of states and possible actions have limitations and cannot be deployed in big calculation complex scenarios such as real time computation. Moreover, determining the optimal action by using the maximum Q-value from the Q-table is infeasible when the number of state-action pairs are very large.

Therefore, Deep Q Learning [27] is used instead of Q-Learning method to replace the Q-table with neuron network layers. Neural networks in Deep Q Network are comprised with multiple layers which are more efficient in computing with large amounts of input state variables and actions.

Unlike other datasets, the possibilities of environmental state value in training an autonomous driving car are unlimited. Moreover, the data of the agent car cannot be

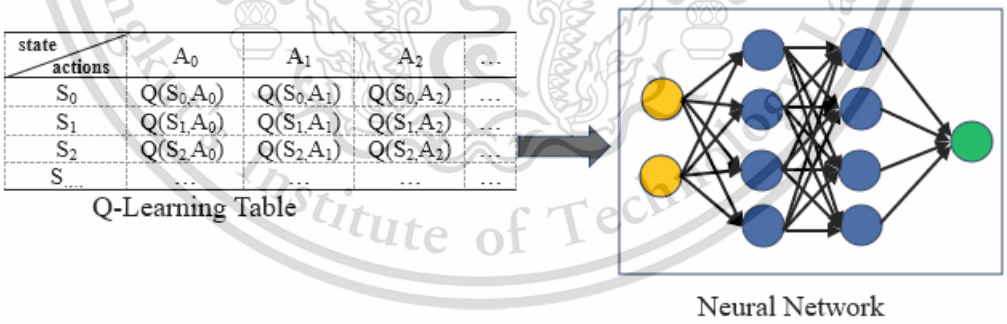


Figure 2-4 Conventional Q table Vs Neural Network

identical even if the same scenarios share the same coordinated points as the rewards of the agent car also rely on the duration of the simulation time. Moreover, it is impossible to plot into a q-table as the data is extremely huge and dynamic. Therefore, instead of conventional reinforcement learning method, which uses q-table, a deep Q network is replaced to make decisions on the input data and environment conditions.

2.3.1 Deep Neutral Network Architecture

An artificial neural network structure is inspired from how the brain cell works in a human brain. Like a human brain, it comprises with numerous interconnected nodes or neurons in a sequence layer structure and the data from the former layer are passed through several functions and only a particular data is transmitted to the succeeding layer to make an output decision. The following diagram **Figure 2-5** shows how a result is calculated from a neuron. Each neuron receives the data (x_1, x_2, ...) from the former layer and calculated with the weight and bias parameters then passed to the activation layer which decides whether it is qualified enough to connect to the neuron in the next layer or not.

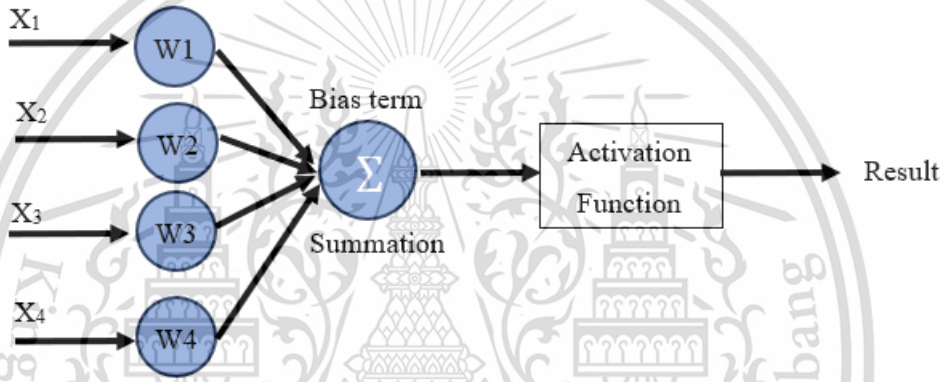


Figure 2-5 Components in a Neuron

$$Result = Activated(((w1 * x1) + (w2 * x2) + \ldots + (wn * xn)) + b) \quad (2.4)$$

The deep neural network is composed of an input layer, several hidden layers, and an output layer [28]. The first layer is called the input layer where the input parameters from the domain or environment are fed into the network. Hidden layers are known as dense layers as numerous neurons in each layer are fully connected to the neurons in the succeeding layer. Each neuron has its own unique weight and bias parameters. The input values to the neurons in the first hidden layer are the weighed sum values from the input layer. The output values of neurons in the first hidden layer are then multiplied with individual weight of the neurons in the second layer and added to the bias term. That weighted sum value is passed through the activation layer to decide whether the result is high enough to activate the neuron in the next hidden layer as input parameters. If the activated result is high enough, that neuron value is connected to every single neuron in the next hidden layer and again connected to the neurons in the next dense hidden layer. The complexity of the network depends on the

number of hidden layers, the number of neurons in each hidden layer and the size of the input data. Finally, the results from the last hidden layer are connected to the output layer of the network which will give the predicted result value. The deeper the network, the calculation cost will be higher, and the training time will be longer.

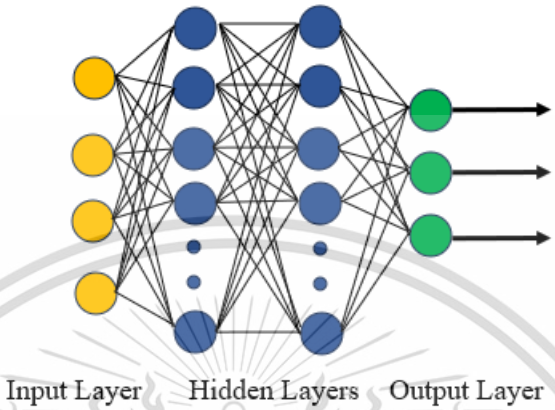


Figure 2-6 A Neuron Network Architecture

2.3.2 How the agent is trained in a DQN Network

Training an agent using deep reinforcement learning method is similar to how humans learn a skill set. The training may take hundreds or thousands of training episodes. In the beginning of the training journey, the agent will have no knowledge and memory data of the mission, therefore, it will explore the environment and commit random action steps. When an episode starts, the agent receives the current state values information of itself in the environment. Based on this current state values and other parameters, a random action or an action calculated from the neural network will be proceeded. Positive and negative rewards will be given as feedback based on the actions of the agent. Based on the feedback rewards and error loss function, the network parameters in the neural network are updated regularly. If the network collected sufficient training experience and trained adequately, the network parameters and the policy will be optimum. The output decision by the neural network will lead the agent to accomplish the mission while returning the highest possible reward for that state value. The trained model can be assumed as a good model if the loss function decreases, and the reward value increases as training episodes increase. Otherwise, the network parameters are tweaked and tuned iteratively to get the best parameters which will return the highest possible reward and performance.

2.3.3 Weight Initialization

The neural network learns the policy by optimization of the weights and bias linked between the neurons. Commonly used weight initialization methods[29][30] are 'Uniform initialization and Normal Initialization'. 'Uniform Initialization' is prone to 'exploding gradients problems' if initialization starts with large weights values, leading the parameters to explode during the back propagation for updating the weights. Whilst 'Normal Initialization' favors 'vanishing gradients problem' when the weights are setup with tiny numbers. This is because no prior information about the number of inputs and outputs is considered in the above two methods. Therefore, the new proposed weight initializer methods such as Xavier Initialization, He Initialization and LeCun Initialization which consider input and output units in each layer are used to mitigate the vanishing and exploding gradients problems.

"He initialization method" is suitable to use with ReLU activation functions as it is used for non-linear activation functions. For "He normal initialization", the weights are set to be normally distributed within mean 0 and standard deviation of sqrt(2/n), where 'n' is number of inputs to the layer. For "He Uniform distribution", the weights values range within [-value, +value], where weight value = sqrt(6/n), and n is the number of inputs to the layer.

2.3.4 Loss Function

A loss function is a measurement of how good the machine learning model is in conducting a certain mission goal. It is the total error sum of the difference between the target desired output and the predicted actual output. If the error is small, it means that our predicted output is close to the desired output value, showing that the network model is working in a good condition. Vice versa, the error will be high if the two output values are different, which indicates that more training is needed, or the parameters should be updated. The loss function or the error value of a machine learning model should decrease gradually if the model is learning the ideal policy after training has been done for a certain long time. The error from the loss function may be reduced using the optimizer function and then network parameters are updated. 'Mean Square Error (MSE)' and 'Absolute Mean Error (AME)' are widely used together with the numerical data models using ReLU and linear activation functions. The loss function can be expressed as

$$Loss = E(target\ Q\ Value) - E(predicted\ Q\ Value) \tag{2.5}$$

$$Loss = E[R_{t+1} + \gamma \max_{\acute{a}} q_*(\acute{s}, \acute{a})] - E\left[\sum_{k=0}^{\infty} r^k R_{t+k+1}\right] \tag{2.6}$$

2.3.5 Activation Functions

Activation functions are key parameters in designing the neural network. Activation layer creates non-linearity in the network which helps and improves the learning process of the agent model as it helps to learn complex features and conditions. If the activation layer is removed, the network will become a linear equivalent process which shows lower performance in complex data computation. The main function of activation functions is to decide whether to transmit the output value to the next layer or not. The input to the activation function is the weighted sum of the output values from the prior layer and the bias (if present), represented as:

$$Activation\ layer\ output = f(((w1 * x1) + (w2 * x2) + \ldots + (wn * xn)) + b) \tag{2.7}$$

where w_1, w_2, ..., w_n are the weights associated with the current neurons, x_1, x_2, ..., x_n are the corresponding output values from the previous layer, and 'b' represents the intercept (bias) term.

The activation function determines whether that weighted sum is triggered and linked to the neurons in the next layer. The neural network's potential and effectiveness are greatly influenced by the activation function. The same activation function is commonly used for the hidden layers in a network and the different type of activation function is used for the output layer depending on the result output type.

In 'Sigmoid Activation Function', the inputs are the real numbers while the outputs range between 0(min) to 1(max). In 'Tanh Activation Function' which is known as hyperbolic tangent function, the output values range from -1(min) to 1(max). As the output values from those two functions are relatively small, the network works properly during the forward pass, however, during the back propagation process for updating the weight parameters, vanishing gradient problem can occur. The derivative of the Sigmoid and Tanh functions are small gradient values so when they are multiplied with gradients from the preceding hidden layers, the results can be as small as zero gradient value. If the gradients are too tiny, the initial weight and biases will not be adequately updated in each training session, therefore, the model network will not learn anything from the training. Although the training process is proceeded, the performance of the

agent will not improve. Therefore, the 'Rectified Linear Unit' (ReLU) became a widely used activation function for deep learning algorithms as it eliminates the vanishing gradient problem.

2.3.6 Rectifier Linear Unit Activation Function

ReLU activation function[31] is commonly used for the hidden layers as it is easy to use and able to overcome the drawbacks of other formerly well-liked activation functions such as Sigmoid and Tanh. During the forward propagation, the equation of the ReLU activation function is that if the input to the function is less than 'zero' it will return zero otherwise the output value is the same as the input value. Its mathematical expression is represented in **Eq (2.8)**.

$$Rectified\ Linear\ Unit, f(x) = \max(0, x) \tag{2.8}$$

For the back propagation process, the derivative of ReLU activation function is required to update the weights and biases of a node. As the derivative of the output zero is "zero" and the linear positive trend is "1", so, the gradient outputs will not vanish during back propagation even multiple hidden layers are used as long as the weighted output from each neuron nodes are positive. The activation function can result in zero value when the input to the activation function is negative. If the zero result occurs seldom, the agent can learn from the training, however, if the output values from the activation layer are predominantly influenced by zeros, then the agent cannot learn from the training. To prevent this kind of phenomenon, assigning the correct weight initializer is important.

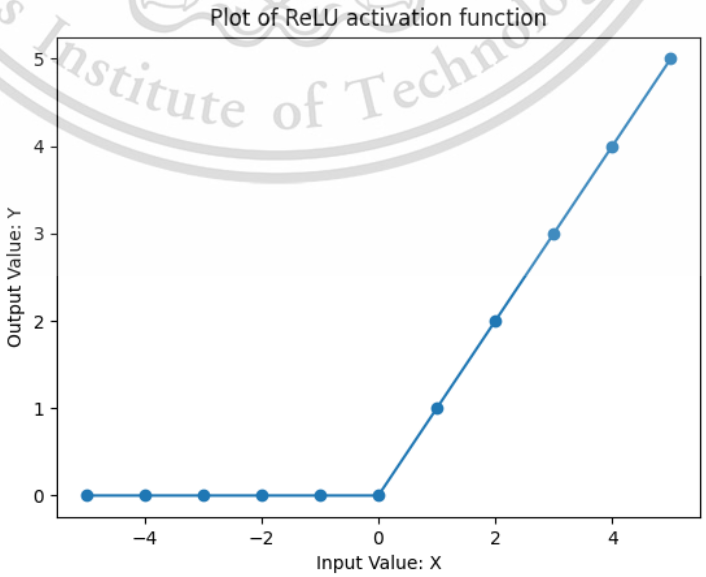


Figure 2-7 Rectified Linear Unit Activation Function

2.3.7 Forward Propagation

Prediction of the output from the first layer to the final layer of a neural network is called the forward pass method. The value of the neuron from the prior layer is multiplied with its respective weight parameter and then added to the bias value (if present). Then the sum of the weighted value is passed over to the activation function and the activated result is linked to the neurons from the following layer. This calculation process continues from the first layer until the last layer concurrently to find out the agent's action which will return highest possible profit in the given state value. This is called the forward pass. After a forward pass, the weights and biases of the network are updated by the back propagation method to get the optimal policy for the desired task.

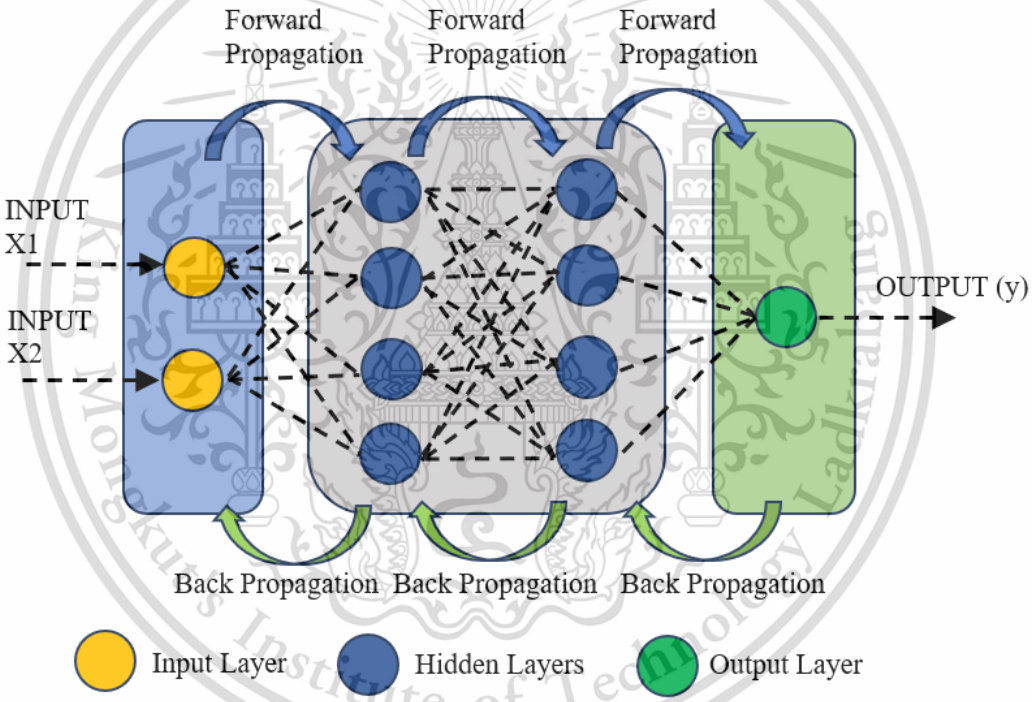


Figure 2-8 Forward Propagation and Backpropagation

2.3.8 Back Propagation

Back propagation [32] is a technique for computing and updating the gradients by calculating partial derivatives of the loss from the loss function with respect to the network's parameters (weights and biases), layer by layer starting from the output layer and moving backward through the layers using the chain rule depicted in **Figure 2-8**. Chain rule is used to differentiate a composite function and the chain rule equation for

the composite function, h =g(f(x)) can be expressed as $\frac{\partial h}{\partial x_i} = \sum_j \frac{\partial h}{\partial u_j} . \frac{\partial u_j}{\partial x_i}$, where, h is the net output, x is the input and u is the output of the inner function.

The weight and biases for each neuron are updated using the gradients through this backpropagation, chain rule method, therefore, the weight value of each neuron depends on the weights of the other proceeding neurons. Therefore, the larger, deeper and the more complex the network becomes, more differentiation steps are conducted to change each neural network parameter to lower the error value. This parameter tuning process will be iterated until the network parameters become ideal, so that it will return correct prediction output values that will result in small error values or low loss function.

2.3.9 Target Network

In deep reinforcement learning, the neural network uses 2 networks, the original network model to predict the optimal result value for a state-action pair and the second network, used to calculate the target result value for that state-action pair. The target network[33] has the identical network architecture as the original network despite its weight values are frozen for a certain time steps to provide more stable and efficient training results.

If both target Q value and predicted Q value are for a state-action pair are calculated from the same network, it will be difficult for the predicted Q value to catch up with the target Q value as the weights and biases are changing equally in every update. Every time the predicted Q value is calculated with the network parameters, the target Q value is also calculated using the same network parameters, therefore, it may be like running after a changing target, which is unable to seize. This method is ineffective and does not guarantee the stability of the training performance.

Therefore, a separate network also known as a target network is used, and it is one of the important factors in deep reinforcement learning. Unlike the earlier target Q value calculation method which shares the same network, the target Q value is calculated by passing the state value $(\acute{s})$ of the following time step to this target network to find the maximum q value $\max_{\acute{a}} q_*(\acute{s}, \acute{a})$ of the state $(\acute{s})$ in the following actions $(\acute{a})$ for the same state action pair used in the original network. The target network comprises of the same network architecture as the original network while the weights and biases (network parameters) are fixed for a certain time. Therefore, it is much easier to approximate the predicted Q value to the target Q output values for the same state-

action pair as the weights of the original network updated consistently towards the fixed targets steadily which helps to improve the training. After training for certain time steps, weights of target network are updated with the original neural network weights to keep pace with the changes and fixed it again for next time t-steps.

2.3.10 Experience Replay

Other machine learning methods like supervised learning, the training data are fed to the neural network to train the network model. However, in our case, there is no pre-trained data to learn. Hence, experience replay memory[34] is used to save the data in a tuple array. The data from each time step, the agent's every course of action and observation starting from the beginning (such as current state, action, reward, and next state values) are kept in a memory array by adding the newest data on top of the other previous data. The size of the memory array is variable and changeable by the model and once the memory is full, the oldest data is automatically removed and replaced by the new ones.

During the training of the model, the memory from the experience replay is randomized and subset into a small size of batches to prevent the catastrophic forgetting process and generalize the training. Choosing a batch of random samples from the replay memory can improve the learning process as the network model faces diverse environment from the older and newer samples. And selecting the samples from the subsequent experience can make the agent's performance good only at a specific location and forget all the training that had learnt a short while earlier, which is not desired. This is another vital reason for using replay memory, to break the correlation between consecutive samples. Instead of training with a big size of experiences consecutively, it is broken down into several small size batches and those limited size training samples in batch by batch are used to train the agent so that the model can generalize the data and reduces the training time.

2.3.11 Epsilon Greedy Strategy

Epsilon is one hyperparameter value that is pre-assigned before the training. It is used as a measuring parameter to decide whether the action of the agent should be randomized, or action resulted from the neural network. The epsilon value (ε) varies between a value of 0 and 1. In the beginning of the training, the epsilon value is assumed to be 1. So, the agent will explore the environment by choosing random actions to collect experiences. At the end of each episode, that epsilon is multiplied by a constant

epsilon decay number to shrink the epsilon value by a small amount. A uniform random number is generated to decide whether to choose the action predicted by the neural network or select the action randomly. Then that newly assigned epsilon value is compared to the randomized number. If the epsilon value is larger, the agent will do the randomly chosen action, otherwise, it will do the selected action from the neural network. The epsilon value will be smaller and finally equal to the minimum epsilon value as numerous training episodes have been trained. Because of the epsilon decay strategy, as time passed, the randomized value will be larger than the newly assigned epsilon in each episode. And this will encourage the agent to be more confident and rely on its updated weight parameters in the network to calculate the output action.

If the agent explores the environment all the time, it will follow random actions and training will be ineffective. On the other hand, if the agent uses exploitation method without having any training experiences, the decision output that the network gives will be ineffective. If the network is sufficiently trained enough, the calculated output value can be optimum however, if the agent always chooses exploitation method, the agent may sometimes stick in the local minima and cannot be able to find the global minimum, a new solution that could lead to better results. It is important to have a balance between exploration and exploitation during training. If the epsilon decay rate is big, then the epsilon value will decrease significantly therefore, the agent cannot be able to save enough experience from random actions. If the epsilon decay rate is small, then, the agent will rely only on the random actions for a long time.

2.3.12 Rewards

The goal or purpose of the agent is to maximize the cumulative reward for each episode but not for a single step of action during the episode. Rewards are the critical parameters in reinforcement learning process as they shape the behaviors of the agent and from the feedback rewards, the agent can learn the optimal policy to accomplish its assigned task. Rewards provide feedback information of what action is desirable and what is undesirable for a specific condition therefore the agent can redo or avoid that kind of similar behavior action if the same state is repeated in the following training.

Positive rewards and negative rewards are commonly used to craft the agent behavior to complete the assigned task. Positive rewards encourage the agent to repeat the similar action while negative rewards are the warning signals to the agent that those are unwanted behaviors during the training process. If the reward conditions are not

well shaped, after several episodes, the agent will probably discover another unexpected solution that can gain more total reward than the desirable way of finishing the assigned task. Moreover, in shaping the rewards, it is crucial to note that the agent is incentivized by the reward it achieved due to its behavior not what the operator intends the agent to do[35].

For instance, if the mission is to get to the final point which gives the agent +10 points and terminates the episode. On the way to the final location, the agent will get a small positive point of (+1) by approaching nearer to the final point. If the reward setting is not well designed, the agent may find out that eating (+1) small points continuously by moving back and forth around that final location till the end of the episode can return higher total points compared to the desired way which is worth only (+10) points. This is one kind of unexpected behavior that the agent misinterpreted the reward which we intended for the other purpose.

And the negative rewards help the agent to terminate the game or training episode as quickly as it can when the agent is losing the rewards continuously, which is not desirable. The penalty size varies with the impact of accident caused by the agent. Sometimes, the agent commits small mistakes consecutively, and if the accumulated small negative penalty is larger than one big mistake, then, the agent will realize that committing this huge mistake results in higher total rewards. Therefore, the agent will choose the bigger one and end the episode. Therefore, even though the human operator crafts the rewards in a certain way, the agent can discover another unexpected behavior solution which is more profitable to the total reward value.

In designing a reward, a regular continuous feedback reward helps the agent to do the task more quickly compared to a sparse reward, which is rewarded only when the desired goal is satisfied. A continuous small reward, either positive or negative, is the result information of the current action. The agent knows its updated current state conditions, therefore, if the agent is following the right path, the agent gains confidence to repeat the same steps when the state condition is similar and even when the action is bad, it can recover from the mistake in a short time. There is no definite way to design the reward functions, and even a small change in reward parameter can cause big differences to the behavior of the agent. Therefore, it is important to craft the reward function that can direct the agent to act exactly what we want it to do, and the best reward function can be derived by trial-and-error method from working on the feedback of the action of the agent.

CHAPTER 3

METHODOLOGY

Chapter 3 covers the research methodology of the project which can be categorized into design scenario, simulator environment and DQN model architecture. The design scenario is a synopsis explanation of this case study and generally describes where and how the agent is spawned in each event and the location of the parking destination. The Carla simulator is used to create the simulation environment, the training agent and interaction between the agent and the environment. DQN model architecture is used as a control algorithm, which is used to find the ideal strategy that can accomplish the desired parking task.

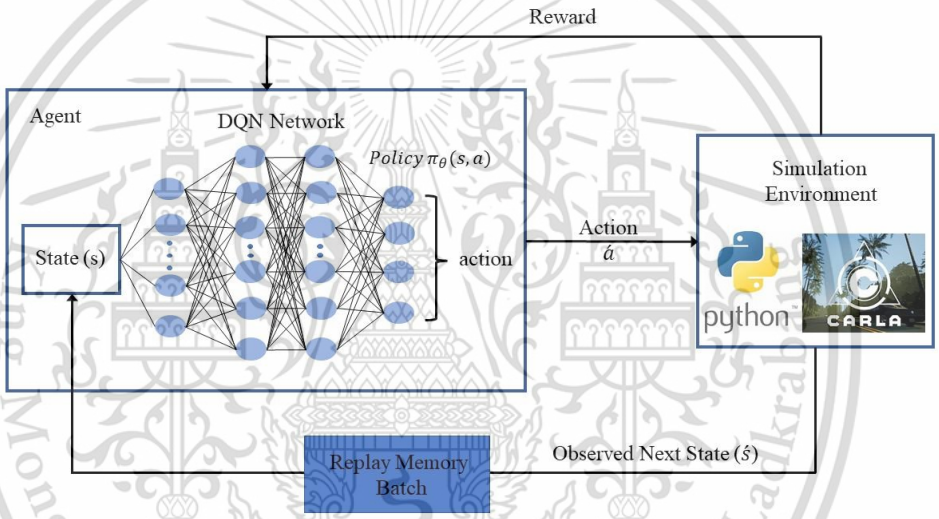


Figure 3-1 Design Model to interact Simulation Environment and DQN Network

3.1 Design Scenario

In vehicle driving systems, parking process is time consuming especially for the novice drivers. After the human driver has driven the car and parked manually at the parking destination, he has to head back to the desired place, which might be wasting time. Especially at the peak hours and at large parking area, it is obvious that parking takes more time and energy drainage than normal hours. Therefore, if autonomous parking system could be replaced for manual parking task, it might save the time and bring more comfort to our daily lives. In our designed scenario, this autonomous parking process is categorized into 3 parts depending on the difficulty level of parking.

Scenario 1: Drive the agent car from a fixed spawned location to a fixed parking destination, which is 10 m distance.

Scenario 2: Drive the agent car from a random spawned location within an assigned boundary, to a fixed parking destination which is 25 m distance.

Scenario 3: Drive the agent car from a randomly spawned location within a parking boundary zone and random yaw spawned angle, to a fixed parking destination.

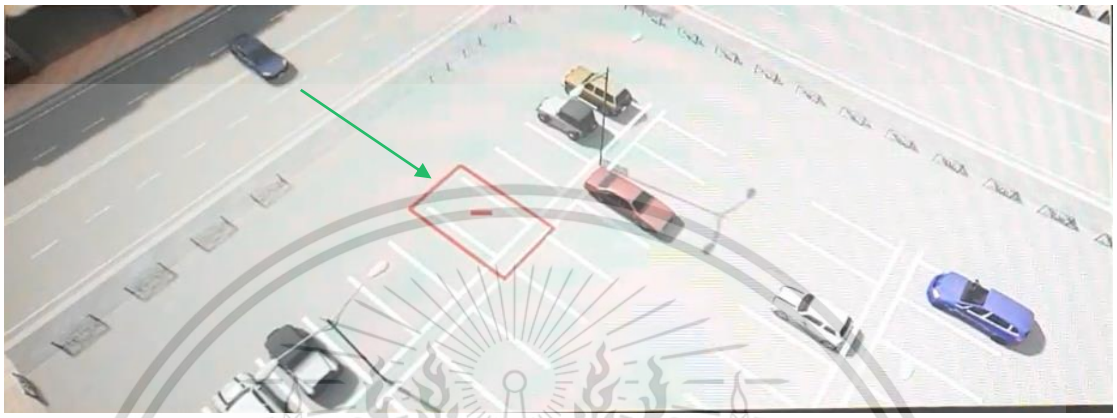


Figure 3-2 Demonstration of Scenario 1

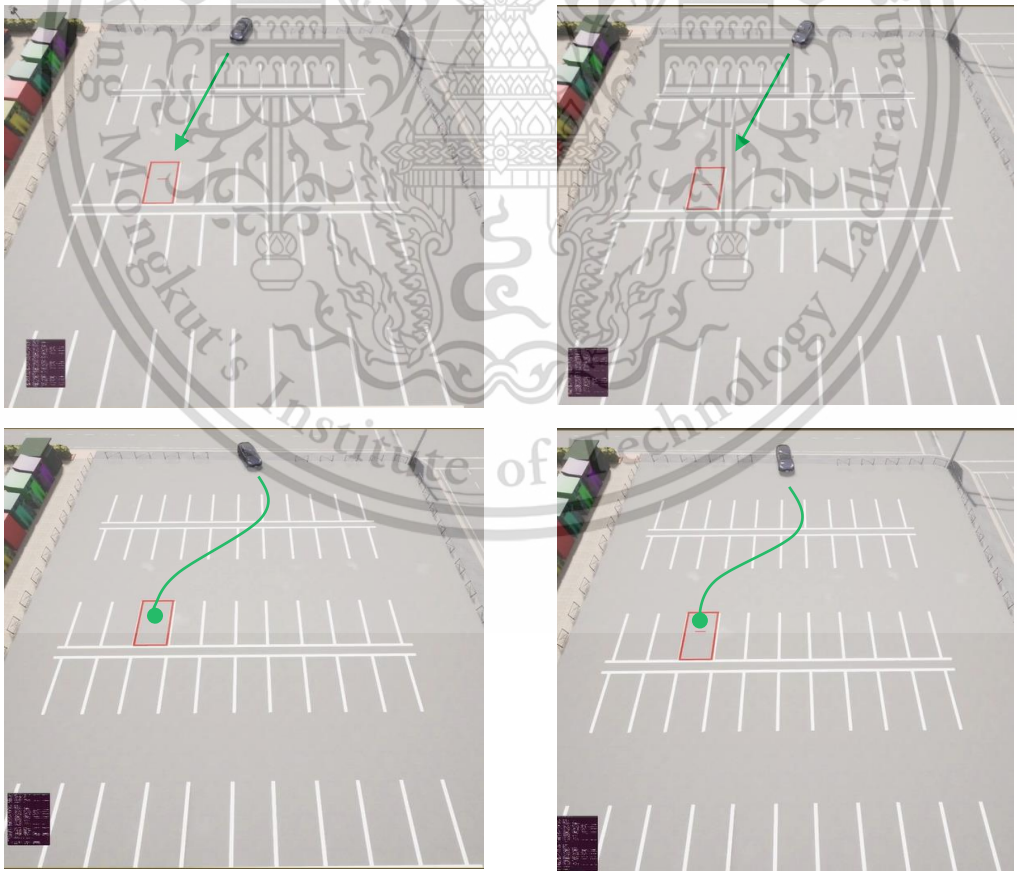


Figure 3-3 Demonstration of Scenario 2 (with different spawn angle and location)

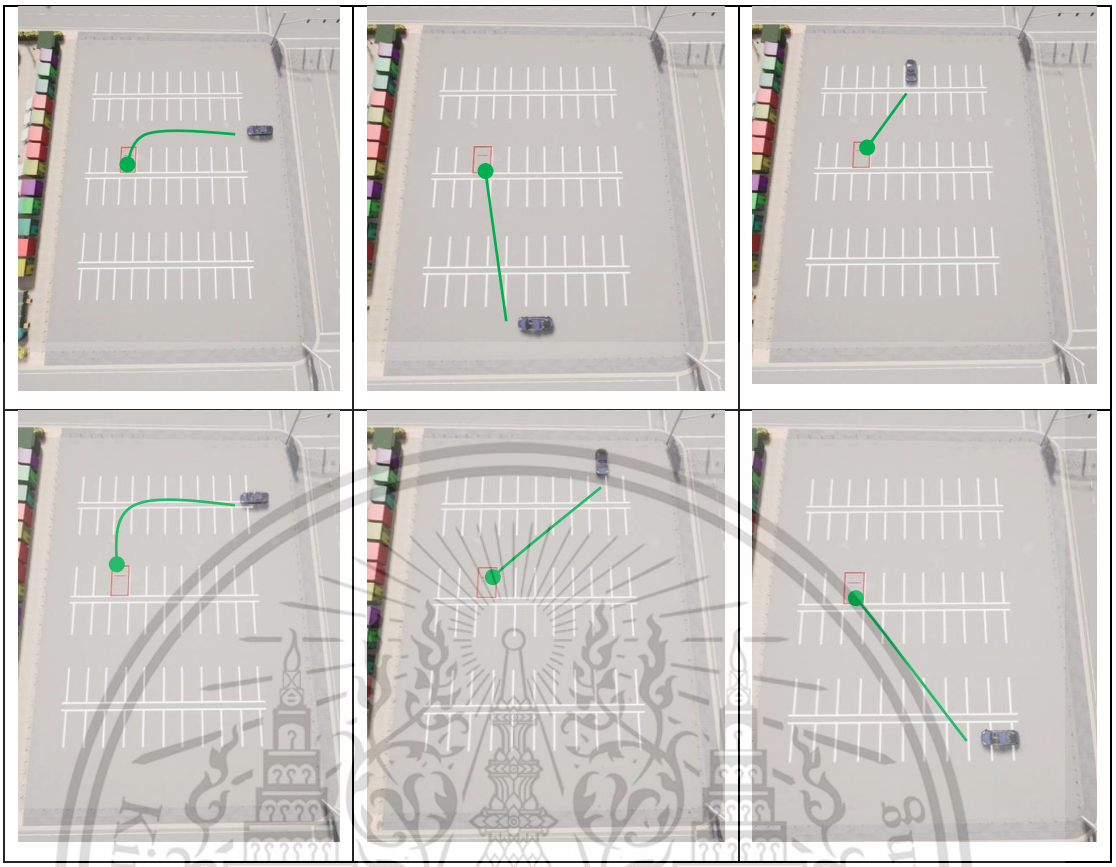


Figure 3-4 Demonstration of Scenario 3 (with different spawn angle and location)

3.2 Carla Simulator

Carla is an open-source 3D visualized simulator which is facilitated with infrastructures, vegetarian and diverse vehicles which support autonomous driving environment. Carla supports maps with different urban designs. All types of vehicles including agents, non-character players and sensors are regarded as actors in this environment. Various types of sensors can be used depending on the needs of the driving event. The Carla simulator comprises the server side and client side. The server side covers the task of initializing and loading the environment and renders the scene. In the client side, the agent will make dynamic action movements which cause changes in agent's state in the environment. Python API is used as a communication tool to transfer data back and forth to the server and client side. All the actions and changes made by the agent on the client side are sent back to the server side to save the changes and render back to update the client side.

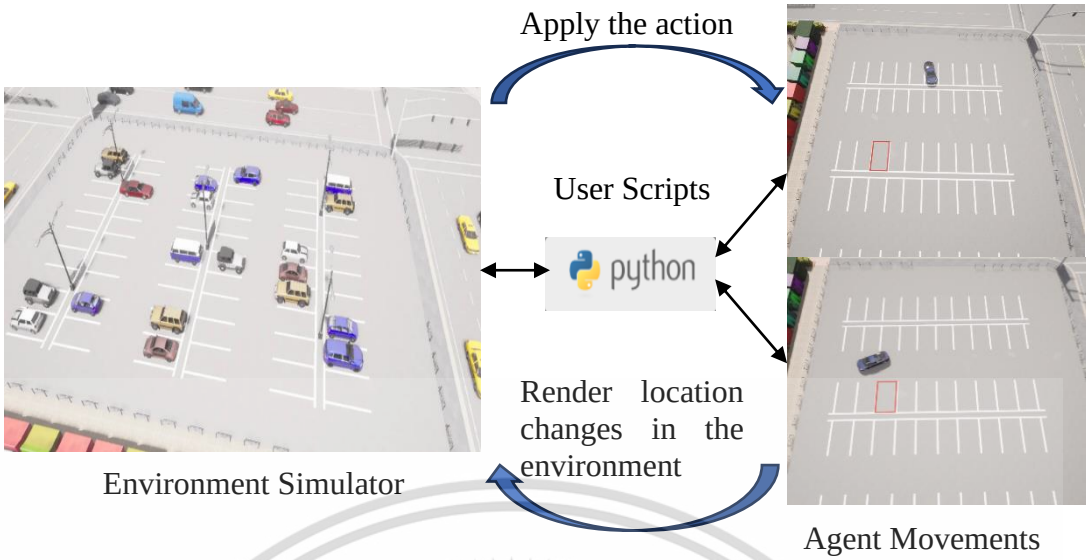
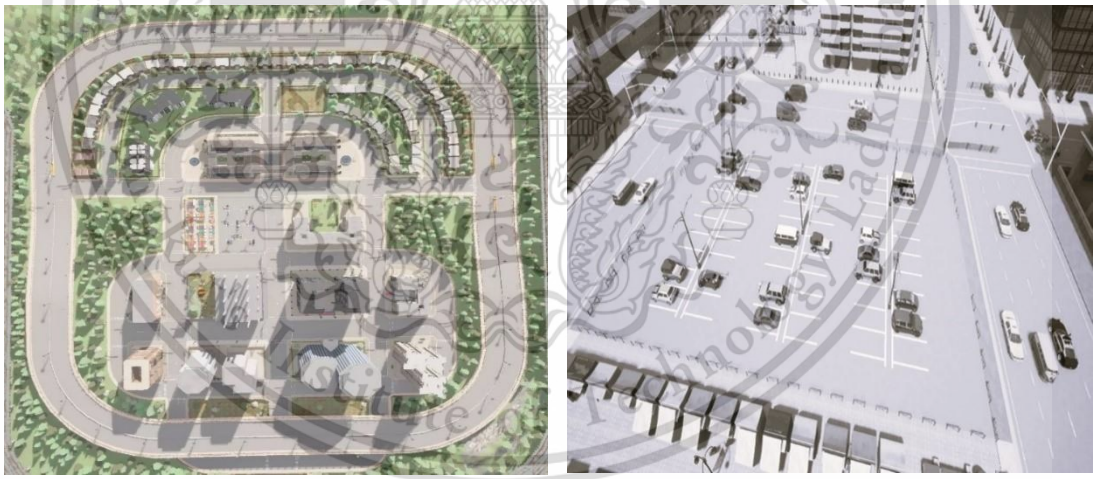


Figure 3-5 Basic Structure of Carla Simulator

3.2.1 Parking Area Town 5

Town 05 is a square-grid town with cross junctions and a bridge and multiple lanes in each driving direction. Carla supported Town 5 is used throughout the training process as it provides a large space for car parking area which occupies one city block that is larger compared to the other parking area in other towns.



Overview of Town 5 Parking Zone Area

Figure 3-6 Urban Design of Town 5

3.2.1.1 Assigning the Parking Area and boundary distance

In this project, a ground level parking lot with double entrances, bounded by a fence is chosen as the training space. The fence acts as a boundary barrier for the agent. Carla simulator supports the point locations and spawn point locations to simplify the usage of the simulation environment, however, our training parking zone area is out of support from spawn point location information. Thus, the environment's x and y

coordinates are manually noted to save the location of the assigned parking space. After the environment has started, x and y coordinates of the 4 corner points of the parking space are stored to assume as the parking zone boundary. Those values are then normalized to set the center coordinate of that boundary which is used in the training.

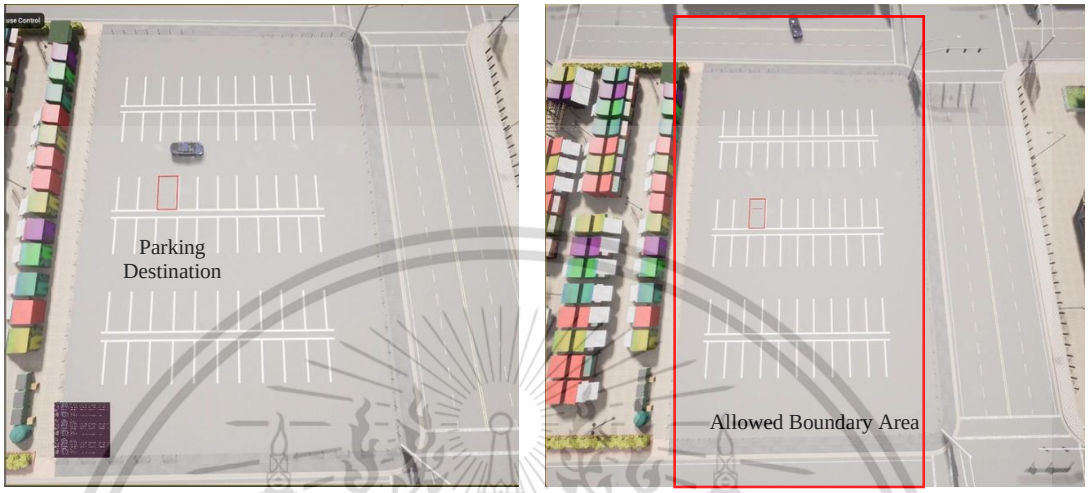


Figure 3-7 Parking Zone Destination and Parking Boundary Area

The boundary limit is set to prevent the agent car driving too far away from the destination point. If the agent car exceeds over the boundary distance limit, the system will assume that the car cannot be able to drive back to the desired location within the remaining training time. As it deviates from our training purpose, the episode is stopped, and a negative penalty is given.

3.2.2 Initialization of the vehicle and sensors

In our Carla simulation scenario, a Tesla Model 3 is used as a training agent car. The vehicle is initialized and selected from the Carla Blueprint Library. The collision sensor is also spawned from the blueprint library, and it is then attached to the host vehicle. Although various and numerous sensors can be used for an autonomous self-driving car, only a collision sensor is used as a blueprint sensor in this training as other sensors are assumed to be mounted on our agent car. Besides the collision sensor, the Euclidean distance of the car, the yaw angle of the car and the speed of the car etc…. are available from the agent car in real-time experiment. The collision impulse can be calculated by

$$intensity = \sqrt{x_{int}^2 + y_{int}^2 + z_{int}^2} \quad , \tag{3.1}$$

where x_{int}, y_{int}, z_{int} = collision intensity in x, y, z direction

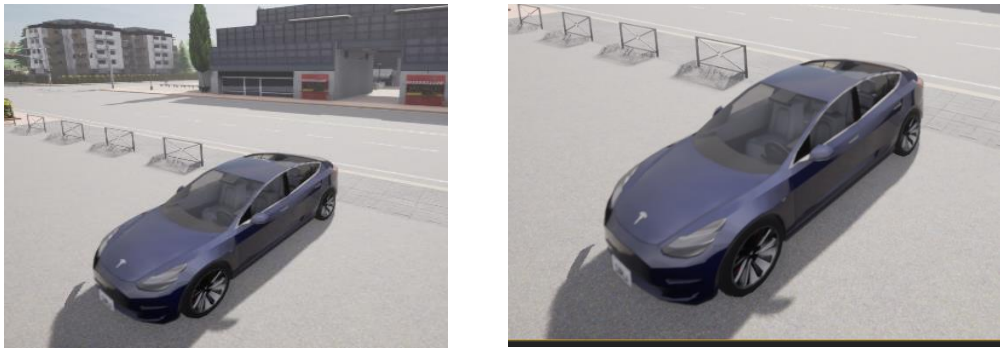


Figure 3-8 Tesla Model 3 in Carla Simulator Environment

3.2.3 Vehicle Control

The turning angle of the agent car Tesla model 3 is limited to a maximum of 70 degrees to the outer direction in both the left and right wheels and 48 degrees for inner wheel from the neutral position while the speed of the forward and reverse can be variable. Resembles to the locomotion of a real car, it supports 4 movement actions: the drive forward command, left and right steering command, the reverse command, and the brake command. The throttle parameter is used in each command to control the speed of the driving action of the agent.

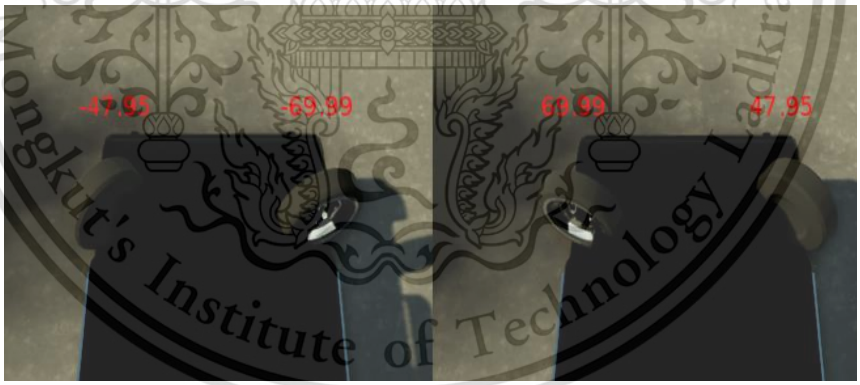


Figure 3-9 Wheel Angle variation for Tesla Model 3

3.3 Neural Network Architecture

A control system is a crucial component in autonomous car as it is responsible for making an action decision of where the vehicle should move for the next time step. The control algorithm can be different depending on the input, output types and size of the data and other simulation facts. In this training event, a deep neural network is used as a control algorithm as autonomous driving system needs enormous environmental information as inputs to the control system. Hence, it needs complex algorithm to

calculate the action solution. The conventional Q table might not be robust enough to handle, therefore, a deep neural network is replaced to predict the most suitable action outcome which brings the highest reward return and accomplish the mission.

Important components in DQN architecture are state values, the model setting, action setting, reward setting and hyperparameter settings. The environment state values of the agent are used as inputs to the input layer. Based on those inputs, the neural network predicts the most appropriated action output for this state by taking account the possible future rewards. The action chosen has the highest possible future rewards among all actions for this state. Then the agent will make a move according to the action, resulting in changing the agent to a new state. The agent receives a reward or penalty for every action it makes and stores these experiences as a memory buffer. They are again used for training the target model, which determines the optimal approach for maximizing rewards. If the neural net is well equipped with reasonable inputs, hyperparameters, outputs and rewards system, then the agent can learn the optimized theory algorithm effectively so it could show great performance in the mission. If one of the parameters is poorly tuned, the agent performance will be significantly worse, or the agent probably does not learn at all even after full training has been done. The details of how those components in DQN model are arranged and setup are explained below.

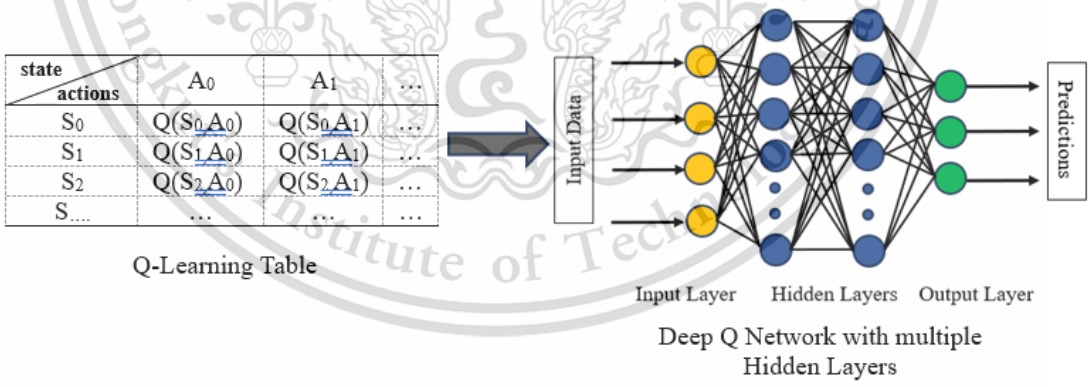


Figure 3-10 Q-Learning Table and Deep Q Network Architecture

The Algorithm for the Deep Q-Learning method with Experience Replay

Initialize replay memory D to capacity N

Initialize action-value function Q with weights θ

Initialize target action-value function $\hat{Q}$ with weights $\theta^{-} = \theta$

For episode = 1, M **do**

Initialize sequence $s_1 = \{x_1\}$ and pre-processed sequence $\phi_1 = \phi(s_1)$

For t=1, T **do**

With probability ε select a random action a_t

Otherwise select $a_t = argmax_a\, Q\,(\phi(s_t), a;\, \theta)$

Execute action a_t in the emulator and observe reward r_t and image x_{t+1}

Set $s_{t+1} = s_t, a_t, x_{t+1}$ $and\ preprocess\ \phi_{t+1} = \phi(s_{t+1})$

Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ $in\ D$

Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ $from\ D$

Set $y_j = \begin{cases} r_j & if\ episode\ terminates\ at\ step\ j+1 \\ r_j + \gamma\, max_{\acute{a}} \hat{Q}\,(\phi_{j+1}, \acute{a};\, \theta^{-}) & for\ non-terminal\ \phi_{j+1} \end{cases}$

Perform gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ w.r.t the network parameter θ

Every C steps reset $\hat{Q} = Q$

End For

End For

3.4 Model Setting

Deep Q Network is a more complex version of a vanilla Q network as it comprises two or more hidden neuron layers, and it is used to perform more complex calculations. There is only one input layer and output layer, however, the number and type of the layers in the middle part can be different depending on the task and the input data types. The number of neurons in the input, output and hidden layer can also be varied as per system environment.

There is no definite rule of assigning the neurons in the hidden layer. The model grows more complicated as layers go deeper and more neurons are added to the hidden layers. The number of neurons in the hidden layer are usually assigned as a power of '2' such as 32,64,128 etc... Starting with a small size network model with a few layers

can probably give a good result in a small easy training process. Then increase the number of neurons and layers gradually to match the complexity of the training task to avoid overfitting and underfitting cases.

3.4.1 Input and Output Hyperparameters Setting

The performance of the agent also depends on the type of input parameters to the neural network and output parameters from the neural network. In particular, if suitable data is not trained as input values, the training performance will not be good as we expected and low reward values will be returned, the mission will not be accomplished. Therefore, it is crucial to include all important data as inputs to the DQN model which will lead the agent to accomplish the training task. For output hyperparameters, 4 action movements can be done by the agent in this study, which are driving in straight direction, left and right steer, brake, and reverse control. In each driving action, the throttle amount can be varied to control the speed of the car. The driving action is operated according to the predicted action from the output layer of the DQN network.

3.4.2 Reward Setting

Setting appropriate reward hyperparameters are crucial as the agent learns the optimum policy from the reward feedback of the actions. Generally, the reward can be categorized into positive and negative rewards. Positive rewards are provided when the actions benefit the agent to reach the goal such as approaching nearer to the desired point, while the negative rewards are used to warn the agent to avoid those similar actions in the further training. The main reward is the total sum of individual rewards calculated based on the Euclidean distance, the angle difference between the yaw angle and the desired heading direction to the destination, and the time consumed. Bonus rewards and penalties are added to the main reward based on the actions of the agent. An extra reward is added for accomplishing the parking task while that extra reward value varies depending on the time the agent spent getting to the destination. Extra negative penalty reward is added when the agent hits the obstacles, or the agent is too far from the destination. There are also small add-on penalties when the agent stands still at someplace in the parking zone without moving at all. Moreover, to encourage the agent to finish the task within the limited simulation time, a small penalty reward is given if the agent is still trying to park when the simulated time has elapsed.

3.4.3 Network Hyperparameter Setting

In Deep Q Network, the prediction accuracy of a machine learning model and the performance of the agent are influenced by the hyperparameter tuning values and the reward setting. Numerous parameters such as number of nodes (neurons) in the hidden layer, number of hidden layers, the learning rate, type of activation function, network weight initialization, etc... are needed to be adjusted to get a good performance DQN model. The learning rate is the decay step size in finding the global minimum. If the learning rate is too high, the decay points will bounce off and will not converge quickly. On the other hand, if it is too small, the decay step is so close that the training duration will be long. After calculation of the weight parameters, those values are passed into activation function which is a strategy to create non-linear outputs. If the activation functions are omitted, the output will be a linear function result. The type of activation function depends on weight initialization and the type of data being processed. ReLU activation function is commonly used to lessen the vanishing gradient and exploding gradient problems. Weight initialization is setting the initial weight values within a suitable range so that vanishing and exploding gradient problem can be solved. He initializer works together with ReLU activation function. Other tunable hyperparameters are the number of epochs, batch size, epsilon decay rate and drop out function, to reduce overfitting.

CHAPTER 4
RESULT AND DISCUSSION

Detailed information of trained scenarios and the results will be discussed in this chapter 4. The detailed information of each scenario will cover boundary conditions of how the environment is set up in the Carla Environment, the hyperparameter settings for the neural network and the reward settings for each individual training. The simulation result for each scenario will be visualized in graph illustration and the performance result will be evaluated.

4.1 Preparation of Simulation Environment

Carla simulation environment is used as the training background throughout the training process. The preliminary preparations such as the parking map initialization, spawning the agent and sensors from the blueprint library and assigning the parking destination are proceeded prior to the training process, which will be discussed in depth in the following content.

4.1.1 Carla Simulation Environment Setup

The virtual environment for this project is set up with Carla Simulator as its driving environment is very realistic and the infrastructures, terrain situations, agents and the sensors can be customized. The training process is performed at the parking zone of Map 5 as described in **Figure 3-6**. As a prerequisite of the training, the location of the spawned position of the agent car is initialized first before the training. The spawn points are not available within the parking area lot; hence, the coordinates of the desired parking area are saved using the separate program and saved as .csv file. Then that data is reloaded in the main program file to set the boundary space for the parking area. The spawned position of the agent and the parking destination point are changed in each scenario to advance the training and will be explained in depth in each training scenario. Tesla Model 3 is used as an agent car throughout the training and vehicle motion is controlled by the steer action, straight forward action, reverse and stop actions. The throttle parameter is used to control the speed of the car during parking which can be varied between 0 to 100 percent. Other sensors are assumed to be mounted on the car and only the collision sensor is mounted as an extra sensor on the agent vehicle.

4.1.2 Parameters Tuning for the Parking Process

In training a neural network, the key parameters that affect the performance and accuracy of the agent are model setting, reward setting, hyperparameter setting and input, output data setting. In this study, various parameters are tuned and tested in each category and the generalized form of the hyperparameter settings will be explained.

4.1.2.1 Network Model, Input and Output Parameter Setting

Although the DQN model is used in all training scenarios, the number of input and output parameters, the number of neurons and layers will be different in each scenario. The network model for the second scenario will be explained as a standard form as it is the best model setting that gives the best performance and result among three scenarios. DQN network for the second scenario comprises 7x64x64x4 structure with 7 data values in input layer and 4 action values in output layer. Two hidden fully connected layers which contain 64 neurons are used to process the data.

The output layer returns 4 action values representing the driving control of the agent in a specific state - the forward, reverse, stop and steer locomotion of the agent. Input data values are the current state information of the car and '7' data in total are fed into the neural network. They are

- the relative distance in x and y direction from the parking coordinate to the current agent position,
- the Euclidean distance,
- the yaw angle of the car,
- the velocity in x direction,
- the angular velocity,
- the angle difference (α) between the yaw angle and the line connecting two coordinates.

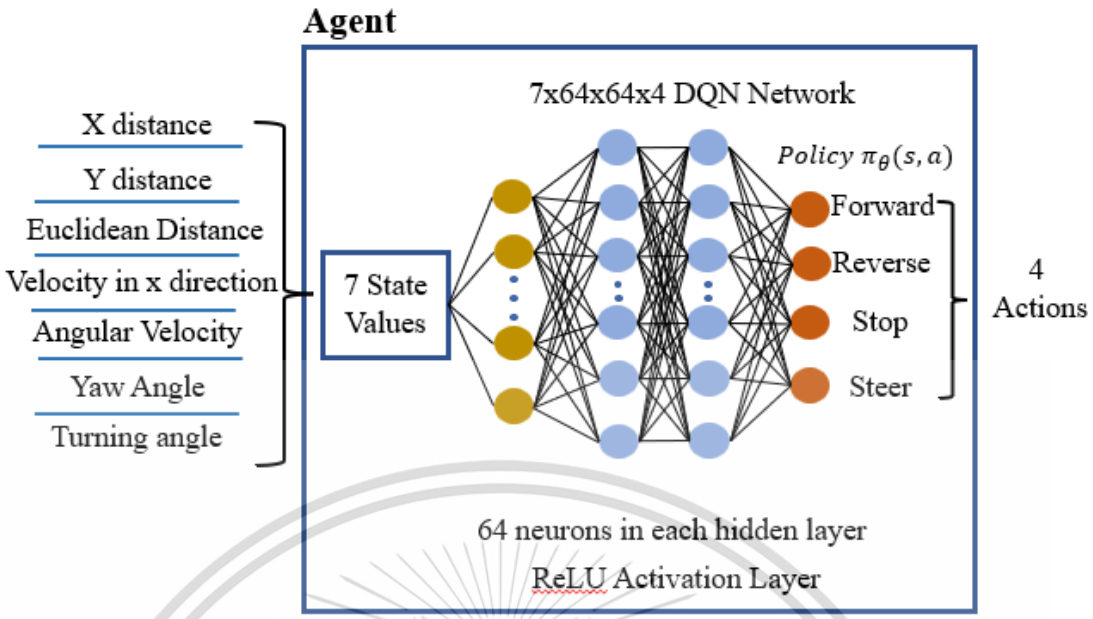


Figure 4-1 Neural Network Architecture for the training

4.1.2.2 Network Hyperparameter Setting

The weights and bias are initialized using He Initializer and the weighted summed values are passed to the Rectified Linear Unit (ReLU) activation function. Only the neuron value larger than the activation function is qualify enough to pass to the neuron in the next layer. The ReLU activation layer will return positive values or zero value only. This ReLU activation function will trigger non-linearity output values which helps the agent to learn complex patterns and non-linear relationships in data. So that, the neural network becomes powerful and can predict the most suitable action outcome even in the complex untrained situations. The weights and biases of the neurons are updated by using the chain rule method in the backpropagation process. 'Adam optimizer' is used and the learning rate is assigned to be 0.001. If the network parameters are optimized, the neural network will choose the most profitable action, which can strike the highest reward and decrease the error value gradually. For the last output layer, linear activation function is used.

4.1.2.3 Reward Setting

Deep neural network is referred to as black box as the working process of how the neurons interconnect and interact with each other cannot be visualized. To teach and train the network to accomplish the desired task is shaped by the reward parameters.

Adjusting the reward settings is one of the challenging tasks as they influence in improving the performance of an agent. Positive reward feedback encourages the agent to repeat the similar action in the next training steps as that is a favorable action which returns high rewards. Vice versa, the negative rewards remind the network that those are unfavorable behaviors and attempting those similar actions frequently can affect the overall training performance adversely.

The goal of the agent is to maximize the total episodic reward which can be achieved by driving the car from the spawned position to the designated parking location using minimum training time while not colliding with other external objects. The reward for each training step is the sum of the **total reward** and **an extra reward**. The total reward for a training step can be divided into three categories: the Distance Reward, Angle Reward and Time Reward which can be expressed as follows:

$$Total\ Reward = (Distance\ Reward + Angle\ Reward) * Time\ Reward \quad (4.1)$$

Distance Reward depends on the relative distance of the location of the agent car to the designated location. The closer the agent car is, the higher the reward becomes. The Euclidean distance between the agent coordinate (x_1, y_1) and the desired coordinate (x_2, y_2) is equal to the square root of the sum of the squares of the difference between the corresponding coordinates.

$$Eulidean\ Distance = \sqrt{(y_2 - y_1)^2 + (x_2 - x_1)^2} \quad (4.2)$$

Then the distance reward is calculated by

$$\boldsymbol{Distance\ Reward} = a * np.\exp(-distance^2 / (2.5 * b^2)) - c \quad (4.3)$$

where distance is equal to the Euclidean distance, which can be accessed from the Carla Simulation and a = 2, b =10 and c=0.5 are tunable hyperparameters which are resulted from the trial-and-error method.

Time reward urges the agent to finish the task as fast as possible. If the car gets into the determined parking lot within 12 seconds from the start, the reward will be a maximum value of 1, otherwise, the reward will decrease gradually from maximum to minimum of ‘zero’ as the training time exceeds over 12 seconds. Simulation time limit is 40 seconds for this scenario and the time consumed is the time difference between the current and the start of simulation.

$$\text{Time consumed} = (time_{current} - time_{start}) \tag{4.4}$$

$$\boldsymbol{Time\ reward} = 1 - \left(\frac{Time\ consumed\ - 12}{simulation\ time\ limit}\right) ** 0.6\ - 0.1 \tag{4.5}$$

Angle reward depends on the angle difference between the yaw angle of the car and the arctan angle of the car. The yaw angle can be accessed from the Carla environment. The arctan angle (θ) is the angle from the positive X axis to the line connecting the current agent location and the desired location which is expressed by $\left(\theta = arctan2\frac{\Delta y}{\Delta x}\right)$. The turning angle$(\alpha)$ of the car is obtained by subtracting the arctan angle from the yaw angle. Both arctan angle $(\theta)$and the turning angle (α) are normalized between -180 to 180 degrees after each step. The angle reward can be achieved by taking the absolute value of the turning angle(α) in the following equation.

$$\boldsymbol{Angle\ Reward} = \cos\left(abs(\alpha) * \frac{\pi}{180}\right) \tag{4.6}$$

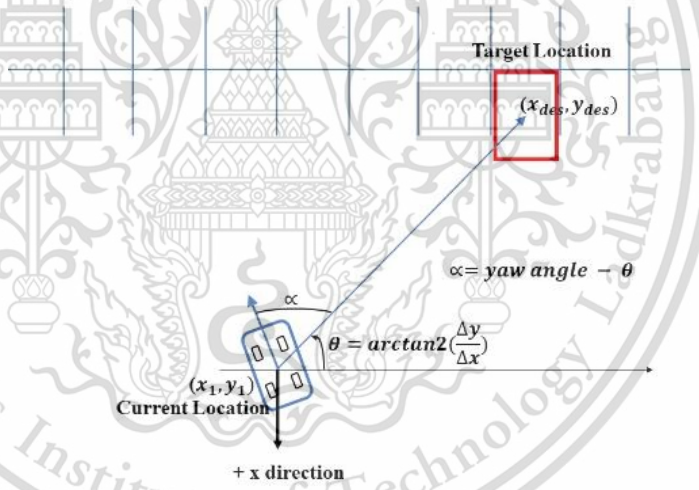


Figure 4-2 Turning Angle of an Agent

The absolute turning angle value (α) is in the range of 0 to 180 degree and the angle reward lies between (+1 and -1). If the turning angle is zero, it means the car is heading straight to the parking location so angle reward will be maximum (+1). If the car is heading in the opposite direction of the parking area, the turning angle will be 180 and return minimum value (-1). The more the car deviates from the parking location, the less the angle reward will be.

Total reward which is the summation of Distance reward, Time reward and Angle reward, is calculated on the current state of the agent in the environment. Extra

Rewards are additional rewards given to the agent when the agent does desirable or undesirable actions.

Table 4-1 Reward Parameters applied in training the Agent for 7000 episodes

State and Action Conditions	**Rewards**
Colliding with external objects	-300
Drive over the distance limit	-300
Training time exceed over time per episode	Total Reward -40
Parking success within Position, Angle and Time Limitation	Total Reward+angle_penalty+1800
Parking success but exceeds time limit	Total Reward+angle_penalty+50
Get into the parking zone	Total Reward+ angle_penalty+2.5
Else During Training	Total Reward

Collision with other external obstacles and driving over the limited Euclidean distance (40m) from the parking lot are undesirable. Therefore, the reward for that time action step is a penalty of -300 and the episode is ended promptly.

When the agent is within the parking zone, it must align within the designated parking boundary to accomplish the parking task. Therefore, in order to park the agent within the parking angle limitation, an angle penalty reward is calculated. This reward encourages the agent to fit into the destination point within parking angle constraints. First, how much angle the head of the car deviates from the 0- or 180-degrees axis is calculated. The cosine value of this angle is calculated and then normalized into the reward range of -1 to 1. The agent will get a maximum reward of +1 when it is very close to the 0- or 180-degrees axis however, the minimum reward of -1 is provided when the head of the agent is perpendicular to the parking destination zone.

$$angle\_penalty = max(abs(cos(angle\ deviation\ from\ \ the\ goal\ angle)), 0.01) * 2 - 1 \quad (4.7)$$

Additional positive rewards are given when the agent accomplishes the task. However, the extra reward will be varied on how the agent accomplished the parking task. The parking task is successful when the Euclidean distance is less than 1.1 m, and the yaw angle is within (± 15) degrees from perfectly straight aligned position. If the

car parks successfully less than 20 seconds of training, a huge extra reward of +1800 is added to the total reward. Else, if the car could park after 20 seconds, a small positive extra reward of +50 is added to the total reward. Else, once the Euclidean distance of the agent is less than 3m from the parking spot, an extra 2.5 points will be added to the total reward to encourage the agent to come closer to destination.

The training is unproductive when the agent makes small movement steps (distance less than 0.1m) continuously for 20-time steps. Therefore, a small negative penalty is given to trigger the agent to make larger movements. If the car is committing such inefficient actions and the current distance from originated spawn location is less than 1m and the training time is over 20 seconds, then that car is standing still at the spawn location by choosing the 'stop' action constantly or the actions chosen by the car are not beneficial. Therefore, a negative penalty of (- 800) points is added to the total reward, and it can be expressed as

$$Reward = Total\ reward - 800 \tag{4.8}$$

If the agent takes small steps and the distance travelled is less than 0.2 m from the previous location, then the car may stick somewhere in the learning process. This may be due to its naivety in the environment, or the insufficient data to make decisions. A penalty reward which is linearly proportional to the delay time is added to the total reward and may be expressed as:

$$Reward = Total\ reward - 2 \tag{4.9}$$

If the agent's condition is excluded from above all conditions, the car is moving and learning the task, and approaching nearer to the desired location. Therefore, the reward may be equal to total reward which is positive.

$$Reward = Total\ reward \tag{4.10}$$

If the agent could not finish the task during the training time (40 seconds), a penalty of (-40) is added to the total reward at the final step to encourage the agent to fulfill the task withing the limited training time in next episodes. Once an episode ends, all the actors (the agent and sensors) are destroyed, and a new episode is loaded, and the new training is started.

$$Reward = Total\ reward - 40 \tag{4.11}$$

4.2 Training the agent

Training a neural network is similar to how a human learns a skill set. At the beginning of the training, the environment is being explored by the agent and saves the experience as it has no prior knowledge at all. The agent learns from the mistakes and rewards and updates the network parameters concurrently according to the action feedback. After several hundred or thousand training episodes, the agent realizes the flow and pattern of the actions and the goal of the training. The agent receives the reward for every action it committed whether it is good or bad. The positive reward encourages the agent to repeat the similar behavior which strikes the highest possible return whilst, the negative rewards notify the agent to avoid similar undesired actions by liberating from that situation. Through iterative training and feedback rewards, the parameters in the network are updated and optimized. The better the network parameter values, the more suitable output action resulted in a higher reward return will be chosen by the neural network. After the agent is trained for several experiences, then it will be able to able to accomplish the task and the total episodic reward can be maximum.

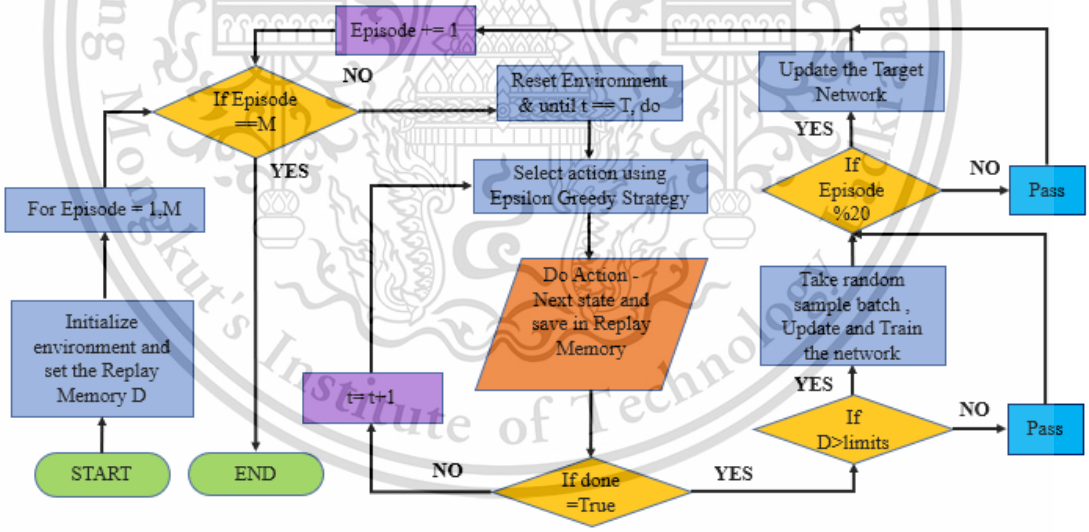


Figure 4-3 Flow Diagram of the training the agent

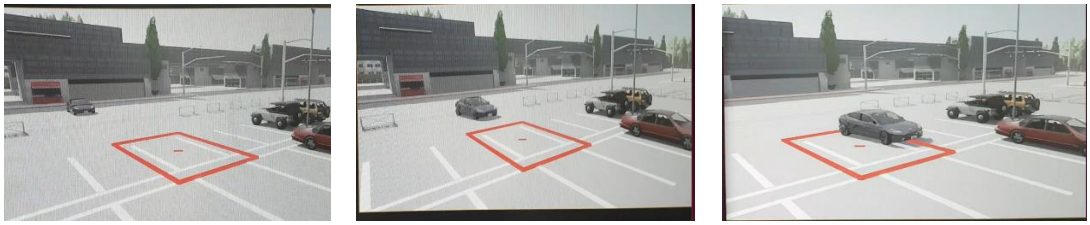
The above flow diagram illustrates how the agent training is done from the start till the end episodically. Firstly, the simulation environment is initialized, and the replay memory and the number of episodes is assigned to a specific amount. The following processes are iterated until the last episode is trained. Once an episode is started, the environment is reset. An action state is chosen either from the neural network or the random choice by using the Epsilon Greedy Strategy. The agent will receive a reward

that is related to the action it made and check if the episode is ended or not. Then, the state, action, reward, next state values are saved into the replay memory. The episode will not end until the agent commits forbidden actions or accomplishes the task. Until the training time is over, the agent will choose another action for the next time step and save the rewards and the necessary data to the replay buffer. This is how game play experiences are collected. When the episode ends, all the created actors are destroyed.

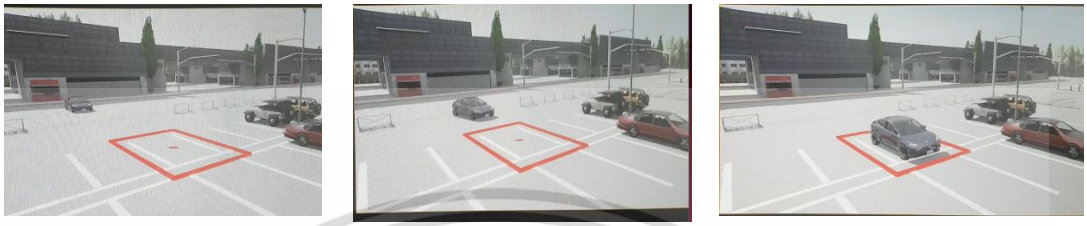
If the number of saved replay memory surpasses a certain threshold, a random set of previously saved game experiences are trained, and loss error value is calculated. The neural network weights are updated by optimizers and backpropagation method using main DQN network and target DQN network. A new epsilon value is set for the next episode by multiplying the current epsilon with an epsilon decay value and accept it if it is larger than minimum epsilon value. After training for a particular episode, the target neural network parameters are updated with parameters from the main network to increase the stability and convergence of the DQN training. After all the episodes are trained, all the trained data is saved in a buffer list.

4.3 FIRST PARKING CASE SCENARIO

For all scenario cases, the same Carla simulation engine with Tesla agent car is used and the DQN algorithm is used for training the agent. However, each scenario plays with different parking zone destination, different start points coordinates, different neural network hyperparameters values and reward functions. The first scenario is the demonstration training of the parking process to test run the car and test whether the agent car can accomplish the basic task successfully. The agent car will be spawned at the entrance of the parking lot and the parking destination is assigned in advance. Parking is assumed successful when the car gets to the designated location, which is a few meters far away from the spawned location in the straight direction. The figure below illustrates the map positions of the car's spawned location and the destination point. The red rectangle is the desired parking space, and the parking coordinate is saved first using the separate program and saved as .csv file. That data is reloaded in the main program file to set the boundary space for the parking area.



Spawned Position During Training Deviate from goal position



Spawned Position During Training Park Successfully

Figure 4-4 Training Demonstration for the 1st Parking Scenario

In this first scenario, the training is done for 20 seconds per episode and the agent is trained for 3000 episodes and 6000 episodes. Designing and training a deep Q network is improved by trial-and-error method, therefore, we assigned different number of training episodes and training time in this scenario. If the training time per episode is too short, the agent does not have enough time to learn the strategy. Conversely, if the training time is too long, the overall training time will take several running hours which consume more processing resources.

4.3.1 Neural Network Architecture

After integration of neurons and network layers with different learning rates and hyperparameters, 8x256x128x5 size, a four-layer DQN network which contains 8 data in the input layer, 256 neurons in the first hidden layer, 128 neurons in the second hidden layer and 5 output agent actions in the last layer, is chosen for this scenario. The input data are

1,2) x and y coordinates of the vehicle,

3,4) the relative distance in x and y direction from the parking coordinate to the current agent position,

5) the Euclidean distance,

6) the yaw angle of the car,

7) the velocity in x direction,

8) the angular velocity.

The outputs are 1) drive straight actions, 2) left steer, 3) right steer, 4) reverse and 5) brake actions. Adam optimizer with a learning rate of 0.001 is used.

If the network architecture is not compatible with the complexity level of the task, the result of the agent will not be good, and the overall average reward trend line is consistently flat around the negative values even after being trained for thousands of episodes. This indicates that the training is unproductive, and the agent is choosing ineffective actions which are inadequate to accomplish the task.

4.3.2 Reward and Epsilon Setting for 1st Scenario

In reward setting, only the linear Euclidean distance reward is assumed as the main total reward while angle reward and time reward are not considered in this scenario. Other extra rewards conditions are used similarly as explained in the above session 4.1.2.3. In this distance reward function, if the agent is still far away from the destination and the Euclidean distance is greater than 12m, the reward value will increase in a small step. If the agent is getting closer to the destination and distance becomes less than 12m, the reward will increase linearly up to a maximum reward value of 25. The maximum additional reward achievable by parking the agent successfully within 13 training seconds is +1300. If the agent managed to park successfully but uses over 13 seconds of simulation time, the extra reward is 1000. As soon as the agent enters the Euclidean distance of less than 1.5 meters, an additional prize of (+500) is provided.

In 3000 episodes, the epsilon value is not raised after a certain number of training episodes. However, for 6000 episodes, the epsilon value is set back to 0.4 after training every 2000 episodes to evaluate the agent's performance and enhance it to try new solutions rather than getting trapped in the local minimum. The epsilon decay rate for both training episodes is 0.999 therefore, in every new episode, the epsilon value decreases gradually hence, the probability of choosing action generated from the neural network becomes higher than the random action choice as the number of training episodes increases.

4.3.3 Result and Discussion for 1st Scenario

4.3.3.1 Discussion for Distance Result

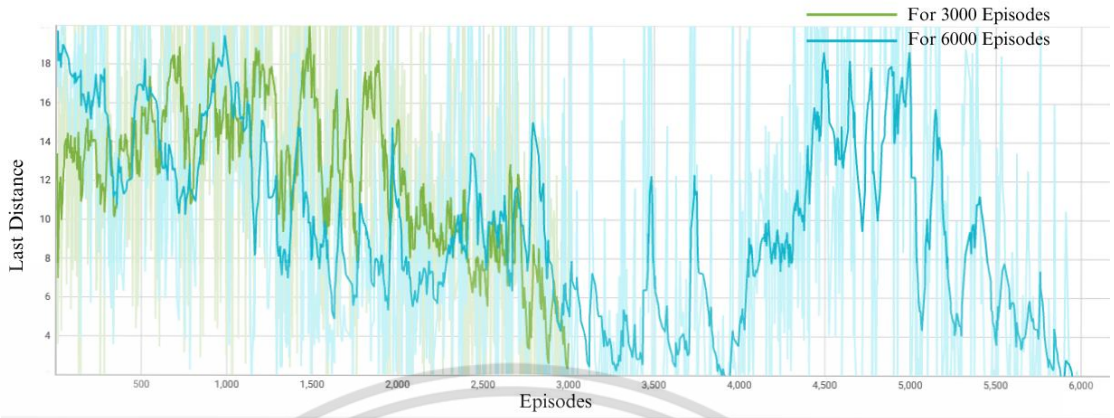


Figure 4-5 Distance Result Comparison for First Scenario

The above graph presents the last distance outcomes obtained from training the agent for 3000 episodes and 6000 episodes respectively. The **green trend** line denotes the terminal Euclidean distance at the end of an episode for 3000 training episodes while the **cyan line** is the result for training 6000 episodes. The spawned position of an agent is about 22 m from the destination. In the initial episodes (1~1000) of the training process, the last distance point may vary significantly in both training courses, and the last distance trend line consistently stabilizes within the range of 14 to 16 meters. This is due to the stochastic action selection mechanism employed by the DQN network, influenced by high epsilon value. Consequently, the DQN network will neglect to find the optimum policy strategy and the agent will prioritize only exploring the environment and accumulating knowledge and experiences. Throughout the training and optimization phases, the network's weight parameters will be updated and changed. As the epsilon value drops, the network will progressively depend less on random selections and more on its output decision.

The terminal Euclidean distance trend line steadily drops after training more than 1000 episodes, indicating that the agent automobile may approach closer to the target with each subsequent episode. This demonstrates how the agent car is picking up on policy and how the DQN network's action recommendations are effective and helpful to the agent. Since parking is considered successful when the Euclidean distance is less than 1.3 meters and hold that position for several time steps, it may be assumed that the agent automobile was able to complete the task after 2700 episodes where the terminal distance is less than 2 meters.

Because of the high epsilon value, the distance cyan line graph grows once again after 2000 and 4000 episodes. Within the range of 3000 and 4000 episodes, the cyan line's distance dropped to the minimum range and did not fluctuate excessively. Following that, the high epsilon value causes the distance line to increase once again. After 5500 episodes, it then returns to its minimal value.

4.3.3.2 Discussion for Reward Result

The trend line for 3000 episodes (blue line) and 6000 episodes (red line) is depicted in the above graph, respectively. The total reward for the chosen state-action value in that time step is linearly proportional to the Euclidean Distance and the greater reward signifies that the agent has approached to a close distance, even if the parking task was not successfully completed in that episode.

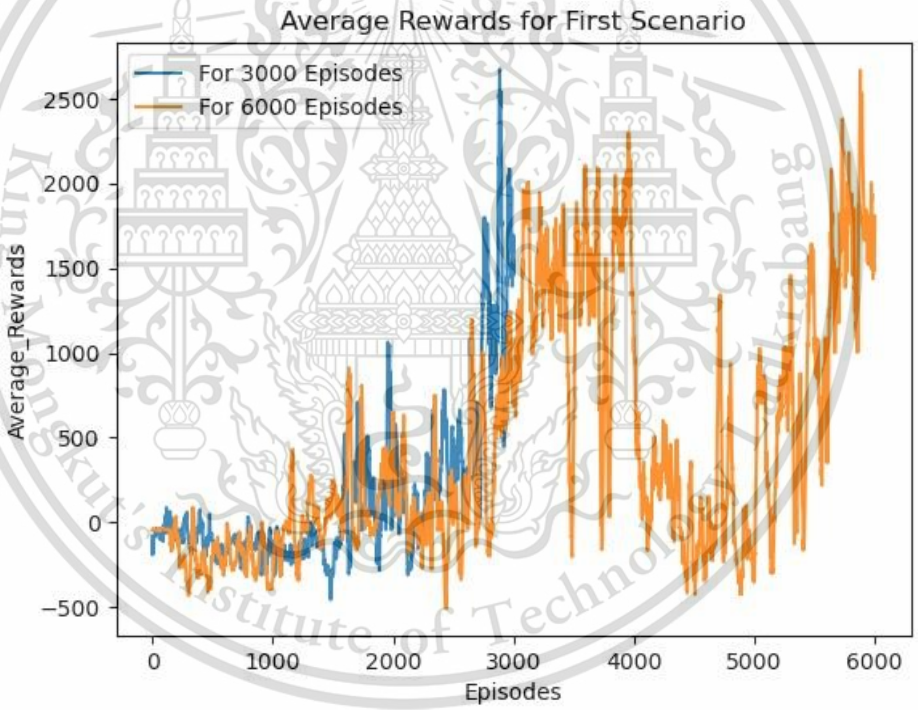


Figure 4-6 Average Episodic Reward Comparison for First Scenario

In the earlier training sessions, the reward trend line was consistent and less than zero because of negative rewards earned from undesirable actions of the agent, resulted either from random chosen actions or the DQN network's generated action recommendations. The network weight parameters are updated in each training. As a result, from roughly 1200 episodes forward, the agent began to gain positive rewards and the reward progressively increased as the agent got close to the destination through the actions chosen by the updated weight parameters of the DQN network. The

performance of the agent reached its optimum level after 2700 episodes as the total episodic rewards exceeded over +1500 value.

4.4 SECOND PARKING CASE SCENARIO

This scenario is an improved version of the first training scenario, with the goal of mimicking the real driving environment and raising the difficulty level by adjusting the agent's yaw angle and position at the spawned point. The spawning place and the car's yaw angle in this second training vary within a range of allowed distances and angles. To complete the parking task in a new assigned parking location, the agent must employ all vehicle control mechanisms, including the brake, steering, and reverse levers. Changing the spawned positions and yaw angles of the agent can generalize the training experiences.

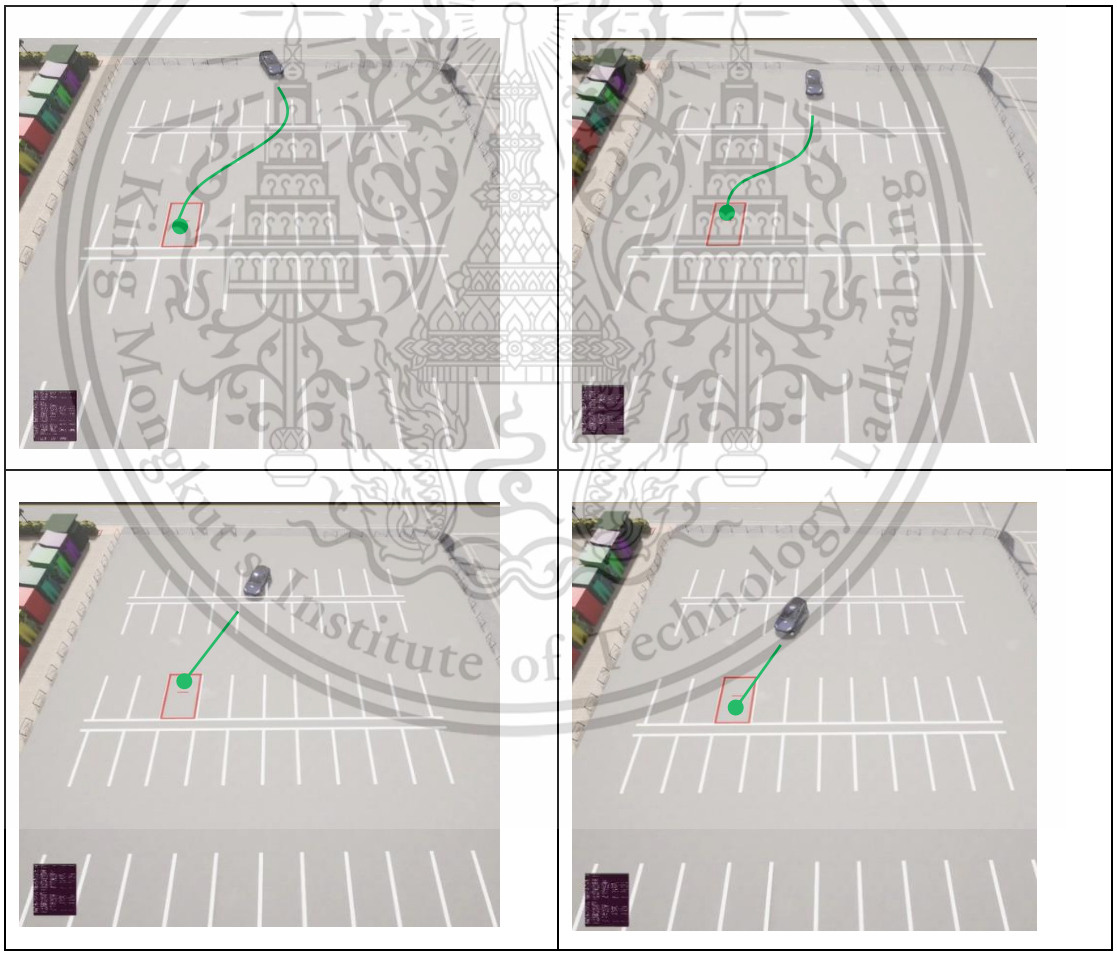


Figure 4-7 Agent Demonstration of Second scenario at around 450 episodes

4.4.1 Environment Modification and Explanation of Different Training process

The simulation environment, sensor and the agent remained the same as the previous scenario while the parking center coordinates, the position and yaw angle of

the agent at the spawn location are changed. The spawned yaw angle of the agent is randomized within the maximum range of ± 20 degrees from a default 180-degree orientation. The spawned coordinate of the agent is also within the maximum distance range of ± 1.5 m from the default spawned position, which was originally situated approximately 26 meters far from the center of the parking area.

Two training processes were executed to assess the agent's performance under distinctly different reward conditions. (a) Conducted the training for 5000 episodes, each lasting 40 seconds and (b) Conducted the training for 7000 episodes with parking angle constraints, each with a duration of 40 seconds. In training (a) 5000 episodes, the total reward is determined solely on the Euclidean distance and turning angle parameters. There are no constraints on parking angle at the parking point for successful completion of the parking task. In training (b) 7000 episodes, the total reward depends on 3 values, which are Euclidean distance, turning angle and the simulation time while the parking angle position at the destination is also restricted.

Table 4-2 List of Hyperparameters and Values for Second Scenario

Hyperparameters	Values	Descriptions
Network architecture	7x64x64x4	2 hidden dense layers with 64 neurons
Activation function	ReLU	Function used to create non-linearity outputs
Replay memory size	100,000	Set of training experiences stored and SGD updates are sampled from here.
Target Network update frequency	20 episodes	The frequency to update the weights of the Target Network
Discount factor	0.95	A value represents how future rewards are crucial
Learning rate	0.001	Tuning parameter that determines how fast the network learns to converge at global minimum value.
Epsilon Decay	0.997	A constant rate to decrease the Epsilon value in every new episode
Minibatch size	128	No. of experiences in each batch
Initial Exploration	1	The initial value of Epsilon
Final Exploration	0.00001	Minimum Epsilon Value
Number of DQN inputs, outputs	7, 4	Number of input and output parameters to Q- network

4.4.2 Neural Network Architecture for 5000 and 7000 episodes

A 7x64x64x4 model with 7 inputs, 64 neurons in each hidden layer, and 4 neurons in the output layer is the new neural network's parameters for second scenario. Input parameters are 1,2) the relative distance in x and y direction from the parking coordinate to the current agent position, 3) the Euclidean distance, 4) the yaw angle of the car, 5) the velocity in x direction, 6) the angular velocity, and 7) angle difference(α) between the yaw angle and the line connecting two coordinates.

The model's performance is significantly improved by adding the angle difference (α) value to the DQN model since it gives the agent information of how much angle should be steered to reach its destination. The output layer generates 4 action values, representing specific vehicle control actions in each state, namely forward, reverse, stop, and steer. Despite this neural network producing only 4 action outputs, the driving mechanism remains the same. Instead of the ‘left’ and ‘right’ separate actions, a singular steer command is utilized to effectuate both left and right turns.

4.4.3 Training scenario for 5000 episodes without parking goal constraints

4.4.3.1 Reward Settings for the training 5000 episodes

In this training (a) of 5000 episodes, the formula for the main total reward in this case is Total reward = Distance reward + Angle reward. The distance reward is a non-linear exponential curve as equated in **equation (4.3)** and as the Euclidean distance approaches the destination, a larger reward is offered. The angle reward is determined by the angle difference between the car's yaw angle and arctan angles, which is demonstrated detailed in **equation (4.6)**. This turning angle also determined the throttle amount and turning direction (left or right) for the agent's steering order. If the Euclidean distance is less than 1.1 meters, the parking task is considered successful. If the simulation time is shorter than 20 seconds while parking, a large additional reward is given; otherwise, a less positive benefit is offered. However, this parking reward resembles a sparse reward, which may not be effective for the training. The remaining extra award categories will be applied in as explained **4.1.2.3**.

4.4.3.2 Result and Discussion for training 5000 episodes

In this training (a) of 5000 episodes, each simulation episode spans 40 seconds. The spawned agent, initialized at random limited distance point and angle at the entrance of the parking lot, encounters challenges in the early stages of training. Due to the randomly assigned initial network weights, neither network generated actions nor the randomly chosen actions is beneficial to the training. The **Figure 4-8** below depicts the trend in terminal Euclidean distance across episodes, illustrating significant fluctuations and spikes during the initial training episodes. As the network parameters aren't in the ideal condition, and the agent is still exploring the surroundings and figuring out the best course of action and policy. Training episodes 1 through 500 clearly demonstrate this fluctuated unstable distance, where the final Euclidean distance ranges between 20 to 10 meters, which is far away from destination. According to the graph, after 500 episodes, the distance value steadily decreases and gradually approaches the horizontal line representing Zero Euclidean distance. Batches of experience are used to train the agent and network parameters are updated when the total experience exceeds the minimum cutoff experience. Consequently, around the "1000" episode, the terminal Euclidean distance began approaching less than 1.1 meters. It shows that the neural network is learning, and the best policy could be achieved by iterative optimization and backpropagation processes. When the network parameters are optimum, the agent could complete the parking task with its peak performance, earning rewards exceeding 1500 points. The network consistently achieves its goal in episodes ranging from 2400 to 3100, from 3300 to 3700 and from 3800 to 4400. During these episodes, the terminal Euclidean distance is as low as zero level indicating that the agent completed the parking task successfully.

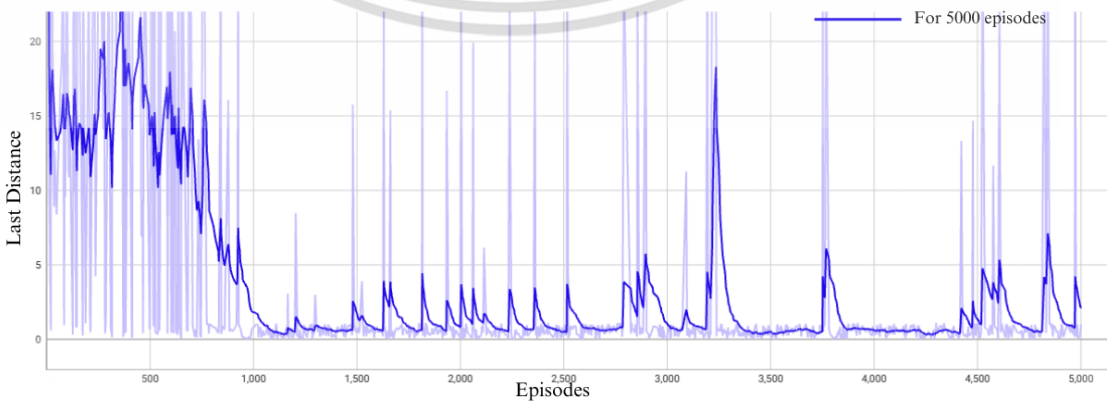


Figure 4-8 Last Distance Result for training 5000 episodes in Second Scenario

The reward graph, illustrating the average episodic reward of the last 30 episodes, is presented in **Figure 4-10** below. The reward graph initiates with a small positive value as the agent is far away from the destination when the episode is terminated. The average reward progressively increases to a higher positive value in every new episode as the policy learnt by the agent is beneficial to the training process. The agent achieves its maximum possible reward, exceeding over 1500 points, when the agent accomplished the parking task successfully. There are several episodes where the average episodic reward falls under 1500, indicating that either the task was not completed within the allotted training time, or the parking task completed after 20 seconds. Even though the agent is learning the policy, it seldom chooses the random action to explore another potential option that could yield higher rewards. Since it is a random activity, either positive or negative reward outcome can be possible. Despite occasional spikes in the distance graph, the average reward trend consistently over the value of 1000. Therefore, it can be inferred that the selected neural network, along with the assigned hyperparameters, is yielding favorable results in this scenario.

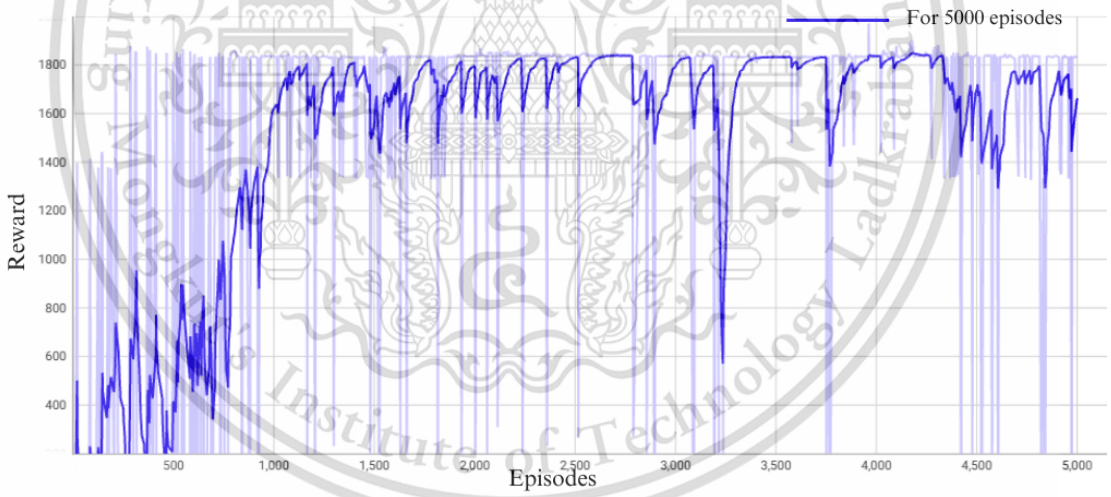


Figure 4-9 Total Reward Earned for training 5000 episodes in Second Scenario

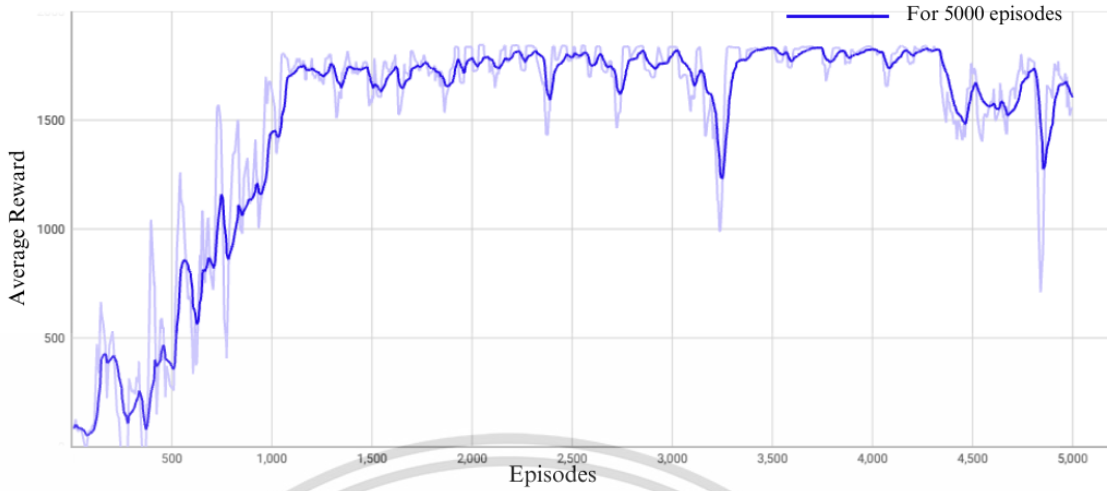


Figure 4-10 Average Reward Earned for training 5000 episodes in Second Scenario

When the agent found the optimum policy and the weight parameters are updated properly, the agent could be able to park successfully despite the yaw angle and coordinate at the spawned location. Durning these episodes depicted below; the epsilon value is approximately 0.0006. The simulation training time for the agent to park successfully and end the episode is less than 15 seconds for these episodes. Therefore, the agent gained the maximum reward possible in each episode.

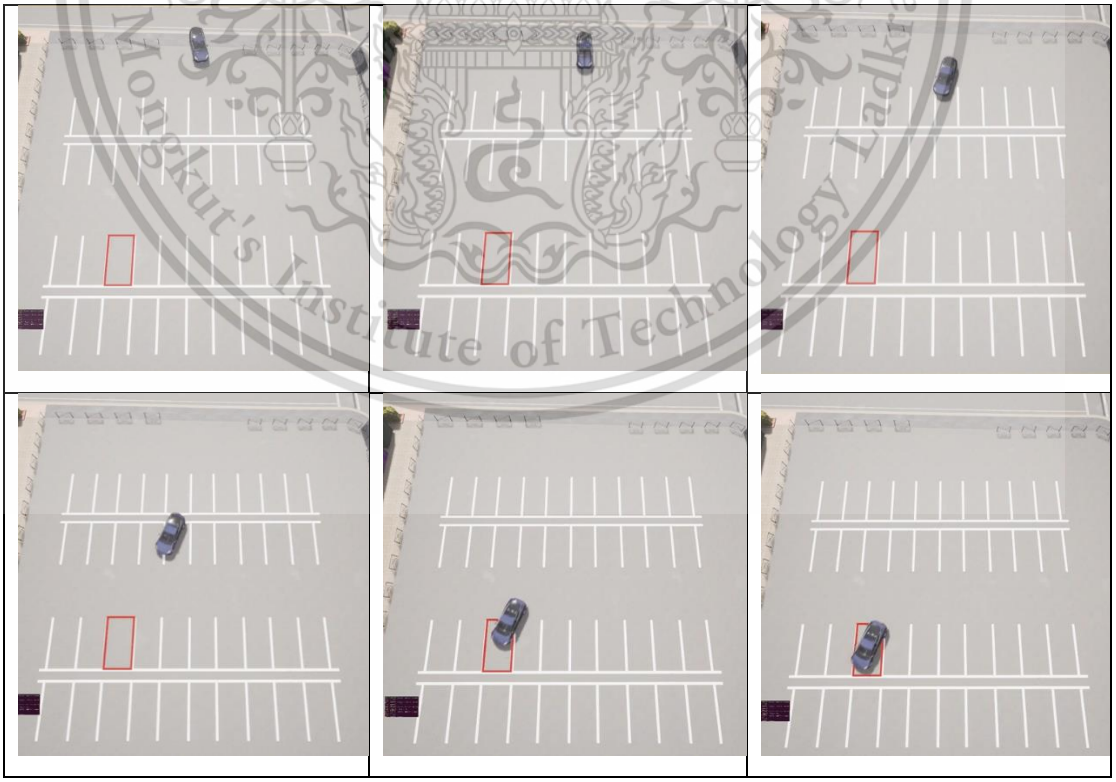


Figure 4-11 Successful Parking Performance without Parking Angle Constraints

Figure 4-12 Successful Parking Performance without Parking Angle Constraints

4.4.4 Training scenario for 7000 episodes with parking goal constraints

4.4.4.1 Reward Settings for the training 7000 episodes

The primary total reward formula for this training (b) of 7000 episodes is Total reward = (Distance reward + Angle reward) * Time reward; with detailed information provided in equations **(4.3), (4.5) and (4.6)** respectively. Time reward is used to inform the agent how much simulation time has elapsed in the current episode. The time reward undergoes a gradual decrease from its maximum value of 1 to a minimum of 0.1 as the agent's simulation time gets longer. The reduction in time reward results in a corresponding decrease in the overall reward for that time step, as it is multiplied with the sum of the distance and angle rewards. The turning angle, in turn, dictates the throttle amount and turning direction (left or right) for the agent's steering command.

If the Euclidean distance is less than 1.1 meters and the parking angle at the destination is within ± 15 degrees from the desired goal angle (± 180 degrees), the parking task is considered successful. Under these conditions, the overall reward is the sum of the total reward and the angle penalty reward. An angle penalty reward is added to encourage the agent to halt within the specified parking angle constraints. ‘Theta’ is the angle difference between the desired goal angle and the current yaw angle. This

'theta' angle undergoes normalization through a cosine function, transforming it into steer amount ranging from -1 to +1 for relocating itself into the desired parking angle.

A substantial extra reward of +1800 is applied to the overall reward (sum of the total reward and the angle penalty reward) if the simulation duration is less than 20 seconds when parking; otherwise, a less positive benefit of (+50) is offered. If the agent is within less than three meters of the destination, a small additional point of 2.5 is added to the total reward. The remaining additional award categories will be applied in accordance with **4.1.2.3** but in order to encourage the agent to refrain from these undesirable activities, the penalties are larger than they were in the prior 5000 training sessions.

4.4.4.2 Result and Discussion for training 7000 episodes

The distance graph, **Figure 4-13**, shows the agent's performance after 7000 episodes of training. Similar to the earlier episodes of the prior 5000 training episodes, the agent's terminal Euclidean distance in the beginning of the training is significantly far away from the target. But within 500 episodes, the agent's performance significantly improved, and the agent could gradually approach closer to the target. The agent began to reach the minimum terminal Euclidean distance at about 800 episodes; however, because of the network's non-optimal weights and the epsilon greedy approach, the outcome was inconsistent. After about 1200 episodes, the agent's and the network's performance hit its peak as the agent could successfully complete the parking assignment and the last Euclidean distance fell below 1.1 meters consistently. However, because of the parking angle restriction and epsilon value, the terminal Euclidean distance may be high in some episodes starting from estimate episode 3800 and beyond.

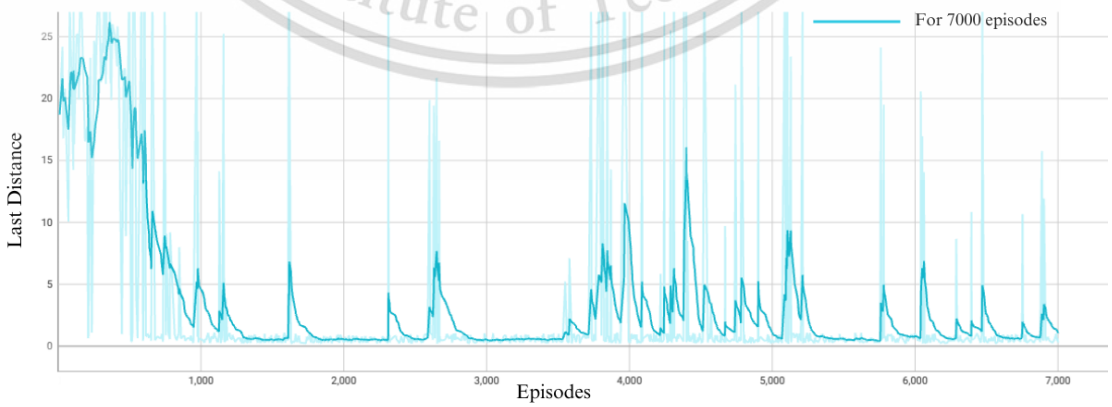


Figure 4-13 Last Distance Result for training 7000 episodes in Second Scenario

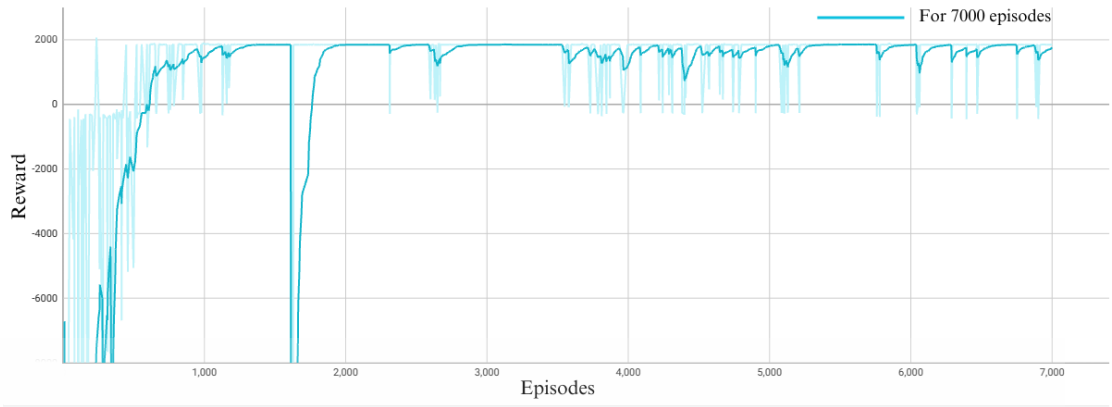


Figure 4-14 Total reward for training 7000 episodes in Second Scenario

Figure 4-15 Average reward for training 7000 episodes in Second Scenario

The **Figure 4-14** and **Figure 4-15** illustrates the reward graphs to evaluate the performance of the agent. The reward begins at the negative points value since this training uses larger negative penalty charges. As the agent refines its policy in each training episode, it successfully generates positive rewards to offset the initially negative average rewards. The average episodic reward gradually trends in a positive direction, crossing the zero-value threshold around the 500th episode. This positive trend persists, with the average reward consistently increasing in each subsequent episode, eventually peaking at approximately +1800 points after completing over 1000 episodes of training. After 2400 episodes, the average episodic reward levels out, indicating that the agent is generally proficient at completing the job. However, there are minor fluctuations in the later stages of training, attributed to instances where the agent struggles to complete the task within the specified 20-second simulation time, resulting in a drop in the episodic reward.

However, those results from unsuccessful cases can be recovered by the subsequent successful training cases, therefore, the overall outcome for the average episodic reward results in positive value. To conclude, the selected network parameters and weights are working well in this training process, hence, the network generated actions are beneficial to the agent performance. As a result, the agent achieved high reward returns by accomplishing the parking task successfully within the limited simulation time in numerous training episodes.

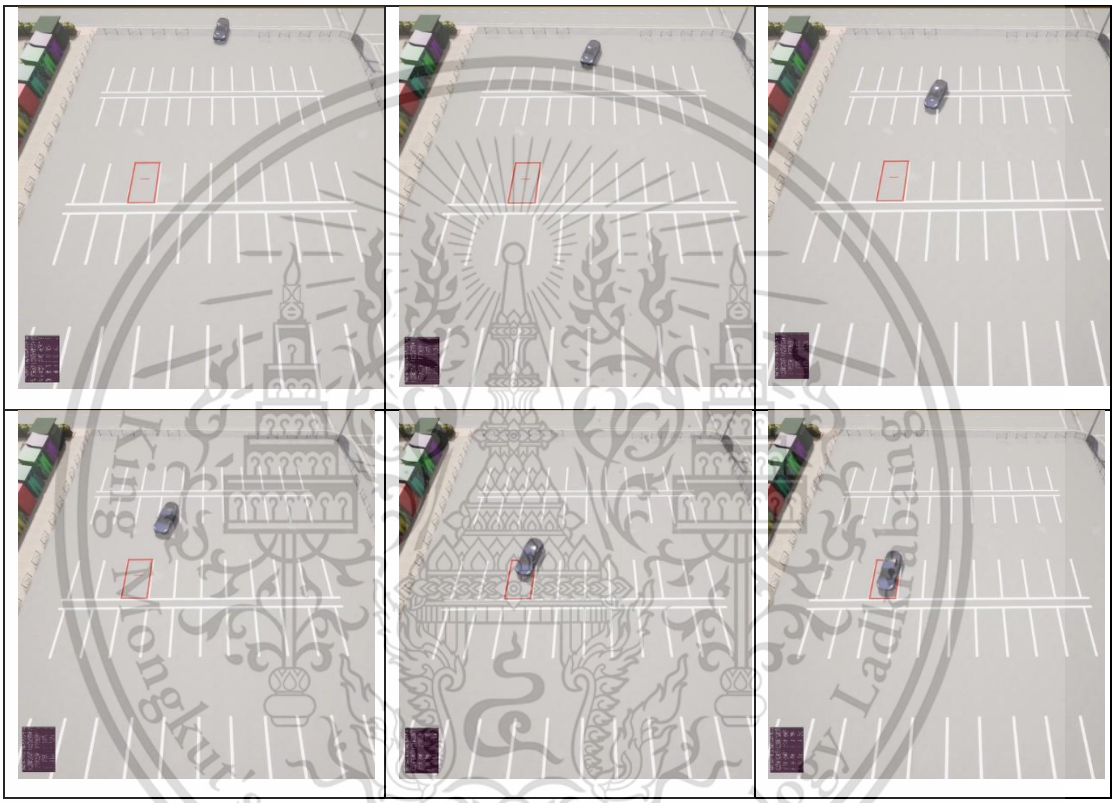


Figure 4-16 Successful Parking Performance I of Agent at around 2450 episodes

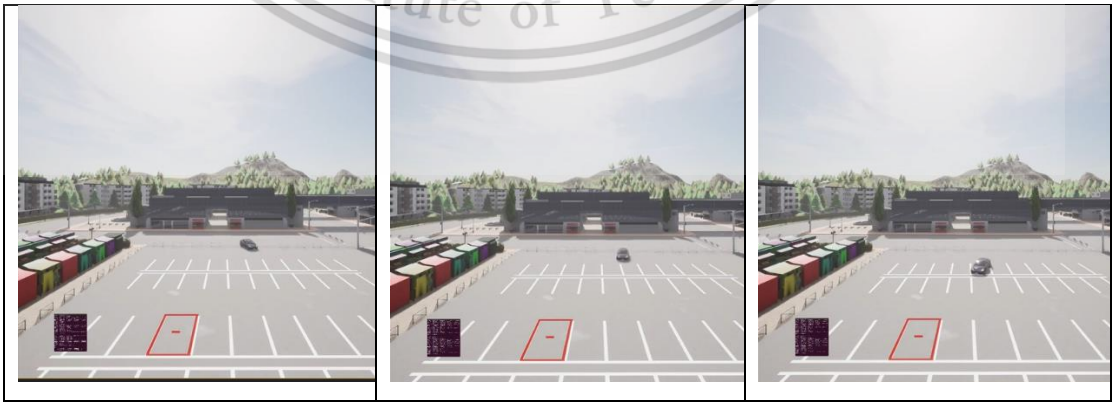


Figure 4-17 Successful Parking Performance II of agent at around 3650 episodes

4.4.5 Results Comparison for different training conditions in 2nd Scenario

Table 4-3 Comparison for 5000 and 7000 Training Episodes in 2nd Scenario

Hyperparameters	Training 5000 Episodes (without Parking Angle Limitation)	Training 7000 Episodes (with Parking Angle Limitation)
Reward Type	***Sparse Time Reward*** (+1800 for parking within 20 seconds else +1300) If the vehicle is parked within 20 training seconds, the extra reward added to the original reward is **+1800**. If the vehicle is **not** parked within 20 training seconds, the extra reward added to the original reward is **+1300**.	***Continuous Time Reward,*** The longer the episode, the lower the reward (+1800 for parking within 20 seconds else +50) If the vehicle is parked within 20 training seconds, the extra reward added to the original reward is **+1800**. If the vehicle is **not** parked within 20 training seconds, the extra reward added to the original reward is **+50**. When the vehicle is able to approach within the parking zone, the extra reward added to the original reward is **+2.5**.
Restrictions	***Do not limit*** the parking angle	***The head angle is limited*** within ± 15 degrees parallel to the parking lot
Network Architecture	7x64x64x4	7x64x64x4
Simulation Time	40 Seconds/ Episode	40 Seconds/ Episode

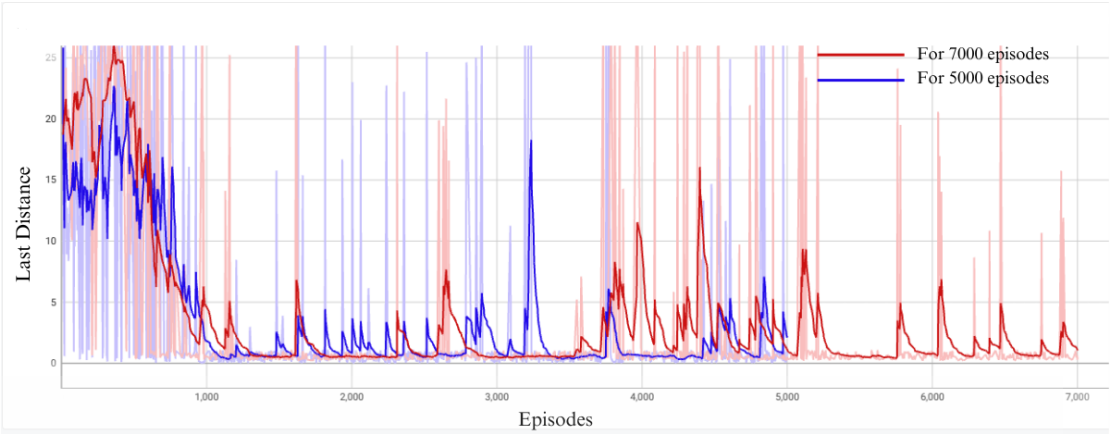


Figure 4-18 Comparison of Last Distance Result for training 5000 and 7000 episodes

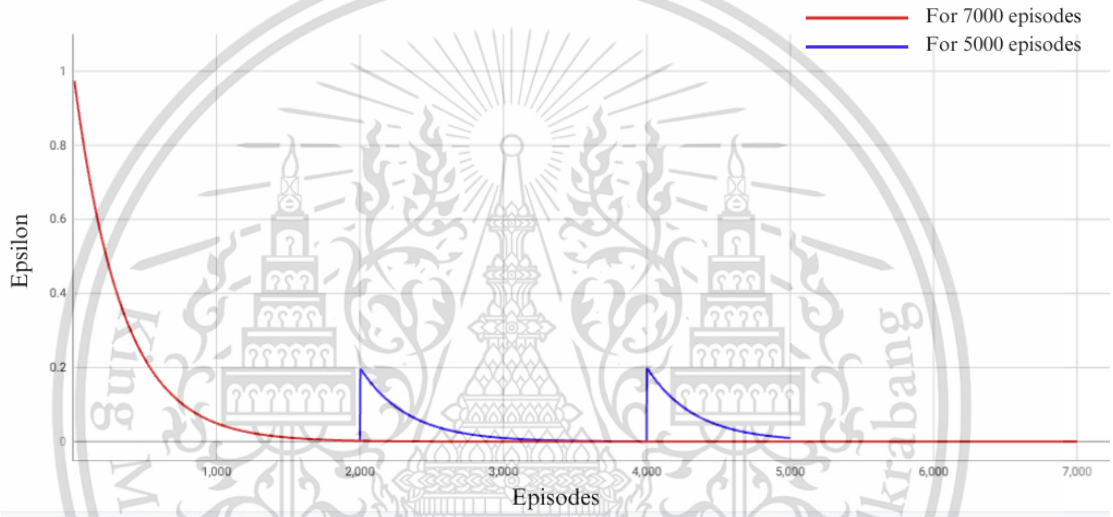


Figure 4-19 Comparison of Epsilon Result for training 5000 and 7000 episodes

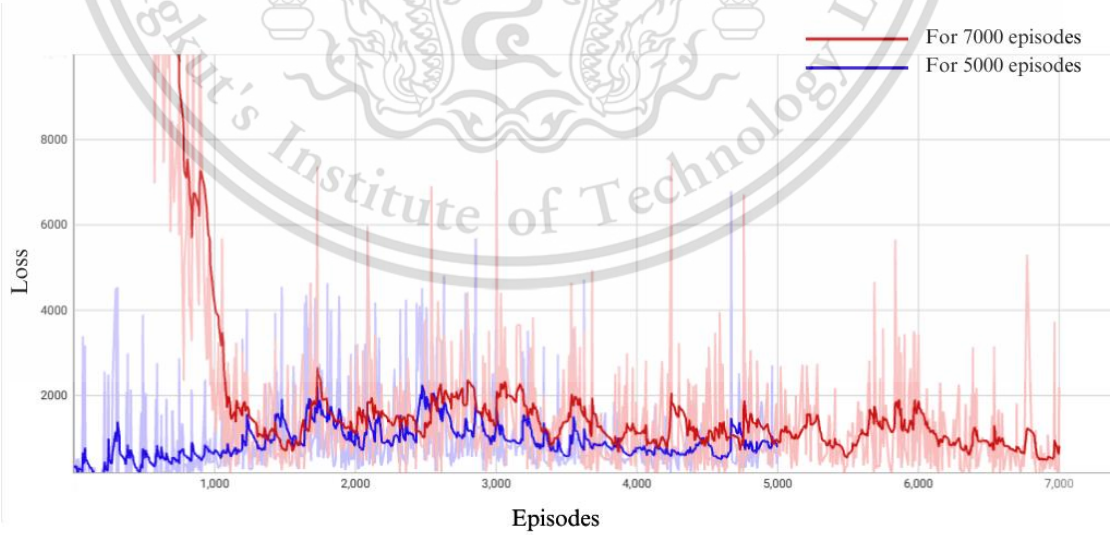


Figure 4-20 Comparison of Loss Function Result for training 5000 and 7000 episodes

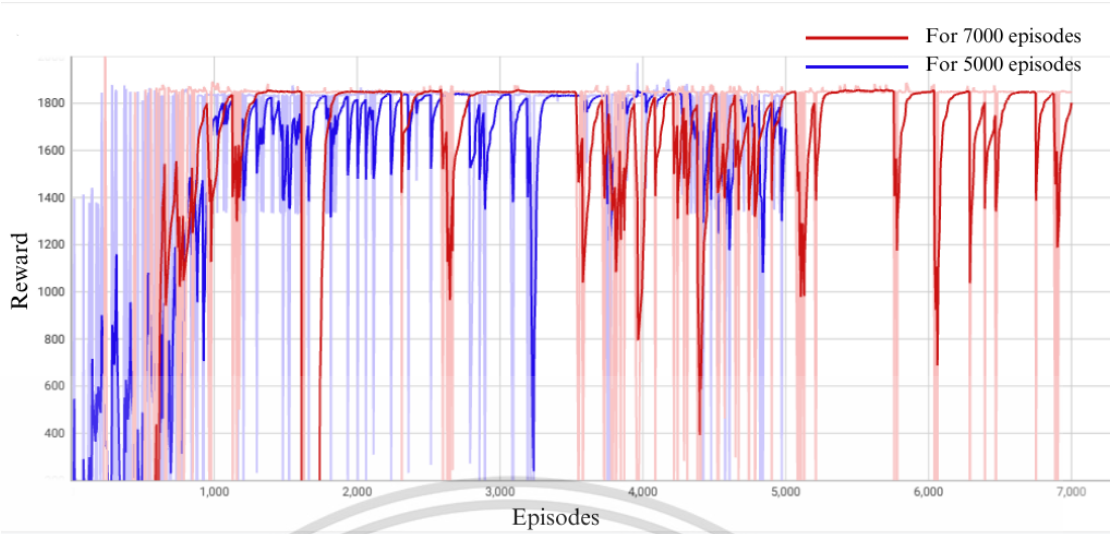


Figure 4-21 Comparison of Episodic Reward for 5000 and 7000 episodes

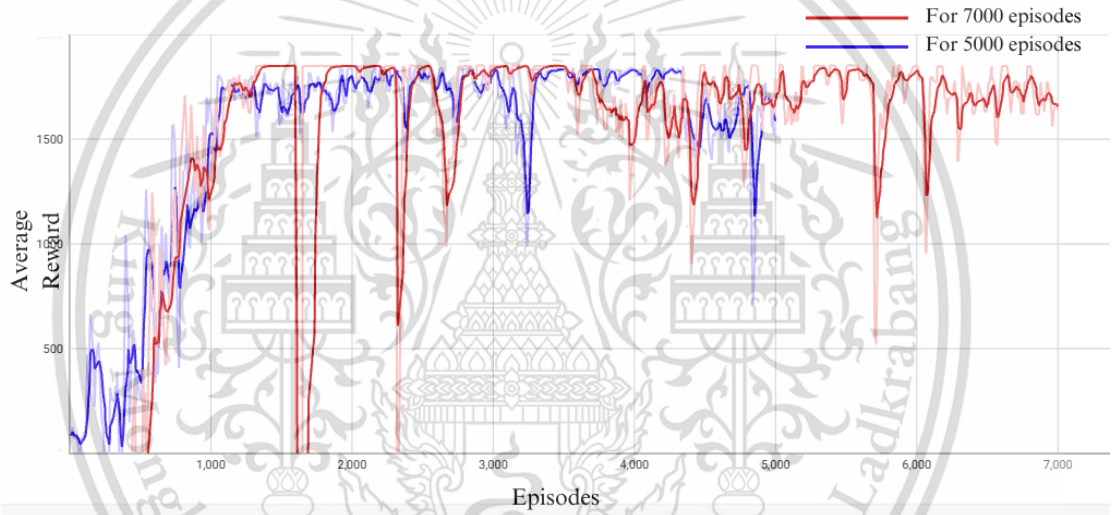


Figure 4-22 Comparison of Average Episodic Reward for 5000 and 7000 episodes

4.5 THIRD PARKING CASE SCENARIO

The third scenario is a modified environment settings of the second scenario. In this scenario, the agent is spawned at random angles and coordinates inside the limited border and the objective is to drive the agent automobile spawn at random position towards the destination and halt at the designated parking space. In real-world scenarios, the agent car will not always start at the same spot. Therefore, this training will evaluate whether the selected neural network and the agent could achieve a comparable level of performance as in the prior trainings, utilizing only the collision sensor and the car's current state variables for position, angle, and velocity. Like in the second training scenario, a car is deemed successfully parked only if its parking angle is within the angle constraints and its Euclidean distance is less than a predetermined distance threshold. During the third scenario's development phase, only the external parking

boundary is restricted, and the spawned angle is randomly chosen from a list of (0,90, -90,180) angles. Therefore, the agent is frequently spawned at unsuitable coordinates —that is, extremely close to the parking area or overlapping on the parking area. As a result, training is ineffective, and even after "5000" sessions of training, the agent's performance did not improve. The 7x64x64x4 DQN model architecture did not function effectively in this training scenario. The updated network weight parameters were unable to generate an appropriate vehicle action to accomplish the parking task and the agent was unable to drive nearer to the destination. Therefore, both an outer and inner boundary are added to improve the spawned position of the agent, which is to be spawned randomly inside the defined outer and inner boundaries. The inner boundary's x coordinate is 4.5 meters away from the parking destination's center in the x direction, and its y coordinate is 2.75 meters away from the parking location's center in the y direction. The spawned coordinates should not be near to the destination nor the boundary to avoid undesirable actions.

The random yaw angle may be generated from (0,90, -90,180) based on the x and y spawned coordinates. If the spawned position is in the first quadrant, the spawned angle may be 90 degree or 180 degrees else if in the second quadrant, the angle may be -90 degree or 180 degrees else if in the third quadrant, the angle may be 90 degree or 0 degree else the spawned angle may either be -90 degree or 0 degree.

Initially, the training simulation duration was raised to 60 seconds in case the agent might need more time to process the parking task as it is spawned at different location in every episode. However, extending the training time did not show significant improvement. Therefore, the training time was reduced to 40 seconds and the agent is trained for 3500 and 5000 episodes using different hyperparameters.

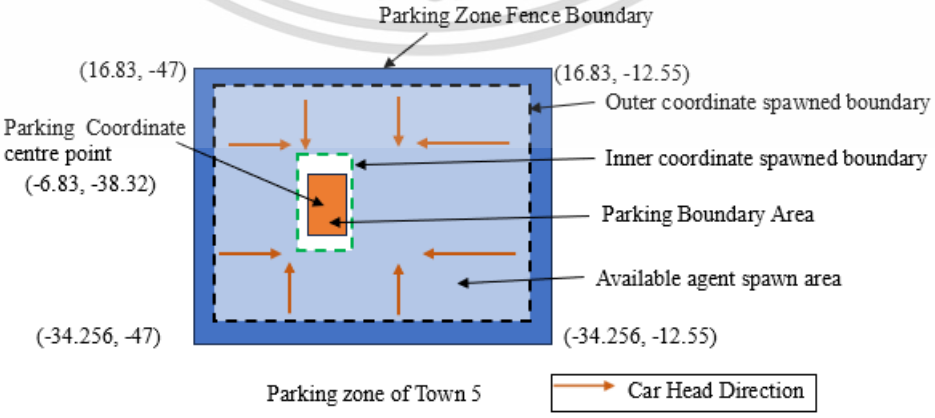


Figure 4-23 Agent Spawning Area for 3rd Scenario

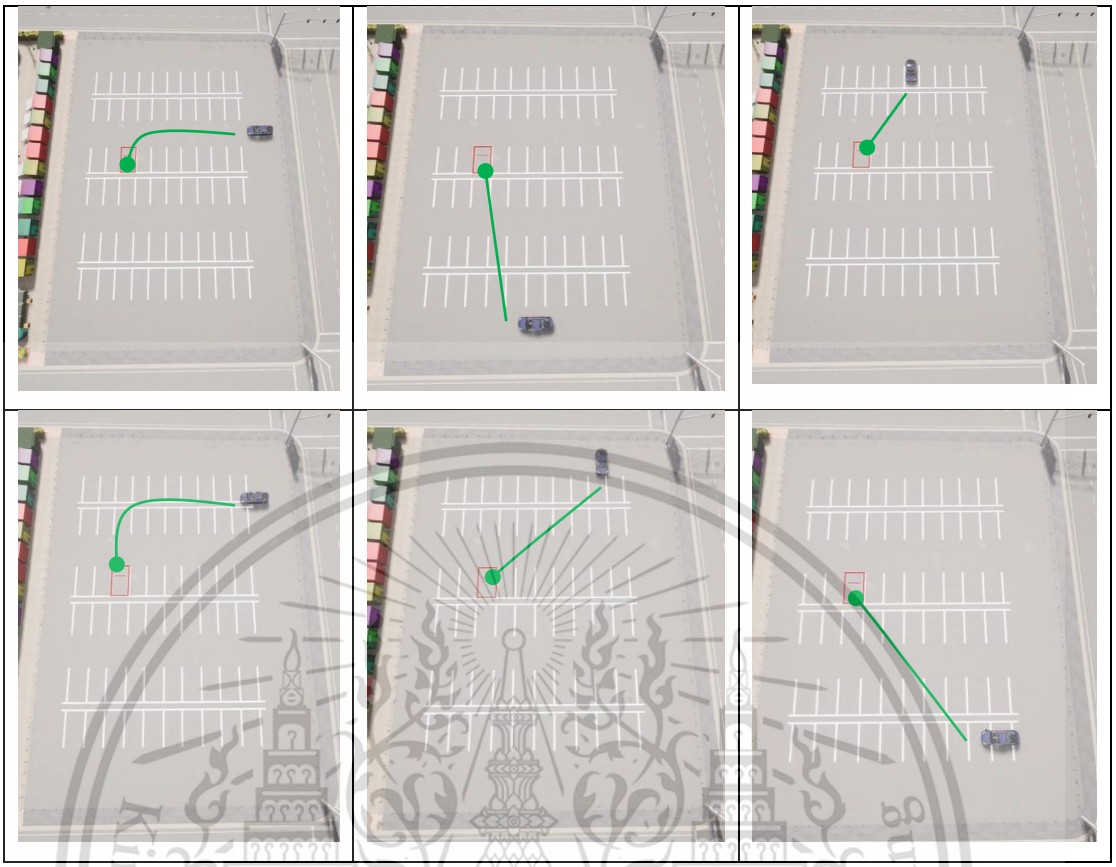


Figure 4-24 Agent Spawning Location for the Third Scenario

4.5.1 Neural Network Architecture

The training performance improved significantly by adding constraints to the spawned angle and the spawned position as explained in the prerequisite session. In this third scenario, DQN model is modified to 7x128x64x32x4 shape which comprises with more hidden layers each consisting of 128, 64 and 32 neurons while sharing the same input and output data. The 'ReLU' activation layer, 'He uniform' kernel initializer, 'Adam' optimizer and the 'Mean Squared Error' loss function method are used which is similar to the second scenario while the learning rate is assigned as 0.005.

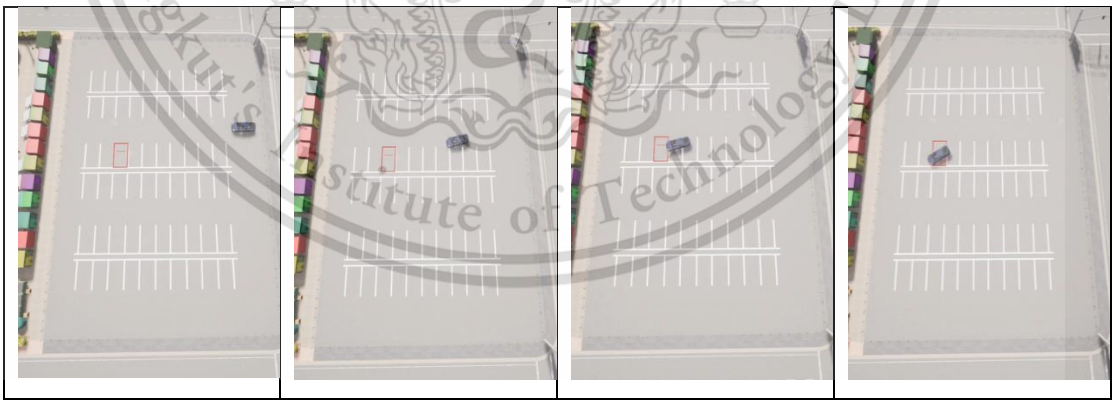
4.5.2 Reward and Epsilon Setting for 3rd Scenario

The third scenario is trained for 3500 episodes and 5000 episodes. A 7x128x64x32x4 neural network with the same learning rate is used in both trainings; the only differences are in the reward and vehicle throttle speed parameters. While agent training in 5000 episodes employs the same settings as in the second scenario, training 3500 episodes is tested using new reward and throttle speed parameters. In training

3500 episodes, when the Euclidean distance is less than a threshold, the throttle speed for ‘steer’ and ‘straight’ action is changed to 0.3 which is originally from 0.5 in training 5000 episodes.

The numerical values of the additional reward hyperparameters are changed, however the reward settings remained the same as the second scenario. The extra parking reward point exceeded over 20 seconds is increased from +50 in 5000 episodes to +150 in 3500 episodes. Extra point for getting into parking area is also increased from +2.5 in 5000 episodes to +5 in 3500 episodes.

Similar to the second scenario, the epsilon value is decreased by a certain amount in each episode, calculated by multiplying the old epsilon with a constant rate of 0.9978. After 1000 episodes, the epsilon value will reach a minimum value of 0.2. The actions are randomly chosen in the early stage of the training however, after several training episodes have passed, the neural network, whose weights are updated periodically, generates the action output. With the right weight adjustments, the neural network's action outputs will help the agent drive closer to the target, yielding the highest reward. Conversely, if rewards parameters are unbalanced or the network algorithm is inadequate, improper weight updates may occur. In such cases, even after being trained for thousands of episodes, the action moves by the agent will not assist to accomplish the desired parking task.



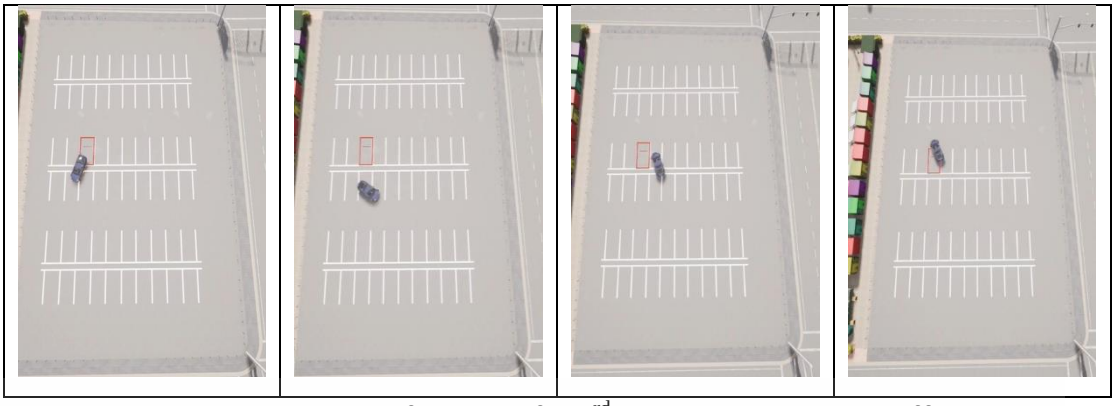


Figure 4-25 Agent Parking performance for 3rd scenario spawned at different location

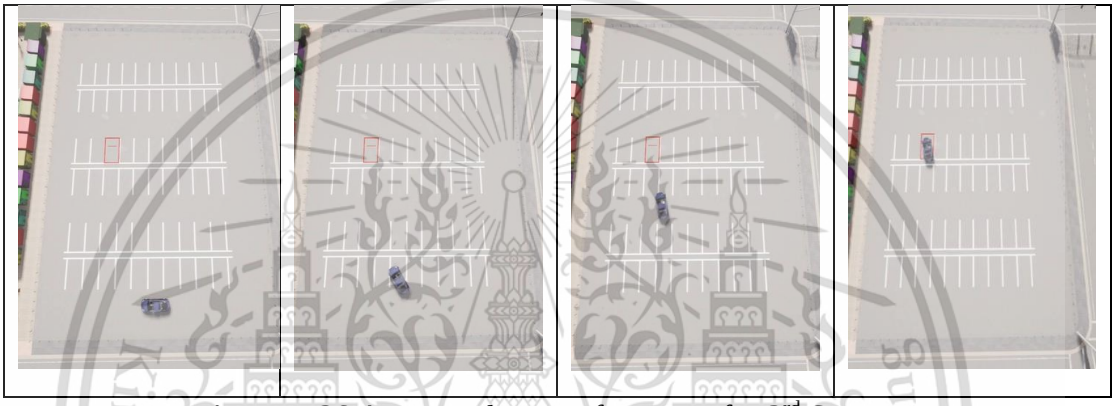


Figure 4-26 Agent Parking performance for 3rd Scenario

Table 4-4 Comparison of 5000 and 3500 Training Episodes in 3rd Scenario

Hyperparameters	Training 5000 Episodes (similar to second scenario)	Training 3500 Episodes
Throttle Speed	**0.5** for 'steer' and 'straight' actions when distance less than a threshold	**Reduced to 0.3** for 'steer' and 'straight' actions when distance less than a threshold
Extra Reward Values	If the vehicle is parked over 20 training seconds, the extra reward added to the original reward is **50**. When the vehicle is able to approach within the parking zone, the extra reward added to the original reward is **2.5**.	If the vehicle is parked over 20 training seconds, the extra reward added to the original reward is changed to **150**. When the vehicle is able to approach within the parking zone, the extra reward added to the original reward is **5**.

4.5.3 Result and Discussion for 3rd Scenario

4.5.3.1 Discussion for Distance Result

The terminal Euclidean distance is the relative distance of the agent from the parking center coordinate when the episode is terminated. The lower the Euclidean distance, the better the performance of the model as the car reached a closer distance to the parking destination. If the car could consistently park successfully in every episode, the fluctuations in the graph will be stagnant and will stay close to a 'zero Euclidean distance' line, which is the ideal scenario.

The figure below depicts the last distance result graph for the third scenario. The **red** line, representing 3500 episodes, demonstrates a superior performance result over the **blue** line, which represents 5000 episodes as the distance trend of the red line decays faster than the blue line. In the start of both training courses, the agent is exploring the environment, resulting in high distances on the graph as randomized action did not help the agent to approach nearer to the destination. The distance declination displayed on the graphs indicates that the agent has learnt policy to generate actions which brings it closer to the destination, thereby maximizing the obtained reward.

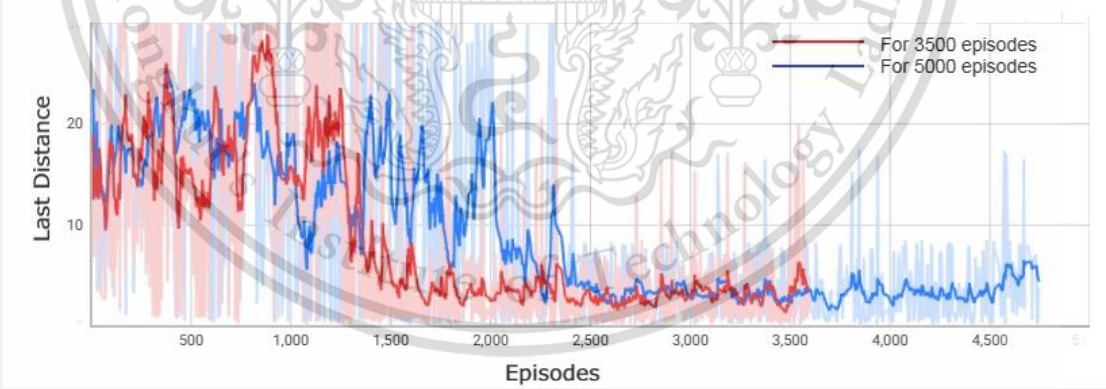


Figure 4-27 Distance Result Comparison for Third Scenario

The blue line starts to drop at 1000 episodes, which is earlier than the red line, however the decay rate is slower than the red line in the latter episodes although the same learning rate of 0.005 in Adam optimizer is applied. This may be due to different reward conditions assigned. Although the decay trend of the blue line continues after 1000 episodes, it frequently exhibits upward shifts, indicating that the agent is still in the process of finding the optimum policy therefore, the result is not stable yet. After

2000 episodes, there is a significant drop in the decay rate and by 2500 episodes, the last distance record stabilizes at around 5 or 6 meters from the parking destination.

Conversely, the red line indicating the final Euclidean distance trend of 3500 training episodes shows a sharp decline from the previous distance of 22 meters at around 1300 episodes to 5 meters at around 1500 episodes. Following 1500 episodes, the rate of declination slows down, and the last distance record remains below an approximate Euclidean distance of 5 meters, occasionally falling as low as 2 m in several episodes.

The red line reached a Euclidean distance of 5 m in just 1500 episodes, compared to almost 2000 episodes for the blue line. However, in the later episodes of both trainings, after reaching a Euclidean distance of 5 meters, the agent struggled to approach a 'zero meter' Euclidean distance. The agent was able to reach the destination, however, it had trouble parking in the designated space within constrained angles as depicted in **Figure 4-25** and **Figure 4-26**. Hence, it spent the entire simulation time attempting to complete the task by circling around the destination. As a result, the episode's final Euclidean distance is only a few meters away from the intended "zero meter" last distance line. The final distance should be less than 1.1 meters while the parking angle constraints are also satisfied to accomplish the parking mission task which is similar to the second scenario's final distance graph.

4.5.3.2 Discussion for Reward Result

The following diagrams depict the reward graph and average reward graph for the third parking training scenario where the agent starts at randomized positions and angles. The average reward represents the mean reward of the last 30 episodes providing a more stabilized result. The reward trend initiated with the negative value because in the earlier training episodes, the agent is still exploring and learning from the environment. Hence, the output actions are randomly chosen from the list of vehicle control actions. Even if the action is generated by the neural network, the network’s weights are not optimized, resulting in actions that do not contribute effectively to the training process. As a result, the network's randomly selected or randomized actions will give a negative impact on the agent in the environment which result in a large negative penalty reward.

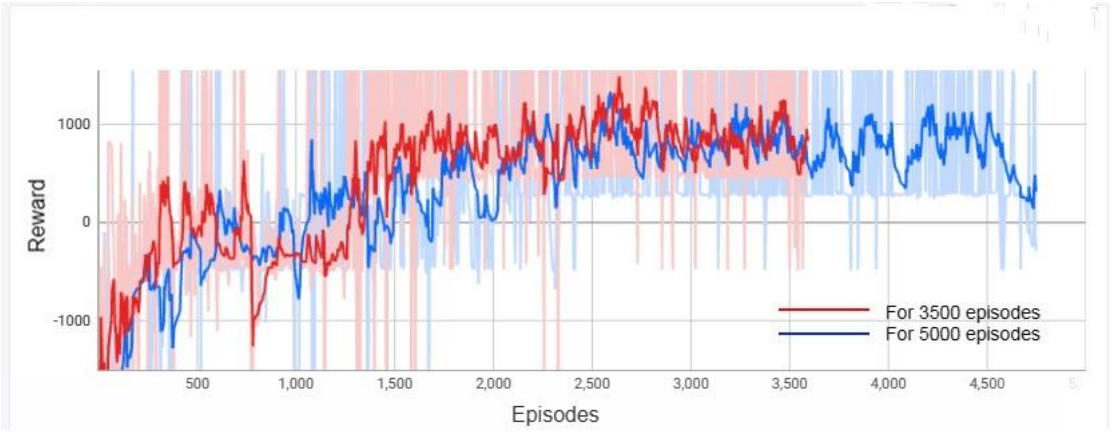


Figure 4-28 Episodic Reward Comparison for Third Scenario

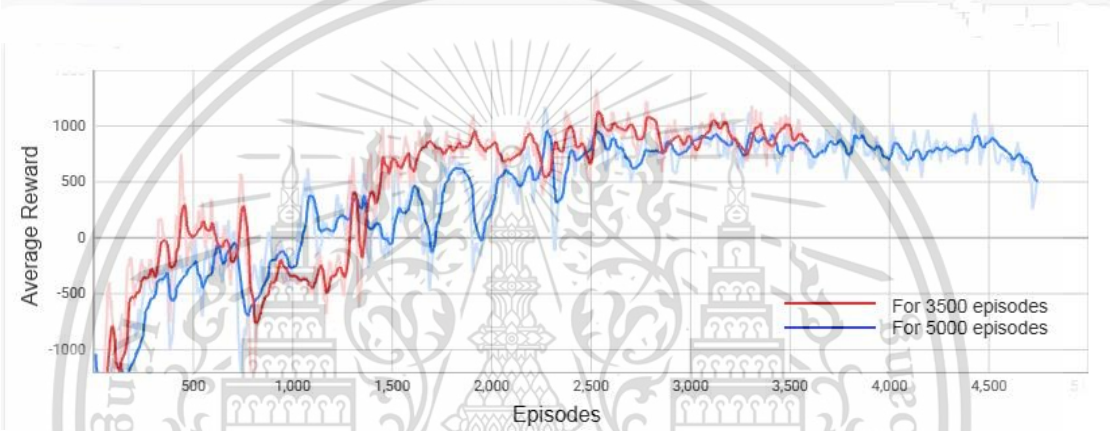


Figure 4-29 Average Episodic Reward Comparison for Third Scenario

The reward graphs steadily increased in a positive direction, indicating an upward trend, as the agent gained sufficient training experience. It shows that the selected actions assist the agent in approaching closer to the target as the agent eventually picks up the policy through iterative training based on prior experiences.

In the average reward graph, it's evident that both the blue line and the red line are trending upward, indicating improvement in the training process, despite starting from a negative value. Since the reward is influenced by the Euclidean distance, a decrease in the Euclidean distance corresponds to a higher reward. The red line experiences a sudden shift from negative to positive rewards around the 1300th episode, coinciding with the Euclidean distance falling below 10 meters for the first time. In subsequent episodes, the reward trend steadily increases and stabilizes around +1000 points, reflecting that the agent's terminal distance remains consistently below 5 meters.

The blue line exceeds over the zero threshold at about 1000 episodes; however, this 5000-episode training shows slower improvement compared to the training of 3500

episodes. The average reward stabilizes around +800 points, indicating that the agent avoided undesired penalty actions and consistently generated the actions which give optimum reward values. The average reward for the red line is higher than the blue line, attributed to slightly larger new reward parameters and the agent drove closer to the destination more frequently in 3500 episodes compared to 5000 episodes.

The parking job was unsuccessful in both training sessions. The agent successfully approached the destination and made attempts to complete the task using actions determined by the network at each time step. However, it ended up in proximity to the goal point when the simulation time elapsed. Therefore, rewards per episode fell short of reaching their maximum value of around +1800. Despite not hitting the maximum reward, the overall total reward of both trainings still achieved a positive score as the agent consistently positioned itself close to the goal destination.

A new reward setting, or algorithm method is obviously needed to assist the agent to fit exactly inside the parking angle constraint, as this is the primary challenge to complete the parking task. The average reward for both training courses leveled out starting with 2000 episodes.

To sum up, in this third scenario, the agent car is spawned with a randomized position and angle within a confined parking space. The spawned position is varied within a defined range and the spawned angle is also randomly assigned at the start of each new training episode. The objective of this mission is to successfully park the agent into the desired parking lot, aligned with the parking boundaries. After training for a thousand episodes, the agent car was able to optimize its strategy to approach closer to the destination, however, it struggled to find an effective method to park the car within the confined Euclidean distance and the parking angle limitation. This might be the result of inadequate modifications in reward and angle calculation. Although the reward and angle algorithms have been changed and tested, the results are unsatisfactory.

4.5.4 Discussion why agent could not park successfully for 3rd Scenario

After modifying the network model and initial position, the agent's performance improves noticeably. The agent can now navigate towards the destination point despite starting from the random position and angle in each episode. To complete the task and

end the episode, the agent needs to hover in the parking zone with correct parking angle alignment for 2-time steps. When the agent enters the parking zone, it attempts to align within the parking boundaries. It sometimes drives exceed over the parking zone due to high speed or fails to maintain the current parking location for 2-time steps or violating the parking angle restrictions. In such cases, the agent executes a turn to reposition itself and retry parking. This process repeats until the parking task is successfully completed. This iterative parking action may be due to parking angle restrictions at the destination goal point, causing the agent car to struggle to park within the specified parking angle restrictions at the destination. The speed of the agent car should be reduced in further training to access full control of the vehicle. Moreover, to overcome the parking angle constraints, the agent car needs an extra steering angle precision control algorithm to accomplish the task successfully.

Parking success case: If the parking angle limitations are neglected, this parking task for the 3rd scenario can be assumed as a success case as the agent car could drive to the destination from different spawned locations and drive over the parking area repeatedly. The agent crossed over the parking coordinate point and the Euclidean distance is less than 1.1m, however the mission is unsuccessful as the parking angle constraints are not satisfied. In real world scenario, the agent must abide by the traffic regulations therefore, parking angle limitation should be taken into consideration. Therefore, further algorithms modifications need to be implemented to boost the performance of the training agent to accomplish the task.

CHAPTER 5

CONCLUSION

In this study, three different parking scenarios of the autonomous driving vehicle at the desired destination are conducted in the CARLA simulated driving environment and the performance results are analysed.

Employing an iterative training method, this research investigated the parking capabilities of the trained agent within a 3D simulation environment, assessing the ability of the agent to navigate from the initial spawned position to the designated parking spot within a specified timeframe. The Carla simulation environment is run through a Python API, and the agent's vehicle is positioned on the Town 5 map, where the parking task occurs. The Deep Q Network technique is utilized for training the agent. The agent initiates training without any prior domain knowledge or experience, accumulating a defined set of experiences through exploration in the environment. The Deep Q neural network model predicts the agent's actions, and the weights of the network neurons undergo updates and optimization by training iteratively using experience batches, stored in a memory buffer. Various network hyperparameters, including learning rate, epsilon decay, layer architecture, neurons per layer, epochs, replay memory size, etc., are adjusted iteratively based on the training performance results.

Factors influencing the training scenario performance in this study include reward configurations, input data for the neural network, and the learning rate hyperparameters. In the first scenario, the agent successfully parked in the designated zone and accomplish the task using a straightforward driving action. However, replicating the training with identical parameters in the second scenario led to subpar performance due to slight changes in the agent's spawn position and parking destination. Despite altering the hyperparameters and model architecture, satisfactory results were elusive until an additional input, the turning angle, was introduced. This new input guided the training agent effectively toward the destination, even with a randomly altered spawn location. Furthermore, this turning angle is also used to compute a new additional angle reward. In the third scenario, the agent, trained with the same reward settings but with a different spawn position, yielded unsatisfactory results. While the agent could navigate towards the destination and enter the restricted parking zone, however, the challenge lay in adhering to parking angle constraints to complete the task.

Through the feedback rewards, the agent acquires knowledge and learns changes associated with the actions taken for each action-state pair at every time step. Key parameters such as weights and biases in the neural network are adjusted in each time step to minimize the error in predictions. The modification of network parameters is done by the Adam Optimizer, and the weight values of neurons in each layer are updated in a reverse direction, starting from the final hidden layer—commonly referred to as the backpropagation method. By engaging in repetitive training with experience batches, the neural network incrementally developed assurance in its existing network policy. This led to the prediction of action values, ultimately generating optimized rewards through deep calculation process.

If the network weights are updated properly, the gradient descent will converge and reach to the local minimum which is ideal. The action predications generated by the agent will assist it to approach nearer to the destination in a short simulation time, resulting in higher reward return. Conversely, if the network weights are not updated properly, the agent may fail to accomplish the task even after being trained for thousands of episodes.

This study presents rooms for enhancement to make it more applicable in real-world situations. Currently, the parking task is limited to an assigned area in the parking lot, and for simplification of the training, the traffic rules are neglected, and the static and dynamic obstacles are excluded. Although the proposed algorithm performs well in trained scenarios, a new training session is required when deploying the agent in a new parking lot. The boundary conditions need to be modified since the data used in this training is specified to this scenario.

In summary, this study focuses on training an autonomous driving car for parking using the Deep Q Network. While the trained agent's performance is satisfactory in these particular scenarios, there is potential for the DQN model algorithm to enhance its performance in diverse driving scenarios. Future research should explore modifications and testing with additional sensors, driving boundary conditions, and new control algorithms to advance the implementation of the car into fully autonomous driving mode.

CHAPTER 6
FUTURE WORK

6.1 ADDITIONAL SCENARIOS

Many global vehicle manufacturers are heading towards the trend of developing autonomous driving cars and their performance is developed and tested on different driving scenarios and environmental conditions. Google's self-driving car project, called Waymo [36], uses a mix of sensors, lidar and cameras and combines all the data of those systems generate to identify everything around the vehicle and predict what those objects might do next. The driving experience data is collected and saved for millions of miles. Map territory in detail is collected via both GPS systems and custom maps. Because driving conditions are dynamic and unpredictable, Waymo Driver's perception system uses sophisticated data from its array of in-car sensors to interpret its surroundings, including pedestrians, cyclists, cars, construction, and traffic laws, using machine learning technology. The Waymo Driver uses all of this data to choose the optimal course of action or path, including its extremely comprehensive maps and information on nearby objects and their potential movements. It quickly ascertains the precise course, velocity, lane, and steering movements required to behave safely during its travel.

WAYVE vehicle[37] uses a pioneering end-to-end AI driving model in its new AV2.0 approach. In this new system, Camera, Radar/ LiDAR and GPS is used as sensors and those are transferred to “Safety Expert Sub-System” AI neural network model, and then, those output from AI Model is directed to its own produced vehicle control system (Vehicle Motion Plan and Drive-By-Wire) to execute the vehicle action movements. In the previous AV 1.0 approach, instead of the end-to-end AI model, perception tasks (Lane Detection, Traffic light and sign detection, Object detection and tracking, Free space detection and localization etc…) are done by using sensor data. Planning task follows perception task, which includes route planning, prediction, behaviour planning, trajectory planning and localization in map. From the data from perception and planning task, the agent car could navigate itself in the environment using PID controller and Model Predictive Control methods and others.

For the control algorithms of the autonomous cars, rule-based systems, neural network systems and hybrid systems are available. However, rule-based systems could not cover all driving scenarios as driving process is unpredictable. Neural network

systems are highly adaptable to new data and situations which makes them to handle complex decision-making process and learning the patterns of the data. However, as it requires an extremely huge amount of training data. Due to its black box nature, the working process and decision making of the neural network cannot be known. Therefore, hybrid system, both rule-based and neural network work together to enable the complex control required for autonomous driving.

Additional scenarios are suggested to improve the driving scenario to resemble the real-world driving environment. In this chapter, two extra driving scenarios will be explored for the future work alongside the necessary reward functions required for each scenario.

Scenario 1: Parking of the car from the spawn position to the desired destination which is between the two parked cars without any collision.

Scenario 2: Parking of the car from the spawn position to the desired destination while avoiding an obstacle midway.

6.2 Scenario 1: Parking of the car between two parked cars

In this suggested scenario, the driving environment is modified by adding two spawned parked cars just beside the goal destination. The driving task is more difficult as the agent needs to avoid collision of those cars besides accomplishing the original parking task. To complete the parking task in a new assigned parking location, the agent must employ all vehicle control mechanisms. According to the previous training scenarios, the agent car could show its good performance after training for 5000 episodes while each episode took 40 seconds for training.

Figure 6-1 Parking Scenario for parking for the car between two parked cars

6.2.1 Environmental Boundary Conditions

Tesla Model 3 car is used as the agent car throughout the simulation training process. The agent car is supposed to be spawned at random yaw angle within the maximum range of ± 20 degrees from a default 180-degree orientation. The spawned coordinate of the agent is within the maximum distance range of ± 1.5 m from the default spawned position, which was originally situated approximately 26 meters far from the center of the parking area. Changing the spawned positions and yaw angles of the agent can generalize the training experiences.

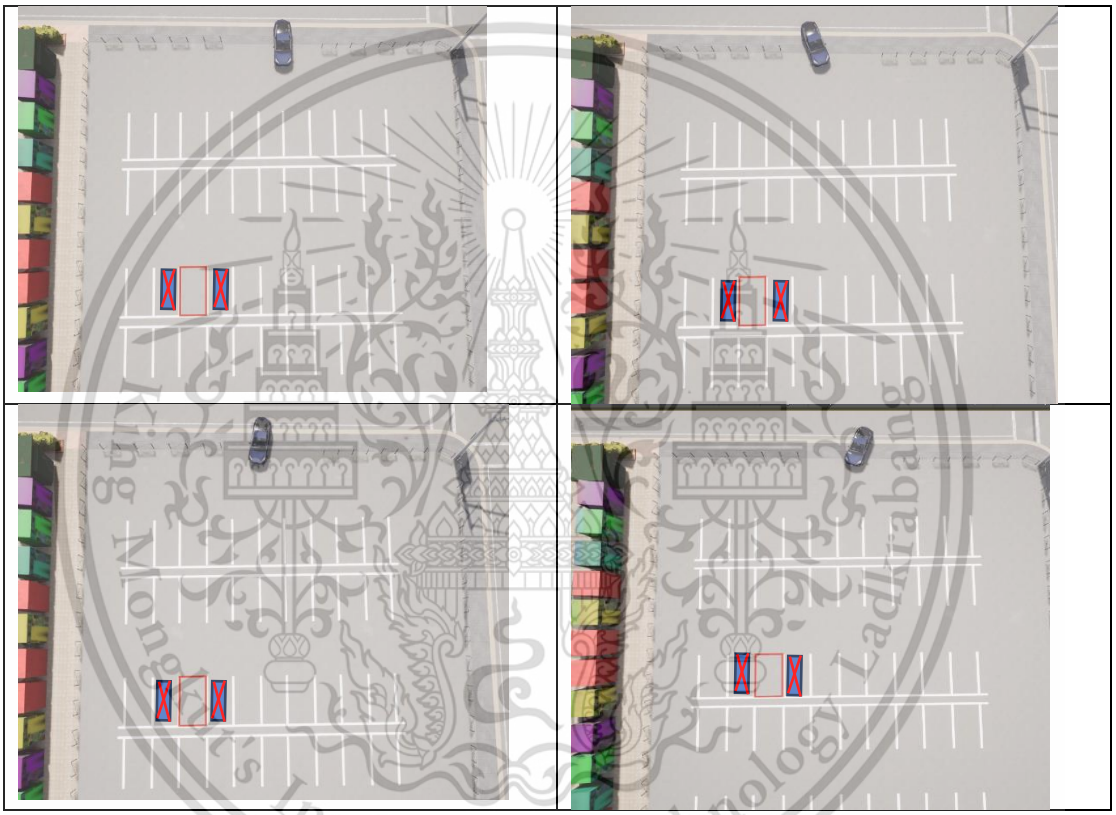


Figure 6-2 Boundary Conditions for Parking between two cars

6.2.2 Proposed Neural Network Architecture

This scenario is similar to the second scenario; hence, 7x64x64x4 size network architecture is proposed, consisting of 7 inputs, 64 neurons in each hidden layer, and 4 neurons in the output layer just like the second scenario. Input parameters are 1,2) the relative distance in x and y direction from the parking coordinate to the current agent position, 3) the Euclidean distance, 4) the yaw angle of the car, 5) the velocity in x direction, 6) the angular velocity, and 7) angle difference(α) between the yaw angle and the line connecting two coordinates.

The model's performance is significantly improved by adding the angle difference (α) value to the DQN model since it gives the agent information of how much angle should be steered to reach its destination. The output layer generates 4 action values, representing specific vehicle control actions in each state, namely forward, reverse, stop, and steer. If the steering value is 'negative', the agent will make a left turn, otherwise, it will make a right turn for the 'positive' value.

Table 6-1 List of Hyperparameters and Values for Additional Scenario

Hyperparameters	Values	Descriptions
Network architecture	7x64x64x4	2 hidden dense layers with 64 neurons
Activation function	ReLU	Function used to create non-linearity outputs
Replay memory size	100,000	Set of training experiences stored and SGD updates are sampled from here.
Target Network update frequency	20 episodes	The frequency to update the weights of the Target Network
Discount factor	0.95	A value represents how future rewards are crucial
Learning rate	0.001	Tuning parameter that determines how fast the network learns to converge at global minimum value.
Epsilon Decay	0.997	A constant rate to decrease the Epsilon value in every new episode
Minibatch size	128	No. of experiences in each batch
Initial Exploration	1	The initial value of Epsilon
Final Exploration	0.00001	Minimum Epsilon Value
Number of DQN inputs, outputs	7, 4	Number of input and output parameters to Q- network

6.2.3 Reward Function

The learning process of the neural network is similar to how humans learn to accomplish a work. Each neuron in a neural network possesses a weight value and they undergo over iterative prediction and backpropagation process for updating the network parameters. The optimal policy to accomplish a certain task would be found when those weight values become optimum. The policy of the network to predict the output is shaped and modified by the **feedback of the reward functions**. If the predicted action of the network benefits the agent and helps it to approach nearer to the destination, a positive reward is provided, and it is saved as a memory. On the other hand, if the predicted action causes the agent hits the obstacles or drives over the distance limit, a

negative penalty reward is provided, and this will also be saved as a memory. The network is trained by using small batches of random samples from the replay memory.

The policy of deep Q Network is shaped by the feedback reward functions. Setting appropriate reward functions is crucial because it defines how great the agent could accomplish the desired task. If the reward function is not well established, the performance of the agent will deviate from our intended mission.

In this scenario, three reward functions would be used to guide the agent to accomplish the parking task. The total reward would be the combination of the distance reward and the angle reward, and the resultant is multiplied by the time reward as explained in **eq(4.1).** Euclidean distance is the relative distance of the current coordinate of the agent car to the destination. **Distance reward** is a non-linear reward, calculated on the Euclidean distance and the closer the agent car to the destination, the higher the reward will become.

Angle reward for angle steering to the destination, firstly, the deviation angle of the car is calculated. The deviation angle in **Figure 4-2** is the difference between the yaw angle of the current agent position and the arctan angle, which is the angle from the positive x axis to the line connecting the current position coordinate to the destination. Hence, if the deviation angle is huge, it means, the agent car deviates too much from the destination, therefore, the angle reward will be minimum. However, if the deviation angle is 'ZERO', the agent car is heading straight to the destination, which results in maximum angle reward.

Time Reward is used to encourage the agent to accomplish the task as fast as possible. Although the allotted time for each episode is 40 seconds, the agent is encouraged to complete the task as soon as possible. In the early training episodes, the agent will spend the entire training period exploring the environment. After accomplishing the parking task successfully, the agent will realize how to approach the destination and its driving actions will become exact and faster compared to the early training period. The agent will receive a maximum time reward of +1, when it can finish the parking task within 12 seconds. In that case, the total reward for the agent for this current step will only depend on the distance and angle reward. If the simulation time exceeds over 12 seconds, the time reward will decrease gradually, which will in turn decrease the total reward.

The **angle penalty reward** is used only when the agent car gets into the parking area and prepares to park. It is used to help the agent to park fit into the parking

boundary. The yaw angle of the agent car at the parking area is limited to ± 15 degrees from the vertical parking boundary, hence, the parking task will be successfully accomplished when the Euclidean distance of the agent is less than 1.1 m from the destination while its yaw angle is less than ± 15 degrees from the vertical parking boundary.

6.2.4 Discussion

In this scenario, extra spawned cars are added to update the training scenario. Collision sensor will notify the network whether the agent car crashed into the obstacles. In the early stages of the training scenario, the agent's actions are unpredictable, for instance, hitting the parked cars or colliding the boundary fence or drive over the distance limit because of the less control of the vehicle action predicted by neural network, or which is picked randomly. During the early training stage, as the neural network does not have enough experience and confidence, the selection of the predicted actions is slow, unproductive and will take the entire training time for exploring the environment.

Whenever the agent car hits the parked cars or the boundary fence, a collision impulse data will be reported, a huge negative penalty reward will be provided to the agent as a notice of the undesired driving action and the current training episode will end on the spot. The total episodic reward will be calculated, which is highly to be a negative value, the training experience data will be saved in the replay memory and a new training episode will be loaded again with the agent car at the spawned location.

The agent will learn from the previous training, and it will gradually avoid the action which causes the negative penalty reward. The agent will gradually learn from the feedback functions and approaches nearer to the destination results in the positive reward which is desired. Therefore, it will repeat the same action and path in the following episodes while avoiding the obstacles. When the agent learns that getting into the parking zone destination within the limited parking angle within a certain time limit can result in the maximum highest reward, it will try to follow the similar policy and will try to accomplish the parking task in the succeeding episodes.

The neural network is updated regularly based on the error feedback between the actual action and the predicted action. After numerous trainings and predictions, and the weights of the neurons will be optimized, the neural network gained confidence in prediction actions which help the agent to approach nearer to the destination and

accomplishing the task, while avoid collision of the parked cars and boundary fence. Moreover, the time taken to accomplish the task will also decrease.

In this scenario, although the parked cars are added besides the destination area, the agent car will be able to accomplish the parking task by using the proposed neural network architecture and reward functions. The policy of the neural network is updated by using the training data and reward functions, therefore, although the agent car could hit the obstacles in the early training episodes, it will learn gradually in each training episode and able to avoid the parked cars or boundary fence.

6.3 Scenario 2: Parking the car while avoiding an obstacle

In this parking scenario, an obstacle is placed midway, hence, the agent car has to drive around it and park at the designated location to assume the parking task is successful. This scenario is addressed to resemble the performance of the car in the real-world scenario. According to the previous training experiments, training for the agent car for 5000 episodes could show the good performance result, while each episode took 40 seconds for training.

6.3.1 Environmental Boundary Condition

The Tesla Model 3 agent car and its spawned positions are used similarly to the previous scenario. The goal destination is also assigned at the same location while the only additional boundary condition is that an extra parked car is added as an obstacle.

Figure 6-3 Boundary Conditions for Parking with an obstacle

6.3.2 Perception of the Environment

In this scenario, a parked car is added to the mid-way of the parking path. The agent car should start from the spawned position, avoid that parked car and park at the destination to assume the parking process accomplished. Autonomous cars which are industrially produced may include multiple sensors for perception of the environment such as the cameras, the Light Detection and Ranging (LiDAR) sensors and the Radio Detection and Ranging (RADAR) sensors. They are crucial for providing real-time data about the environment, which the vehicle's systems process to make decisions about how to control the vehicle.

Figure 6-4 Perception of an agent car using Lidar Sensor

In this scenario, in addition to the collision sensor, the LiDAR sensor is proposed to mount on the roof of the agent car for detection of the obstacles. LiDAR uses laser light to detect and measure distances between objects, creating a 3D map of the environment. The distance of the surrounding objects can be calculated by the time taken for the laser beam to return to the laser sensor from the laser emitter. It is highly accurate and effective in low-light conditions and can accurately detect objects in rain, fog, and snow. The LiDAR measurements may include the coordinate of the space points to indicate the obstacle and the intensity of the laser beam during the travel. The LiDAR system has a limited amount of horizontal and vertical views; therefore, the sensor is rotated on every simulation step to cover a specific angle view of surroundings.

The lidar sensor data is used to find the obstacles in the surrounding environment so that the agent could estimate the location of the obstacles and could avoid it. [38]

6.3.3 Proposed Neural Network Architecture

In this scenario, a neural network architecture of 9x64x64x4 size is proposed. A**dditional LiDAR sensor is used**, therefore, the number of inputs to the neural network is increased to 9 values. The neural network comprises of 9 inputs, 64 neurons in each hidden layer, and 4 neurons in the output layer. Input parameters are 1,2) the relative distance in x and y direction from the parking coordinate to the current agent position, 3) the Euclidean distance, 4) the yaw angle of the car, 5) the velocity in x direction, 6) the angular velocity, 7) angle difference(α) between the yaw angle and the line connecting two coordinates and 8,9) x and y distance of the obstacle from the agent detected by the lidar sensor.

The output layer generates 4 action values, representing specific vehicle control actions in each state, namely forward, reverse, stop, and steer. If the steering value is 'negative', the agent will make a left turn, otherwise, it will make a right turn for the 'positive' value.

The hyperparameters of the neural network could be assigned with the same values as mentioned in **Table 6-1** in the previous scenario as the training process is similar. For parameter tuning for better results, the learning rate could be modified.

6.3.4 Reward Functions

The optimal policy to predict the action output using the Deep Reinforcement learning method depends on how reward functions are designed. In previous scenarios, only distance reward, angle reward and time reward are used to shape the policy and the performance of the agent. The **distance reward** is used to encourage the agent to get closer to the destination as the closer to the destination, the higher the reward becomes. The **angle reward** is used to guide the direction of the agent to drive forward to the destination. If the agent is heading to the destination, the reward is maximum, however, the bigger the deviation angle between the car's yaw angle to the destination coordinate, the smaller the reward will become. The **time reward** is used to urge the agent to accomplish the task as fast as possible.

In addition to the above reward systems, the **lidar reward** is added as a new reward function to shape the policy so that the agent car could avoid the obstacle and drive around it. According to [39]the LiDAR coverage distance in the environment

could be set up with the following specifications: a lateral distance range of 3 meters from the car, distance from the road to 1.3 meters on the vertical axis, and the longitudinal distance of 70 meters in the longitudinal axis, so if there is an obstacle in this area, it can be detected. The LiDAR sensor returns the point clouds containing the x,y,z location in the space, showing the location of the obstacle. Therefore, the difference between the point cloud coordinates and the current location of the agent is used as the minimum distance.

When the distance of the detected obstacle is farther than 30 meters, as the driving speed is slow, the agent car has enough time to change direction, therefore, no penalty reward is provided. When the lidar sensor detects an obstacle less than 30 meters, the lidar penalty reward is applied as a warning. When the lidar sensor detects the obstacle within 10 meters, it is an emergency so, the negative penalty value increases as the agent approaches nearer to the obstacle. The LiDAR reward function is referred from [39]it is imitated from human's behavior, and it can be expressed as

$$Lidar\ Reward = a_1 d_1^4 + a_2 d_1^3 + a_3 d_1^2 + a_4 d_1 + a_5 \quad (6.1)$$

$$d_1 = min\sqrt{x_0^2 + y_0^2 + z_0^2} \quad (6.2)$$

where, x_0, y_0, z_0 are the coordinates of the detected points, which are the distance magnitude of the detected position to the agent. The coefficient $a_{1,}a_{2,}a_{3,}a_4\ and\ a_5$ are -0.0011, 0.05, -6.60,145 and -3040 respectively, which are derived from the trial-and-error method. Adding this LiDAR reward can help the agent to avoid the obstacle within the safe distance, resulting in avoiding collisions which cause negative penalties.

Therefore, in this scenario, the total reward is the sum of distance, angle and lidar reward then the sum is multiplied with time reward, and it can be expressed as

$$Total\ Reward = (Distance\ Reward + Angle\ Reward + Lidar\ Reward) * Time\ Reward \quad (6.3)$$

In this scenario, an obstacle is placed midway of the path. Once the agent is spawned, it will receive the collision sensor data and LiDAR sensor data from the environment. If the detected obstacle is over 30 meters far from the agent, it is assumed as a safe distance, therefore, no penalty will be applied. The LiDAR reward value is negligible, and the total reward will be the same as in the earlier scenarios using only distance, angle, and time reward. There will be no effects on updating the neural

network weights and algorithm's decision-making steps. When the detected obstacle is less than 30 meters but higher than 10 meters, the agent car still possesses enough time to steer and avoid it, but a small negative penalty is given as a warning. When the detected obstacle is less than 10 meters far from the agent, it is highly possible that the agent would hit the obstacle therefore, a huge negative penalty is provided which changes the decision-making policy of the agent.

So, in the beginning of the training, the agent will collide with the obstacles resulting in negative total rewards, however, as the agent has been trained for sufficient episodes using the past experiences, it generally finds the policy which makes the agent to drive around that obstacle and then, drive to the destination to return the highest total reward.

6.4 Omitting white lanes

In Carla simulator, each lane supports waypoints ID to locate itself and hold information of the left and right lane markings, a data of whether it is a junction or lane changing permissions. However, way point data is not supported in the parking area. Therefore, in this training case, several driving rules are omitted. In the real driving environment, the agent car could abide by the rules and could not neglect the while lane boundary lines. However, in our study, we only observed the development of the neural network which is suitable for our proposed driving scenarios while omitting the considerations of the rules.

6.5 Summary

To conclude, the additional scenarios of the parking task can be accomplished by changing the reward functions to influence the neural network policy and adding the additional LiDAR sensor on the agent for the perception purposes. As the Deep-Q-Network is reward based algorithm, assigning appropriate reward functions could improve the driving performance and accomplish the desired parking task without colliding with the obstacles and parked cars. Moreover, the LiDAR sensors help the agent to get wider information of the surrounding environment, therefore, reducing the risk of collision accidents.

REFERENCES

[1] "Road traffic injuries." Accessed: Dec. 24, 2023. [Online]. Available: https://www.who.int/news-room/fact-sheets/detail/road-traffic-injuries

[2] "Path To Autonomous." Accessed: Dec. 24, 2023. [Online]. Available: https://www.gm.com/commitments/path-to-autonomous

[3] "Zoox - Mobility designed around you." Accessed: Dec. 24, 2023. [Online]. Available: https://zoox.com/vehicle

[4] "Waymo Driver." Accessed: Dec. 24, 2023. [Online]. Available: https://waymo.com/waymo-driver/

[5] M. Toromanoff, E. Wirbel, and F. Moutarde, "End-to-End Model-Free Reinforcement Learning for Urban Driving using Implicit Affordances," Nov. 2019, [Online]. Available: http://arxiv.org/abs/1911.10868

[6] H. Liu, Z. Huang, J. Wu, and C. Lv, "Improved Deep Reinforcement Learning with Expert Demonstrations for Urban Autonomous Driving," in *IEEE Intelligent Vehicles Symposium, Proceedings*, 2022. doi: 10.1109/IV51971.2022.9827073.

[7] L. Claussmann, M. Revilloud, D. Gruyer, and S. Glaser, "A Review of Motion Planning for Highway Autonomous Driving," *IEEE Transactions on Intelligent Transportation Systems*, vol. 21, no. 5, 2020, doi: 10.1109/TITS.2019.2913998.

[8] G. M. Hoffmann, C. J. Tomlin, M. Montemerlo, and S. Thrun, "Autonomous automobile trajectory tracking for off-road driving: Controller design, experimental validation and racing," in *Proceedings of the American Control Conference*, 2007. doi: 10.1109/ACC.2007.4282788.

[9] M. Treiber and A. Kesting, "The Intelligent Driver Model with Stochasticity - New Insights into Traffic Flow Oscillations," in *Transportation Research Procedia*, 2017. doi: 10.1016/j.trpro.2017.05.011.

[10] D. C. Gazis, R. Herman, and R. W. Rothery, "Nonlinear Follow-the-Leader Models of Traffic Flow," *Oper Res*, vol. 9, no. 4, 1961, doi: 10.1287/opre.9.4.545.

[11] M. Bojarski *et al.*, "End to End Learning for Self-Driving Cars," Apr. 2016, [Online]. Available: http://arxiv.org/abs/1604.07316

[12] G. Williams *et al.*, "Information theoretic MPC for model-based reinforcement learning," in *Proceedings - IEEE International Conference on Robotics and Automation*, 2017. doi: 10.1109/ICRA.2017.7989202.

[13] Y. Pan *et al.*, "Imitation learning for agile autonomous driving," *International Journal of Robotics Research*, vol. 39, no. 2–3, 2020, doi: 10.1177/0278364919880273.

[14] V. Mnih *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, Feb. 2015, doi: 10.1038/nature14236.

[15] D. Li, D. Zhao, Q. Zhang, and Y. Chen, "Reinforcement Learning and Deep Learning Based Lateral Control for Autonomous Driving [Application Notes]," *IEEE Comput Intell Mag*, vol. 14, no. 2, 2019, doi: 10.1109/MCI.2019.2901089.

[16] Ó. Pérez-Gil *et al.*, "Deep reinforcement learning based control for Autonomous Vehicles in CARLA," *Multimed Tools Appl*, vol. 81, no. 3, 2022, doi: 10.1007/s11042-021-11437-3.

[17] B. Thunyapoo, C. Ratchadakorntham, P. Siricharoen, and W. Susutti, "Self-Parking Car Simulation using Reinforcement Learning Approach for Moderate Complexity Parking Scenario," in *17th International Conference on Electrical Engineering/Electronics, Computer, Telecommunications and Information*

Technology, ECTI-CON 2020, 2020. doi: 10.1109/ECTI-CON49241.2020.9158298.

[18] A. Dosovitskiy, G. Ros, F. Codevilla, A. López, and V. Koltun, "CARLA: An Open Urban Driving Simulator."

[19] D. Li and O. Okhrin, "Modified DDPG car-following model with a real-world human driving experience with CARLA simulator," Dec. 2021, [Online]. Available: http://arxiv.org/abs/2112.14602

[20] L. Junzuo and L. Qiang, "An Automatic Parking Model Based on Deep Reinforcement Learning," in *Journal of Physics: Conference Series*, IOP Publishing Ltd, Apr. 2021. doi: 10.1088/1742-6596/1883/1/012111.

[21] M. Khalid, L. Wang, K. Wang, N. Aslam, C. Pan, and Y. Cao, "Deep reinforcement learning-based long-range autonomous valet parking for smart cities," *Sustain Cities Soc*, vol. 89, Feb. 2023, doi: 10.1016/j.scs.2022.104311.

[22] R. Takehara and T. Gonsalves, "Autonomous Car Parking System using Deep Reinforcement Learning," in *2021 2nd International Conference on Innovative and Creative Information Technology, ICITech 2021*, Institute of Electrical and Electronics Engineers Inc., Sep. 2021, pp. 85–89. doi: 10.1109/ICITech50181.2021.9590169.

[23] W. Terapaptommakol, D. Phaoharuhansa, P. Koowattanasuchat, and J. Rajruangrabin, "Design of Obstacle Avoidance for Autonomous Vehicle Using Deep Q-Network and CARLA Simulator," *World Electric Vehicle Journal*, vol. 13, no. 12, 2022, doi: 10.3390/wevj13120239.

[24] "Carla Simulator Documentation." Accessed: Dec. 22, 2023. [Online]. Available: https://carla.readthedocs.io/en/latest/

[25] Richard S.Sutton and Andrew G.Barto, *Reinforcement Learning An Introduction*.

[26] Open AI, "Key Concepts in RL." Accessed: Dec. 24, 2023. [Online]. Available: https://spinningup.openai.com/en/latest/spinningup/rl_intro.html

[27] V. Mnih *et al.*, "Playing Atari with Deep Reinforcement Learning," Dec. 2013, [Online]. Available: http://arxiv.org/abs/1312.5602

[28] "What is a neural network?", Accessed: Dec. 24, 2023. [Online]. Available: https://aws.amazon.com/what-is/neural-network/

[29] "Keras 3 API Documentation/Layers API/ Layer weight initializers," https://keras.io/api/layers/initializers/. Accessed: Dec. 22, 2023. [Online]. Available: https://keras.io/api/layers/initializers/

[30] "Better Weight Initialization Methods in Deep Learning." Accessed: Dec. 24, 2023. [Online]. Available: https://www.linkedin.com/pulse/better-weight-initialization-methods-deep-learning-sasidhar-reddy/

[31] "A Gentle Introduction to the Rectified Linear Unit (ReLU)." Accessed: Dec. 24, 2023. [Online]. Available: https://machinelearningmastery.com/rectified-linear-activation-function-for-deep-learning-neural-networks/

[32] Deep Lizard, "Backpropagation Intuition." Accessed: Dec. 24, 2023. [Online]. Available: https://deeplizard.com/learn/video/XE3krf3CQls

[33] Ketan Doshi, "Reinforcement Learning Explained Visually (Part 5): Deep Q Networks, step-by-step," *towardsdatascience.com*.

[34] Jordi TORRES.AI, "Deep Q-Network (DQN)-II." Accessed: Dec. 24, 2023. [Online]. Available: https://medium.com/towards-data-science/deep-q-network-dqn-ii-b6bf911b6b2c

[35] Bonsai, "Deep Reinforcement Learning Models: Tips & Tricks for Writing Reward Functions." Accessed: Dec. 24, 2023. [Online]. Available:

https://medium.com/@BonsaiAI/deep-reinforcement-learning-models-tips-tricks-for-writing-reward-functions-a84fe525e8e0

[36] “Waymo”, Accessed: Apr. 30, 2024. [Online]. Available: https://waymo.com/waymo-driver/

[37] “Wayve Autonomous Car ”, Accessed: May 01, 2024. [Online]. Available: https://wayve.ai/technology/

[38] “Carla Simulator>>Docs>>Sensor Ref.” Accessed: Apr. 30, 2024. [Online]. Available: https://carla.readthedocs.io/en/latest/ref_sensors/#lidar-sensor

[39] W. Terapaptommakol, D. Phaoharuhansa, P. Koowattanasuchat, and J. Rajruangrabin, “Design of Obstacle Avoidance for Autonomous Vehicle using Deep Q-Network and CARLA Simulator,” 2022, doi: 10.3390/wevj1010000.

APPENDIX A:
CONFERENCE PARTICIPATION

講演番号	047	文献番号	20235047

Low Speed Car Parking Manoeuvring using Deep Reinforcement Learning

Khin Khin Kyi [1,2] Wasinee Terapaptommakol[2] Supat Kittiratsatcha [1] Masaki Yamakita [3] and Jartuwat Rajruangrabin [2]

1) School of Engineering, King Mongkut's Institute of Technology Ladkrabang, Bangkok,10520, Thailand
2) Rail and Modern Transports Research Center, National Science and Technology Development Agency, Pathum Thani 12120, Thailand
3)School of Engineering, Tokyo Institute of Technology, Tokyo 152-8550, Japan

ABSTRACT: Autonomous driving research have been conducted in different driving scenarios and environments. This study presents the parking scenario of a four-wheel vehicle in Carla 3D Simulation Environment using Deep Reinforcement Learning. The agent vehicle learns and trains in environment concurrently based on the rewards and punishments in each episode. Euclidean Distance is used as the input of the agent and the agent actions are chosen by the weights of the Deep-Q-Network. The agent reached the optimum training and testing condition with the stable high rewards in each episode by parking the car in the determined location without having any collision.

KEY WORDS: **electronics and control, autonomous driving system,** Deep Q-Network, Carla Parking Simulation (E1)

The author participated in **2023 JSAE Annual Congress (Spring) conference** which is held in May 2023.

AUTHOR BIOGRAPHY

Author: Ms. Khin Khin Kyi
Nationality: Myanmar
Email: 64601177@kmitl.ac.th

Educational Record

2023 Master of Automotive and Advance Transportation Engineering
Faculty of Engineering, King Mongkut's Institute of Technology Ladkrabang

2018 Bachelor of Engineering, Mechatronics Engineering
Yangon Technological University

Scholarship/ Research Grant — Thailand Advanced Institute of Science and Technology and Tokyo Institute of Technology (TAIST-Tokyo Tech), 2021

Conference Participation Publications:

[1] K. K. Kyi, W. Terapaptommakol, S. Kittiratsatcha, M. Yamakita, and J. Rajruangrabin, "Low Speed Car Parking Manoeuvring using Deep Reinforcement Learning." 2023 JSAE Annual Congress (Spring), Proceedings (Spring May 2023)