AUTONOMOUS DRIVING USING DEEP REINFORCEMENT LEARNING

By

Mohammed Abdou Tolba

A Thesis Submitted to the
Faculty of Engineering at Cairo University
in Partial Fulfillment of the
Requirements for the Degree of
MASTER OF SCIENCE
in
ELECTRONICS AND COMMUNICATIONS ENGINEERING

FACULTY OF ENGINEERING,
CAIRO UNIVERSITY
GIZA, EGYPT
2017

AUTONOMOUS DRIVING USING DEEP REINFORCEMENT LEARNING

By
Mohammed Abdou Tolba

A Thesis Submitted to the
Faculty of Engineering at Cairo University
in Partial Fulfillment of the
Requirements for the Degree of
MASTER OF SCIENCE
in
ELECTRONICS AND COMMUNICATIONS ENGINEERING

Under the Supervision of
Prof. Dr. Hanan Kamal
Professor of Control Systems Engineering
Faculty of Engineering, Cairo University

FACULTY OF ENGINEERING,
CAIRO UNIVERSITY
GIZA, EGYPT
2017

Engineer's Name: Mohammed Abdou Tolba
Date of Birth: 28/05/1991
Nationality: Egyptian
E-mail: Mohammed.Abdou@valeo.com
Phone: 01112238861
Address: 10st El Maahad El deni, Maadi, Egypt

Registration Date: 01/10/2013
Awarding Date: 2017
Degree: Master of Science
Department: Electronics and Communications Engineering

Supervisors:

Prof. Dr. Hanan Ahmed Kamal

Examiners:

Prof. Dr. Hanan Ahmed Kamal (Thesis Main Advisor)
Prof. Dr. Omar Ahmed Nasr (Internal Examiner)
Prof. Dr. Ahmed Mohamed El-garhy (External Examiner)
Professor of Control Systems, and the Dean of Faculty of Engineering Helwan University

Title of Thesis:

Autonomous Driving using Deep Reinforcement Learning

Key Words:
Autonomous Driving, Reinforcement Learning, TORCS Simulator, Q-Learning, Deep Deterministic Policy Gradient

Summary:

Autonomous Driving is one of the difficult problems faced the automotive applications. This is due to many corner cases which can be formulated in the unexpected behavior of the autonomous vehicles during the interaction with the other vehicles. The presented work used the Reinforcement Learning field, a strong Artificial Intelligence paradigm that teaches machines through the environment interaction and learning from their mistakes, in order to reach to having an Autonomous Driving vehicle. This work compared between two main categories: Discrete Action Algorithms like: Q-Learning, Double Q-Learning Algorithms, and Continuous Action Algorithms like: Deep Deterministic Policy Gradient (DDPG) Algorithm. It was proven as expected that Continuous Action Algorithms have better performance, so we applied some enhancements over the DDPG algorithm like solving the Long learning time which faced all machine learning problems. These enhancements were depending on disabling some restricted conditions and compensating them with the Reward term. The proposed work depends on Simulator called TORCS to reach to our aim.

Acknowledgments

All thanks and praises to **ALLAH** for providing me the strength and the time to complete this thesis beside my work and for guiding and helping me in this work.

I would like to express my gratitude to all who supported me in this work. In particular, I want to thank **Dr. Hanan Kamal** who gives me the guidance, supports me as usual, and puts her confidence in me.

I would like also to thank **Dr. Ahmad El-Sallab** very much who guides me with his ideas, supports me with his knowledge, helps me with his guide in performing the whole experiments, and helps me in choosing the thesis point.

I would like also to thank my team leader **Hussein Hesham** who assigned me for the research tasks that helps me in performing this thesis.

I would like also to thank my team leader **Ibrahim Sobh** who directed me to add some important details in this thesis.

Finally, I would like to express my thanks to **my Mother**, **my Father**, **my Brothers** and **my Fiancee**. They were supporting and encouraging me to achieve my goal.

Table of Contents

List of Tables

List of Figures

List of Abbreviations

AI	*Artificial Intelligence*
DDPG	*Deep Deterministic Policy Gradient*
GPS	*Global Positioning System*
GPU	*Graphics Processing Unit*
IOT	*Internet of Things*
MDP	*Markov Decision Process*
MCDM	*Multiple Criteria Decision Making*
NN	*Neural Network*
POMDP	*Partially Observable Markov Decision Process*
RL	*Reinforcement Learning*
SCR	*Simulated Car Racing*
TD	*Temporal Difference*
TORCS	*The Open Racing Car Simulator*
UDP	*User Datagram Protocol*
V2I	*Vehicle to Infrastructure*
V2V	*Vehicle to Vehicle*

Abstract

Autonomous Driving is one of the difficult problems faced the automotive applications. Nowadays, it is forbidden due to the presence of some restricted Laws that prevent cars from being autonomous for the fear of accidents occurrence. However, researchers try to reach autonomous driving as a new area for research for the aim of having a strong push against these restricted Laws. Autonomous Driving has many corner cases which can be formulated in the unexpected behavior for the autonomous cars during the interaction with other cars.

Reinforcement Learning is considered as a strong Artificial Intelligence paradigm which can teach machines through the environment interaction, and learning from their mistakes. Actually it proves its success when applied on Atari games, but it does not prove its success in the automotive applications.

The presented work will use the Reinforcement Learning field in order to reach to our goal of having Autonomous Driving Vehicle. Reinforcement Learning has two main categories: Discrete Action Algorithms and Continuous Action Algorithms. This work will compare between each of them in order to know which one is better to apply our enhancements on. Reinforcement Learning is one of the machine learning categories, so it has two main phases: Learning phase, and Testing phase. Sure, the applied work will not do training on a real vehicle, so it will use TORCS Simulator for that aim. This wok will achieve having autonomous driving using both of: Discrete Action Algorithms like: Q-Learning and Double Q-Learning Algorithms, and Continuous Action Algorithms like: Deep Deterministic Policy Gradient (DDPG) Algorithm. It was proven that DDPG have smoother and better performance than Q and Double Q-Learning Algorithms. Long Learning time is considered as one of the main problem that faced any machine Learning problem, so the proposed work will also decrease Learning time as much as possible by disabling some restricted conditions and compensating them with the Reward term.

Chapter 1 : Introduction

Nowadays, Autonomous Driving is an up-rising Research field. Although there are many restriction laws that prevent having fully autonomous vehicle, there is a great effort exerts in this field of research. As if this research proves stability and high performance for the vehicle during testing, there will be a great push against these laws. Driving is a multi-agent interaction problem. As a human driver, it is much easier to keep within a lane without any interaction with other cars than to change lanes in heavy traffic. The latter is more difficult because of the inherent uncertainty in behavior of other drivers. Driving a vehicle is a task that requires high level of skill, attention and experience from a human driver. Although computers are more capable of sustained attention and focus than humans, fully autonomous driving requires a level of intelligence.

There are two different approaches for the thinking of having Autonomous Driving. The First approach is the vehicle has various sensors installed to cover a 360^{o} angle around the vehicle as shown in Fig. (1.1) [1]. These sensors have the ability to provide the vehicle with the surrounded environment and the sufficient information like: the other vehicles on the same track, lane markings, traffic signs and lights, pedestrians, motor cycles…etc. There are many sensors can be used for the previous aims: Camera, Lasers and Radars. Camera sensor has the ability detect objects and classify them if they are: pedestrians, cyclists, cars, truck…etc, but the Camera doesn't have the accurate precision for calculating the distances away from the other objects. This allows us to think about different sensor that calculates the distances in a right way. Laser sensors have the ability to get the Long range distances away from the vehicle located in the range of the sensor, but it doesn't have the ability to classify the object like the Camera sensor. Radar sensors have the ability to get the short range distances away from the vehicle located in the range of the sensor.

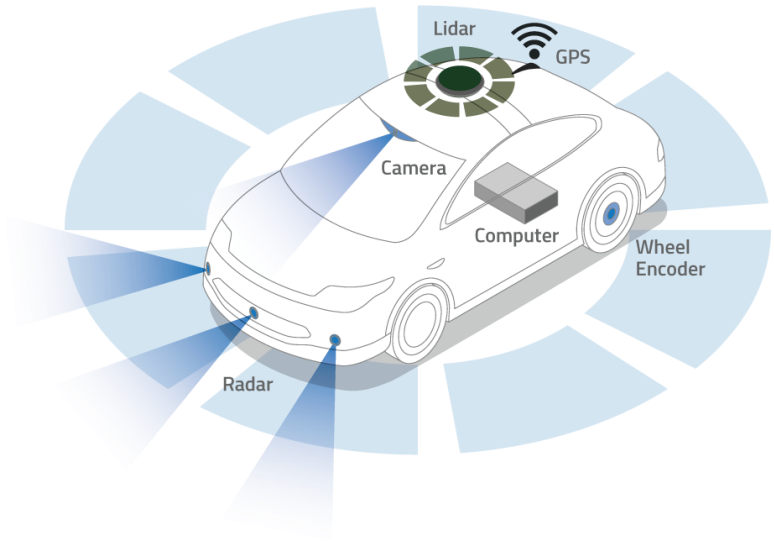

Fig. 1.1 Sensors covers 360^{o} angle around a vehicle

All of the previous sensor readings are fused together in order to have accurate, robust, and sufficient information for the surrounded environment. These sensor readings are used in great algorithms like: tracking, lane keep assist, traffic jam assist…etc. These great algorithms are designed for the aim of reaching to autonomous driving vehicle. Sensor Fusion is one of the most important phases as the proposed work will benefit from having various sensors which increases the certainty in sensor readings.

The second approach also will depend on having sensor readings, but the proposed work will use Machine Learning and its extension Deep Learning. Machine Learning has three different categories for learning: Supervised Learning, Unsupervised Learning, and Reinforcement Learning. This work will use the Reinforcement Learning way. The number of interacting vehicles, their geometric configuration and the behavior of the drivers could have large variability and it is challenging to design a Supervised Learning dataset with exhaustive coverage of all scenarios. Human drivers employ some sort of online Reinforcement Learning to understand the behavior of other drivers such as whether they are defensive or aggressive, experienced or in-experienced, etc. This is particularly useful in scenarios which need negotiation, namely entering a roundabout, navigating junctions without traffic lights, lane changes during heavy traffic, etc.

1.1. Problem Definition

The main challenge in autonomous driving is to deal with corner cases which are unexpected even for a human driver, like recovering from being lost in an unknown area without GPS or dealing with disaster situations like flooding or appearance of a sinkhole on the ground. The Reinforcement Learning paradigm models uncharted territory and learns from its own experience by taking actions. Additionally, Reinforcement Learning may be able to handle non-differentiable cost functions which can create challenges for Supervised Learning problems.

In all Machine Learning or Deep Learning problems, we have two phases: Learning phase, and Testing phase. During the Learning phase, the proposed work will sure not letting a real vehicle trains on a specific track because the vehicle in this phase do uncontrolled random actions for the aim of learning from its mistakes. This will damage the vehicle completely and we will not sure if it learns something or not. As a solution for that point, we will go to the Simulators path in order to prove of the concept firstly, and then we can put the learnt model on a real vehicle. During the testing phase, we can use another track for testing other than training track in order to prove the concept of having a generalized model for autonomous driving. This work chose the training track wisely for the aim of having a good model learnt how to move in a straight line, take right curves, and take left curves in a good way.

Learning Phase is the phase of building a solution model for the Machine Learning problem. Specifically the Learning phase for the aim of having autonomous driving vehicle takes very long time to build the reasonable model. This allows us to think about an enhancement for decreasing the Learning Phase.

1.2. Thesis Objectives

The proposed work aims to have an Autonomous Driving Vehicle using Reinforcement Learning field. We faced many problems because the Autonomous Driving problem is a very critical and difficult problem. This work depends on Machine Learning field especially Reinforcement Learning category that is based on learning from mistakes during the interaction with the environment.

We have many algorithms in the Reinforcement Learning field: a) Discrete Action, and b) Continuous Action. Our aim is to compare between the performances of both of them in order to know the better one based on the vehicle motion on track. This was the keyword for enhancing the performance of these algorithms.

Any Machine Learning Problem consists of two main phases: Learning phase, and testing phase. During the learning phase, this work let the vehicle moving on a specific track contains straight line, left curves and right curves in order to build a convenient model sufficient to succeed in driving on other new tracks. As usual, the Learning phase takes much long time; one of the objectives of this thesis is solving the long time taken to build the convenient model.

1.3. Organization of the thesis

This thesis consists of six chapters including the Introduction chapter. The following chapters are organized as follows:

Chapter 2 defines the problem of how we can reach to Autonomous driving, describes some literature review for previous work that tries to solve this problem, then gives an overview for Reinforcement Learning field and how we can map this field to be a Markov Decision Problem (MDP) problem. After that it will illustrate the basic solution for the MDP problem with some discretization.

Chapter 3 describes TORCS Simulator which allows us to apply our algorithms through based on the provided sensors. These sensors provide us with the sufficient information about the surrounded environment. After that, it illustrates how we can consider TORCS as an MDP through its states and actions with the help of the provided sensors.

Chapter 4 describes how we applied Reinforcement Learning on TORCS Simulator through describing the Game Loop in TORCS and the interface between the client and the server for taking actions from the states sent. After that, we decomposed our algorithms into two main categories: Discrete Action Algorithms, and Continuous Action Algorithms. Then, it will show some experimental results for applying both of these categories on TORCS and measure the performance of the vehicle while moving by each of these algorithms. Finally, it has a decision for working on Continuous Action Algorithm category in order to apply some enhancements over it.

Chapter 5 describes TORCS normal conditions that restrict the vehicle motion which also prevents the vehicle from exploring the Learning track. It will also have some enhancements done over the Continuous Action Algorithm especially DDPG which has the best performance comparing with other algorithms we have dealt with. It will show the effect of TORCS restriction conditions on the time needed to build the autonomous driving model through some experiments done on TORCS Simulator.

Chapter 6 illustrates the thesis conclusion and the future works in the field of Autonomous Driving.

Chapter 2 : Reinforcement Learning

In this part, we will deal with some Literature Review for the aim of having Autonomous Driving, and then we will describe the Reinforcement Learning framework and several classical techniques for solving the Reinforcement Learning Problem. Section 2.1 will describe what the Reinforcement Learning problem is, and section 2.2 will describe how we could solve this problem. Section 2.3 will describe how we could disceretize or approximate the values of the states in a continuous environment especially tile-coding. Section 2.4 will describe the mechanism of the action selection.

2.1. Autonomous Driving Literature Review

Autonomous Driving has been said to be the next big disruptive innovation for the next years. Considered as being predominantly technology driven, it is supposed to have massive societal impact in all kinds of fields. Having a closer look at the history of Autonomous Driving, as explained in the IEEE Spectrum (Ross, 2014) in Figure 2.1 it can be observed that the technological development and main milestones of the autonomous driving field started already a few decades ago. Leading to a vast analysis of some semi-autonomous features, development of present technologies and understanding on the future problematic while focusing in the near future in the connected car [2].

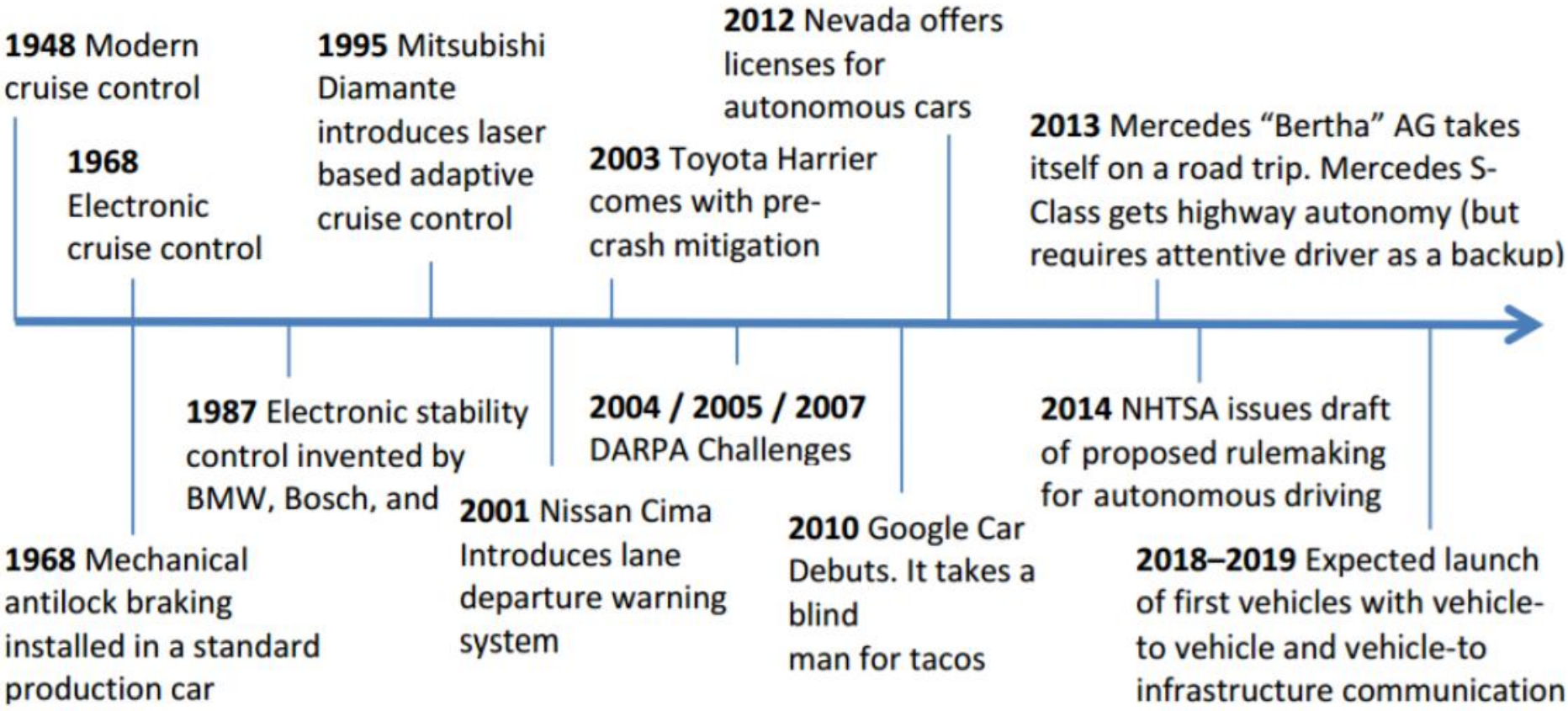

Fig. 2.1 Autonomous Driving historical review and Road Map

Some Researches present an autonomous driving research vehicle with minimal appearance modifications that is capable of a wide range of autonomous and intelligent behaviors. These include smooth and comfortable trajectory generation as: lane keeping and lane changing; intersection handling with or without Vehicle to Infrastructure (V2I), Vehicle to Vehicle (V2V) Communications, and pedestrian, bicyclist, and work zone detection. Safety and reliability features include a fault-tolerant computing system; smooth and intuitive autonomous-manual switching; and the ability to fully disengage and power down the drive-by-wire and computing system upon E-stop. The vehicle has been tested extensively on both a closed test field and public roads [3].

Other researches address the topic of real-time decision making for autonomous city vehicles, i.e., the autonomous vehicles' ability to make appropriate driving decisions in city road traffic situations. These researches explain the overall controls system architecture, the decision making task decomposition, and focuses on how Multiple Criteria Decision Making (MCDM). MCDM is used in the process of selecting the most appropriate driving maneuver from the set of feasible ones. Experimental tests show that MCDM is suitable for this new application area [4].

Traditionally, the vehicle has been the extension of the man's ambulatory system, docile to the driver's commands. Recent advances in communications, controls and embedded systems have changed this model, paving the way to the Intelligent Vehicle Grid. The car is now a formidable sensor platform, absorbing information from the environment (and from other cars) and feeding it to drivers and infrastructure to assist in safe navigation, pollution control and traffic management. The next step in this evolution is just around the corner: the Internet of Autonomous Vehicles. Pioneered by the Google car, the Internet of Vehicles will be a distributed transport fabric capable to make its own decisions about driving customers to their destinations. Like other important instantiations of the Internet of Things (e.g., the smart building), the Internet of Vehicles will have communications, storage, intelligence, and learning capabilities to anticipate the customers' intentions. The concept that will help transition to the Internet of Vehicles is the Vehicular Cloud, the equivalent of Internet cloud for vehicles, providing all the services required by the autonomous vehicles. In this article, we discuss the evolution from Intelligent Vehicle Grid to Autonomous, Internet-connected Vehicles, and Vehicular Cloud [5].

Other Researches present a motion planner for autonomous highway driving that adapts the state lattice framework pioneered for planetary rover navigation to the structured environment of public roadways. The main contribution of this research is a search space representation that allows the search algorithm to systematically and efficiently explore both spatial and temporal dimensions in real time. This allows the low-level trajectory planner to assume greater responsibility in planning to follow a leading vehicle, perform lane changes, and merge between other vehicles. This research show that algorithm can readily be accelerated on a Graphics Processing Unit (GPU), and demonstrate it on an autonomous passenger vehicle [6].

2.2. What is Reinforcement Learning?

Reinforcement Learning is derived from training animals in Biology especially from a famous experiment done by Skinner in 1932 in Biology [7] as shown in Fig. 2.2. Skinner trained a caged rat on pressing on a bar because of giving it a small food as a reward every time it pressed the bar. We can conclude from this experiment that we learned the rat to take an action based on the interacting with the environment which is represented in the cage and the bar.

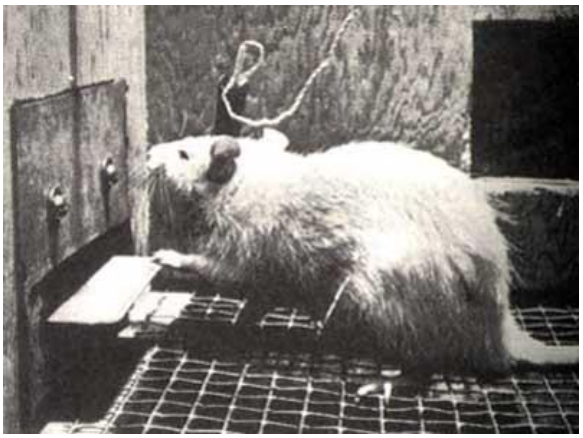

Fig. 2.2 Caged Rat

Thinking of the same idea but in computer programs (agents) as agents can be learnt to deal with the environment and taking actions and observing their rewards as shown in Fig. 2.3, but sure in this case the reward will not be food like the caged rat experiment, it will be a higher numerical value.

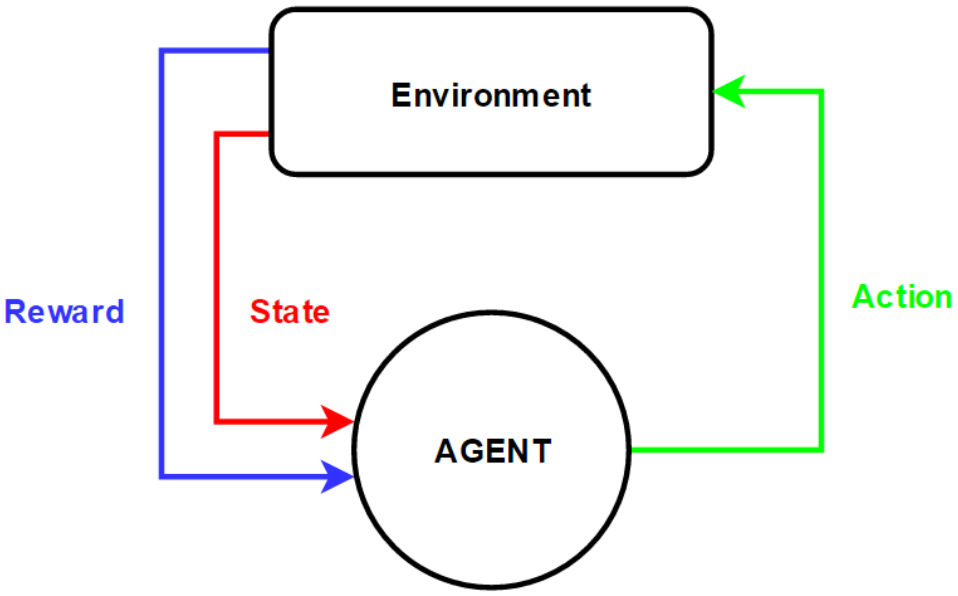

Fig. 2.3 Reinforcement Learning

There are two approaches while dealing with reward concept:

a- Accumulated Rewards: For every action done by the agent, there is a reward numerical value is given and it depends if this action will make the agent come nearer to the target, so the numerical value will be high.
b- Delayed Rewards: It is the traditional way of thinking as the agent is not given any reward numerical value if and only if it reaches to the target.

In Reinforcement Learning, there is no a teacher that tells the agent what is the best action that must be done. The agent must learn the effect of its actions by trying them and observe the reward. The agent tries to maximize the reward and this can be done by exploring the effect of all actions that can be done, then it could exploit the gained knowledge or experience in order to select the action that gives the higher rewards. This leads to the appearance of two main definitions: Exploration and Exploitation.

In short, Reinforcement Learning can be formulated by an agent try to map situations to actions, and then actions are chosen in order to maximize the numerical value of reward. This can be done by formulating the environment as a mathematical framework called Markov Decision Process (MDP).

2.2.1. Markov Decision Process (MDP)

It [9][15] is a mathematical framework for modeling decision making in situations that outcomes partly under random and partly under the control of a decision maker. It is most commonly used in optimization problems solved by dynamic programming and by Reinforcement Learning. MDP is a discrete time stochastic control process because it is modeled as discrete steps (t = 0, 1, 2 …).

MDP is formulated as a tuple consists of 5 main parameters (S, A, T(.,.), R(.,.), $\boldsymbol{\gamma}$) where:

S: Finite set of states,

A: Finite set of actions,

T(s,s'): Transition Probability from state (s) at discrete time (t) to state (s') at discrete time (t+1) when an action (a) is taken by the agent at discrete time (t) as in Fig. 2.4

$$Pr(st+1 = s' \mid st = s, at = a)$$

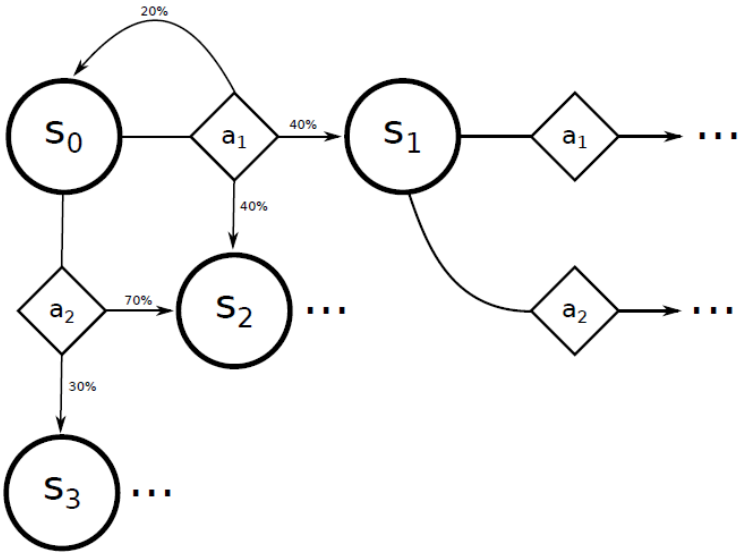

Fig. 2.4 MDP Example

R(s,a): The immediate reward received by the agent from the environment after the transition from (s) to (s') by taking an action (a)

γ: Discount factor identifies the difference between the importance between future rewards and the present reward where ($\gamma \in [0,1]$)

There must be a pre-processing phase on the raw information in order to reduce the number of states that the agent can move through them freely, as this leads to reduce a lot of computations, but this doesn't mean do a huge approximation that may lead to lose information, so it is a compromise problem.

Note that the transition probability (T(s,s')) depends on the current state (s) and the agent's action (a), this means that it is independent of all other previous states and actions. This property is called Markov's Property.

One of the most important assumption is that the environment is fully observable, this means that the state (s), action (a) and transition probability (T(s,s')) are the whole requirements in order to predict the next state (s'). Note that we have some environments are partially observable. These environment can be solved the generalization MDP which is called Partial Observable Markov decision Process (POMDP) [30], but it is not our concern.

2.3. Solving a Reinforcement Learning Problem

We have defined the Reinforcement Learning problem framework and now we will focus on how we can make a solution for the MDP. The solution for an MDP is called Policy and it is denoted by (π) which specifies an action for every state of the agent. Remember that we are aiming to maximize the accumulated rewards. The policy achieves that is called the optimal policy and it is denoted by (π*) that is defined the policy that achieve the highest expected return. The highest expected return for the accumulated Rewards can be formulated as follows:

$$R_{acc} = \sum_{t=0}^{\infty} R(s_t, a_t) \tag{2.1}$$

Where: R_{acc}: Accumulated Reward

In an infinite problem, the agent might never reach to a terminal state and this will lead to create an infinite sequence of states and so infinite sum of rewards, so we can avoid this infinite sum of rewards by using the discount factor (γ) which help us in discounting the Reward values over time as follows:

$$R'_{acc} = \sum_{t=0}^{\infty} \gamma^t * R(s_t, a_t) \tag{2.2}$$

Where: R'_{acc}: Discounted Accumulated Reward

This will ensure that for infinite number states, we will have a finite sum of Rewards values. For any environment, we can define the boundaries of the Rewards values, so let us define that the maximum Reward value be (R_{max}), so the maximal return of a sequence of states will be as follows:

$$\sum_{t=0}^{\infty} \gamma^t * R(s_t, a_t) \leq \sum_{t=0}^{\infty} \gamma^t * R_{max} = R_{max}/(1-\gamma) \tag{2.3}$$

Now we want to define the expected value of a particular policy, this will let us define the current state of the agent and know the reward that the environment will send it to the agent given that the action taken for that current state. It is also depends on the whole possible states that the agent can move to it for this specified current state. Bellman introduced the Value function which is a way to express the expected value of a state in terms of its reward and the future returns given a fixed policy (π). Bellman equation can be defined as follows:

$$V^{\pi}(s) = R(s, \pi(s)) + \gamma \sum_{s'} T(s'|s, \pi(s)) V^{\pi}(s') \tag{2.4}$$

In case of dealing with the optimal policy (π*), Bellman equation changes to Bellman Optimality equation that can be defined as follows:

$$V^*(s) = \max_a R(s,a) + \gamma \sum_{s'} T(s'|s,a)\, V^*(s') \tag{2.5}$$

If we ensure taking the optimal action every state, so we will ensure reaching to the optimal policy. To ensure that, we need to define a function that calculates the value of taking an action in a certain state. This function is called (Q-function) that can bed defines as follows:

$$Q^*(s,a) = R(s,a) + \gamma \sum_{s'} T(s'|s,a)\, V^*(s') \tag{2.6}$$

This equation differs from equation (2.5) in the fact of starting from any action instead of the optimal action (maximal action).

We can define the optimal policy as follows:

$$V^*(s) = \max_a Q^*(s,a) \tag{2.7}$$

Or

$$\pi^*(s) = \operatorname{argmax}_a Q^*(s,a) \tag{2.8}$$

2.3.1. Dynamic Programming

We stated before that MDP is most commonly used in the optimization problems that are solved by Dynamic Programming. It can be used to compute the optimal policy in the Reinforcement Learning problem. The most well-known Dynamic Programming algorithms are:

a- Value Iteration
b- Policy Iteration

2.3.1.1. Value Iteration

Bellman introduced the value function and the optimal value function as stated in equations (2.4) and (2.5). Suppose that we have (n) states, this means that we have (n) Bellman equations and so (n) Bellman Optimality Equations, therefore, we have (n) unknowns which are the states values. This means that we have (n) equations in (n) unknowns. Equation (2.5) contains max-operator and this means that the equations are non-linear; this means that we can't solve these equations by Linear Algebra.

Bellman introduced an algorithm called Value Iteration that provides an Iterative solution by starting with arbitrary values that are used to compute the right hand side of Equation (2.5) which is used to update the left hand side. Repeat the previous step till convergence or reaching the equilibrium. If we assumed the value of state (s) at the i-th iteration is $V_i(s)$, so the update which is called Bellman Update can be defined as follows:

$$V_{i+1}(s) \leftarrow \max_a R(s,a) + \gamma \sum_{s'} T(s'|s,a)\, V_i(s') \tag{2.9}$$

2.3.1.2. Policy iteration

Instead of using the Value Iteration to find the optimal value function, Policy Iteration manipulates the agent's policy directly. We can start with an arbitrary policy which is calculated by taking the expected value for equation (2.4). Once the value of each state in the current policy is known, the action corresponding to the highest expected return will be used because for sure it will be the optimal action and it will give the optimal policy if we repeat this step iteratively using the update equation as follows:

$$\pi'^{(s)} \leftarrow \operatorname{argmax}_a \left[R(s,a) + \gamma \sum_{s'} T(s'|s,a)\, V_\pi(s') \right] \tag{2.10}$$

2.3.2. Q-Learning

Dynamic Programming has many drawbacks especially in the high computational cost and the need of the Transition matrix that represent the transition from state to another based on the taken action; this means that we need a model for the environment which is considered as a special case. Q-Learning is a model-free algorithm due to its simplicity that comes from its cost, but it needs a lot of updates in order to reach to the optimal policy.

From its name, for sure, it comes from the Q-function in equation (2.7) that map states to actions by computing the values of actions in certain states instead of calculating state values. Q-Learning is derived from plugging equation (2.7) into equation (2.6), so we can have a Q-function independent on the state values as follows:

$$Q^*(s,a) = R(s,a) + \gamma \sum_{s'} T(s'|s,a) \max_a Q(s',a) \tag{2.11}$$

This leads to have the updated rule as follows:

$$Q(s,a) \leftarrow Q(s,a) + \alpha\,[R(s,a) + \gamma \max_a Q(s',a) - Q(s,a)] \qquad (2.12)$$

Where (α) is called the learning rate which identifies to how extend the new information overrides the old information.

It is obvious that this algorithm depends on the optimal future value that will ensure reaching to the optimal solution by an iterative method. It concerns with the current state, the next state and the action taken. Note that the maximal optimal future value is achieved according to the following scenario; suppose that the agent decided to take action (a) and the current state is (s), so the next state will be (s'). The maximal optimal future value is calculated when considering the current state is (s') then choosing the action whose Q value is the maximum for the whole possible actions we have in the policy.

Due to working on continuous environment and since the computer is a discrete system, so we are in need for some discretization in order to store the Q values. This allows us to search for a Function Approximator.

2.3.3. Double Q-Learning

Q-Learning Algorithm may suffer from many problems as the overestimation problem which appears in case of noisy environment, this problem comes from using only estimator, so the confidence level is very high and the uncertainty level tends to be zero. Double Q-learning Algorithm solves this problem by doubling the number of estimators.

Double Q-learning Algorithm [16] can be formulated as follows:

$$Q_A(s_t,a_t) \leftarrow Q_A(s_t,a_t) + \alpha\,(s_t,a_t).(R + \gamma \max_a Q_B(s_{t+1},a^*) - Q_A(s_t,a_t)) \qquad (2.13)$$

$$\text{where: } a^* = \operatorname{argmax}\{Q_A(S_{t+1},a)\}$$

$$Q_B(s_t,a_t) \leftarrow Q_B(s_t,a_t) + \alpha\,(s_t,a_t).(R + \gamma \max_b Q_A(s_{t+1},b^*) - Q_B(s_t,a_t)) \qquad (2.14)$$

$$\text{where: } b^* = \operatorname{argmax}\{Q_B(S_{t+1},a)\}$$

As we see that we have two Q-learning equations, this means that we will construct two Q-tables, but this is not the important part in the algorithm. The most important part in this algorithm is that we have TWO estimators because we have terms for the maximum optimal future; this will ensure solving overestimation problem. The confidence level is not high because one estimator update reflects on the other estimator's Q-table and vice versa. Let's see the following Scenario: we want to update Q-table (A), so the maximum optimal future part comes from taking the maximum Q-value when entering with the same state that maximizes action taken by the agent.

At the end of this algorithm, we will have two converged Q-tables because of the inverted updating happens between the two equations.

2.4. Function Approximation

Q-Learning algorithm is based on discrete states and a Look-up table called Q-Table. This table contains the Q-values for the whole possible actions for each state. Suppose [8] that the agent has (a) number of actions and the environment has (n) dimensions, each dimension has (m) values, then the total number of entries in the constructed Q-Table would be $(n^m * a)$ entries which means that the constructed Q-table may be huge. Learning from a huge Q-table would take a very long time because each entry must be visited many times before the Q-values become accurate. This means that the Q-table itself is infeasible representation for problems with continuous environment.

It is not necessary to represent the Q-Learning as a Q-Table. It could be approximated by a function approximator either Linear approximation or non-Linear approximation.

In case of dealing with linear approximation, the Q-function could be approximated as follows:

$$Q_t(s,a) = \theta_t^T \emptyset(s,a) \tag{2.15}$$

Where (θ_t^T) is the parameter vector at time (t), and $(\emptyset(s,a))$ is the feature vector of state (s) and action (a). Gradient Descent is the most commonly used for this approximated Q-function and our main objective function is to minimize the mean squared error for the observed examples (Training data) by adjusting the parameters (θ_t). The update rule for the Q-function can be defined as follows:

$$\theta_{t+1} = \theta_t + \alpha\, \delta_t\, \nabla_{\theta_t} Q(s_t,a) \tag{2.16}$$

Where (δ_t) is considered as the temporal difference error that is defined as follows:

$$\delta_t = r_{t+1} + \gamma \max_a Q_t(s_{t+1},a) - Q(s_t,a) \tag{2.17}$$

Due to equation (2.15), the update rule for Q-function can be written as follows:

$$\theta_{t+1} = \theta_t + \alpha \delta_t \emptyset(s, a) \quad (2.18)$$

In case of dealing with non-Linear approximation, gradient descent could be more complicated, so the most commonly used type for non-linear approximation is the Artificial Neural Networks, but it is not out concern here. Let's see the comparison between Linear and non-Linear approximation in the following table:

Table 2.1 Linear vs. Non-Linear Approximator

Linear Approximation	Non-Linear Approximation
The agent can learn quickly	The agent takes longer time to learn
It is easily to be implemented	It is difficult to be implemented, due to higher complexity
It can be analyzed easily	It is difficult to be analyzed
It is easily to be understand	It is difficult to be understand
It is less robust to the features choice	It is more robust to the features choice

Pyeatt [10] compared between the performance of the Q-table, neural network, and some forms of adaptive decision trees as function approximators for Q-Learning especially in (Car Racing). It is concluded that the Q-table was not feasible representation as a function approximator because of the huge Q-table generated which is big enough to learn nothing within defined time. In case of using neural networks, it is failed to converge in a defined time. However, in case of using decision trees with a mechanism based on number of tests, it has the best performance.

Although neural networks prove a great success while dealing with Reinforcement Learning, it doesn't prove that success in the (Car Racing), so now we will deal with the advantages and disadvantages of neural networks especially in that case in the following table:

Table 2.2 Advantages and Disadvantages of Neural Networks in Car Racing

Advantages of neural Networks in (Car Racing)	**Disadvantages of neural Networks in (Car Racing)**
It can easily generalize towards new states	It assumes stationary target function, but targets in Q-learning are continually changing during learning
It can handle easily continuous inputs	It doesn't perform local updates, but it changes in the whole network during each update. This may lead to the update in a state affects on the update of another state, This makes the network becomes over trained (over-fitting problem)
It solves larger problems than Q-table	It may lie on a local minimum, so it is not guaranteed to converge in these kinds of problem

The most commonly used in Linear and non-Linear approximation and at the same time proves success in (Car Racing) is Tile Coding which is an easy approach to be implemented and applied achieving suitable approximation for the change in the value function.

2.4.1. Tile Coding

It is considered as a Linear function approximation that has the ability to choose $(\emptyset(s, a))$ in equation (2.15). It can approximate state values or approximate Q-values. Unlike previous ways, every action has a value for each state, so it is considered as the simplest way for approximating value function.

Linear function approximation with tile coding [8] is the easiest way to have linear combination for many overlapped Q-tables with high resolution nearer to the state space. More specifically, state dimensions are partitioned into tiles (features) which are considered as fields, and a portioning of each dimension is called tiling. If the input value lies within the tile boundaries, so this tile is considered as active tile; otherwise it is considered as inactive tile. Active tiles take value equal to (1), while the inactive tiles take value equal to (0) in order to have simple computations.

Tile coding generalizes the learned states because of the following:

a- States that results in having the same active tiles are considered as the same state.
b- Each update affect on all states that result in having active tiles

Tiling may be considered as a uniform grid map with many dimensions as shown in Fig. (2.5):

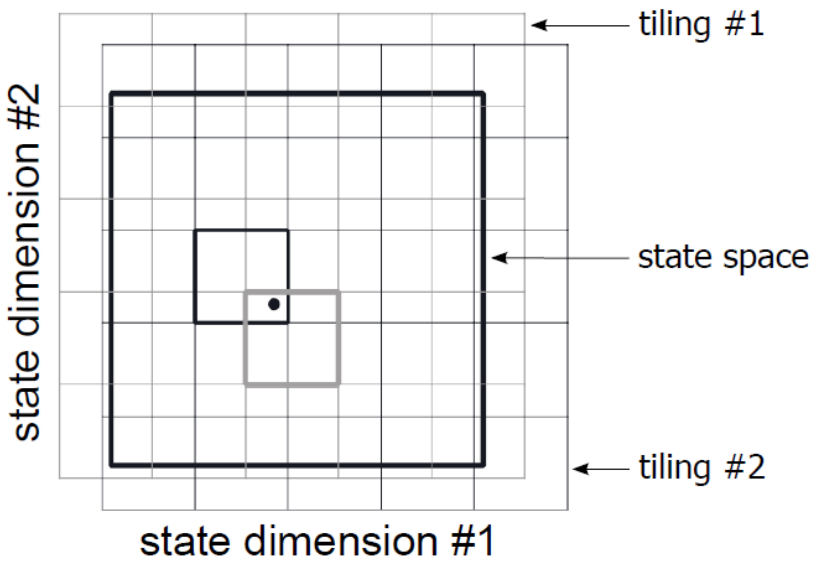

Fig. 2.5 Value functions approximation with tile coding Action Selection

In the previous figure, there are two dimensions, continuous problem. The dot marks the actual state. However, note that all dots within the intersection of the two active tiles will be considered as the same state.

In this part, we refer to the exploration and the exploitation terms in which the agent uses trial and error method to take an action and then explore the effect of it by learning. The big aim of that is to learn to take the action that maximizes the estimated return. This means that we have two main options: the first option exploits current knowledge to optimize short-term return and the second one can be said to explore the values of the other actions as it helps to increase the performance of the agent and prevent from the over fitting.

There must be a balance between exploitation and exploration in order to achieve the advantages of both of them which can be done by using the most popular mechanism which is ϵ-greedy. This mechanism enforces the agent to choose a random action with a small probability (ϵ) and this action is independent on the other actions. This algorithm or mechanism can be formulated by:

$$\pi(s_t) = \begin{cases} \text{Random action } a \in A(s_t) & \text{if } (\rho < \epsilon) \\ \text{argmax}_a\, Q(s_t, a) & \text{otherwise} \end{cases} \qquad (2.19)$$

Where (ρ) is a random number in which ($0 < \rho < 1$) and it is chosen every time we choose an action. There is a tuning parameter which is (ϵ), this is why the mechanism is called (ϵ-greedy) [34].
In order to achieve a higher performance, we proposed another mechanism depending on (ϵ-greedy) too, but we want to make the agent visits more unknown action, so the percentage of the exploration increases.

The new mechanism can be formulated as follows:

$$\pi(s_t) = \begin{cases} \text{informed action } a \in A(s_t) & \text{if } (\rho < \eta) \\ \text{Random action } a \in A(s_t) & \text{if } (\rho < \eta + \epsilon) \\ \text{argmax}_a\, Q(s_t, a) & \text{otherwise} \end{cases} \quad (2.20)$$

Where (η) has the same aim as (ϵ) but taking an informed action. This mechanism ensures that the degree of freedom to choose an action is increased due to the added choice which is the informed action. Informed action is based on a simple heuristic that exploits domain knowledge. For example, in the car racing example, we can make the car moves in the middle of the road with a constant speed, it is just make the car survive from being out of the road.

2.5. Conclusion

Reinforcement Learning can be modeled as an MDP problem which can be solved using Dynamic programming in addition to some basic algorithms like: Q-Learning and Double Q-Learning. It is only the problem of the ability to create: states and actions in addition to define the Reward function. All of the basic algorithms to solve the Reinforcement Learning or the MDP problem need some discretization. In the following chapter, TORCS Simulator will be explained in details.

Chapter 3 : TORCS

In this part, we will deal with TORCS game simulator [11] and how we can apply Reinforcement Learning algorithms on it. Section 3.1 will describe a historic explanation for using Artificial Intelligence in games. Section 3.2 will describe an overview on TORCS and how we can adapt it to be compatible to have a racing car. Section 3.3 will describe how we can adapt TORCS to have an MDP. Section 3.4 will describe how we can implement reinforcement learning based on MDP.

3.1. Artificial Intelligence in games

Game-playing was an area of research in Artificial Intelligence [32] from its inception. Since Arthur Samuel wrote a checkers program for one of the first computers (IBM 701) in 1952, he also introduced a learning algorithm in the optimization field whose aim was the high correlation to win in checkers in 1959. This program actually was based on a heuristic look-ahead search based on the minimax-algorithm introduced by Shannon in 1950 to determine the best available move. This was one of the first applications in Reinforcement learning and it was one of the great achievements in machine learning [12].

Over the years, computer games proved to be an ideal way to test the Al algorithms [35]. Games are much simpler than problems in reality, but complex enough to be a challenge. Since Samuel's algorithm, this field has grown rapidly. There are many games (e.g. checkers and backgammon) [13] are considered as turn-based, deterministic and each agent can perceive the complete state of the game at all the times. This made a strong platform or base for the symbolic, search-based AI algorithms which are considered as the main interest at that time. One of the great achievements in this field is that the computer program is playing chess, as shown in Fig. 3.1(a), with a human, and the big surprise is that the computer program wins that game. Another great achievement in this field is that a computer program is playing backgammon, as shown in Fig. 3.1(b), and has the ability to make some techniques like human; this is done by training a simple neural network to play this game.

Fig. 3.1(a) Chess

Fig. 3.1 (b) Backgammon

After that video games industry has grown dramatically, so many researches turned to apply AI algorithms on video games. This step was very interesting because of the huge facilities provided by video games like the presence of many opponents.

One of the types of AI algorithms that combine well with games is Reinforcement Learning. The concept of RL shortly can be formulated as there is an agent interfaces with an environment, this agent learns to perform a behavior depending on actions taken given the states plus the feedback reward sent from the environment to the agent that describes if the agent is near or far from its target. RL is currently used in many applications especially in video games like: racing games, fighting games, real time strategy games, and first-person shooters. Our main focus will be how we can use RL in order to teach an agent to achieve and reach to a certain goal in the game.

3.2. TORCS

As we know that our aim is to use RL to reach to a High-way Driving Vehicle, and due the presence of the learning phase in the RL, we need a Car Simulator. This is why we go to TORCS [11] which stands for The Open Racing Car Simulator. It is considered as a stand-alone application in which all opponents have a race through, and it is considered as a game engine for a car that is designed as a racing game as shown in Fig. 3.2. The advantage of choosing TORCS [29] is that it is an open source that we can control it easily. One of the most important questions is what we will base on while taking actions. Due to the open source Game engine, there is a plug-in called Simulated Car Racing (SCR plug-in) enables and facilitates that as we will see.

Fig. 3.2 TORCS Screenshot

3.2.1. SCR plug-in

There is an SCR [33] championship held during several conferences such as the IEEE World congress on Computational Intelligence, IEEE Congress on Evolutionary Computing [31] and the Symposium on Computational Intelligence in Games. It is a highly customizable plug-in added on TORCS that provides a sophisticated physics engine, 3D Graphics, various game modes, several diverse tracks and car models.

SCR plug-in ensures that all cars in TORCS have the access to all information in either the environment or the tracks, but actually it is not a normal situation in real life. Therefore, the software interface of TORCS-SCR can be formulated as shown in the architecture in Fig. 3.3. The architecture is composed of two main parts: 1) Server and 2) Client.

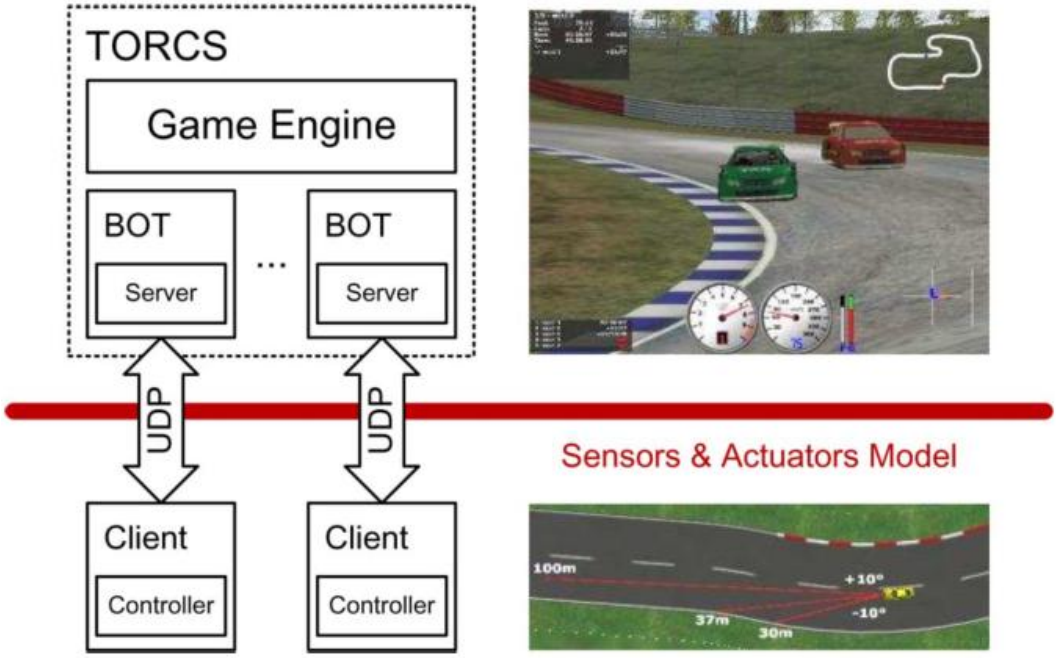

Fig. 3.3 TORCS-SCR Architecture

The Server is TORCS with SCR which provides the client with all the sufficient information about both of the environment and the track. These information are considered as the sensor readings for the sensors provided by SCR as shown in Fig. 3.4

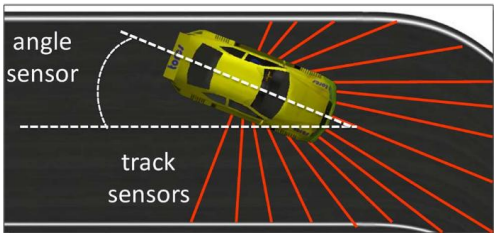

Fig. 3.4(a) Angle and Track Sensors

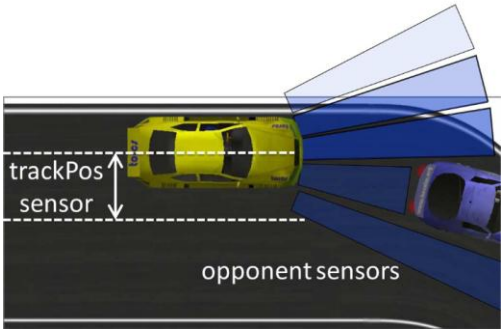

Fig. 3.4(b) TrackPos and opponent Sensors

The main sensors that we depend on are: 1) angle sensor, 2) track position, 3) trackPos sensor, and 4) opponents' sensors, actually we will deal with many other sensors, but these ones are the key to control the motion of the car. The Client is considered as the controller that we are able to design in order to achieve our aim of controlling the motion of the car. The client takes the sensor readings and work on them in order to take the action based on these readings. The bidirectional way of communication between the server and the client is a UDP connection in which the sensor readings are sent from the server to the client and the taken action by the client is sent to the server. This way of communication is very intelligent because it separates the controller from the environment. This is why we can call the environment as The Agent (Remember: Reinforcement Learning Agent).

3.2.2. TORCS-SCR Sensors

The following table (3.1) will describe the built-in provided sensors by TORCS:

Table 3.1 TORCS Sensors

Name	Description
Angle	It is the angle between the car direction and the track axis.
CurLapTime	It is the current Lap time
DistFromStartLine	It is the distance covered starting from the ready position
Gear	Current Gear: -1: Reverse 0: neutral 1-6: Various Gear Values
LastLapTime	It is time taken to complete the last lap
Opponents	It is considered as array of 36 sensors with a 10o range for each one and it detects the distance from the car with the opponents
Trackpos	It is distance between the car and the track axis: 0: The car is moving on the axis -1: The car is moving on the left side of the track +1: The car is moving on the right side of the track The values which are less than (-1) or greater than (+1) means that the car is outside the track
SpeedX	It is the Speed in the longitudinal axis of the car
SpeedY	It is the Speed in the traversal axis of the car
Track	It is considered as array of 19 sensors, with 10o range, that installed in the front of the car and it represents the distance from the car to the track edge.

3.2.3. TORCS-SCR Control Actions

The following table (3.2) will describe the control actions sent by controller/Agent to the server:

Table 3.2 TORCS Control Actions

Name	Description
Accel	It is a virtual Gas Pedal: 0: no gas 1: full gas
brake	It is a virtual Brake Pedal: 0: no brake 1: full brake
Gear	It is the Gear value
Steering angle	It is the steering angle value: -1: full left +1: full right The value of the steering angle is corresponding to 45o
Meta	It is the meta-command: 0: Do nothing 1: Ask the server to restart the race

The following part will describe how the proposed work mapped MDP components to be compatible with TORCS and then how it applied the model-free Reinforcement Learning algorithm to replace the Transition model due to the infinite number of transitions while controlling the autonomous car.

3.3. TORCS as MDP

As we know that MDP consists of the following components or concepts: states, actions, reward model, and Transition model.

3.3.1. States

No one can deny that the sensor readings came from the Server are very useful to control the car, but actually it contains a lot of unnecessary information. The proposed work will concern with the most important sensor readings as shown in the following Table (3.3):

Table 3.3 TORCS Main States

Sensor Model	State Dimension
SpeedX	It is considered as the Longitudinal Speed in the car axis
TrackPos	It is considered as the Latitudal Deviation away from the track
Angle	It is considered as the angle with respect to the track axis
Track [5,(6;7;8),9,(10;11; 12),13]	It is considered as the distance of the [-40^o, -20^o, 0^o, 20^o, 40^o] sensors. The (-40^o) sensor distance comes directly from Track[5] The (0^o) sensor distance comes directly from Track[9] The (40^o) sensor distance comes directly from Track[13] But The (-20^o) sensor distance comes from the average of: Track[6], Track[7], and Track[8] The (20^o) sensor distance comes from the average of: Track[10], Track[11], and Track[12]

Note that all of these state dimensions are considered as continuous, and this may lead to have infinite number of the states, so we used Tile-coding function approximator as we stated it before that.

3.3.2. Actions

The proposed work has five action dimensions available in TORCS which will be described in the following table which are: 1) accelerate, 2) Brake, 3) Gear, 4) Steer, and 5) Meta. Since brake is the opposite of accelerate, so they will be considered as one parameter and Meta is considered as the Restart in case of failure. The following table (3.4) will show TORCS actions:

Table 3.4 TORCS Main Actions

Action Model	Action Dimension
Accel	It is a virtual Gas Pedal: 0: no gas 1: full gas
brake	It is a virtual Brake Pedal: 0: no brake 1: full brake
Gear	It is the Gear value which can be controlled based on the RPM of the car.
Steer	It is the steering angle value: -1: full left +1: full right The value of the steering angle is corresponding to 45^o
Meta	It is the meta-command: 0: Do nothing 1: Ask the server to restart the race

The presented work can control Gear based on the current RPM of the car, so we must have some disceretized boundaries for changing the Gear box.

The presented work also have a discretized steering angle values as we will see in the following table, but actually steering must be compatible with both accelerate and brake, so they are strongly related with each other. Initially, we had [-1, -0.5, -0.1, 0, 0.1, 0.5, 1] values for steering angle, but in order to avoid sharp edges steering curves, so we removed both of (-1) and (+1). In addition to these values, we may have accelerate or brake with can be modeled as (+1 or -1), so we have the following combinations.

Table 3.5 TORCS Designed Actions for Steering, Acceleration, and Brake

Steer	Accelerate	Neutral	Brake
Extreme Left (0.5)	0	1	2
Left (0.1)	3	4	5
Straight (0)	6	7	8
Right (-0.1)	9	10	11
Extreme Right (-0.5)	12	13	14

3.3.3. Rewards

As we know that our aim is to control the car to drive autonomously, so this means that this work has some good actions and some bad actions due to the random choice for the actions taken. Autonomous Driving is a well-defined target, but we must prevent the car from: 1) going out of the road, 2) stopping in a stuck position, and 3) doing sharp curves while moving in straight road.

All of the previous concepts allow us to define the reward model as in the following table:

Table 3.6 TORCS Reward Function with Scenarios

Scenario	Reward
The car is moving fast in the middle of the road	The reward will be strongly related with the distance covered between the current state and the previous state. We will use an accumulated reward function if the distance covered is long and the car survive between the road boundaries, so it will take a positive and high reward. We also defined the reward model to have a continuous values of range [-1,1]; where the negative reward means undesired action and the more positive, the more we desire that action.
The car goes out of the track	It is a forbidden scenario as the controller sends the META action in order to re-do this episode another time. We put the highest negative reward value (-1) as a penalty at the end of the episode.
The car is in a stuck position (Actually it is calculated when the traveled distance < 0.01m)	It is considered as an undesirable behaviour and unwanted blocking state. This is an exception reward value, so we gave this scenario (-2) reward value to prevent doing this another time.

3.4. Conclusion

TORCS is considered as a basic framework for applying Reinforcement Learning algorithms. TORCS Simulator has many sensors which provide us with the sufficient information about the surrounded environment. It also provides us with the car model, physics model, and different tracks which can be used for training and testing. It helps us to define both states and actions which are essential for solving the MDP problem. In the following chapter, we will deal with how we will apply Reinforcement Learning algorithms on TORCS Simulator.

Chapter 4 Reinforcement Learning on TORCS

In this chapter, we will deal with applying Reinforcement Learning algorithms on TORCS. The proposed work will mainly concern with two main algorithms; the first one is discrete action algorithm, and the second one is continuous action algorithm. As an example for the discrete action algorithm, this work will concern with: Q-Learning and Double Q-Learning Algorithms. In addition to that, this work will concern with Deep Deterministic Policy Gradient algorithm as an example for the continuous action algorithm. After that, the proposed work will show some measurement results for the success of these algorithms in the field of having Autonomous vehicle.

4.1. Reinforcement Learning Implementation

As a formulation for all the previous information about TORCS-SCR, MDP and Reinforcement Learning; Reinforcement Learning has two main concepts which are the agent and the environment; where the agent tries to take actions based on the environment feedback or reward. TORCS-SCR; a game engine with a plug-in; has two main parts the server and the client; where the client receive sensor readings and takes the reasonable action based on these readings. The proposed work will show the following strong relationship as we can have a direct mapping as follows: the environment in Reinforcement Learning (RL) is mapped to be the server in TORCS-SCR that sends the sensor readings, and the agent in the RL is mapped to be the client which is the controller. Fig. 4.1 describes the merging or the mapping between the RL and TORCS-SCR; it also shows that the controller is far away from the environment which is the most powerful point that allows anyone to design his controller and interface with the environment easily.

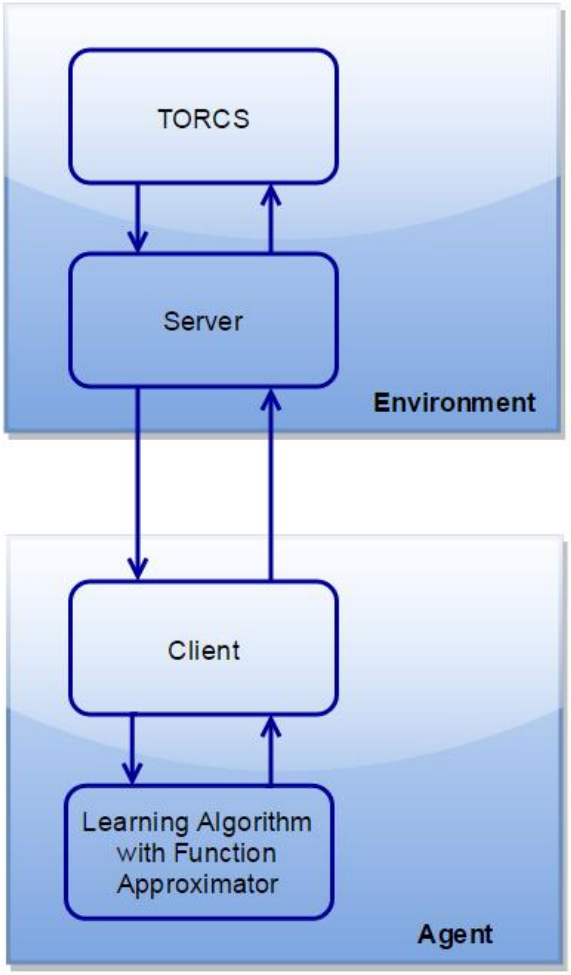

Fig. 4.1 TORCS-SCR as RL

The following section will describe the interaction Loop between the client and the server as in Fig. (4.1)

4.2. The Game Loop

Server sends the sensor readings to the client every game tick (1 Game tick = 20 ms). These sensor readings are discretized based on the function approximator (Tile Coding), then the client takes the action that maximized the maximal optimal return for the reward function. The client calculates update either the look-up table, then it sends the action taken to the server again and so on. Actually this is called One Episode.

We may face some problems like the car goes away from the road, so when the sensor readings sent to the client for this situation. The client immediately send the Meta action to start a new episode, but it is not only that action as the reward model compensate that by giving a highly negative reward value that affects on the update by decreasing its value. Another problem like the car stuck or moving at a lower velocity, due to the survival of the car within the road boundaries and at the same time it is

considered as an undesired behaviour, this work will compensate that with an exceptional negative reward value in addition to start.

As we know that we can solve the Reinforcement Learning using many algorithms. Now, this work will concern with the two main algorithms: a) Discrete Action algorithms: These algorithms, like Q-Learning and Double Q-Learning, send discrete actions to TORCS, b) Continuous Action algorithms: These algorithms, like Deep Deterministic Policy Gradient (DDPG), send continuous actions to TORCS.

4.3. Discrete Action Algorithms

Actually we will deal with two main algorithms for the Discrete Action Algorithms which simply solve the MDP problem: Q-Learning Algorithm and Double Q-Learning Algorithm.

4.3.1. Q-Learning

It is an algorithm that solves the reinforcement learning problem which is modeled as an MDP. The equation constraint is as follows:

$$Q(s,a) \leftarrow Q(s,a) + \alpha\,[R(s,a) + \gamma\, max_a\, Q(s',a) - Q(s,a)] \qquad (4.1)$$

This means that we construct only One Q-table, this table is the mapping for the states and actions Q-Values. These Values are updated depending on the reward value came from the environment for the current state and action. The most important value is the optimal future Q-value that maximizes the return value of the reward which ensures taking the optimal or best action and achieves the optimal policy for the problem by an iterative method across the episodes applied.

During the Game tick, we know that current state, so the remaining is choosing the applied action based on that state (Remember: we have dealt with how we can choose the action).

Initially, the Q-table is constructed as zero values for the discretized number of states and number of actions. If we know the state and the action, we will know which Q-value needed to be updated depending on the previous equation.

Applying this algorithm iteratively will lead to construct the Q-table which can be taken as an input and follow it in the testing phase for the autonomous car. Actually, learning the car on one track will not be enough for the Q-table to be compatible for other tracks, so the testing phase must be done on a sufficient number of tracks.

4.3.2. Double Q-Learning

After testing the Q-Learning algorithm, we found that the car suffers from non-smooth actions; this allows us to think of another algorithm achieves the convergence like the Q-Learning and achieves smooth actions. Double Q-Learning depends on constructing two Q-Tables where the maximal optimal future term for one table is come from the other one. This will reduce the dependency on one estimator, so the equations constraints are as follows:

$$Q_A(s_t, a_t) \leftarrow Q_A(s_t, a_t) + \alpha(s_t, a_t).(R + \gamma \max_a Q_B(s_{t+1}, a^*) - Q_A(s_t, a_t)) \quad (4.2)$$
$$where: a^* = argmax\{Q_A(S_{t+1}, a)\}$$

$$Q_B(s_t, a_t) \leftarrow Q_B(s_t, a_t) + \alpha(s_t, a_t).(R + \gamma \max_b Q_A(s_{t+1}, b^*) - Q_B(s_t, a_t)) \quad (4.3)$$
$$where: b^* = argmax\{Q_B(S_{t+1}, a)\}$$

The criteria to apply Double Q-learning is exactly the same as Q-Learning, but the main differences are: 1) We construct two Q-tables, and 2) The chosen Q-table to be updated is updated with the other Q-table in the maximal optimal future term.

Applying this algorithm iteratively will lead to construct two Q-tables where anyone can be taken as an input and follow it in the testing phase for the autonomous car. Actually, learning the car on one track will not be enough for the Q-table to be compatible for other tracks, so the testing phase must be done on a sufficient number of tracks.

4.3.3. Measurement Results for Discrete Action Algorithms

Now we will deal with some measurement results noticed while applying both of Q-learning and Double Q-Learning algorithms on TORCS

Table 4.1 Some Measurement Results for Q and Double Q-Learning

Measurement Result	Q-Learning	Double Q-Learning
Number of Episodes to complete One Lap	1200 episodes	1400 episodes
Number of Completed Laps in Training Phase	2 Laps	10 Laps
Number of Episodes needed to build the Model	1500 episodes	1800 episodes
Maximum Speed for the Car	160 km/h	148 km/h
Time to complete One Lap	01:24:09	01:15:22

Q-learning Algorithm suffers from taking discrete actions which affects on the car motion. This is very obvious from the Q-Learning path in Fig. (4.2).

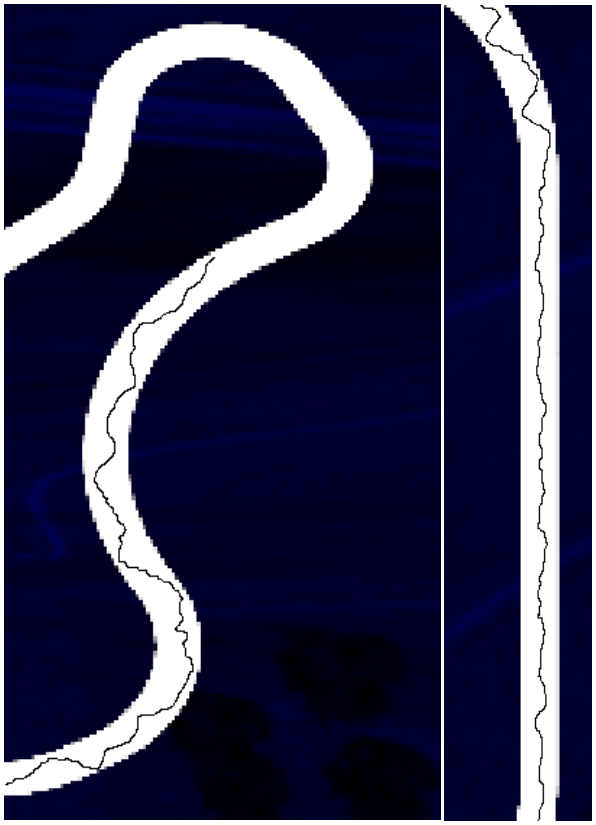

Fig. 4.2 Q-Learning Algorithm Performance in Straight and curved parts

Actually, we chose an accurate testing track that contains both: straight and curved parts. As a reporting way for the performance of the Q-Learning Algorithm in the straight lines, Right curves, and Left curves, we have some screenshots for each of them as shown in Fig. (4.3).

Previously, we thought that the Double Q-Learning algorithm will achieve smoother actions more than the Q-Learning algorithm up to having very smooth actions. Unfortunately, the Double Q-Learning Algorithm enhances the control actions a little bit but with decreasing the maximum allowable speed for the car, actually this point has an advantage and disadvantage. The advantage of the previous point is to have smooth actions especially the curvature part of the track, while the disadvantage is represented in the aim of racing of this algorithm versus other opponents.

Fig. 4.3(a) Q-Learning Algorithm Performance in Straight line part

Fig. 4.3(b) Q-Learning Algorithm Performance in Left Curve part

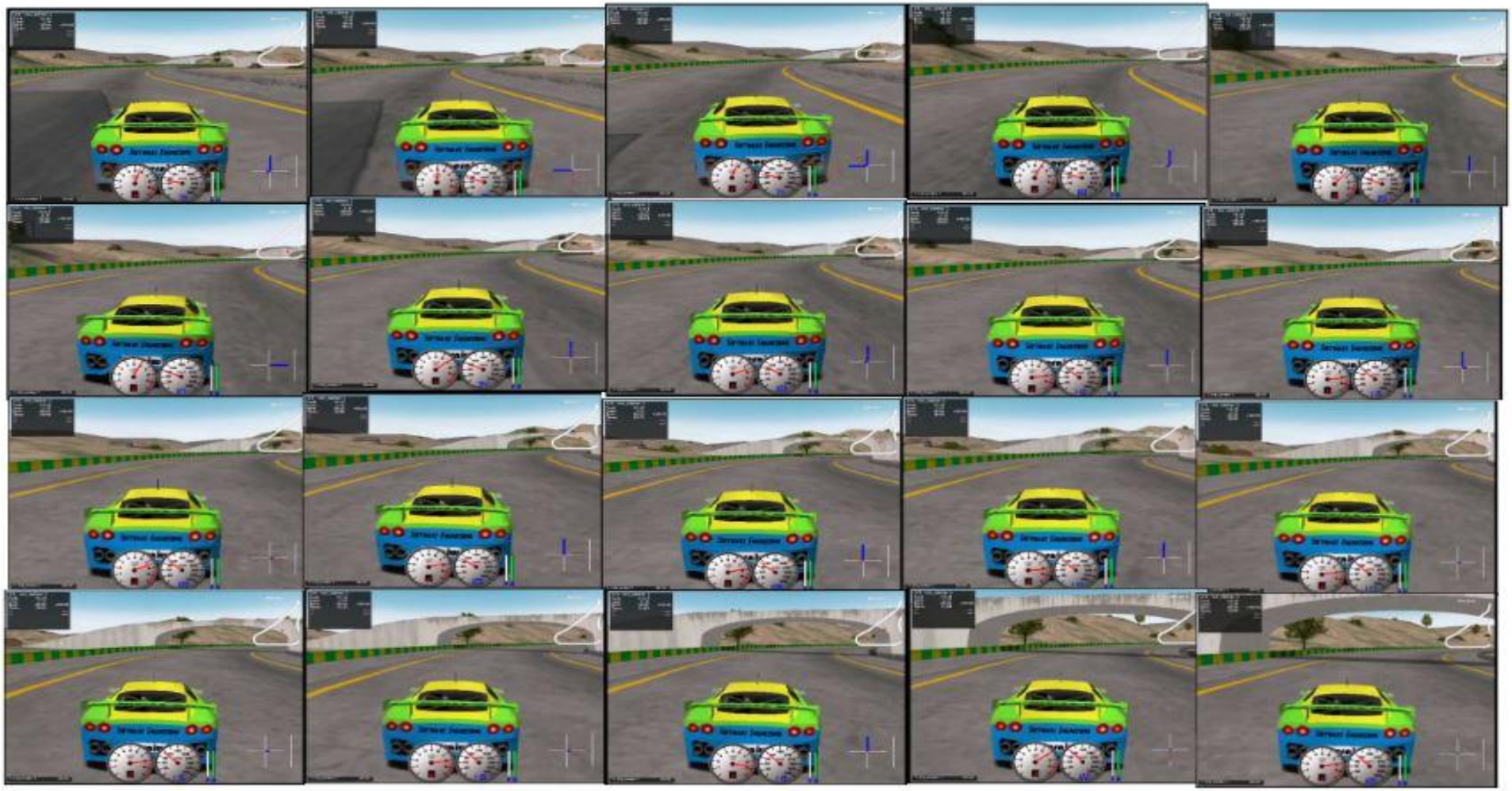

Fig. 4.3(c) Q-Learning Algorithm Performance in Right Curve part

Double Q-Learning Algorithm suffers from more computations than Q-Learning Algorithm, but as we can notice from Table (4.1) that Double Q-Learning Algorithm covered more laps than the Q-Learning. This means that Double Q-Learning Algorithms gives higher stability for the created model than the Q-Learning Algorithm especially the Q-Learning algorithm fail from achieving or covering more number of Laps than 2 Laps while the Double Q-Learning covered more than 10 Laps.

4.4. Continuous Action Algorithms

Actually the proposed work will deal with one of the main algorithms for the Continuous Action Algorithms which solves the MDP problem: Deep Deterministic Policy Gradient Algorithm (DDPG), but firstly we want to deal with the Actor Critic Algorithm because the DDPG Algorithm is considered as the extension of the Actor Critic Algorithm as we will see.

4.4.1. Actor Critic Algorithm

Actor Critic algorithm [18][27] is essentially a hybrid method that combines both: policy function with value function method together as in Fig. 4.4.

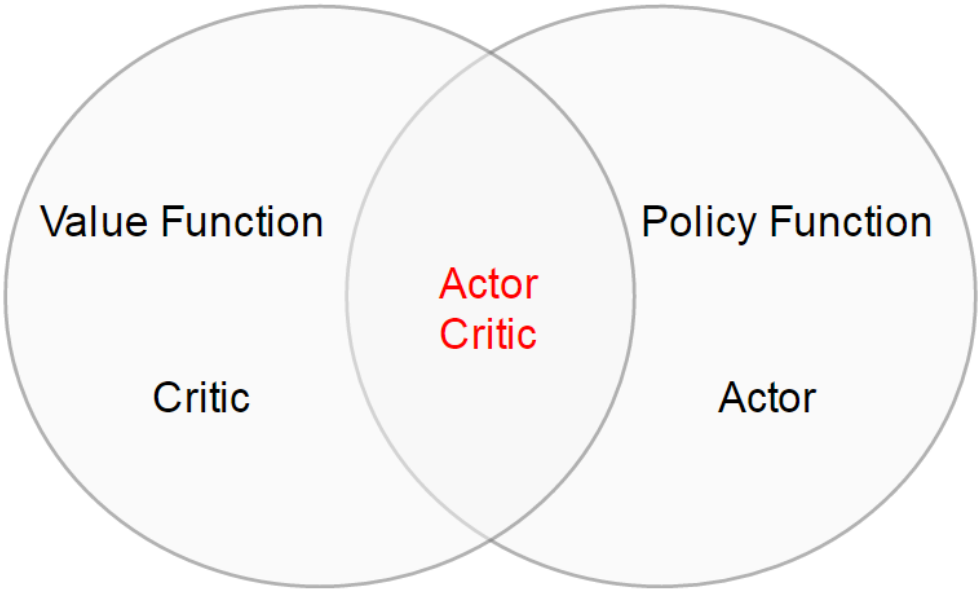

Fig. 4.4 Actor Critic Combination

This algorithm consists of two main parts: Actor part and Critic part, where the policy function is known as the Actor part while value function is known as the Critic part. The Actor part is responsible for taking actions given the current state of the environment, while the Critic part is responsible for producing signals or values to criticize the actions taken by the Actor part as in Fig.4.5.

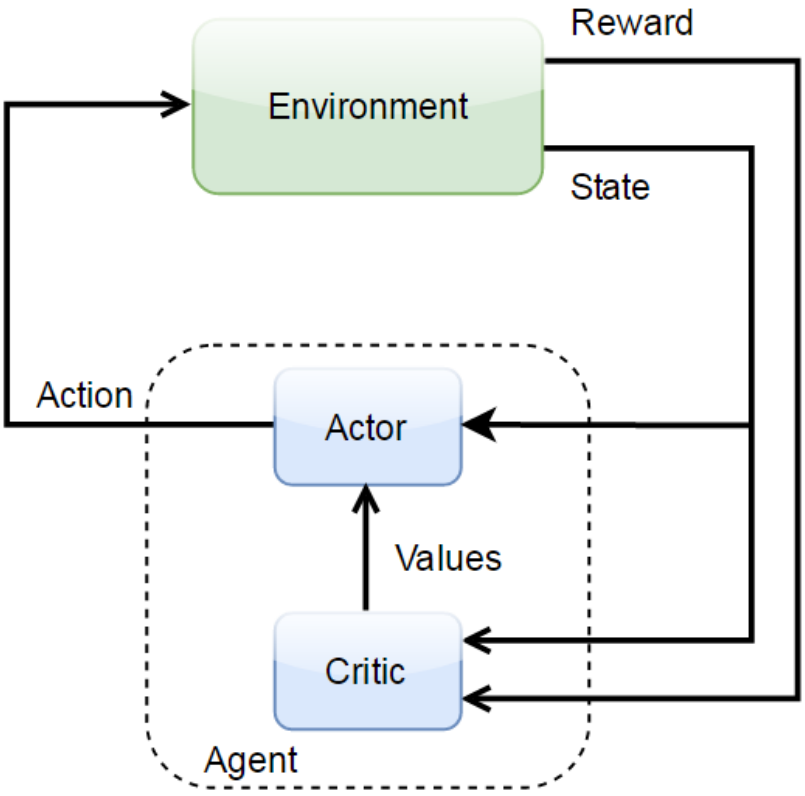

Fig. 4.5 Actor Critic replaces one part of the Agent

This algorithm is very close to our nature, as for example: junior employee does natural work can be represented as the Actor part, and his boss criticizes his work, hopefully he can do better next time, can be represented as the Critic part.

4.4.1.1. Critic Part

It uses a state-action value function not an explicit function for the policy. For continuous state and action spaces, this will be an approximate state-action value function. These methods learn the optimal value function by finding online an approximate solution to the Bellman equations:

$$V^*(s) = \max_a R(s,a) + \gamma \sum_{s'} T(s'|s,a)\, V^*(s') \tag{4.4}$$

$$Q^*(s,a) = R(s,a) + \gamma \sum_{s'} T(s'|s,a)\, V^*(s') \tag{4.5}$$

A deterministic policy is calculated by using an optimization procedure over the value function:

$$\pi^*(s) = \operatorname{argmax}_a Q^*(s,a) \tag{4.6}$$

4.4.1.2. Actor Part

Policy gradient methods are actor-only and do not use any form of a stored value function. Instead, the majority of actor-only algorithms works with a parameterized family of policies and optimizes the cost functions directly over the parameter space of the policy. A major advantage of actor-only methods over critic-only methods is that they allow the policy to generate actions in the complete continuous action space.

A policy gradient method is generally obtained by parameterizing the policy $\boldsymbol{\pi}$ by the parameter vector $\boldsymbol{\theta}$. The parameterized policy can be defined as $\boldsymbol{\pi_\theta}$. By assuming that the parameterized policy is differentiable with respect to $\boldsymbol{\theta}$, so the cost function can be calculated using the Chain rule as follows:

$$\nabla_\theta J = \frac{\partial J}{\partial \pi_\theta} \frac{\partial \pi_\theta}{\partial \theta} \tag{4.7}$$

Then, by using standard optimization techniques, we can reach to an optimal solution for the Cost function. The gradient $\boldsymbol{\nabla_\theta J}$ is estimated per time step and the parameters are then updated in the direction of this gradient. For example, a simple gradient ascent method would yield the policy gradient update equation:

$$\theta_{k+1} = \theta_k + \alpha \nabla_\theta J_k \tag{4.8}$$

4.4.1.3. Merging Actor and Critic parts

Actor-critic methods aim to combine the advantages of actor-only and critic-only methods. Actor-critic methods are capable of producing continuous actions, while the large variance in the policy gradients of actor-only methods is countered by adding a critic. The role of the critic is to evaluate the current policy prescribed by the actor. In principle, this evaluation can be done by any policy evaluation method commonly used, such as TD (Temporal Difference) [28]. The critic approximates and updates the value function using samples. The value function is then used to update the actor's policy parameters in the direction of performance improvement. These methods usually preserve the desirable convergence properties of policy gradient methods, in contrast to critic-only methods.

4.4.2. Deep Deterministic Policy Gradient (DDPG) Algorithm

This Algorithm [17] is considered as the natural extension for the Actor Critic Algorithm by replacing the Actor and Critic parts with Actor and Critic Networks or models as in Fig. 4.6.

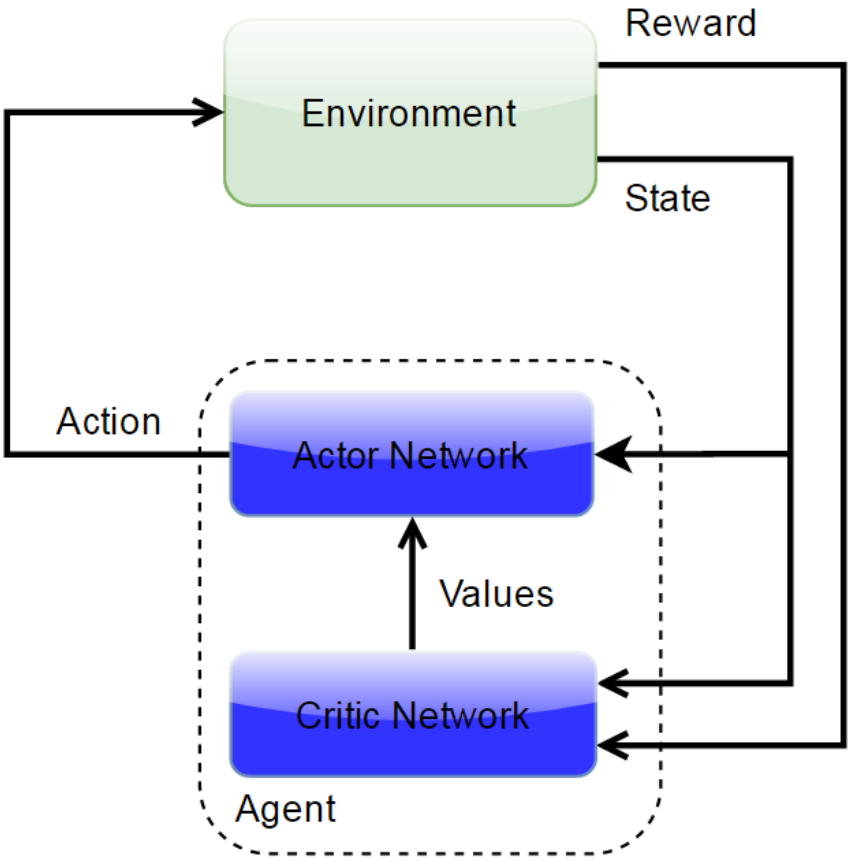

Fig. 4.6 DDPG Algorithm

In TORCS, we used Policy Gradient method as the Actor model, and Q-Learning Algorithm with some modifications as the Critic model. Now, we will describe the basic structure of the DDPG Model down to up starting from defining the policy network till having deep networks for both actor and critic parts.

4.4.2.1. Policy Network

The aim of this network is to take the states of the game (car velocity, distance between the car and the track axis...etc) and decide the right action (steer left or right, accelerate, decelerate...etc). It is called Policy-Based Reinforcement Learning because we will deal with parameterized policy as follows:

$$\pi_\theta(s, a) = P[a|s, \theta] \tag{4.9}$$

Where s: the state, a: the action, and θ: the parameters of the policy networks.

4.4.2.2. Deterministic

It means that for a given state, we have a particular action should be taken based on the learnt policy a = μ(s).

4.4.2.3. Policy Objective Function

As we know that the policy aims to do the mapping from the state to the taken action. For example, suppose that a car is trying to learn how to make a left turn. At the beginning, the car may simply not steer the wheel and hit the curb, then it will receive a negative reward, so the network will update the parameters θ such that in the next time will try to avoid hitting the curb. After many attempts, it will learn how to do that correctly.

Let's define the total discount future reward

$$R = r_0 + \gamma r_1 + \gamma^2 r_2 + \cdots \gamma^n r_n \tag{4.10}$$

Due to the policy objective Function, we need to define our objective as the total expectation of the total discount reward

$$L(\theta) = E[r_0 + \gamma r_1 + \gamma^2 r_2 + \cdots |\pi_\theta(s, a)] \tag{4.11}$$

which can be written as follows:

$$L(\theta) = E_{x \sim p(x|\theta)}[R] \tag{4.12}$$

where the expectations of the total reward R is calculated under the probability distribution P[x, θ] parameterized by some θ.

For the continuous case for the Q-function, we can use:

$$Q(s_t, a_t) = R_{t+1} \tag{4.13}$$

therefore, we can write the Gradient of a deterministic policy a = μ(s) as follows:

$$\frac{\partial L(\theta)}{\partial \theta} = E_{x \sim p(x|\theta)}[\frac{\delta Q}{\delta \theta}] \tag{4.14}$$

Now, we can apply chain rule:

$$\frac{\partial L(\theta)}{\partial \theta} = E_{x\sim p(x|\theta)}[\frac{\delta Q^{\theta}(s,a)}{\delta a}\frac{\partial a}{\partial \theta}] \quad (4.15)$$

4.4.2.4. Deep

Instead of applying the normal Q-Learning algorithm, we will use Deep Q-Network in which we can replace the Q-Function by a neural network $Q^{\pi}(s,a) \approx Q(s,a,w)$, where w is the weights of the neural Network.

Therefore, we have the Deep Deterministic Policy Gradient Formula which is used in the Back-propagation of the neural network to optimize the cost function:

$$\frac{\partial L(\theta)}{\partial \theta} = \frac{\delta Q(s,a,w)}{\delta a}\frac{\partial a}{\partial \theta} \quad (4.16)$$

where the policy parameters θ can be updated via stochastic gradient ascent optimizer.

Loss Function of the neural network could be calculated as follows:

$$Loss = [r + \gamma Q(s', a') - Q(s,a)]^2 \quad (4.17)$$

4.4.3. Measurement Results for Continuous Action Algorithm

Now we will deal with some measurement results noticed while applying DDPG algorithm on TORCS

Table 4.2 Some Measurement Results for DDPG Algorithm

Measurement Result	**DDPG Algorithm**
Number of Episodes to complete One Lap	800
Number of Completed Laps in Training Phase	10
Number of Episodes needed to build the Model	1000
Maximum Speed for the Car	165 km/h
Time to complete One Lap	53:01

DDPG Algorithm is characterized by continuous actions which affects in a better way on the car motion. This is very obvious from the DDPG path in Fig. (4.7).

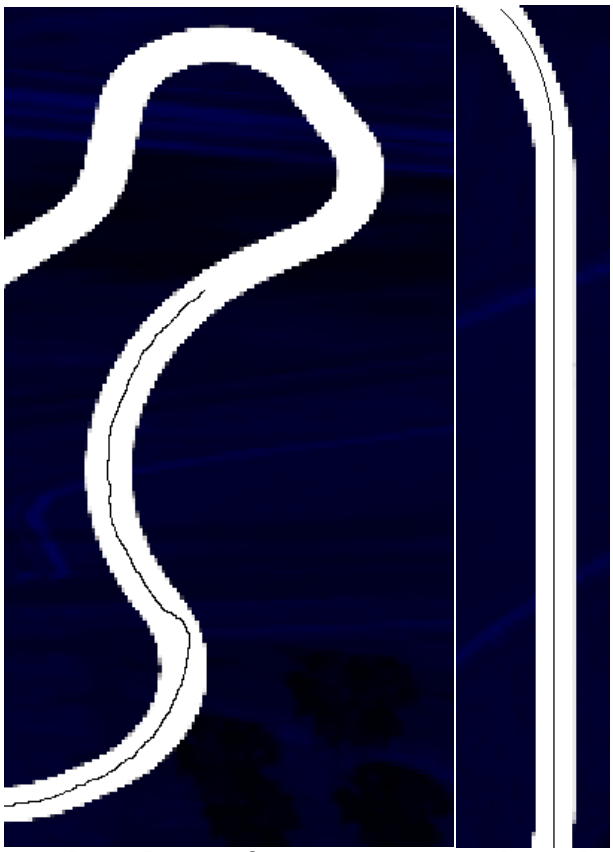

Fig. 4.7 DDPG Algorithm Performance in Straight and curved parts

Actually, we chose an accurate testing track that contains both: straight and curved parts. As a reporting way for the performance of the Q-Learning Algorithm in the straight lines, Right curves, and Left curves, we have some screenshots for each of them as shown in Fig. (4.8).

DDPG Algorithm is characterized by having continuous actions, so we achieved smooth actions and performance. The number of episodes needed to complete One Lap is less than Discrete actions algorithms which means that the DDPG algorithm learns faster how to complete one lap on that same track. During the Training phase, DDPG algorithm covers many laps in a smooth way, and actually this performance is extended also to be found the testing phase on different tracks. As a result of learning in shorter time, it is obvious that the number of episodes to build the model will be low. In addition to that, the Maximum speed of the car is relatively high especially when going in a straight line or taking curves.

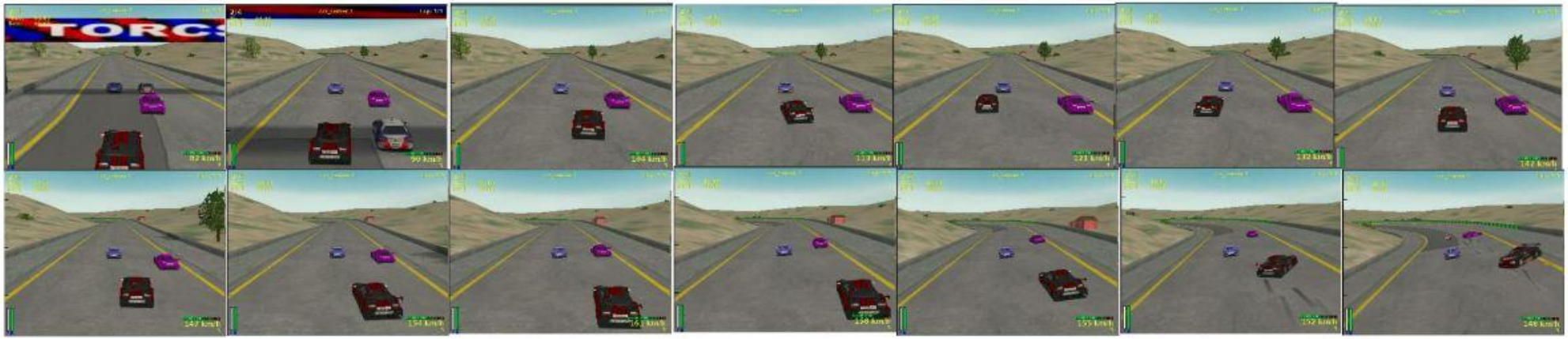

Fig. 4.8(a) DDPG Algorithm Performance (Blue Car) in Straight line part

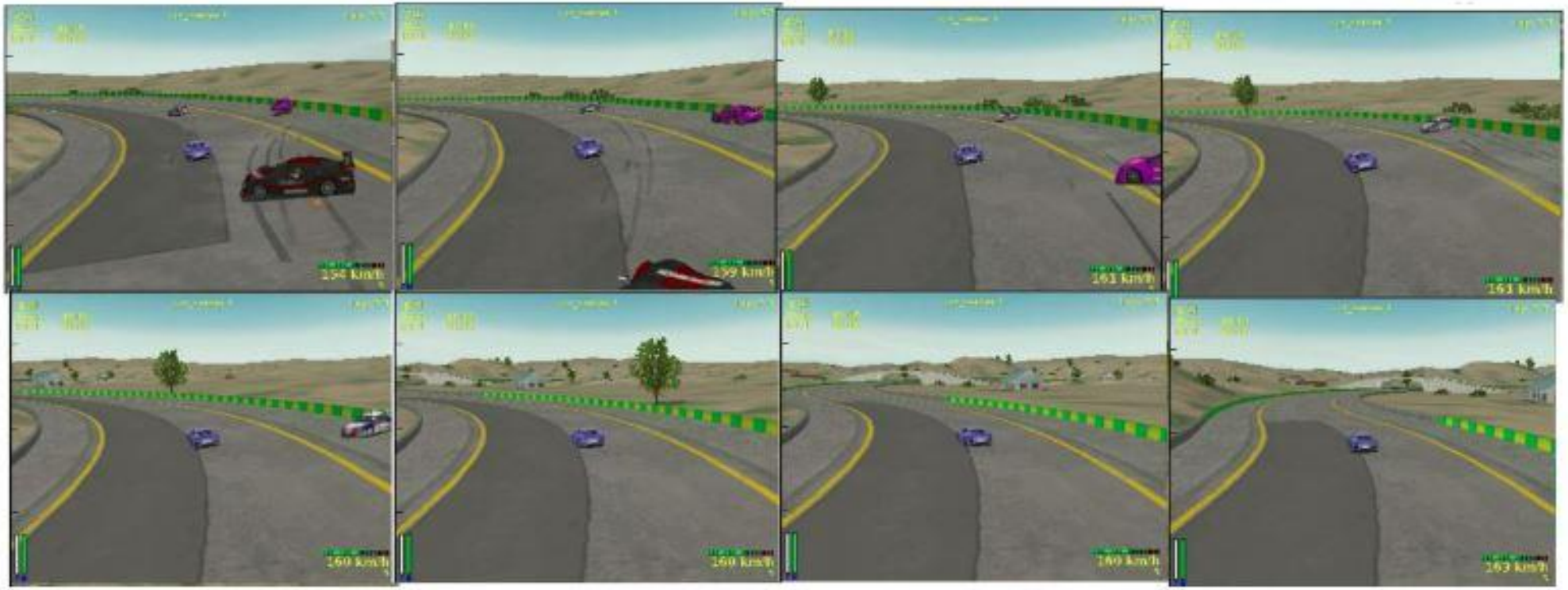

Fig. 4.8(b) DDPG Algorithm Performance (Blue Car) in Left curve part

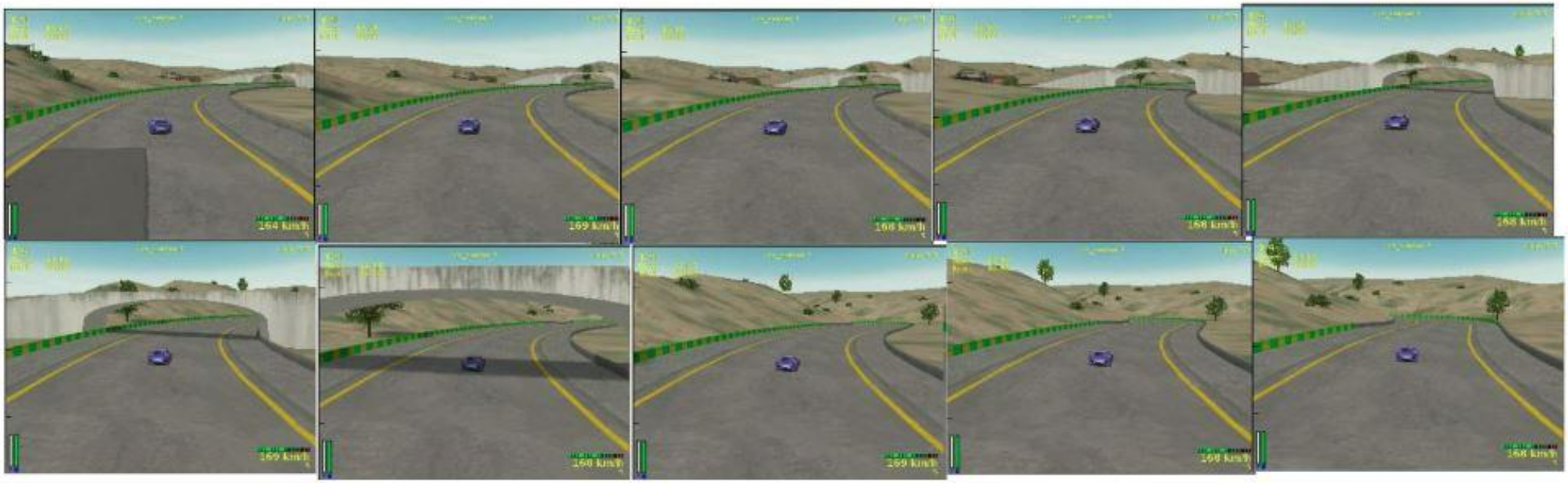

Fig. 4.8(c) DDPG Algorithm Performance (Blue Car) in Right curve part

4.5. Discrete vs. Continuous Actions Performance

In this part, we will formulate and compare results of Discrete and Continuous Actions performance when we applied them on TORCS. Actually, we will deal with many points to illustrate which Algorithm could be used for the aim of reaching Autonomous Driving. From tables (4.1) and (4.2), we can conclude that Continuous Actions Algorithms have better performance then Discrete Actions Algorithms: a) Number of episodes to complete One Lap is less in the Continuous Actions Algorithms which means that the vehicle has the ability to explore and learn faster, b) Number of Completed Laps in Training Phase is large in the Continuous Actions Algorithms which means that the vehicle will have the ability to build good model, c) Number of Episodes needed to build the target model are less in the Continuous Actions Algorithms which means that it is better so as to have less time in training phase, and d) Maximum Speed for the vehicle is high. In addition to the two tables of our experimental results, Fig. (4.9) illustrates the incremental of the number of episodes vs. number of Laps covered.

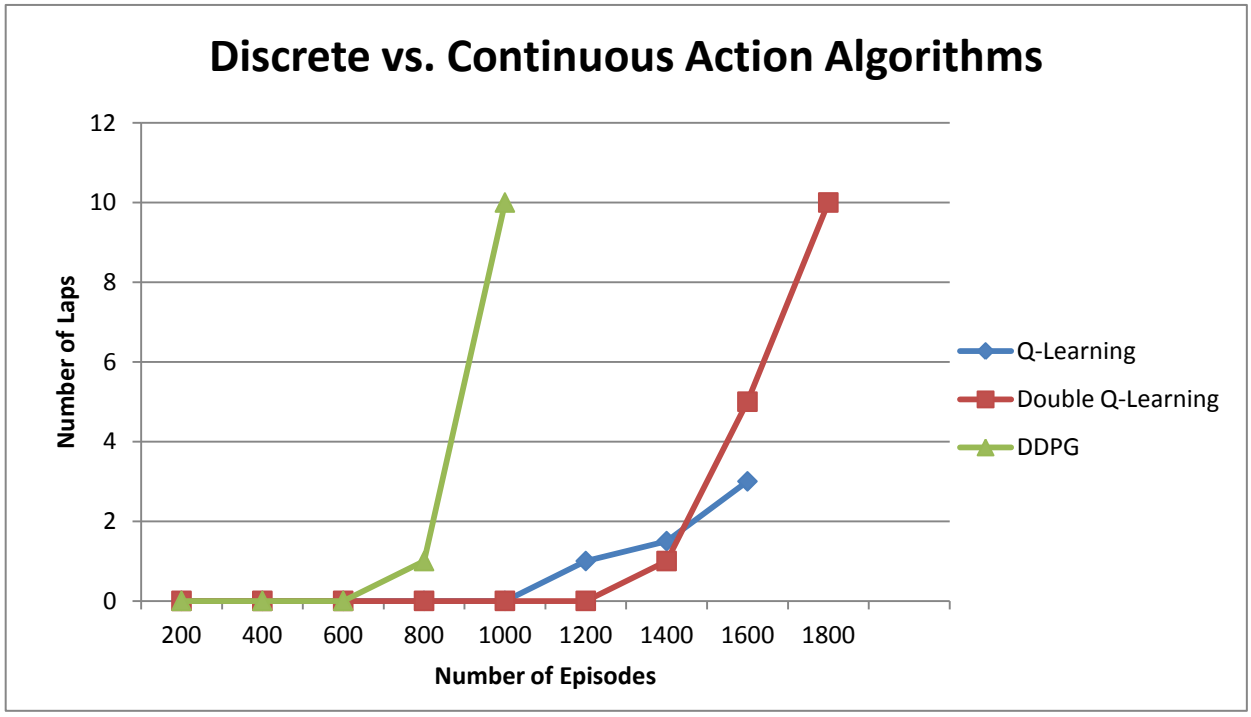

Fig. 4.9 Number of Laps vs. Number of Episodes for Discrete and Continuous Actions Algorithms

From Fig. (4.2) and (4.7), it was noticed that the Continuous Actions Algorithms have better performance on Straight lines, Right curve, and Left curve parts of the track. Fig. 4.10 combines Q-Learning, Double Q-Learning, and DDPG algorithms.

Table 4.3: Q-Learning, Double Q-Learning, and DDPG Experimental Results

Measurement Result	**Q-Learning Algorithm**	**Double Q-Learning Algorithm**	**DDPG Algorithm**
Number of Episodes to complete One Lap	**1200**	1400	**800**
Number of Completed Laps in Training Phase	2	**10**	**10**
Number of Episodes needed to build the Model	**1500**	1800	**1000**
Maximum Speed for the Car	**160 km/h**	148 km/h	**165 km/h**
Time to complete One Lap	01:24:09	**01:15:22**	**59:23**

From the previous table (4.3), we can calculate some enhancement percentages specialized for the Continuous Action Algorithms in green rather than Discrete Action Algorithms in red:

1- Number of Episodes to complete One Lap is decreased by more than 33.3% which is a great enhancement for the car to explore the training track as fast as possible.

2- Number of Completed Laps in Training phase is the same.

3- Number of episodes needed to build the model is decreased by more than 33.3% which is a great enhancement for the simulated car to build the sufficient model in around 66.6% from the previous learning time.

4- Maximum Speed for the Car is increased by more than 3.1% from the previous Car Speed which is a good enhancement for the aim of having a Racing Car.

5- Time to complete One Lap is decreased by more than 21.2% which is a great enhancement for the aim of having a highly stable vehicle on the road in addition to have a Racing car.

Note: The previous calculations are done based on the best and optimistic case and for the Discrete Action Algorithms.

We can conclude from the previous that in case of applying any enhancements, it will be applied on the Continuous Actions Algorithms because they are already better than Discrete Actions Algorithms. As a result, we will apply our enhancements to decrease the learning phase time on Continuous Actions Algorithms especially on DDPG Algorithm. These enhancements will deal with some restricted conditions applied on TORCS, so we will try to remove these limitations to achieve the best results.

4.6. Conclusion

Reinforcement Learning can be applied on TORCS Simulator. We have two main categories from the algorithms: Discrete Action Algorithms, and Continuous Action Algorithms. We have applied both of these categories and measured some experimental results in order to know which category is reasonable for applying some enhancements. We have compared between them with the help of the performance of the vehicle and we found that the Continuous Action Algorithms such as DDPG have more stable and better performance than the Discrete Action Algorithms such as Q-Learning and Double Q-Learning algorithms. This means that in the following chapter we will use the Continuous Action Algorithms as a basic category then we will do the enhancement over it.

Chapter 5 Enhanced approach for Continuous Action Algorithms

In this chapter DDPG algorithm will be used which proves its success in having better performance than Q-Learning and Double Q-Learning Algorithms. TORCS normal conditions applied will be illustrated while we are running our algorithms, and then defining why they are considered as restricted conditions. After that the effect of these restricted conditions will be figured out through the model Convergence time. Finally, some solutions will be proposed for these conditions leading to have better results and performance.

5.1. TORCS Normal Conditions

As a recall for our main aim, the goal was reaching to autonomous driving. It is expected that: a) the vehicle will not move out of the track and b) the vehicle will not get stuck or stopped while it is moving on the track. In order to avoid the occurrence of these situations, TORCS normal conditions restricted these situations. As during the Learning phase, when the vehicle is exposed to: get out of the track, or get stuck on the track, TORCS is totally terminated and the vehicle takes a negative reward then starts a new episode on the same track.

We have applied both of Discrete and Continuous Actions Algorithms on TORCS with the same restricted conditions. From the results in the previous chapter, we decided to work on the Continuous Actions Algorithms due to having: a) higher performance, b) less time to build the model, c) smooth actions in both straight line and curves parts of the track, and d) less time to complete one lap on the same track applied overall the other algorithms. All of the previous allows us to enhance only in the Continuous Actions Algorithms.

5.2. TORCS Restricted Condition

As we mentioned before that TORCS has some restricted conditions. These conditions are named by this name because when one of these conditions is satisfied, so TORCS terminate the episode, then the car starts again from a new episode which means “Let’s Start a new trail, for the hope of learning from its previous trail, and for the aim of surviving next time and succeeding in passing it in a correct way”. Restriction criterion is important as it indicates whether the training phase has converged to the expected goal. TORCS has many restricted conditions used especially on the DDPG Algorithm: a) Out of Track and b) Stuck on the track. We will deal with them in details.

5.2.1. Out of Track Restricted Condition

During the Learning phase of our vehicle needed to learn how to drive autonomously, when the vehicle gets out of the track, the environment gives the agent highly negative reward. After that TORCS is terminated and re-launched again which means the current episode is finished, so the vehicle will start a new episode again. This means that the vehicle will start from the beginning of the track. We can measure getting out of the track due to having a built-in 19 sensors on TORCS-SCR plug-in which allows us to get the distance from the vehicle to the Lane Boundaries. These sensors are called: Track sensors. You can refer to Fig. 3.4(a).

5.2.2. Stuck Restricted Condition

During the Learning phase of our vehicle needed to learn how to drive autonomously, when the vehicle gets stuck on the track, the environment gives the agent highly negative reward. After that TORCS is terminated and re-launched again which means the current episode is finished, so the vehicle will start a new episode again. ***This means that the vehicle will start from the beginning of the track.*** We can measure Stuck from our opinion as follows: We give the car around 100 time steps, which is equivalent to the initial motion of the car (initial acceleration), then we measure the distance covered from the previous state till the current state. If the progress doesn't exceed a certain limit (tuning parameter), this means that the vehicle gets stuck. The tuning parameter for the stuck condition depends on the vertical speed of the car (speedX) in which the stuck concept is if the speed reaches to (5 km/h). This also means during the learning phase, we prevent the car from approaching to the low speeds, so it is expected that during the testing phase, the car will not approach from these low speeds. This condition is very useful for the aim of Racing.

5.2.3. Out of Track with Stuck Restricted Conditions

TORCS has both of Out of Track and Stuck Restricted Conditions. This means that if one of them is satisfied, TORCS will be terminated and re-launched again which means the current episode is finished, so the vehicle will start a new episode again. ***This means that the vehicle will start from the beginning of the track.***

5.3. Convergence Time vs. Restricted Conditions on TORCS

In this part we will illustrate the effect of Restricted Conditions on the Convergence Time. However, we need first to define the concept of the convergence time.

5.3.1. Convergence Time Definition

It is considered as the time needed by the vehicle to build the sufficient model for autonomous driving. Actually the task of building the sufficient model is not an easy task, as we don't have a way of measurement for that aim. However, we have two proposed ways for that task: a) the first one is depending on the monitoring the performance of the vehicle by the human eyes, and b) the second one is when the vehicle succeeded in completing 15 Laps per one episode.

The first way is more efficient than the second one because the enhancement in the performance of the vehicle could be monitored easily by the following scenario: Suppose that the car is moving on a straight line, then it is about to turn to the left. Firstly, the vehicle will not have the ability to turn left, so it will go out of the track. This will lead to Terminate TORCS and re-launch it again with a new episode, after the environment gives the agent a highly negative reward value which update the agent's model and tell it "Don't do that again". The vehicle tries many times till its success in turning to the left. The previous scenario is easily to be monitored, but the time needed to build this model will be more than 24 hrs which means that it is a long time.

The second way is something tricky because the vehicle may build the sufficient model without any certain of that. However, we reached to this number of laps by trial and error, as firstly we put it by 10 laps and unfortunately it was not enough to build the model. After that we put it by 20 laps, but we found some OverFitting Problem appears during the testing phase (i.e. OverFitting Problem is a very dangerous problem which means that the model will learn how to pass this track ONLY), so we put it 15 Laps gives better accuracy and better performance while testing on other tracks.

5.3.2. Restricted conditions effect on Convergence Time

It is obvious that the restricted conditions affect badly on the Convergence Time because TORCS is terminated when one of these conditions is satisfied, so this leads to finish the current episode and start new episode. Actually, all of the previous is considered the main problem as the vehicle will start from the beginning of the track which means that ***the vehicle will cover a part from the track which is already learnt in the previous episodes***.

Due to applying TORCS two Restricted conditions of out of track and stuck on, we found that the vehicle didn't have the ability to explore the track well due to the termination and re-launching for every satisfaction of a condition. From Fig. (4.9), we found that the vehicle had smooth and great performance, but we have a restriction on getting out of the track or stuck. We know that the number of episodes in DDPG algorithm is less than the Q-Learning and Double Q–Learning algorithms, but still we need to do some enhancements in order to have the ability to: a) explore the track in a faster way than, and b) build the sufficient model for autonomous driving in less number of episodes.

5.4. Enhanced Approach for Restricted Conditions on DDPG Algorithm

For the restrictions of these conditions which have many disadvantages on the performance of the algorithm and its convergence time. This allows us to think out of the box through modifying TORCS Restricted Conditions not only enhancing the algorithms run as an agent for the TORCS environment.

5.4.1. Proposed Solution: Disable Restricted Conditions

We disabled the whole Restricted Conditions on TORCS as we let the vehicle moves in any part of the track without any restricted condition. We replaced the presence of these conditions by giving the vehicle highly negative incremental rewards if it does wrong actions like: getting out of the track or being in a stuck mode. This proposed solution ensures some of the great points as: a) the vehicle now has the ability to explore the whole track as fast as possible which is a great achievement, b) the vehicle now has the ability to complete one lap in the training phase in less number of episodes, c) we will benefit from the smooth performance of the DDPG algorithm without restricting it by some limitations, and d) We will also benefit from the maximum speed of the DDPG algorithm for the aim of racing.

We have applied our proposed solution through different experiments like: a) Disable Out of Track Restricted conditions, b) Disable Stuck Restricted Conditions, and c) Disable All termination conditions. After that, we monitored the enhancements for each experiment and report some measurement results like what we had done in Q-Learning, Double Q-Learning, and DDPG algorithms based on TORCS normal conditions.

5.4.2. Measurement Results for DDPG Algorithm without Restricted Conditions

We found that Disabling all these conditions will give the best result for decreasing the Learning phase. We reached to this result based on Table (5.1) as follows:

Table 5.1 DDPG Algorithm without any restricted condition

Measurement Result	**DDPG Algorithm No restricted Condition**
Number of Episodes to complete One Lap	5
Number of Completed Laps in Training Phase	15
Number of Episodes needed to build the Model	300
Maximum Speed for the Car	171 km/h
Time to complete One Lap	53:01s

We can notice that: a) the vehicle completed one lap in less number of episodes comparing with Q-Learning and Double Q-Learning algorithms in addition to the normal DDPG algorithm applied with normal conditions, b) the vehicle covers the maximum number of laps that were needed to be the references for building the model, c) disabling for these conditions was useful for building the model in less number of episodes, and d) disabling these conditions enable the vehicle to move freely without the fear of terminating TORCS and starting a new episode, so we acquired the highest speed for the vehicle which is useful for Racing.

The following table (5.2) will formulate the experimental results for all of the: Q-Learning, Double Q-Learning, DDPG with restricted conditions, and DDPG without restricted conditions algorithms

Table 5.2 Q-Learning, Double Q-Learning, DDPG with/without restricted conditions experimental results

Measurement Result	Q-Learning Algorithm	Double Q-Learning Algorithm	DDPG Algorithm with restricted conditions	DDPG Algorithm without restricted conditions
Number of Episodes to complete One Lap	1200	1400	**800**	**5**
Number of Completed Laps in Training Phase	2	10	**10**	**15**
Number of Episodes needed to build the Model	1500	1800	**1000**	**300**
Maximum Speed for the Car	160 km/h	148 km/h	**165 km/h**	**171 km/h**
Time to complete One Lap	01:24:09	01:15:22	**59:23**	**53:01s**

From the previous table (5.2), we can calculate some enhancement percentages specialized for the Continuous Action Algorithms without restricted conditions in green rather than Continuous Action Algorithms with restricted conditions in red:

1- Number of Episodes to complete One Lap is decreased by more than 99.3% which is a great enhancement for the car to explore the training track as fast as possible.

2- Number of Completed Laps in Training phase is increased by more than 50% which is a great enhancement that shows the more stability acquired after disabling the restricted conditions.

3- Number of episodes needed to build the model is decreased by more than 70% which is a great enhancement for the simulated car to build the sufficient model in around 30% from the previous learning time.

4- Maximum Speed for the Car is increased by more than 3.6% from the previous Car Speed which is a good enhancement for the aim of having a Racing Car.

5- Time to complete One Lap is decreased by more than 11% which is a great enhancement for the aim of having a highly stable vehicle on the road in addition to have a Racing car.

The following Fig. (5.1) describe the effect of the previously described restricted conditions on the moving vehicle with DDPG algorithm.

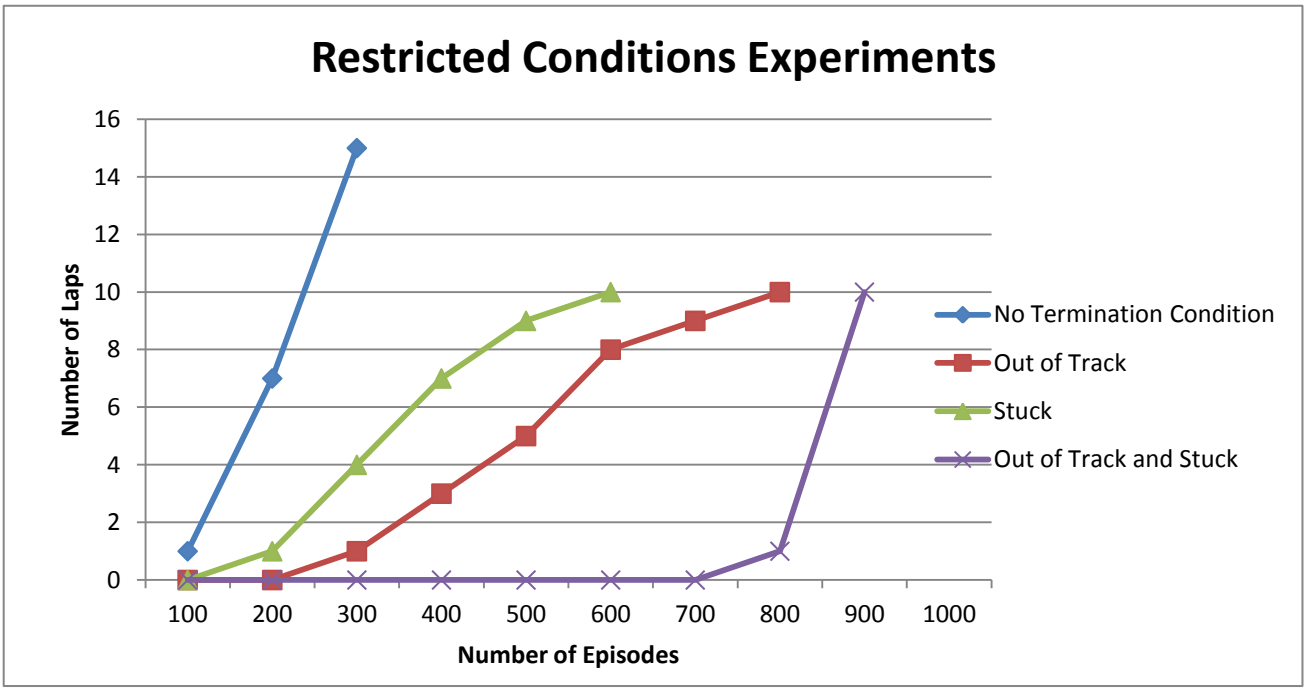

Fig. (5.1) DDPG Restricted Conditions Experiments

5.5. Conclusion

After we knew that the Continuous Action Algorithms have more stable and better performance than the Discrete Action Algorithms, we have done some enhancements over the Continuous Action Algorithms to reach to the best performance. TORCS Restricted conditions were the key-point for the enhancements as we have done many experiments which prove that it affect on the Learning phase time, so affect on the convergence time to build the model. Disabling these restricted conditions gives us a better performance when it is applied on Continuous Action Algorithms.

Chapter 6 Conclusion and Future Works

In this chapter, we will deal with two main parts which are the overall conclusion from this thesis in addition to the future research works which could help in the extension of this thesis.

6.1. Overall Conclusion

Driving a vehicle is considered as difficult process due to multiple interactions between drivers with the surrounded environment from: other vehicles, pedestrians, road boundaries, lanes, traffic signs, and traffic lights. These are not only the factors affect on driving a vehicle but also we have the climate, changing lanes, congestion…etc. The driver has his sensors like his eyes, hands, ears, and legs, he can use his eyes in order to detect other objects, his hands in order to take actions of the steering angle and his legs in order to accelerate or decelerate. For the aim of having autonomous driving, the proposed work will replace these sensors with other sensors. Actually, this work can use the: Camera Sensor in order to detect other objects, Laser and Radar in order to calculate accurately long and short distances.

By Nature, the driver is making Reinforcement Learning, as the driver can do: a) Accelerate if the road is empty with obeying the road restricted maximum speed limit, b) Decelerate if the road is congested with the avoidance of not hitting other vehicles from the back, c) Maneuver right and left if there is another vehicle beside him, and d) Avoid Road Boundaries. This means that the driver interact with the environment naturally and learn how to take an action based on his sensor reading.

The proposed work allowed us to conclude the following points:

1- Open the way to use Reinforcement Learning to participate in the automotive applications especially in a very difficult field which is Autonomous Driving. This is especially after the proposed work proved its success.

2- Succeed in applying Reinforcement Learning Algorithms on TORCS Simulator. This in order to avoid applying these algorithms on a real vehicle, as for sure will lead to its crash especially during the training phase.

3- Compare between Discrete Action Algorithms and Continuous Action Algorithms in order to know which category has the best performance.

4- Conclude that the Continuous Action algorithms have the better performance than the Discrete Action Algorithms, as this was clear from the vehicle maneuvering either right or left. In addition to the enhancement percentages calculated especially the time needed to build the model in the continuous action algorithm is about 66.6% from the discrete action algorithms.

5- Work on the Continuous Action Algorithms especially DDPG ensures having a smooth performance.

6- After that, solve having long learning time by disabling some restricted conditions like: stuck and out of track, introduce a new field of research which is studying the effect of the restricted conditions on the convergence time. In addition to that the enhancement percentages calculated especially the time needed to build the model in the continuous action algorithm without restricted conditions is about 70% from the continuous action algorithm with restricted conditions.

7- Conclude that the more we put restriction conditions, the slower convergence time to learn and we have proven this conclusion through experiments.

6.2. Future Works

The research done in this thesis can be extended simply as follows:

1- DDPG model can be participated in great competitions especially in Racing, as DDPG algorithm raced against other TORCS built-in algorithms and won the race.

2- DDPG Model can be put on an Embedded Target or Hardware, then test the model when it put on a real vehicle in order to check TORCS simulator nearness from real environment.

3- We can use Data Aggregation way in order to have a more generalized model for Autonomous Driving.

4- We can search for having different Continuous Action Algorithms and compare between them in order to get the best of the best algorithm.

5- We can try to achieve more and more optimization for the Learning time.

6- We will apply Autonomous Driving on Public transportation in order to be fully automated and prevent any accidents may be happened.

References

1. http://www.sigratech.de/research.php
2. **Juan Rosenzweig, Michael Bart**. A Review and Analysis of Literature on Autonomous Driving, OCTOBER 2015
3. **Junqing Wei, Jarrod M. Snider, Junsung Kim, John M. Dolan, Raj Rajkumar and Bakhtiar Litkouhi**. Towards a Viable Autonomous Driving Research Platform, May 2017
4. **Andrei Furda and Ljubo Vlacic**. Real-Time Decision Making for Autonomous City Vehicles, March 5, 2010
5. **Gerla, Mario, et al**. "Internet of vehicles: From intelligent grid to autonomous cars and vehicular clouds." Internet of Things (WF-IoT), 2014 IEEE World Forum on. IEEE, 2014.
6. **McNaughton, Matthew, et al**. "Motion planning for autonomous driving with a conformal spatiotemporal lattice." Robotics and Automation (ICRA), 2011 IEEE International Conference on. IEEE, 2011.
7. **McLeod, S. A. Skinner** - Operant Conditioning, 2015. Retrieved from www.simplypsychology.org/operant-conditioning.html
8. **Daniel Karavolos**, "Q-learning with heuristic exploration in Simulated Car Racing", A thesis submitted in partial fullifilment of the requirements for the degree of Master of Science in Artificial Intelligence at the University of Amsterdam, The Netherlands, August 2013.
9. https://en.wikipedia.org/wiki/Markov_decision_process
10. **Pyeatt, L. D. and Howe**, A. E. Learning to race: Experiments with a simulated race car, 1998. In Proceedings of the Eleventh International Florida Artificial Intelligence Research Society Conference, pages 357-361
11. http://torcs.sourceforge.net/
12. **Sutton, R. S. and Barto**, A. G. Reinforcement Learning: An Introduction. MIT Press, Cambridge, MA (1998).
13. **Schaeffer, J.** The games computers (and people) play. Advances in Computers, 52:189-266 (2000).
14. **El Sallab, A., Abdou, M., Perot, E., and Yogamani, S.** Deep Reinforcement Learning framework for Autonomous Driving, Autonomous Vehicles and Machines, Electronic Imaging 2017
15. **Martijn van Otterlo , Marco Wiering (2012)**. Reinforcement Learning and Markov Decision Processes, chapter (1).
16. **Hasselt, H.** V. Double Q-learning. Advances in Neural Information Processing Systems, 2010, (pp. 2613-2621).
17. **Silver, D., Lever, G., Heess, N., Degris, T., Wierstra, D., & Riedmiller, M,** Deterministic policy gradient algorithms ICML, 2014.

18. **Jan Peters, Sethu Vijayakumar, and Stefan Schaal,** United Kingdom. Natural Actor Critic, 2005

19. **Huval, B., Wang, T., Tandon, S., Kiske, J., Song, W., Pazhayampallil, J., Andriluka, M., Cheng-Yue, R., Mujica, F., Coates, A. and Ng, A.Y.** An Empirical Evaluation of Deep Learning on Highway Driving, 2015. arXiv preprint arXiv:1504.01716.

20. **Hadsell, Raia, et al.** "Deep belief net learning in a long-range vision system for autonomous off-road driving." Intelligent Robots and Systems, 2008. IROS 2008. IEEE/RSJ International Conference on. IEEE, 2008.

21. **Hadsell, Raia, et al.** Learning long-range vision for autonomous off-road driving, 2009 Journal of Field Robotics 26.2: 120-144.

22. **Bagnell, James Andrew, et al.** Learning for autonomous navigation, 2010. Robotics & Automation Magazine, IEEE 17.2: 74-84.

23. **Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., … & Wierstra,** D. Continuous control with deep reinforcement learning, 2015. arXiv preprint arXiv:1509.02971.

24. **Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., ... & Petersen, S.** Human-level control through deep reinforcement learning, 2015 Nature, 518(7540), 529-533.

25. **Wulfmeier, Markus, Peter Ondruska, and Ingmar Posner.** Maximum Entropy Deep Inverse Reinforcement Learning, 2015 arXiv preprint arXiv:1507.04888.

26. **Vanhoucke, Vincent, Andrew Senior, and Mark Z. Mao.** Improving the speed of neural networks on CPUs, 2011, Proc. Deep Learning and Unsupervised Feature Learning NIPS Workshop. Vol. 1.

27. **Ivo Grondman, Lucian Bus¸oniu, Gabriel A.D. Lopes and Robert Babuska.** A Survey of Actor-Critic Reinforcement Learning: Standard and Natural Policy Gradients, 2012

28. **Abdullahi, A. A. and Lucas, S. M. (2011).** Temporal difference learning with interpolated ntuples: Initial results from a simulated car racing environment. In 2011 IEEE Conference on Computational Intelligence and Games (CIG11), pages 321-328. IEEE Press

29. **Athanasiadis, C., Galanopoulos, D., and Tefas, A.** (2012). Progressive neural network training for the open racing car simulator. In Proceedings of the IEEE Symposium on Computational Intelligence in Games (CIG '12), pages 116-123. IEEE.

30. **Butz, M. V., Linhardt, M. J., and L□bnneker, T. D.** (2011). Effective racing on partially observable tracks: Indirectly coupling anticipatory egocentric sensors with motor commands. IEEE Transactions on Computational Intelligence and AI in games, 3(1):31-42.

31. **Eiben, A. E. and Smith, J. E.** (2003). Introduction to Evolutionary Computing. SpringerVerlag.

32. **Galway, L., Charles, D., and Black, M.** (2008). Machine learning in digital games: a survey. Artificial Intelligence Review, 29:123-161.

33. **Loiacono, D., Cardamone, L., and Lanzi, P. L.** (2012). SCR Championship - Competition Soft-ware Manual. Retrieved at June 6, 2013, from http://sourceforge.net/projects/cig/files/SCR

34. **Tokic, M.** (2010). Adaptive e-greedy exploration in reinforcement learning based on value differences.

35. In Dillmann**, R., Beyerer, J., Hanebeck, U., and Schultz, T., editors, KI 2010**: Advances in Artificial Intelligence, volume 6359 of Lecture Notes in Computer Science, pages 203-210.Springer Berlin Heidelberg.

القياده الذاتيه باستخدام متعلم التقويه العميق

الملخص

القيادة الذاتية هي واحدة من المشاكلات الصعبة التي تواجه تطبيقات السيارات. في الوقت الحاضر، يحظر القيادة الذاتية بسبب وجود بعض القوانين المقيدة التي تمنع السيارات من أن تسير مستقلة خوفا من الحوادث. ومع ذلك، يحاول الباحثون الوصول للقيادة الذاتية كمجال جديد للبحوث و ذلك بهدف وجود دافع تجاه هذه القوانين. للقياده الذاتيه الكثير من المشكلات صعبه الحل قد تتلخص فى السلوك الغير متوقع للسيارات المستقله خلال التعامل مع السيارات الاخرى.

ويعتبر متعلم التقويه نموذج قوي للذكاء الاصطناعي الذى يمكن أن يعلم الآلات من خلال التفاعل مع العالم المحيط، والتعلم من الاخطاء. في الواقع فإن متعلم التقويه أثبت نجاحه عند تطبيقه على ألعاب الأتاري، لكنه لم يثبت نجاحه في تطبيقات السيارات.

يعتمد العمل المقترح على استخدام مجال متعلم التقويه من أجل الوصول إلى هدفنا المتمثل في وجود مركبة ذاتية القيادة. متعلم التقويه لديه فئتين رئيسيتين: خوارزميات العمل المنفصلة وخوارزميات العمل المستمر. وهذا العمل يقارن بين كل واحد منهم من أجل معرفة أي واحد هو أفضل لتطبيق بعض التحسينات عليه. يعتبر متعلم التقويه واحد من فئات تعليم الآلة، لذلك لديه مرحلتين رئيسيتين: مرحلة التعلم، ومرحلة الاختبار. بالتأكيد، فإن العمل المطبق لم يتم على تدريب سيارة حقيقية، لذلك تم استخدام المحاكى TORCS. اثبت العمل المقترح امكانيه تحقيق القيادة الذاتية باستخدام كل من: خوارزميات العمل المنفصلة مثل: خوارزميات المتعلم Q و المتعلم Q المزدوج، وخوارزميات العمل المستمر مثل: خوارزمية السياسة العميقة حتمية التدرج DDPG. لقد تم اثبات ان للخوارزمية السياسة العميقة حتمية التدرج DDPG أداء سلس وأفضل من خوارزميات المتعلم Q و المتعلم Q المزدوج من خلال بعض التجارب العلميه. بجانب ذلك, تعتبر مشكله طول وقت التعلم واحدة من المشكلات الرئيسية التي تواجه تعليم الآله، وبالتالي فإن العمل المقدم استطاع ان يقلل من طول فتره التعلم عن طريق تعطيل الشروط المقيده والتعويض عنها من خلال مصطلح المكافأه.

تحتوى هذه الرساله على سته فصول بما فى ذلك الفصل التمهيدى على النحو التالى:

الفصل الثانى يشرح كيفيه الوصول للقياده الذاتيه, يستعرض بعض من الاعمال السابقه التى اهتمت بالوصول لهذا الهدف من قبل, ثم بعد ذلك يعطى نبذه عامه عن مجال التقويه العميق و كيفيه ربطه بعمليه اختيار القرار لماركوف (MDP). الى جانب ذلك, عرض الحل المبدأى لعمليه اختيار القرار لماركوف.

الفصل الثالث يشرح المحاكى TORCS الذى سمح لنا بتطبيق بعض الخوارزميات من خلال الاستشعارات المقدمه. هذه الاستشعارات تمدنا بالمعلومات الكافيه عن العالم المحيط. تناول الفصل بعد ذلك كيفيه ربط المحاكى TORCS بعمليه اختيار القرار لماركوف و ذلك من خلال: حالتها و افعالها.

الفصل الرابع يشرح كيفيه تطبيق متعلم التقويه على المحاكى TORCS عن طريق شرح حلقه التفاعل بين العميل و الخادم التى تكمن فى ارسال قراءات و استقبال الافعال. بعد ذلك يتم تقسيم الخوارزميات المطبقه الى نوعين: خوارزميات العمل المنفصلة وخوارزميات العمل المستمر و من ثم عرض كيفيه تطبيق كلاهما على TORCS من خلال بعض التجارب العلميه. و فى النهايه, عمل مقارنه بين اداء السياره عند تطبيق النوعين السابقين. و لقد اثبتنا ان خوارزميات العمل المستمر و بالاخص خوارزمية السياسة العميقة حتمية التدرج DDPG لديها اداء افضل من خوارزميات العمل المنفصلة. ليس هذا فقط بل تم حساب بعض نسب التحسين التى امتازت بها خوارزميات العمل المستمر عن خوارزميات العمل المنفصلة و بالاخص: الوقت اللازم للتعلم حيث ان اصبح هذا الوقت حوال 66.6% من الوقت السابق.

الفصل الخامس يشرح الشروط الموجوده فى المحاكى TORCS و التى تقيد حركه السياره و تمنعها من استكشاف الطريق. و يشرح ايضا اجراء بعض التعديلات على خوارزميات العمل المستمر و خاصه خوارزمية السياسة العميقة حتمية التدرج. تكمن هذه التعديلات فى تعطيل الشروط المقيده والتعويض عنها من خلال مصطلح المكافأه. و لقد اثبتنا ان خوارزميات العمل المستمر و بالاخص خوارزمية السياسة العميقة حتمية التدرج DDPG بدون الشروط المقيده لديها اداء افضل من خوارزميات العمل المستمر مع تطبيق الشروط المقيده. ليس هذا فقط بل تم

حساب بعض نسب التحسين التى امتازت بها خوارزميات العمل المستمر بدون شروط عن خوارزميات العمل المستمر مع تطبيق الشروط و بالاخص: الوقت اللازم للتعلم حيث ان اصبح هذا الوقت حوالى 70% من الوقت السابق.

الفصل السادس يوضح الاستنتاجات من الرساله ثم يعرض بعض الافكار المستقبليه المتعلقه بمجال القياده الذاتيه.

و تحتوى الرساله على الاضافات التاليه:

1- فتح الباب امام متعلم التقويه للمساهمه فى مجال السيارات و بالاخص فى واحد من اصعب المجالات الا وهو: القياده الذاتيه الذى اثبتنا من خلال العمل المقترح انه من الممكن الوصول له من خلال العديد من الخوارزميات.
2- تطبيق انواع كثيره من الخوارزميات و يعتبر العمل المقدم هو الاول فى المقارنه بينهم من خلال التجارب العلميه على المحاكى مثل: خوارزميات العمل المنفصلة وخوارزميات العمل المستمر. و من امثله خوارزميات العمل المنفصلة: خوارزميات المتعلم Q و المتعلم Q المزدوج, و من امثله خوارزميات العمل المستمر: خوارزمية السياسة العميقة حتمية التدرج DDPG.
3- اثبات ان خوارزميات العمل المستمر و بالاخص خوارزمية السياسة العميقة حتمية التدرج DDPG افضل من خوارزميات العمل المنفصلة مثل: خوارزميات المتعلم Q و المتعلم Q المزدوج. و ذلك من خلال:
   أ- أداء السياره فى كلا النوعين.
   ب- حساب بعض نسب التحسين و التى تتمثل فى: الوقت اللازم للتعلم, سرعه السياره, عدد الجولات اللازمه لاكمال لفه واحده لهذا الطريق.
4- حساب بعض نسب التحسين التى امتازت بها خوارزميات العمل المستمر عن خوارزميات العمل المنفصلة و بالاخص: الوقت اللازم للتعلم حيث ان اصبح هذا الوقت حوال 66.6% من الوقت السابق.

5- حساب بعض نسب التحسين التى امتازت بها خوارزميات العمل المستمر بدون شروط عن خوارزميات العمل المستمر مع تطبيق الشروط و بالاخص: الوقت اللازم للتعلم حيث ان اصبح هذا الوقت حوالى 70% من الوقت السابق

6- يعتبر العمل المقدم هو الاول فى الأخذ فى الاعتبار مشكله طول فتره التعلم, لذلك فقد تم تقليل فتره تعلم السياره من خلال تركها لاستكشاف الطريق و ذلك بعد تعطيل الشروط المقيده لحركه السياره و التعويض عنها من خلال مصطلح المكافأه.

مهندس: محمد عبده طلبه ابراهيم

تاريخ الميلاد: 1991\05\28

الجنسية: مصرى

تاريخ التسجيل: 2013\10\01

تاريخ المنح: 2017

القسم: هندسه الالكترونيات و الاتصالات الكهربيه

الدرجة: ماجستير العلوم

المشرفون:

ا.د. حنان أحمد كمال

الممتحنون:

أ.د. حنان أحمد كمال (المشرف الرئيسي)

أ.د. عمر أحمد نصر (الممتحن الداخلي)

أ.د. أحمد محمد الجارحى (الممتحن الخارجي)

أستاذ التحكم و عميد كليه الهندسه جامعه حلوان

عنوان الرسالة:

القياده الذاتيه باستخدام متعلم التقويه العميق

الكلمات الدالة:

القياده الذاتيه, متعلم التقويه, المحاكى TORCS, المتعلم Q, السياسة العميقة حتمية التدرج.

ملخص الرسالة:

القيادة الذاتية هي واحدة من المشاكل الصعبة التي تواجه تطبيقات السيارات. ويرجع ذلك إلى العديد من المشكلات التي يمكن ان تتلخص فى السلوك الغير متوقع من للسيارات المستقله خلال التعامل مع السيارات الأخرى. يعتمد العمل المقدم على استخدام مجال متعلم التقويه، وهو نموذج قوي الذكاء الاصطناعي الذي يمكن ان يعلم الآلات من خلال التفاعل مع العالم المحيط والتعلم من أخطائهم، وذلك من أجل الوصول إلى وجود مركبه ذاتيه القياده. يقارن هذا العمل بين فئتين رئيسيتين: خوارزميات العمل المنفصلة مثل: خوارزميات المتعلم Q، ومتعلم Q المزدوج، وخوارزميات العمل المستمر مثل: خوارزميه السياسة العميقه حتمية التدرج DDPG. ولقد ثبت كما هو متوقع أن خوارزميات العمل المستمر لديها أداء أفضل، لذلك طبقنا بعض التحسينات على خوارزمية السياسه العميقه حتميه التدرج DDPG مثل حل مشكله طول فتره التعلم التى تواجه تعليم الآله. وتعتمد هذه التحسينات على تعطيل بعض الشروط المقيدة والتعويض عنها من خلال مصطلح المكافأة. العمل المقترح يعتمد على محاكى يدعى TORCS للوصول إلى هدفنا.

القياده الذاتيه باستخدام متعلم التقويه العميق

اعداد

محمد عبده طلبه ابراهيم

رسالة مقدمة إلى كلية الهندسة - جامعة القاهرة

كجزء من متطلبات الحصول على درجة **ماجستيرالعلوم**

في

هندسه الالكترونيات و الاتصالات الكهربيه

يعتمد من لجنة الممتحنين:

الاستاذ الدكتور: حنان أحمد كمال (المشرف الرئيسى)

الدكتور: عمر أحمد نصر (الممتحن الداخلى)

الاستاذ الدكتور: أحمد محمد الجارحى (الممتحن الخارجى)
أستاذ التحكم و عميد كليه الهندسه جامعه حلوان

كليــة الهندســة - جامعــة القاهــرة

الجيـزة - جمهوريـة مصـر العربيــة

2017

القياده الذاتيه باستخدام متعلم التقويه العميق

اعداد

محمد عبده طلبه ابراهيم

رسالة مقدمة إلى كلية الهندسة - جامعة القاهرة

كجزء من متطلبات الحصول على درجة **ماجستيرالعلوم**

في

هندسه الالكترونيات و الاتصالات الكهربيه

تحت اشراف

أ.د. حنان أحمد كمال

أستاذ هندسة نظم التحكم

كلية الهندسة، جامعة القاهرة

كليــة الهندســة - جامعــة القاهــرة

الجيـزة - جمهوريـة مصـر العربيــة

2017

القياده الذاتيه باستخدام متعلم التقويه العميق

اعداد

محمد عبده طلبه ابراهيم

رسالة مقدمة إلى كلية الهندسة - جامعة القاهرة
كجزء من متطلبات الحصول على درجة **ماجستيرالعلوم**
في
هندسه الالكترونيات و الاتصالات الكهربيه

كليــة الهندســة - جامعــة القاهــرة
الجيـزة - جمهوريـة مصـر العربيــة
2017