

Deep Reinforcement Learning for Autonomous Driving Strategy

A project report submitted

to

MANIPAL ACADEMY OF HIGHER EDUCATION

For Partial Fulfillment of the Requirement for the

Award of the Degree

of

Bachelor of Technology

in

Computer and Communication Engineering

by

Ashwin S H

Reg. No. 180953352

Under the guidance of

Dr. Rashmi Naveen Raj

Associate Professor

Department of I & CT

Manipal Institute of Technology

Manipal, India



MANIPAL INSTITUTE OF TECHNOLOGY

MANIPAL

(A constituent unit of MAHE, Manipal)

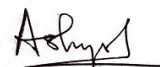
May 2022

DECLARATION

I hereby declare that this project work entitled **Deep Reinforcement Learning for Autonomous Driving Strategy** is original and has been carried out by me in the Department of Information and Communication Technology of Manipal Institute of Technology, Manipal, under the guidance of **Dr. Rashmi Naveen Raj, Associate Professor**, Department of Information and Communication Technology, M. I. T., Manipal. No part of this work has been submitted for the award of a degree or diploma either to this University or to any other Universities.

Place: Manipal

Date :25-05-2022



Ashwin S H



MANIPAL INSTITUTE OF TECHNOLOGY

MANIPAL

(A constituent unit of MAHE, Manipal)

CERTIFICATE

This is to certify that this project entitled **Deep Reinforcement Learning for Autonomous Driving Strategy** is a bonafide project work done by **Mr. Ashwin S H (Reg.No.:180953352)** at Manipal Institute of Technology, Manipal, independently under my guidance and supervision for the award of the Degree of Bachelor of Technology in Computer and Communication Engineering.

Dr. Rashmi Naveen Raj

Associate Professor

Department of I & CT

Manipal Institute of Technology

Manipal, India

Dr. Smitha N. Pai

Professor & Head

Department of I & CT

Manipal Institute of Technology

Manipal, India

ACKNOWLEDGEMENTS

I would like to express my gratitude to my guide Dr. Rashmi Naveen Raj, Associate Professor, Department of Information and Communication Technology, for her invaluable guidance, constant support, and encouragement throughout my project period. I am also thankful to Dr. Sanjay Singh, Professor and Project Coordinator, Dr. Smitha N. Pai, Professor and Head, Dept of I&CT and Dr. (Cdr) Anil Rana, Director, MIT Manipal for their cooperation and support in carrying out this work.

I express my sincere thanks to all the faculty and staff members of the Department of I&CT for their help and cooperation.

I am, indeed, grateful to my parents and the Almighty for helping me constantly throughout my student life.

ABSTRACT

The constantly increasing vehicles on the road and the carelessness of the drivers, result in huge number of accidents and fatalities all over the globe. In addition to this, people spend a lot of their time on roads driving their vehicles in busy traffic. Because of these, autonomous vehicles have garnered momentous attention. This will also address several enduring challenges related to traffic congestion, vehicle parking, and many other issues in the transportation sector. From the various recent literature, it is found that to tackle these issues, Reinforcement Learning, one of the finest Machine learning applications is the best-suited approach.

To develop an Autonomous Driving Strategy, Deep Deterministic Policy Gradient, one of the RL algorithms is used. This algorithm deals with continuous state spaces, which is the essential requirement of autonomous driving. Out of the various simulators available, the TORCS simulator satisfies the requirements to implement this project.

The objectives of lane keep operation of the car using Reinforcement Learning and avoiding collisions by detecting obstacles using sensors have been achieved successfully. After 14000 episodes, the simulated car learns to drive by itself without any hiccups. Elderly people, patients on medication, persons with disabilities, and busy executives can make use of this technology.

[Computing Methodologies]: Machine Learning—learning paradigms, reinforcement learning

Contents

Acknowledgements	iii
Abstract	iv
List of Tables	vii
List of Figures	viii
Abbreviations	viii
Notations	x
1 Introduction	1
2 Literature Survey	6
3 Problem Statement and Objectives	11
3.1 Problem Statement	11
3.2 Objectives	11
4 Methodology and Implementation	12
4.1 DDPG Algorithm	13
4.1.1 State Space	15
4.1.2 Action Space	17
4.1.3 Reward Function	18
4.2 Implementation	19

4.2.1	Lane Keep	19
4.2.2	Front obstacle detection	20
4.2.3	Overtaking	21
4.2.4	Side obstacle detection	22
5	Result Analysis	23
6	Conclusion and Future Scope	30
6.1	Conclusion	30
6.2	Future Scope	31
	References	32
	Project Details	34

List of Tables

2.1	Details of states, actions and rewards	10
4.1	Table of States [1]	17
4.2	Table of Actions [1]	18
6.1	Project Details	35

List of Figures

1.1	Standard blocks of Autonomous Driving System [2]	2
1.2	Classification of Machine Learning algorithms [3]	3
1.3	General scenario of Reinforcement Learning [4]	4
4.1	Deep Deterministic Policy Gradient block diagram	13
4.2	Driving Environment [1]	16
5.1	TORCS main screen	23
5.2	Track selection screen	24
5.3	Agent driving on the track	24
5.4	Agent goes off-track	25
5.5	Agent slowly getting back on track	25
5.6	Training of the agent	26
5.7	Rewards vs speed, angle, and position (track 1)	27
5.8	Rewards vs speed, angle, and position (track 2)	28
5.9	Rewards vs speed, angle, and position (track 3)	28
5.10	Rewards vs speed, angle, and position (track 4)	29

ABBREVIATIONS

RL	: Reinforcement Learning
AI	: Artificial Intelligence
ML	: Machine Learning
DL	: Deep Learning
DQN	: Deep Q Network
DPG	: Deterministic Policy Gradient
DDPG	: Deep Deterministic Policy Gradient
DDQN	: Double Deep Q Network
GNSS	: Global navigation satellite system
SUMO	: Simulation of Urban Mobility
TORCS	: The Open Racing Car Simulator
LHD	: Left Hand Drive
GPS	: Global Positioning System
LiDAR	: Light Detection And Ranging
RADAR	: Radio Detection And Ranging

NOTATIONS

δ	:	offset between centre of the lane and agent
ζ	:	distance between the edge of the lane and agent
θ	:	departure angle between the lane and agent
v	:	velocity
a	:	acceleration
s	:	state
a	:	action
s'	:	next state
a'	:	next action
r	:	reward
Q	:	critic network
μ	:	actor network
Q'	:	target critic network
μ'	:	target actor network
θ	:	weights for the networks
τ	:	update factor for target networks
μ	:	actor network

Chapter 1

Introduction

Autonomous vehicles are a class of vehicles that utilize multiple sensors such as LiDARs, RADARs, GPS-GNSS, cameras etc. to perceive its surroundings and move with scant or no human interaction [2]. In the present world, lots of tasks are being automated to provide humans more convenience and safety. There can be scenarios where the drivers are not in a good state to drive, and it may lead to accidents. Instead, if the task of driving is given to an adequately trained machine, it will perform its task with maximum efficiency every single time. The web article [5] speaks about the evolution of autonomous vehicles. It states that the cars became autonomous first in 1920s and were called “Phantom Autos”. They were called so because they were remote controlled. Later in the 1980s self-managed autonomous cars were introduced by Mercedes, which were not fully automatic. It was performing lane change with human intervention. Because of the advancements in automobiles, information technology and the increasing interest in designing the autonomous cars by many young researchers, this will result in low-cost, reliable and efficient cars on the road in the near future.

Autonomous driving systems consist of a few standard blocks as shown in Figure 1.1 [2]. Scene understanding, localization, mapping, planning, driving policy and control are a few problems addressed by these modules.

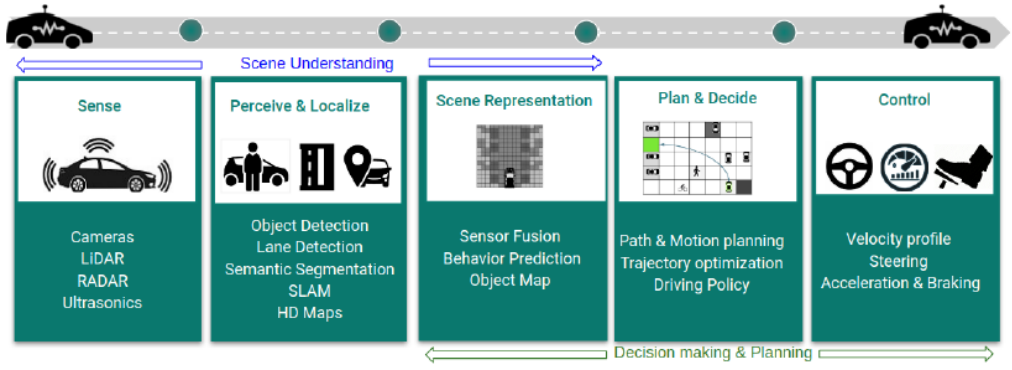


Figure 1.1: Standard blocks of Autonomous Driving System [2]

Machine Learning (ML) models can be used to predict, classify/categorize and perform many more tasks. Usually, ML algorithms are classified into three broad categories: reinforcement, supervised and unsupervised learning. In supervised learning algorithms, the model is trained beforehand using labelled data to perform a certain task whereas in unsupervised learning the model is forced to learn from unlabelled data. Labelled data is a collection of information with one or more labels. These labels are useful tags with the help of which ML models can quickly interpret the data. Labelled data are the cornerstone of supervised learning [6]. The third category is Reinforcement Learning (RL) where an agent improves its performance in a task by constantly interacting with its surrounding. The classification and a few of its applications are as shown in Figure 1.2.

For the process of autonomous driving which is a sequential decision problem, RL is the optimal choice as the vehicle needs to actively interact with its environment to drive safely. It is impossible to train the vehicle for all possible scenarios beforehand thus ruling out supervised and unsupervised learning approaches. There are a few challenges of using RL though, such as bridging the gap between simulation and reality, sample efficiency etc. Research in applying RL for autonomous driving requires tremendous effort from academicians, researchers and automotive industry experts from various fields to bring this

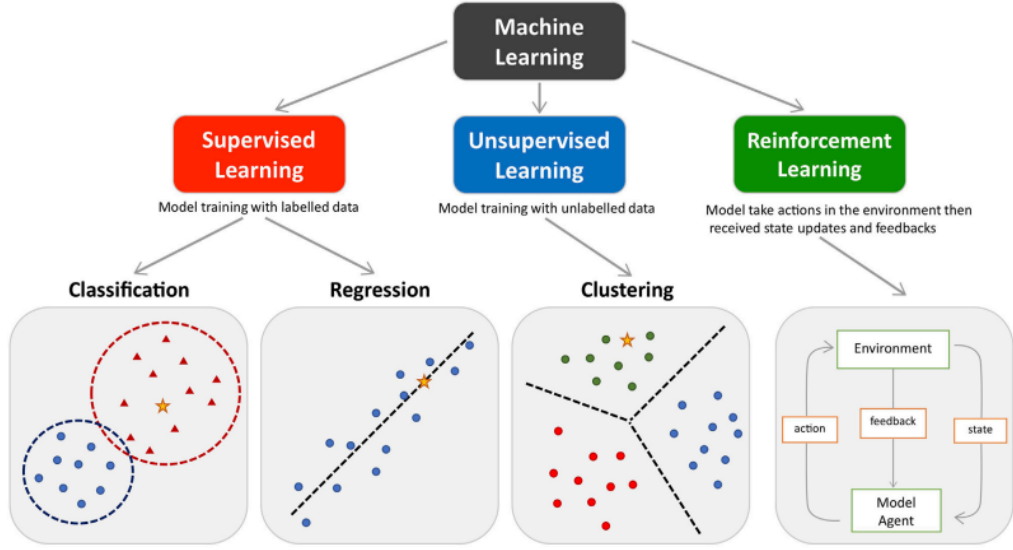


Figure 1.2: Classification of Machine Learning algorithms [3]

idea into reality.

In the RL model, an autonomous agent masters its performance by persistent interaction with its environment. Reward function is a performance criteria which is used to evaluate the RL agent [4]. The general scenario of RL is as shown in Figure 1.3. At any time instance t , the agent in state s_t which belongs to the state space S , chooses an action a_t belonging to action space A and receives a reward $r_t \in R$ (which can be positive or negative) from its environment based on the practicality of its decision. The agent then moves to the next state s_{t+1} and proceeds to choose an action for this state. The cycle continues until the agent has learned the task completely or reached the final state. The agent's objective is to maximize the cumulative rewards obtained during the entire lifetime of the task. Eventually, by exploiting the knowledge learnt, agent can increase the lifetime reward. It expands its knowledge by trial and error method [4].

The prime challenge in RL is managing the balance between exploration and exploitation. To enhance the rewards it receives, an agent must exploit its knowledge by choosing actions that prove to be highly rewarding. In contrast, to ascertain such favourable actions, it has to make certain risky decisions.

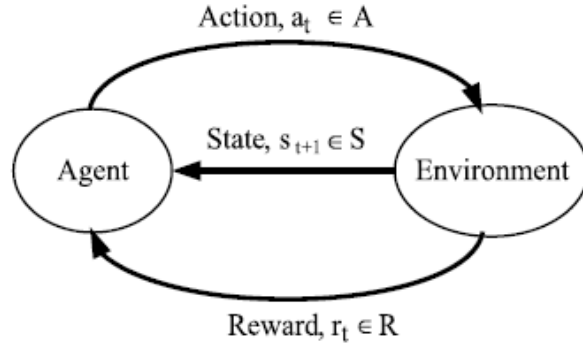


Figure 1.3: General scenario of Reinforcement Learning [4]

This may or may not result in higher rewards [7]. The strategy used by the agent to choose an action for a given state is known as a policy. If a particular policy gives maximum reward, such a policy is termed as optimal policy.

RL can be used to solve many real world problems but sometimes, there exists certain situations where conventional RL algorithms fail to provide desirable results. This is mainly due to the fact that the state and/or action spaces involved in these problems are very high dimensional e.g. camera images, infrared images etc. and it is impossible to store all state- action pairs. To solve such problems, Deep Learning(DL) is used along with RL and it is referred to as Deep Reinforcement Learning(DRL). Usually, in DRL the policy is represented as a Neural Network.

Deterministic Policy Gradient (DPG) is an algorithm that makes use of policy gradient function. Instead of representing the policy as a probability distribution, DPG uses gradient descent to create a policy that is deterministic in nature. The main advantage of DPG compared to other stochastic policy algorithms is that it is simpler and values can be computed more efficiently. But it comes with its own drawback, i.e. it becomes unstable when it is used to solve problems with high dimensional state or action spaces.

Conventional Q-learning algorithm calculates the Q values for state- action pairs and stores it in a tabular form. However, when the state and/or action

spaces are very huge, it becomes infeasible to create table of Q values. Instead, neural networks are used as the memory and computation required would be less. Deep Q-Learning function approximator is used to solve such a problem. This learning algorithm is called Deep Q-Network (DQN). The primary advantage is that it can be used with high dimensional state and action spaces but it sometimes leads to over estimation of Q value due to random initialization of the same at the beginning.

Chapter 2

Literature Survey

In the field of autonomous driving, a lot of research is going on. Recent papers have been reviewed and findings from a few of them are listed below.

The work by Zhao et al., 2020 [8] aims to model the decision making and interactions between various vehicles which run on highways. Double Deep Q-Network (DDQN) is employed to train the host vehicle. An open source simulation platform called 'SUMO - Simulation of Urban Mobility' is used to implement the work. The driving environment is created by having three driving lanes and randomly distributing 20 cars on the highway. While driving, the host constantly measures the distance between itself and the car ahead of it. In case the distance reduces between successive measurements, it begins to apply brakes to avoid collision. The speed of the host is altered accordingly by the algorithm.

Deep Deterministic Policy Gradient (DDPG) is one of the significant algorithms in the field of RL. It has been thoroughly explained in the work by Huang et al., 2019 [1]. In any autonomous driving system, mapping of driving state to driving action is an important criteria. This work establishes the same mapping using DDPG algorithm on TORCS simulation software. Here, actor network and target actor are used for outputting the best action for a particular state. Q-value is estimated using critic and target critic network.

The transitions obtained from exploration are stored in replay buffer.

The work done by Zhang et al., 2018 [9] focuses on using Double Deep Q Learning to build the vehicle speed control model. In Q Learning, an agent selects an action with best Q value which might not be an optimal choice. This is considered as over-estimation of action value which results in a complicated learning process. To resolve the over-estimation, two separate Q value estimators (Double Q) can be used. Unbiased Q-value estimation of the actions can be done by using these two separate estimators. The implementation of this concept using Deep Neural Networks is known as Double Deep Q Learning. The conventional supervised learning approach is not used so that the system is forced to approximate the values of the function by interacting with the environment. It also designs a reward network where each state is mapped to a reward expressing the expectation of the state. Double Deep Q Learning is efficient in both value accuracy and policy quality. The authors have reported that DDQN model's score is very high when compared with that of Deep Q-Network (DQN).

The work by Chopra et al., 2020 [10] is aimed at steering the vehicle in its path with the help of Deep Q-Learning algorithm. The model uses raw images, sensor inputs and calculated rewards to create a Q-value approximator which is used to steer the car. Authors were able to develop an algorithm to achieve the same in TORCS simulator. The drawback of this work is that it is very time consuming to train the model. This drawback provides a future scope of implementing Imitation Learning to make the algorithm faster by first training it using labelled data and then running RL on it. Table 2.1 shows the state space, action space and reward function of various existing work related to autonomous driving using reinforcement learning.

This survey paper by Elallid et al. 2022 [11] focuses on deep learning and reinforcement learning based approaches on autonomous vehicles with major functionalities such as scene understanding, decision making, motion

planning, vehicle control etc. This paper reviews the literatures pertaining to autonomous vehicles that use DL/RL from 2016 to 2021. An extensive comparison is also made by the authors with respect to the functionalities listed above. It thoroughly explains the various sensors and their uses in autonomous vehicles. It also highlights various challenges which need to be addressed by the researchers in the days to come. One of the main challenges are behaviour of AVs in different weather and lighting conditions. The authors are also unsure about the reliability of the Neural Network models that are trained using datasets.

The work by Rasheed Hussain et al. [12] focuses on the results that have been accomplished till now and the challenges that lie ahead for researchers working in the field. The automobile industry attained significant milestones in designing reliable, safe, and efficient vehicles throughout the last century. Because of momentous progress in computation and communication technologies, autonomous vehicles are becoming a dream come true.

The work by Zhu et al. [2020, 13] is a car-following simulation model i.e. the agent learning to follow the lead vehicle that is being controlled by a human in front of it. Rather than utilizing a plethora of sensors, they have just used the distance between the agent and the lead vehicle to calculate time for collision, jerk in driving etc. The reward function is developed by observing human driving data captured real time from the lead car and later combining it with driving related features such as efficiency, comfort and safety. They have used 'Next Generation Simulation' software for the implementation.

This paper by Omeiza et al. [2021, [14]] is about 'explainable autonomous driving'. 'Explainability' is an essential prerequisite for AVs. AVs must be able to explicate what they see, do, and might do in the environments in which they interact. This will build confidence and trust in AVs. Four main points are addressed in this paper. They are motivation for explanations, requirements of explanations for autonomous vehicles, review of previous work on expla-

nations for AVs, and finally, authors have given a conceptual framework for autonomous vehicles explainability.

The paper by Grigorescu et al. 2020 [15] addresses the major areas which form open challenges for the budding researchers in the field of autonomous driving. They make a strong statement that Deep Learning and Artificial Intelligence will play a phenomenal role in overcoming these challenges. Among the major areas they addressed, functional safety and real time computing and communication prove to be really challenging.

The paper by Yang et al. 2020 [16] is a complete survey of significant works in the area of AVs. It aims to analyze the use of Artificial Intelligence in building the primary applications of autonomous driving. These applications are perception, localization & mapping, and decision making. It explores the recent approaches to comprehend how AI can be made use of and what are the challenges associated with their implementation. Based on the survey of current practices and advancement of the technologies, this paper further provides acumens into impending opportunities regarding the use of AI in juxtaposition with other developing technologies.

In the work done by Cao et al. 2022 [17] the authors have discussed about Trustworthy improvement Reinforcement Learning (TiRL). It has been mentioned that in spite of RL algorithms having the ability to constantly improve, there are instances where they are unreliable due to rule or model based algorithms. To get the best of both worlds the authors have designed a decision making framework that utilizes RL and rule/model based algorithms. By doing so, their final framework has the potential to self-learn with better reliability. They have done simulations with more than 42000 kilometres of driving which was calibrated with human driving data. They have proved that TiRL can do better than any other arbitrary model based policy.

Table 2.1: Details of states, actions and rewards

Work by	State space	Action space	Reward function
Zhao et al., 2020 [8]	$S = \{pos_x, pos_y, v_x, v_y\}$	$A = \{acceleration\}$ <i>value</i>	$R = 1 - \frac{V_{max} - V_x}{V_{max}}$
Huang et al., 2019 [1]	$S = \{v_{x,y,z} \in R^3, \zeta \in R^{19}, \delta, \theta\}$	$A = \{accelerate, brake, steer\}$	$R = v \cos \theta - v \sin \theta - v \delta$
Zhang et al., 2018 [9]	GPS position, acceleration, relative speed and position	$A = \{a a \in \{accelerate, decelerate, maintain\}\}$	$R = \{-2, -1, 0, 1, 2\}$ depending upon the action taken
Chopra et al., 2020 [10]	Camera images	$A = \{left, half_left, no_action, half_right, right\}$	$R = v \cos \theta$

Chapter 3

Problem Statement and Objectives

3.1 Problem Statement

From the literature review it is noted that, in most of the work RL algorithms have been extensively used to control the autonomous vehicles. However most of these work concentrated on only one car in the track/road and always focusses on increasing the reward function depending on the speed of the car. The problem statement of this work is to detect other cars and obstacles present on road and drive the autonomous vehicle keeping in mind all those obstacles.

3.2 Objectives

Based on the problem statement defined in the previous section, the following objectives are formed:

1. To deploy a suitable RL algorithm to control the speed and facilitate lane keep operation of the car.
2. To avoid collisions by detecting obstacles using sensors.

Chapter 4

Methodology and Implementation

After extensive literature survey, it is found that Carla, TORCS, Airsim, SUMO are a few simulators which are considered for autonomous driving applications. Each of them has its own pros and cons. Carla and Airsim, being relatively new softwares, are more suitable for vision based driving and are known for their photo-realistic driving scenes which require higher graphic computations. SUMO does not have its RL capabilities very well developed. On the other hand TORCS simulator satisfies the requirements to implement this project. Hence, TORCS-1.3.7 in Ubuntu 16.04 OS with Python 3.5 with tensorflow and keras deep learning framework is used. It is an open source, lightweight simulator software which is preferred due to its simplicity and variety of driving dynamics. The server code to control the car is installed with the software during installation. The client code to communicate with the server is written in python language as it offers immense libraries to code RL algorithms. The simulator offers a plethora of sensors and readings. Only the required sensors need to be activated lest it throws too many unwanted values during the race. Once these sensor readings are interpreted, it needs to be fed into the client code to establish control over the car during execution.

4.1 DDPG Algorithm

DDPG is one of the RL algorithms which can be applied on tasks which deal with continuous state spaces. It is an off-policy model-free algorithm. It combines ideas from Deterministic Policy Gradient (DPG) and DQN. It uses experience replay and slow-learning target networks from DQN, and it is based on DPG, which can operate over continuous action spaces. The working of the algorithm is as illustrated in Figure 4.1 and the algorithm pseudo-code is as shown in Algorithm 1

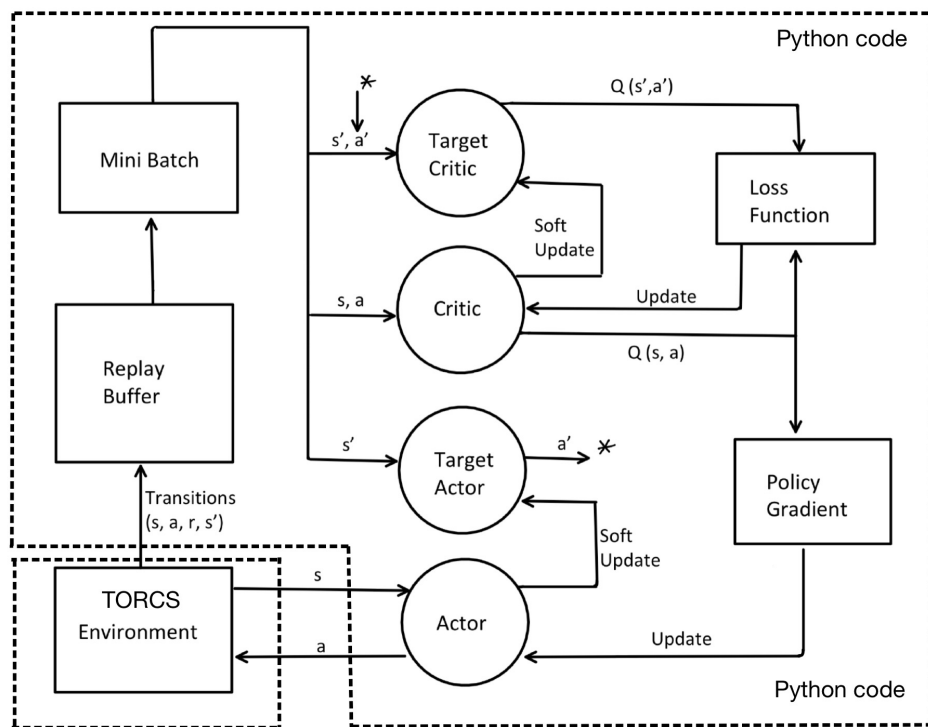


Figure 4.1: Deep Deterministic Policy Gradient block diagram

Initially when the simulator execution is started, the agent is in a new state represented by s . It sends its state to the actor. The actor decides the best possible action a and sends it back to the agent. Since agent cannot

Algorithm 1 DDPG Algorithm

- 1: Randomly initialize critic network $Q(s, a|\theta^Q)$ and actor $\mu(s|\theta^\mu)$ with weights θ^Q and θ^μ
 - 2: Initialize target network Q' and μ' with weights $\theta^{Q'} \leftarrow \theta^Q, \theta^{\mu'} \leftarrow \theta^\mu$
 - 3: Initialize replay buffer R
 - 4: **for** episode = 1, M **do**
 - 5: Initialize a random process N for action exploration
 - 6: Receive initial observation state s
 - 7: **for** t=1, T **do**
 - 8: Select action $a_t = \mu(s_t|\theta^\mu) + N_t$ according to the current policy
 - 9: Execute action a_t and observe reward r_t and observe new state s'_t
 - 10: Store transition (s_t, a_t, r_t, s'_t) in R
 - 11: Sample a random minibatch on N transitions (s_t, a_t, r_t, s'_t) from R
 - 12: Set $y_i = r_i + \gamma Q'(s', \mu'(s'|\theta^{\mu'})|\theta^{Q'})$
 - 13: Update critic by minimizing the loss: $L = \frac{1}{N} \sum_i (y_i - Q(s_i, a_i|\theta^Q))^2$
 - 14: Update the actor policy using the sampled policy gradient:
$$\nabla_{\theta^\mu} J \approx \frac{1}{N} \sum_i \nabla_a Q(s, a|\theta^Q)|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^\mu} \mu(s|\theta^\mu)|_{s_i}$$
 - 15: Update the target networks:
$$\theta^{Q'} \leftarrow \tau \theta^Q + (1 - \tau) \theta^{Q'}$$
$$\theta^{\mu'} \leftarrow \tau \theta^\mu + (1 - \tau) \theta^{\mu'}$$
 - 16: **end for**
 - 17: **end for**
-

decide anything on its own, it follows the actions sent by the actor. After implementing the action, it gets a reward r (which can be anything depending upon how good the action was) and goes to a new state s' . The entire transition comprising of (*state s , action a , reward r , new state s'*) is stored in a replay buffer. From this buffer, random sampling is done and a few transitions are stored in a mini batch. The state s and action a are then sent to the critic to evaluate the action taken by the actor. The critic generates a Q value for the state-action pair and this value shows how good the action was for that particular state. From the mini batch, the next state s' is sent to the target actor. Target actor is expected to give the action a' that is most suitable for the next state s' . This next state s' and next action a' is then sent to target critic. The role of target critic is to evaluate the action a' that was recommended by the target actor for the state s' and generate a Q value for (s', a') pair. Using the Q values for both (s', a') and (s, a) pairs, the loss function is calculated which is used to update the critic network. This helps the critic evaluate actor's actions in a better way. Furthermore, the Q value of (s, a) is used to calculate the policy gradient which is in turn used to update the actor network. This helps the actor to make improved decisions. Target networks are not updated with calculated values. Instead, a soft update is carried out after the actor and critic networks are updated. Soft update involves updating the target network by a very small factor called τ with a typical value being 0.001. Once all these steps are carried out, the agent would have reached a new state and the cycle continues. The actor makes use of neural network to calculate the desirable action a whereas the critic uses neural network to compute Q values [18].

4.1.1 State Space

The state space consists of velocity of the car, tangential angle of the car with respect to road, horizontal distance between the centre of the lane and car,

distance between lane edge and agent, distance between the agent and vehicles ahead, and on the right side of the agent. According to the agent state, the actor network encodes a state to action at time t . This action is sent to the simulator through shared memory to control the car and return the next state, reward for the same. In each iteration, the set $\langle state, action, reward, next_state \rangle$ is collected and is stored into the replay buffer for network training. The velocity vector of the car is divided into three orientations $x, y, z \in \mathbb{R}^3$. The offset $\delta \in \mathbb{R}^1$ is the distance between the agent and centre of the track is normalized between $(-1, 1)$. The distance between the agent and the edge of the lane $\zeta \in \mathbb{R}^{19}$ is measured in $(45, -19, -12, -7, -4, -2.5, -1.7, -1, -0.5, 0.5, 1, 1.7, 2.5, 4, 7, 12, 19, 45)$ degrees with respect to front of the agent. The parameter $\theta \in \mathbb{R}^1$ is angle between direction of the track and direction of the agent and it is measured in radians between $(-\pi, \pi)$. These values are as shown in Figure 4.2 and Table 4.1. DistF and DistR are two variables that define the distance between the agent and obstacles at the front and at the right side respectively. Since 4 sensors spread across the front of the agent simultaneously detect for obstacles ahead upto a distance of 200 metres, DistF consists of a set of 4 values. DistR consists of a set of 14 values with maximum range of 200 metres as 14 sensors, that are located on the right side of the agent, are used to detect obstacles on the right side.

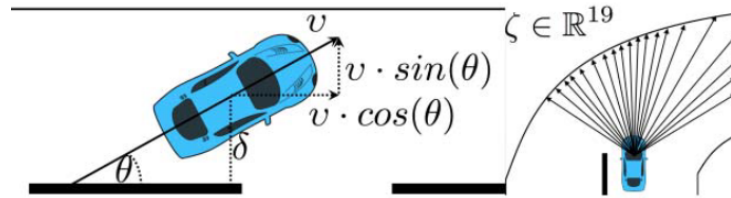


Figure 4.2: Driving Environment [1]

Table 4.1: Table of States [1]

States Symbol	Description	Range
$v(km/h)$	Agent's velocity	$(0, 300)\epsilon R^3$
δ	Offset between centre of the lane and the agent horizontally	$(-1, 1)\epsilon R^1$
$\theta(radian)$	Departure angle between the lane and agent	$(-\pi, \pi)\epsilon R^1$
$\zeta(m)$	Distance between edge of the lane and agent	$(0, 200)\epsilon R^{19}$
$DistF(m)$	Distance between the agent and obstacle(s) ahead of it	$(0, 200)\epsilon R^4$
$DistR(m)$	Distance between the agent and obstacle(s) on the right side	$(0, 200)\epsilon R^{14}$

4.1.2 Action Space

Agent action space $a_t = (accelerate, brake, steer)$ represents agent acceleration, braking, and steering angle respectively as illustrated in Table 4.2 in detail. Acceleration value of 1 implies that the agent has applied full throttle i.e. in human driving terminology, it has pressed the accelerator pedal completely. If acceleration value is 0, it implies that the agent has completely let go of the accelerator pedal and it is moving only due to its momentum. Brake value of 0 states that the brake pedal is not at all pressed and a value of 1 implies that it is completely pressed. Steering value of -1 represents full right and +1 represents full left turn.

Table 4.2: Table of Actions [1]

Action Symbol	Description	Range
accelerate	Current acceleration	$(0, 1) \in \mathbb{R}^1$
brake	Current braking	$(0, 1) \in \mathbb{R}^1$
steer	Current steering angle: negative for right, positive for left	$(-1, 1) \in \mathbb{R}^1$

4.1.3 Reward Function

One of the major tasks is designing an appropriate reward function. This function needs to be properly adjusted so that the car knows its priorities well and drives in a controlled manner. To briefly explain a scenario, consider an autonomous car X which is steadily going in its lane. It is getting some positive reward. Suppose X detects some other car, named Y, which is seemingly rushing towards X and it might end up crashing onto the car X, then ego-car (X) must receive higher rewards if it decides to take an action of applying brake and maneuvering it away from Y. In order to achieve the objective of lane keep, the car needs to be trained with altered reward function wherein it must get a higher reward for staying in its lane and a lesser reward for swerving away from the required lane. Another possible approach here is to make use of the sensors to limit the car in one lane thereby creating a pseudo boundary for the car.

The current reward function is $R = v\cos(\theta) - v\sin(\theta) - v|\delta|$ where θ is the departure angle, v is the speed of the agent and δ is the offset between agent and centre of lane as shown in Figure 4.2. If the departure angle is less, then $\cos(\theta)$ will be high while $\sin(\theta)$ will be a minute value. If the agent is near the centre of the lane, only a small value will be subtracted. In case it is near the edge of the lane, a higher value will be subtracted from the reward function. The second part of the work involves detection of obstacle using sensors. Two

different sensors are used for this process, namely *opponents* and *track*. The *track* sensor is as represented by the ζ value which is shown in Table 4.1. It consists of 19 range-finder sensors which contain the distance between the centre of the car and edge of the lane. These sensors are stored in a python list in the implementation. The 36 range-finder values given by the *opponents* sensor are also stored in a python-list. These are located throughout the car at every 10 degrees and they have a maximum range of 200 metres. In case no other cars are present, this sensor outputs the maximum value i.e. 200 metres from each of the sensors. Otherwise it sends the distance between the agent and other cars present on track.

4.2 Implementation

4.2.1 Lane Keep

This is the primary objective of this project. Here, *trackPos* sensor is used to implement the lane keep operation. Whenever the agent is in a new state, the *trackPos* sensor outputs a value which defines the relative position of the agent on the track. It is represented by the greek symbol δ .

In case the agent is driving on the left edge of the left lane, *trackPos* sensor outputs a value of +1 and similarly if it is on the right edge of the left lane, the sensor outputs a value of -1. If the car needs to be at the centre of the left lane, it must be constantly maintaining a value of around 0. Of course, it cannot maintain driving at the centre of the lane at all times. Owing to some tight corners, it does deviate from the centre a bit.

There are two methods implemented in this work to enable lane keep. The first one is a part of the reward function. The third term in the reward function is $v|\delta|$. If the agent is driving at the centre of the lane, δ will be almost close to 0. In case it has deviated a lot from the centre, $v|\delta|$ a considerably large value,

will be subtracted from the reward function. When the reward has reduced, the agent learns that it is not the optimal path to follow and tries to increase the same by diligently following the lane in the future. The first method is very efficient in teaching the agent to drive in the centre of its lane. However, the second approach is also implemented to leave no room for error. In this method, a pseudo boundary is created between $+0.2$ and -0.2 corresponding to the *trackPos* sensor. If the agent's *trackPos* value is greater than $+0.2$, it means that the agent is steering left from the centre of the lane. Hence the *steer* value is negated to enable the agent go steer back to the centre of the lane. If the value is lesser than -0.2 , it signifies that the agent is steering right from the centre of the lane. Just like it was done earlier, the *steer* value is negated so that the agent can steer back to the centre of the lane.

4.2.2 Front obstacle detection

Though the simulator gives the freedom of implementing both Left Hand Drive(LHD) and Right Hand Drive(RHD), the former driving style was chosen as it is more common around the world. Four sensors out of the 36 *opponents* sensors are used for front obstacle detection. Whenever the car enters a new state, it waits for a few millisecond to receive instructions from actor network on how to proceed. During this time interval, all the sensor readings are collected and analysed to see if any obstacles are present ahead of the vehicle. If any obstacle is detected beyond 50 metres, it is not required to apply brakes as there is ample distance between the agent and the obstacle. In case front *opponents* sensors detect a car within a range of 30-50 metres ahead, the agent stops accelerating i.e. directly sets its value to 0. Setting the value directly to zero is analogous to taking the foot off the accelerator pedal in real world driving scenario. At this point the agent keeps moving due to its momentum and gradually slows down due to friction. After a few seconds, if the obstacle

ahead is less than 30 metres, it applies brakes slightly i.e. it sets the brake value to 0.3 as this is the ideal value to gradually slow down the vehicle. If the value is greater than 0.3, the braking action will involve jerky motion but if it is less than 0.3, it may not slow down at the desired rate. If the obstacle is less than 20 metres away, it applies brakes a tad bit stronger with a value of 0.7. At this value, the vehicle reduces the speed quickly as the obstacle is not very far and if it does not apply brakes with this value, the agent might end up very close to the obstacle with a considerably higher speed after which collision is inevitable. In case the obstacle is less than just 10 metres away, it applies full brakes i.e. sets brake value to 1, so that it comes to a complete halt. If the obstacle is detected early enough, braking action is done in a step by step manner over a distance of 50 metres and it smoothly comes to a halt. In case the obstacle suddenly comes in the path of the agent with less distance between them, then the agent applies full brake inorder to avoid collision. In such a scenario, braking will not be smooth but that is secondary since it is more important to avoid collision.

4.2.3 Overtaking

This is interconnected with front obstacle detection. When driving in a lane and any obstacle appears suddenly, the only thing that comes to the mind of any driver is to apply the brakes and stop swiftly and that is exactly what the agent does too. On the other hand, if an obstacle is detected well in advance, the agent can plan to overtake the same. When the distance between the agent and the vehicle ahead is around 20 metres, agent takes another set of *opponents* sensor readings to check if there are any vehicles that are currently on the right lane. It uses the same sensor to detect the cars on the right lane that it used to detect the cars or obstacle ahead of it. If it is clear of any traffic, it steers to the right lane. After overtaking a vehicle, it checks if it can

come back to its previous lane. If the path is again clear, it steers back to its lane and continues driving.

The agent's speed is limited to 90 kilometre per hour. While overtaking, imagine a scenario where the agent has completed half of the overtaking process by going to the right lane. At this point, if the vehicle which was being overtaken, suddenly increases its speed and if this is equal to or more than the speed of the agent, the agent decides to abort the overtaking procedure. The agent cannot measure the speed of other vehicles, instead it knows that other vehicle's speed has increased when it senses that it is driving paralelly at its maximum speed with the vehicle that it just tried to overtake. At such a scenario, it slows down a little by setting the acceleration value to zero and checks if it can go back to its previous lane. If the road is clear, it steers left and follows the car which it had intended to overtake.

4.2.4 Side obstacle detection

To detect the obstacles at the sides of the agent, a total of 28 sensors are utilized. If all the vehicles that are plying over a stretch of road meticulously follow their lane, then agent takes no action. However if any vehicle deviates from its lane and enters the lane that belongs to the agent, at that time the problems arise. If the agent senses any vehicle approaching very close to it from sides, it slightly steers away to avoid collision. It moves away from the approaching vehicle only if there is space left on the road. In case it was already driving on the edge of the road and if some vehicle approaches from sides, then it slows down letting the other vehicle pass. If such a situation occurs in the real world, drivers first press the horns of their vehicle to alert the driver who is deviating from their lane. Since the simulator does not have that privilege, it simply slows down as that is the best the agent can do to avoid a sideward collision.

Chapter 5

Result Analysis

Initially TORCS-1.3.7 was installed in the system along with all the python modules required to code RL algorithms. Then, a basic python client code is written to communicate with the TORCS server which enables the user establish control over the car. Controlling includes the speed variation, gear selection and steering control. Figures 5.1, 5.2, 5.3, 5.4 and 5.5 are the screen-shots obtained during the execution of the program.

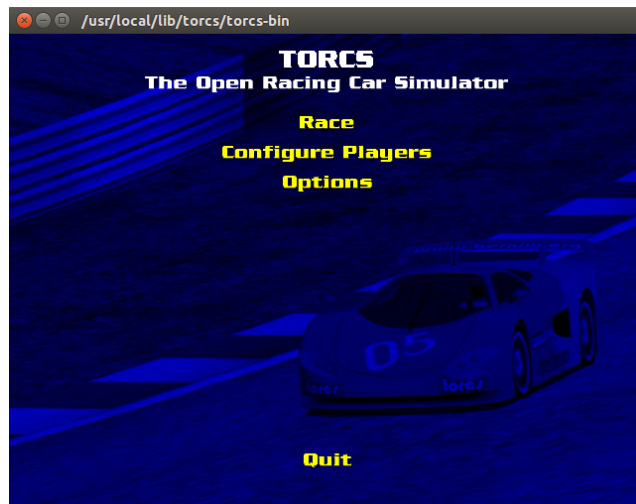


Figure 5.1: TORCS main screen



Figure 5.2: Track selection screen



Figure 5.3: Agent driving on the track



Figure 5.4: Agent goes off-track



Figure 5.5: Agent slowly getting back on track

Figure 5.1 shows the main menu of TORCS simulator . Figure 5.2 shows one of the numerous tracks which is available for selection. Figure 5.3 portrays a scenario where the agent is driving on the track. Since the agent is not perfectly trained, it sometimes deviates completely off the track as shown in Figure 5.4. However, as soon as it goes out of the track, its rewards begins to diminish. Hence it soon finds its way back to the track as shown in Figure 5.5.

After the initial training comprising of around 8000 iterations, the agent had learnt to drive on the track but it was not at all moving in a straight

line. It was swerving abruptly in its lane. As a solution to this, the learning rate of the actor and critic was altered from 0.001 and 0.002 to 0.0005 and 0.001 respectively to give more time for the agent to learn. This resulted in the agent taking around 14000 iterations to completely re-learn driving from scratch. This is visualized in Figure 5.6. Reward obtained in each episode of training is plotted against the episode number.

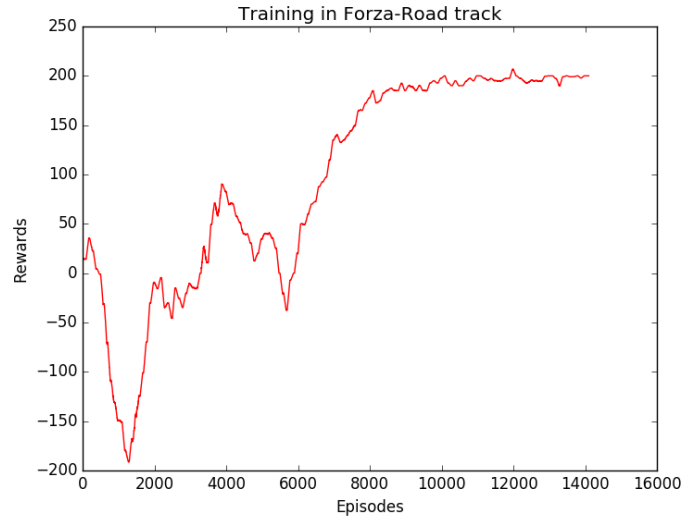


Figure 5.6: Training of the agent

Figure 5.7 shows the variation of rewards with respect to speed(speedX), distance from lane centre(trackPos) and angle between lane and car(theta value). At instances where the speed is less, the rewards follow suit. At times, there are instances where in spite of having sufficiently good speed, the rewards are less. This is due to the fact that the other parameters collectively bring down the rewards as the vehicle may have deviated from the centre of the lane and may have not aligned itself properly with respect to the direction of the lane. In case the rewards are higher than the speed, it shows that the agent has received some extra reward for avoiding some form of probable collision.



Figure 5.7: Rewards vs speed, angle, and position (track 1)

Figure 5.8 further explains the dependency of speed, angle and distance from the centre for rewards. Despite the agent aligning itself properly on the track, it swerved left and right owing to high-speed cornering.

On careful observation of Figure 5.9, it can be seen that trackPos(distance of the car from the centre of the lane) is very large. In this case, it is larger than +1. This easily translates into the fact that the agent was not driving on the road but beside it. This scenario is created specifically for testing how the rewards would look if the agent is made to not drive on the road.

Before the agent was trained with an improved learning rate, it was moving in a zig-zag manner in spite of maintaining its lane. Figure 5.10 shows how the reward varies if the agent isnt properly trained to drive itself smoothly.

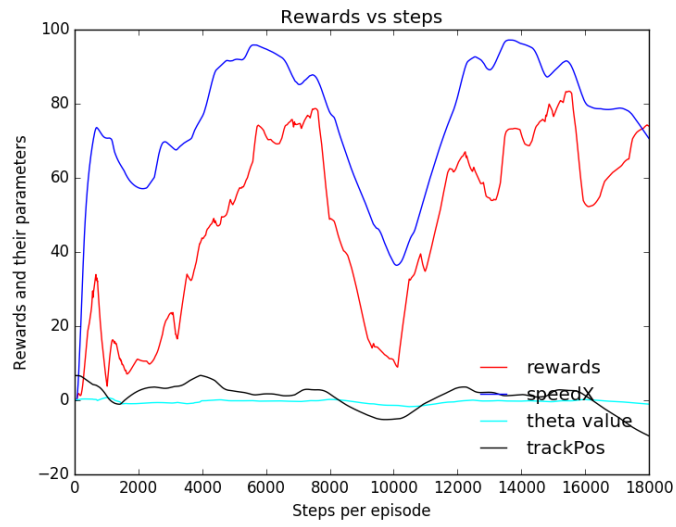


Figure 5.8: Rewards vs speed, angle, and position (track 2)

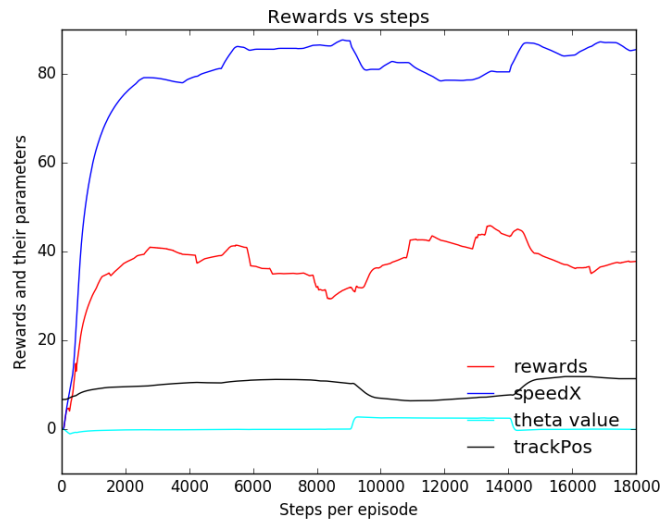


Figure 5.9: Rewards vs speed, angle, and position (track 3)

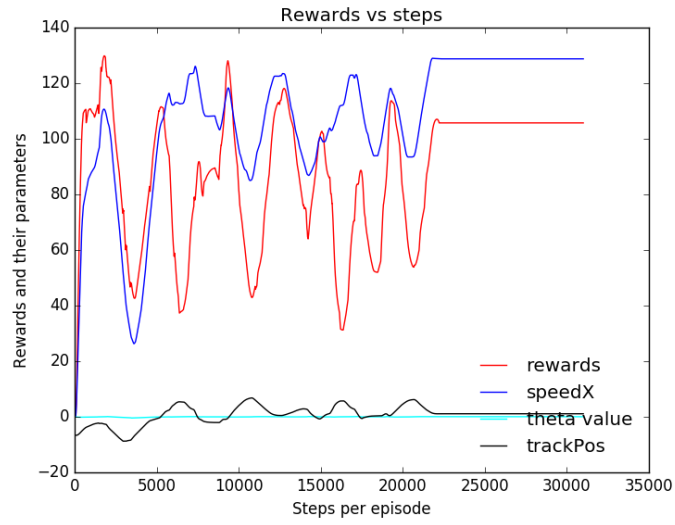


Figure 5.10: Rewards vs speed, angle, and position (track 4)

In all the simulations, the agent successfully managed to drive itself smoothly on the track by constantly interacting with the dynamic environment. It has learnt to detect the twists and turns in the track and applies brakes accordingly. It efficiently detects the obstacles and manages to take precautions or abrupt decisions to avoid collisions. Since all these cannot be depicted here, it is shown in the form of a video available through this link: [Result Video Link](#)

Chapter 6

Conclusion and Future Scope

6.1 Conclusion

Today, self driving cars is the need of the hour because of the increasing traffic on the roads. Having self driven cars with highest possible efficiency and reliability are required very much for safe travelling. Even though many researchers work on this topic by making use of AI/ML tools, still 100% efficiency has not been achieved. In this work, one of the Reinforcement Learning algorithms, DDPG is used for controlled driving of a car. In the second phase, object detection and collision avoidance is achieved with the help of various sensors present on the car. Thus, the agent can handle situations like front, side obstacle detection, overtaking and can maneuver itself to avoid collision.

Vehicles have become an integral part of our life. Just a few decades ago, autonomous driving was just a distant dream. Over the past few years, there has been such a tremendous improvement in technology which is enabling us to bring such dreams into reality.

Elderly people, patients on medication, persons with disabilities who will have to depend on other persons for mobility can now move around independently. In addition, busy executives can attend to their work undisturbed while travelling and this saves their time. With more and more autonomous

vehicles on the road, it can be hoped that traffic accidents can be reduced to a minimum since all these vehicles can be designed to strictly follow the traffic rules.

6.2 Future Scope

This project does not take into account the changes in weather conditions, illumination levels etc. There can be some variations in the sensor values if one or more of these change and it can hamper the performance. It needs to be checked if all these sensors give accurate values after changes in weather and light. If not, it needs to be calibrated accordingly.

Since the simulator has no provision to include traffic signals, the same could not be implemented. Hence training the agent to sense the signals and act accordingly is a major task for the future.

The agent heavily relies on lane markings to detect lanes. In the real world, if the lane markings are erased off in certain areas or if the lane markings are not at all there, it struggles to maintain its lane and drive efficiently. In the future, it can be trained to use a camera to detect the edge of the road and perform seamless driving.

References

- [1] Z. Huang, J. Zhang, R. Tian, and Y. Zhang, “End-to-end autonomous driving decision based on deep reinforcement learning,” in *2019 5th International Conference on Control, Automation and Robotics (ICCAR)*. IEEE, 2019, pp. 658–662.
- [2] B. R. Kiran, I. Sobh, V. Talpaert, P. Mannion, A. A. Al Sallab, S. Yogamani, and P. Pérez, “Deep reinforcement learning for autonomous driving: A survey,” *IEEE Transactions on Intelligent Transportation Systems*, 2021.
- [3] J. Peng, E. C. Jury, P. Dönnies, and C. Ciurtin, “Machine learning techniques for personalised medicine approaches in immune-mediated chronic inflammatory diseases: Applications and challenges,” *Frontiers in Pharmacology*, p. 2667, 2021.
- [4] R. Naveen Raj, A. Nayak, and M. S. Kumar, “A survey and performance evaluation of reinforcement learning based spectrum aware routing in cognitive radio ad hoc networks,” *International Journal of Wireless Information Networks*, vol. 27, no. 1, pp. 144–163, 2020.
- [5] A. LaFrance. (2016) Your grandmother’s driverless car. [Online]. Available: <https://www.theatlantic.com/technology/archive/2016/06/beep-beep/489029/>
- [6] T. Fredriksson, D. I. Mattos, J. Bosch, and H. H. Olsson, “Data labeling: an empirical investigation into industrial challenges and mitigation strate-

- gies,” in *International Conference on Product-Focused Software Process Improvement*. Springer, 2020, pp. 202–216.
- [7] M. Rahmati, M. Nadeem, V. Sadhu, and D. Pompili, “Uw-marl: Multi-agent reinforcement learning for underwater adaptive sampling using autonomous vehicles,” in *Proceedings of the International Conference on Underwater Networks & Systems*, 2019, pp. 1–5.
- [8] J. Zhao, T. Qu, and F. Xu, “A deep reinforcement learning approach for autonomous highway driving,” *IFAC-PapersOnLine*, vol. 53, no. 5, pp. 542–546, 2020.
- [9] Y. Zhang, P. Sun, Y. Yin, L. Lin, and X. Wang, “Human-like autonomous vehicle speed control by deep reinforcement learning with double q-learning,” in *2018 IEEE Intelligent Vehicles Symposium (IV)*. IEEE, 2018, pp. 1251–1256.
- [10] R. Chopra and S. S. Roy, “End-to-end reinforcement learning for self-driving car,” in *Advanced computing and intelligent engineering*. Springer, 2020, pp. 53–61.
- [11] B. B. Elallid, N. Benamar, A. S. Hafid, T. Rachidi, and N. Mrani, “A comprehensive survey on the application of deep and reinforcement learning approaches in autonomous driving,” *Journal of King Saud University-Computer and Information Sciences*, 2022.
- [12] R. Hussain and S. Zeadally, “Autonomous cars: Research results, issues, and future challenges,” *IEEE Communications Surveys & Tutorials*, vol. 21, no. 2, pp. 1275–1313, 2018.
- [13] M. Zhu, Y. Wang, Z. Pu, J. Hu, X. Wang, and R. Ke, “Safe, efficient, and comfortable velocity control based on reinforcement learning for au-

- onomous driving,” *Transportation Research Part C: Emerging Technologies*, vol. 117, p. 102662, 2020.
- [14] D. Omeiza, H. Webb, M. Jirotko, and L. Kunze, “Explanations in autonomous driving: A survey,” *IEEE Transactions on Intelligent Transportation Systems*, 2021.
 - [15] S. Grigorescu, B. Trasnea, T. Cocias, and G. Macesanu, “A survey of deep learning techniques for autonomous driving,” *Journal of Field Robotics*, vol. 37, no. 3, pp. 362–386, 2020.
 - [16] Y. Ma, Z. Wang, H. Yang, and L. Yang, “Artificial intelligence applications in the development of autonomous vehicles: a survey,” *IEEE/CAA Journal of Automatica Sinica*, vol. 7, no. 2, pp. 315–329, 2020.
 - [17] Z. Cao, S. Xu, X. Jiao, H. Peng, and D. Yang, “Trustworthy safety improvement for autonomous driving using reinforcement learning,” *Transportation research part C: emerging technologies*, vol. 138, p. 103656, 2022.
 - [18] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, “Continuous control with deep reinforcement learning,” *arXiv preprint arXiv:1509.02971*, 2015.

Table 6.1: Project Details

Student Details

Student Name	Ashwin S H		
Registration Number	180953352	Section/Roll No.	A/70
Email Address	ashwinsh22@gmail.com	Phone No.(M)	8618287408

Project Details

Project Title	Deep Reinforcement Learning for Autonomous Driving Strategy		
Project Duration	4 Months	Date of Reporting	10-01-2022

Organization Details

Organization Name	Department of Information and Communication Technology		
Full Postal Address	Manipal Institute of Technology, Manipal - 576104		
Website Address	www.manipal.edu		

Internal Guide Details

Faculty Name	Dr. Rashmi Naveen Raj		
Full Contact Address with PIN Code	Department of Information and Communication Technology, Manipal Institute of Technology, Manipal-576104		
Email Address	rashmi.naveen@manipal.edu		

Report

ORIGINALITY REPORT

3%

SIMILARITY INDEX

1%

INTERNET SOURCES

2%

PUBLICATIONS

0%

STUDENT PAPERS

PRIMARY SOURCES

- | | | |
|-------|--|------|
| 1 | <p>Zhiqing Huang, Ji Zhang, Rui Tian, Yanxin Zhang. "End-to-End Autonomous Driving Decision Based on Deep Reinforcement Learning", 2019 5th International Conference on Control, Automation and Robotics (ICCAR), 2019</p> <p>Publication</p> | 1 % |
| <hr/> | | |
| 2 | <p>huggingface.co</p> <p>Internet Source</p> | 1 % |
| <hr/> | | |
| 3 | <p>Akhil Hannegudda Ganesh, Bin Xu. "A review of reinforcement learning based energy management systems for electrified powertrains: Progress, challenge, and potential solution", Renewable and Sustainable Energy Reviews, 2022</p> <p>Publication</p> | <1 % |
| <hr/> | | |
| 4 | <p>mafiadoc.com</p> <p>Internet Source</p> | <1 % |
| <hr/> | | |
| 5 | <p>Yali Yuan, Robert Tasik, Sripriya Srikant Adhatarao, Yachao Yuan, Zheli Liu, Xiaoming Fu. "RACE: Reinforced Cooperative</p> | <1 % |

Autonomous Vehicle Collision Avoidance", IEEE Transactions on Vehicular Technology, 2020

Publication

6

deepai.org

Internet Source

<1 %

7

nbn-resolving.de

Internet Source

<1 %

8

Hao Li, Siddharth Misra. "Reinforcement
learning based automated history matching
for improved hydrocarbon production
forecast", Applied Energy, 2020

Publication

<1 %

Exclude quotes On

Exclude matches < 3 words

Exclude bibliography On