Artificial agent for car game using Reinforcement Learning.

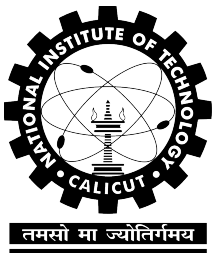
CS4099D Project
End Semester Report

Submitted by

Ankit Meshram (B190688CS)
Rahul Bamniya (B190395CS)
L Ravikiran (B190827CS)

Under the Guidance of

Dr. A Sudarshan Chakravarthy

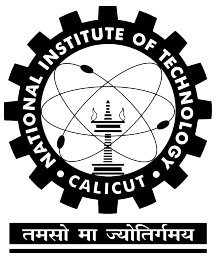


Department of Computer Science and Engineering
National Institute of Technology Calicut
Calicut, Kerala, India - 673 601

May 2023

NATIONAL INSTITUTE OF TECHNOLOGY CALICUT
KERALA, INDIA - 673 601

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING



CERTIFICATE

Certified that this is a bonafide report of the project work titled

ARTIFICIAL AGENT FOR CAR GAME USING REINFORCEMENT LEARNING

done by

Ankit Meshram
Rahul Bamniya
L Ravikiran

of Eighth Semester B. Tech, during the Winter Semester 2020-'21, in partial fulfillment of the requirements for the award of the degree of Bachelor of Technology in Computer Science and Engineering of the National Institute of Technology, Calicut.

(Dr. A Sudarshan Chakravarthy)
(Assistant Professor)
Project Guide

12-05-2023
Date

DECLARATION

I hereby declare that the project titled, **Artificial agent for car game using Reinforcement Learning**, is our own work and that, to the best of our knowledge and belief, it contains no material previously published or written by another person nor material which has been accepted for the award of any other degree or diploma of the university or any other institute of higher learning, except where due acknowledgement and reference has been made in the text.

Place : NIT Calicut
Date : 12-05-2023

Name : Ankit Meshram
Roll. No. : B190688CS
Name : Rahul Bamniya
Roll. No. : B190395CS
Name : L Ravikiran
Roll. No. : B190827CS

Abstract

Deep reinforcement learning has opened up new opportunities for achieving tasks that require intricate navigation and control. One such task is the development of autonomous self-driving cars. To design an effective self-driving car, it is essential to create an agent that can automatically navigate and control a car. This can be accomplished by building an agent that can play a car racing game. We have designed a deep reinforcement learning agent that is capable of playing a car racing game. While existing solutions have been successful in specific routes, our objective is to create an agent that can effectively navigate and control a car on all types of tracks. To achieve this, we have utilized deep reinforcement learning techniques, which enable the agent to learn and improve its behavior through trial and error. Our approach involves training the agent using a large data set of racing scenarios, and we have designed a reward system that encourages the agent to make decisions that lead to the successful completion of the race.

ACKNOWLEDGEMENT

We would like to express our sincere and heartfelt gratitude to our guide and mentor **Dr. A Sudarshan Chakravarthy**, who have guided us throughout the course of the final year project. Without their active guidance, help, cooperation and encouragement, we would not have made headway in the project. We would like to thank our parents and the faculty members for motivating us and being supportive throughout our work. We also take this opportunity to thank our friends who have cooperated with us throughout the course of the project.

Contents

List of Figures

Chapter 1

Introduction

A car-driving agent can analyze the environment without human interaction and make decisions accordingly. A number of sensors are connected and used to detect road and path signals from the environment. Deep reinforcement resulted in new capabilities for complex navigation and control tasks.

Self-driving agents make decisions and communicate with each other. Autonomous cars combine different technologies to sense their environment using sensors. Sophisticated control systems interpret sensory information to identify appropriate navigational paths and appropriate signs. Potential The benefits of autonomous cars include increased safety, increased mobility, increased customer satisfaction, and fewer accidents.

Chapter 2

Problem Statement

The aim of this project is to create an artificial agent that can successfully play a racing game on different tracks using deep reinforcement learning. The main challenge of this task is that the agent must navigate through terrain with varying conditions which can make it difficult to maintain control and achieve high performance. Moreover, the agent must learn quickly and effectively in this complex environment.

To address these challenges, the project will leverage deep reinforcement learning techniques. This involves training a neural network to make decisions based on input data, such as the agent's current state and the environment's feedback. By doing so, the project aims to create an agent that can learn effectively and perform well on different kinds of race tracks.

The successful development of such an agent could have significant implications for various applications, including autonomous vehicles, robotics, and gaming. By creating an agent that can effectively navigate through complex environments, we could advance the capabilities of artificial intelligence and enable it to solve a wide range of real-world problems.

Chapter 3

Literature Review

3.1 Reinforcement Learning

In order to develop the automated car agent, many problems were faced. Achieving autonomous agents is a very complicated task.[3] There are multiple practices in machine learning to train agents to learn and act. The first one is supervised learning,[3] In which the dataset contains the data and the ground truth labels. Agents will try to predict true labels. The second one is unsupervised learning. Instead of having labels, this time only data is provided and the agents need to correlate and group the data together. The last one, which is our method, reinforcement learning, creates its own data by interacting with the environment. This is the best-suited approach since most of the time, there will not be any ground truth actions for agents to learn.[3]

3.2 Virtual Simulators

3.2.1 TORCS (The Open Racing Car Simulator)

Since driving simulations are fairly important before real-life autonomous implementations, there are multiple driving-racing simulations for testing purposes.[1] The Open Racing Car Simulation (TORCS) is a highly portable open-source car racing -self-driving- simulation. While it can be used as a game in which human players compete with scripted agents, TORCS provides an observation and action API to develop an artificial intelligence agent.[1] We are primarily concentrating on the issue of a car agent that operates effectively and efficiently on uneven tracks.

3.2.2 CARLA (Car Learning to Act)

CARLA is an accessible driving simulator for cities. To facilitate training, development, and validation of autonomous driving models, encompassing both perception and control, CARLA has been built from the ground up. It is intended to operate as a server-client system, with the server handling both simulation and scene rendering. The client API, which is built in Python, is in charge of facilitating communication across sockets between the autonomous agent and the server. The client communicates with the server via instructions and meta-commands while also receiving sensor readings. The vehicle is controlled by commands, which also include steering, accelerating, and braking.[8]

The agent's sensor suite can be configured in a flexible manner thanks to CARLA. The customer can choose the number of cameras, their type, and their placement, and it has RGB cameras and pseudo-sensors. The field of view, depth of field, 3D position, and 3D orientation to the car's coordinate system is all camera settings. Unlabeled, building, fence, other,

pedestrian, pole road line, road, sidewalk, vegetation, wall, traffic sign, sky, ground, bridge, rail track, guardrail, traffic light, static, dynamic, water, and landscape are only a few of the 22 semantic classes offered by the pseudo-sensor. A variety of metrics related to the agent's condition and adherence to traffic laws are provided by CARLA. Vehicle location and orientation about the global coordinate system, speed, acceleration vector, and cumulative impact from collisions are all measurements of the agent's state. It also provides a collision detector, lane invasion detector, obstacle detector, radar, and RSS[8].

3.2.3 Air Sim (Aerial Informatics and Robotics Simulation)

It is an open-source, cross-platform AI research platform built on Epic Games' Unreal Engine 4. It simulates drones, ground vehicles like cars, and several other items. It was created by Microsoft and may be used to test out algorithms for deep learning, computer vision, and reinforcement learning in autonomous vehicles. This makes it possible to test autonomous solutions without worrying about potential harm to the real world. Numerous programming languages can be used to access the APIs.

3.2.4 OpenAI Gym

The gym is a toolkit for developing and comparing reinforcement learning algorithms. It makes no presumptions about the agent's structure. For each track tile that is visited, the payout is -0.1 for each frame and +1000/N, where N is the total number of track tiles. When all tiles have been visited, the episode is over. The state RGB buffer and a few indications are displayed at the bottom of the window. Actual speed, four ABS sensors, steering wheel position, and gyroscope are listed from left to right.[8]

3.3 Improving training performance by Maximilian Jaritz and his team

According to Maximilian Jaritz and his team, There was slow development in the field of self-driving agents for the car game. There has been a number of deep learning approaches to solve end-to-end control (aka "behavioral reflex") for games, but still, very few were applied to end-to-end Driving. They propose a method from recent asynchronous learning and building on their preliminary work to train an end-to-end agent in World Rally Championship 6 (WRC6), a realistic car racing game with stochastic behavior (animations, light).[5] To improve car training, they used the actor-critic asynchronous perk to train the car. At each time step, the algorithm receives the game state, performs actions (acceleration and steering), and receives a reward as a supervisory signal.[5] Compared to prior research, they learned the full control (lateral and longitudinal) including hand brake for drifts, in a completely self-supervised manner.[5]

3.4 Tackle the problem of learning the optimal racing line by Emilio Capo

According to a team led by Emilio Capo Previous work, mainly focused on a low-level input design shows that artificial agents are able to learn to stay on track starting from no driving knowledge; however, the final performance is still far from those of competitive driving.[6] In their work, they used deep reinforcement learning to develop a series of artificial agents using The Open Racing Car Simulator (TORCS) and the algorithm Deep Deterministic Policy Gradients (DDPG). They develop artificial agents, considering both numerical and visual inputs, based on deep neural networks. The agents learn to compute a target point for the vehicle - or, additionally, a correction to

the maximum target speed in the current position - which is then turned into actionable commands by custom low-level logic.[6] Their results show that our approach is able to achieve a fair performance, though extremely sensitive to low-level logic. In their research, there is still a margin for improvement to be explored in order to understand how to fully exploit a high-level control design.[6]

3.5 Optimal Deep Reinforcement Learning Agents for TORCS by Kivanc Guckiran

An international team of researchers led by Turkish computer scientist Kivanc Guckiran has published a study on Deep Reinforcement Learning Agents for TORCS environments at the National University of Defense Technology (NUN) in Gothenburg, Sweden. They have used: TORCS, Rainbow DQN, and Soft actor-critic. TORCS provides an API for AI agents to act and learn from. They have used Rainbow DQN in a car racing game agent by first training the agent using a variety of different DQN algorithms. Soft actor-critic is a reinforcement learning algorithm that combines aspects of both Q-learning and policy gradient methods. They have had remarkable success with the implementation of two alternative TORCS algorithms. It has been found that agents generalize effectively on hidden tracks.[8]

3.6 Two-Dimensional Car Racing Environment with Continuous Action Space by Henry Teigar

The authors proposed a method for training an agent to play 2D car racing games using deep reinforcement learning. They used a similar approach to

ours, which involves using a CNN to extract features from the game environment and a policy network to output actions based on those features.

The authors found that their approach was able to learn effective policies for playing the game and outperformed some other methods. They also conducted experiments to evaluate the effect of different hyperparameters on the training process and found that the choice of hyperparameters can have a significant impact on the performance of the agent.

Overall, this provides valuable insights into the use of deep reinforcement learning for playing 2D car racing games and highlights the importance of carefully choosing hyperparameters for achieving good performance.

Chapter 4

Motivation for using Reinforcement Learning in Self-Driving Car Agent

Prior to this work, we had worked on Self Driving Cars using supervised learning. The following work is done by us using Supervised Learning:-

4.1 Work Done for Self-Driving Car with Supervised Learning

4.1.1 Map of Environment

For training our agent, we used the Udacity self-driving car simulator. This simulator provides a 3D environment in which the agent can learn to drive a car in a more realistic and complex setting. The simulator provided a realistic environment for training the agent, allowing it to learn how to make decisions based on real-world scenarios.

Figure 4.1: 3D Map to Train the Model

4.1.2 State Space

The state space of the environment for the Udacity self-driving car simulator includes various components that represent the current state of the car and its surroundings. These components include:

Position and Orientation: The position of the car in the simulator environment, represented by x, y, and z coordinates, as well as the orientation of the car, represented by roll, pitch, and yaw.

Velocity: The current speed and direction of the car, represented by the linear velocity in x, y, and z directions, as well as the angular velocity in roll, pitch, and yaw directions.

Sensor Data: Information from various sensors on the car, including lidar, radar, and camera data, which provide information about the distance, speed, and position of nearby objects.

Figure 4.2: Car in 3D Environment

Road Information: Information about the road, such as the position and orientation of lanes, road markings, and other objects.

4.1.3 Action Space

The action space for the Udacity self-driving car simulator is the set of all possible actions that the car can take in response to the current state of the environment. The action space consists of several discrete and continuous actions that can be performed by the car's control system.

The discrete actions that can be performed include accelerating or decelerating, and turning. The continuous actions that can be performed by the car include adjusting the steering angle, throttle, and brake inputs. The steering angle can be adjusted to control the direction of the car, while the

throttle and brake inputs can be adjusted to control the car's speed.

The actions that are taken by the self-driving car algorithm are determined based on the current state of the environment and the goal of the algorithm.

4.1.4 Observation Space

The observation space for the Udacity self-driving car simulator also includes images captured by the car's front, left, and right cameras. The center image provides a forward-facing view of the road ahead, while the left and right images provide views of the adjacent lanes. These images can be used to detect the presence of road markings.

Figure 4.3: Center Observation

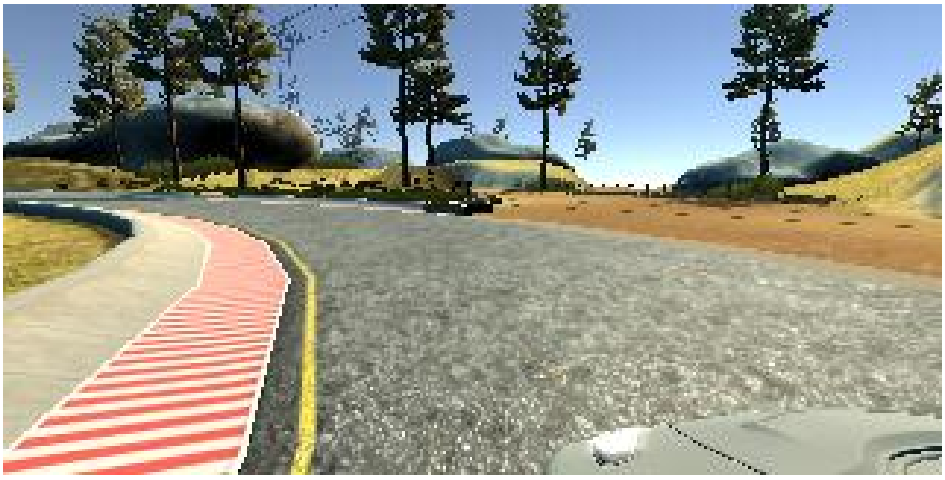


Figure 4.4: Left Observation

Figure 4.5: Right Observation

4.1.5 Training Phase of Agent with Supervised Learning

The images and driving log generated from the Udacity self-driving car simulator are used to train the agent to predict the appropriate steering angle, throttle, brake, and other actions based on the current state of the environment.

In the Udacity self-driving car simulator, the data collected during driving often contains many instances where the steering angle is 0.0, which indicates that the car is driving straight. While this is a common occurrence during driving, it can cause an imbalance in the data, as there may be significantly more examples of driving straight than of turning or changing lanes.

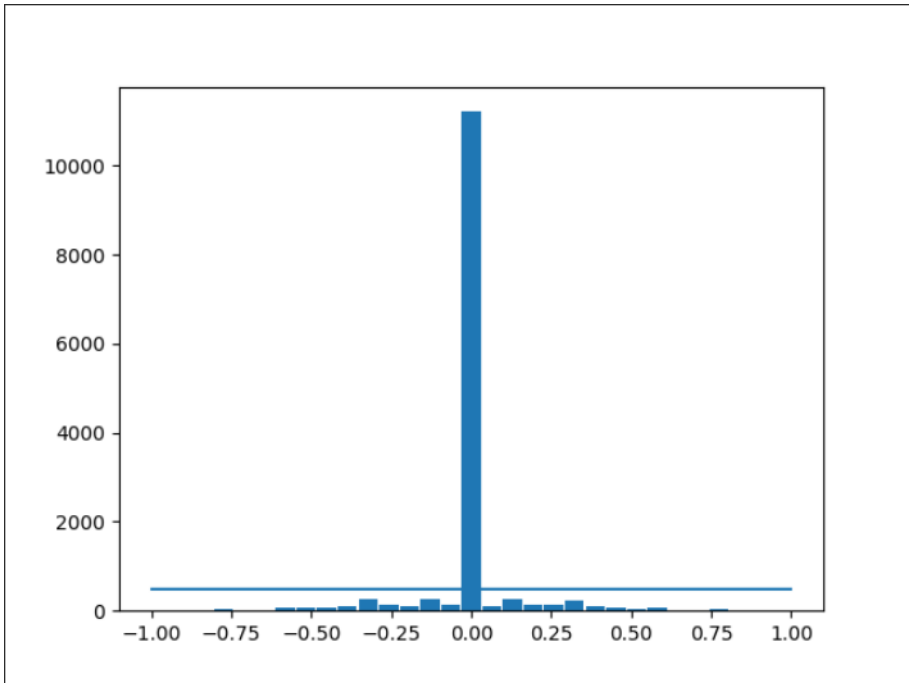


Figure 4.6: Data Before Balancing

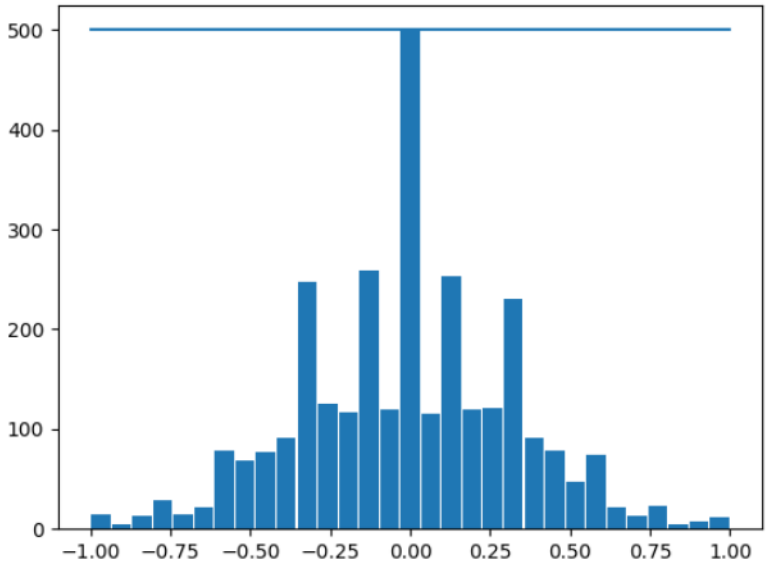


Figure 4.7: Data After Balancing

To address this issue, it is common practice to balance the data by removing some of the instances where the steering angle is 0.0. This can be done by randomly selecting a subset of the data where the steering angle is 0.0 and removing it from the dataset. While removing some of the instances where the steering angle is 0.0 can help balance the data.

4.1.6 Image Augmentation and Pre-processing

We used data augmentation to increase the examples of training. Data augmentation is a technique used to increase the number of training examples by generating new examples from the existing data set. This is achieved by applying various transformations to the input images, such as flipping them horizontally, adjusting the brightness and contrast, or adding random noise. For example, by flipping the images horizontally, we effectively double the

size of the data set and provide the agent with more examples of turning in both directions

```
FUNCTION augmentImage(imagePath, steering)
    img = imread(imagePath)
    IF random number < 0.5 THEN
        pan = Affine(translate_percent = {'x':(-0.1, 0.1), 'y':(-0.1, 0.1)})   image
        img = pan.augment_image(img)
    END IF
    IF random number < 0.5 THEN
        zoom = Affine(scale=(1, 1.2))
        img = zoom.augment_image(img)
    END IF
    IF random number < 0.5 THEN
        brightness = Multiply((0.4, 1.2))
        img = brightness.augment_image(img)
    END IF
    IF random number < 0.5 THEN
        img = flip(img, 1)
        steering = -steering
    END IF
    RETURN img, steering
END FUNCTION
```

Figure 4.8: Pseudo Code for Image Augmentation

We have also done preprocessing of the image to focus more on road. This involves applying various transformations to the images to make them more suitable for use in the agent.

One common preprocessing step is to crop the images to remove any irrelevant or distracting information from the edges of the image.In the case of the self-driving car simulator, this might include the dashboard, the car hood, or the surrounding scenery. By cropping the images, we can focus the agent's attention on the road ahead and reduce the amount of noise in the input data.

Another common preprocessing step is to resize the images to a consistent size. This is important because the agent expects input images to be of a certain size and shape, and resizing the images ensures that they are all the same size and shape. This can help simplify the agent architecture and reduce the risk of overfitting.

4.1.7 Agent with Supervised Learning

The agent consists of 9 convolutional neural network (CNN) layers and is organized as a sequential agent. The input to the agent is a batch of preprocessed images, which are fed through the various layers of the agent to produce a steering angle prediction.

The first layer of the agent is a convolutional layer that applies a set of filters to the input image to extract features such as edges, corners, and textures. This is followed by a series of additional convolutional layers that progressively extract more complex features from the input image.

After the convolutional layers, the agent includes several fully connected layers that perform classification and regression tasks. These layers are used to predict the steering angle, throttle, brake, and other control signals for the self-driving car.

Our final agent was with the following structure:

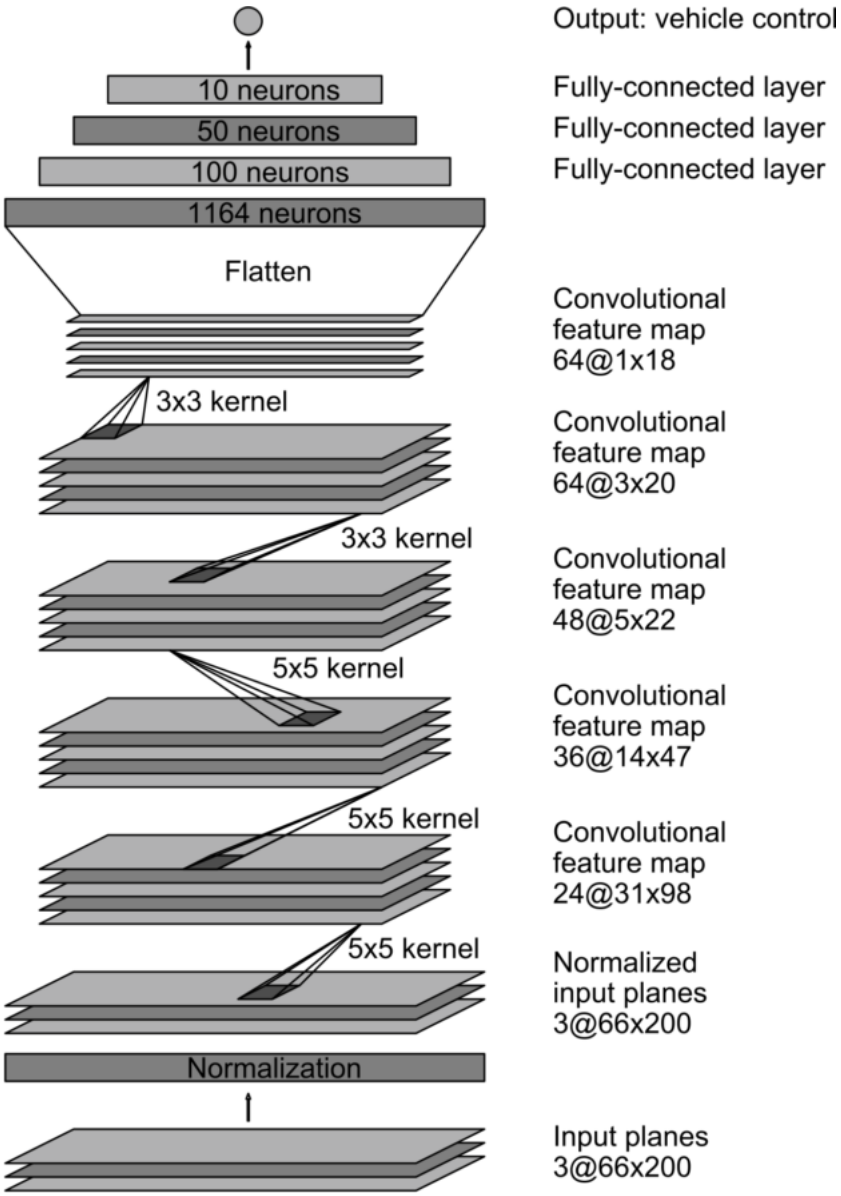


Figure 4.9: Model Architecture

4.2 Results of Agent trained with Supervised Learning

The agent was trained for nearly 30,000 epochs, with a final loss of around 9%. This corresponds to an accuracy of approximately 91%, indicating that the agent is able to predict the steering angle with a high degree of accuracy.

During the training process, the agent was fed batches of preprocessed images and corresponding steering angles, which were used to update the agent's weights and biases. The training process involved iteratively adjusting the agent's parameters to minimize the difference between the predicted and actual steering angles.

To monitor the performance of the agent during training, the loss was calculated at each epoch and plotted over time. The goal was to minimize the loss over time, indicating that the agent was improving its predictions and generalizing well to new data.

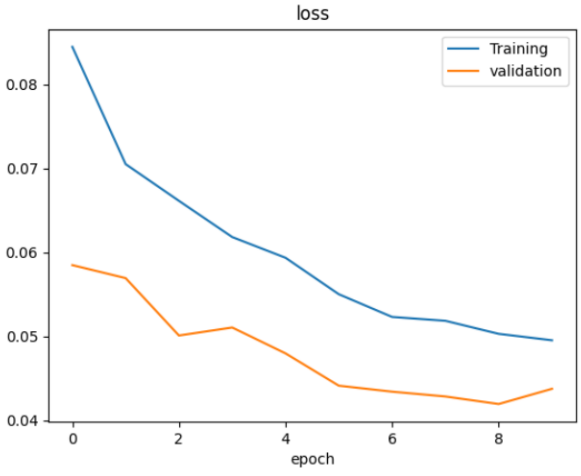


Figure 4.10: Training and Validation.

4.3 Conclusion

We worked on Self Driving Cars using supervised learning. However, we faced some limitations and drawbacks of supervised learning. For example, supervised learning requires a large amount of labeled data, which can be difficult to obtain for autonomous driving applications. Additionally, the agent trained with supervised learning may struggle to generalize to new and complex scenarios that were not present in the training data. As a result, we decided to explore alternative methods for training self-driving car agents. After careful consideration and analysis, we chose to switch to the use of reinforcement learning. This decision was based on the potential benefits of reinforcement learning, such as the ability to learn from trial and error and to handle complex and uncertain environments. With the experience and knowledge gained from our previous work, we were able to apply these insights to develop a successful reinforcement learning approach for self-driving car agents.

Chapter 5

Proposed Work

This project proposes the development of an artificial agent for a car game using reinforcement learning techniques. The objective is to train an agent to play the game efficiently.

5.1 Work Done for Artificial Car Racing Agent using Reinforcement Learning

5.2 Design

The design of our artificial agent for Car Racing v0 is based on the reinforcement learning paradigm. In our case, the agent receives a reward signal based on its performance in the game environment, and the goal is to maximize the total reward over time.To implement this, we used a deep reinforcement learning algorithm called Deep Q-Networks (DQN). DQN is a type of neural network that is trained using a variant of Q-learning, which is a popular reinforcement learning algorithm. The network takes the current observation of the game environment as input and outputs a Q-value for each possible action. The action with the highest Q-value is then chosen by the agent.

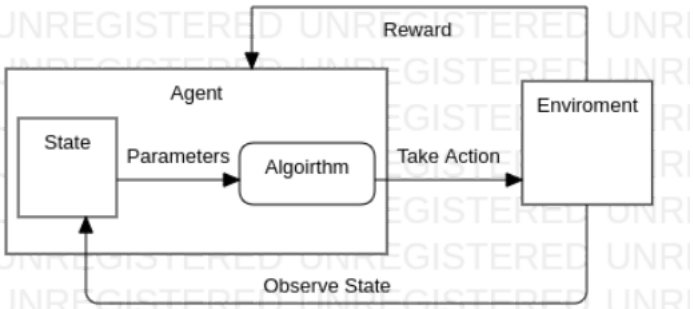


Figure 5.1: Design

We also used a technique called experience replay to improve the stability and efficiency of the training process. Experience replay involves storing the agent's experiences (i.e., the observations, actions, rewards, and next observations) in a buffer and randomly sampling from this buffer to train the network. This helps to reduce the correlation between the agent's experiences and makes the training process more robust.

Overall, our design for the artificial agent involved using a combination of deep reinforcement learning techniques to train a neural network to make decisions in the Car Racing v0 environment. This approach allowed us to create an agent that can learn to navigate through the game world and make decisions in response to the game's challenges.

5.2.1 Map of Environment

We used Car Racing v0 from OpenAI gym 2D map to train our artificial agent using reinforcement learning. The map has been designed to simulate the game environment and includes various elements such as rewards, turns, and checkpoints. These elements help the agent learn how to navigate the game

environment and make decisions by trial and error. By receiving feedback in the form of rewards or penalties, the agent can improve its performance and develop better strategies to achieve its objectives in the game.

5.2.2 State Space of the Environment

The state space we used for the Car Racing v0 game is an RGB image with three channels. The first two channels represent the boundaries of the track, while the third channel indicates the car's position and orientation.

The first channel provides information about the left-hand side boundaries of the track, while the second channel represents the right-hand side boundaries. By analyzing these channels, the agent can avoid colliding with the track boundaries. The third channel contains information about the car's position and orientation. It helps the agent make decisions about how to navigate the track.

In addition to the RGB image, the state space includes the car's speed, represented as a two-dimensional vector. This vector provides information about the car's velocity, enabling the agent to adjust its speed and direction optimally.

By defining the state space with these variables, our artificial agent can learn from its experience and develop strategies to maximize its rewards and minimize its penalties. This, in turn, helps the agent improve its performance in the game.

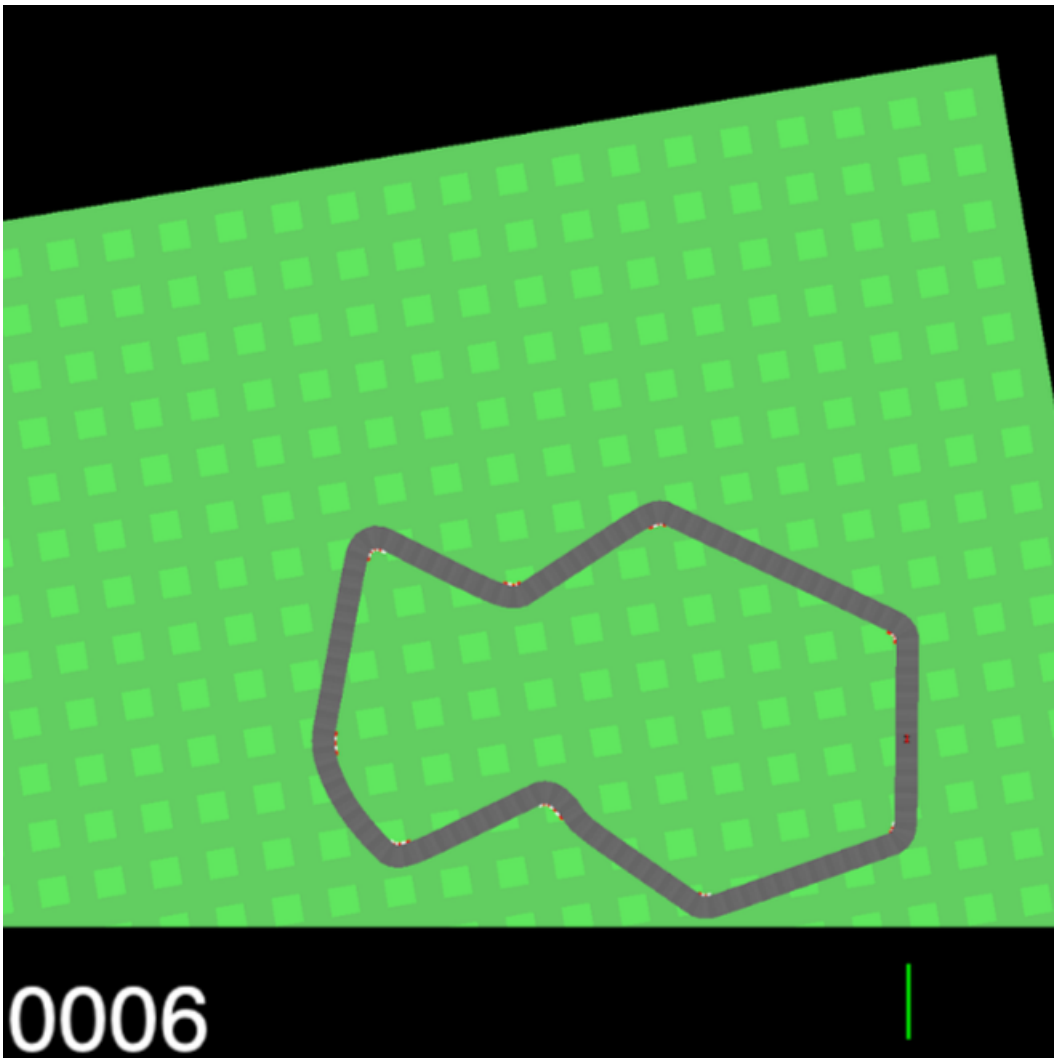


Figure 5.2: 2D-Map

Figure 5.3: Learning to be on track

5.2.3 Action Space

Our Artificial Agent in the Car Racing v0 game can perform five possible actions:

1. Do nothing: This action refers to the car maintaining its current speed and direction without any changes.
2. Steer left: The "steer left" action enables the car to turn left.
3. Steer right: The "steer right" action enables the car to turn right.
4. Gas: The "gas" action allows the car to accelerate.
5. Brake: The "brake" action allows the car to slow down or stop.

To train the agent, we utilized a Reinforcement Learning algorithm that adjusts the agent's actions based on the rewards it receives. The agent gradually learned the optimal policy for navigating the track and achieving the goal, By defining the Action Space consisting of these five actions, we pro-

vided our Artificial Agent with the ability to make decisions and navigate the game environment effectively.

5.2.4 Reward System and Actions

In the Car Racing v0, we defined a set of actions that our artificial agent could take, and a reward system to encourage the agent to take actions that would lead to a successful completion of the game. The actions available to the agent were accelerating, braking, and turning left or right. The agent's goal was to complete an episode of the track in the shortest time possible while avoiding collisions with the track boundaries.

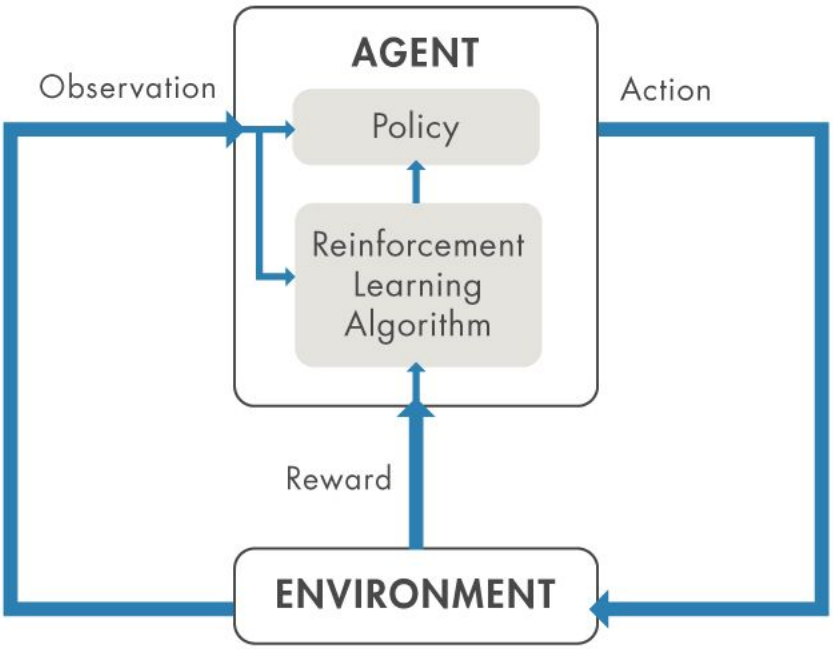


Figure 5.4: Reward System Image

To incentivize the agent to take actions that would help it achieve this

goal, we defined a reward system that assigned positive and negative rewards to the agent based on its actions. For example, if the agent completed an episode of the track, it would receive a positive reward. Conversely, if the agent collided with the track boundaries. it would receive a negative reward.

We also included intermediate rewards to encourage the agent to take actions that would help it complete the episode faster. For example, the agent would receive a positive reward for passing through a checkpoint or driving at high speeds. To determine the optimal policy for our agent, we used a reinforcement learning algorithm that iteratively improved the agent's performance by adjusting its actions based on the rewards it received. By learning from trial and error, the agent could gradually improve its performance and develop strategies to complete the game successfully.

In summary, we defined a set of actions and a reward system for our artificial agent to incentivize it to achieve the goal of completing an episode of the track in the shortest time possible while avoiding the collision. Using reinforcement learning, the agent could improve its performance over time and develop optimal strategies to succeed in the game.

5.2.5 Observation Space

The observation space in our Car Racing v0 environment consists of a top-down 96x96 RGB image of the car and the race track. This image is generated by the game agent and is used as the input for our AI agent. The image provides a bird's-eye view of the track and allows the agent to perceive its position and orientation in the game world. It also includes information about the location of the track boundaries. By processing this image, the agent can make informed decisions about its actions and navigate through the track effectively.

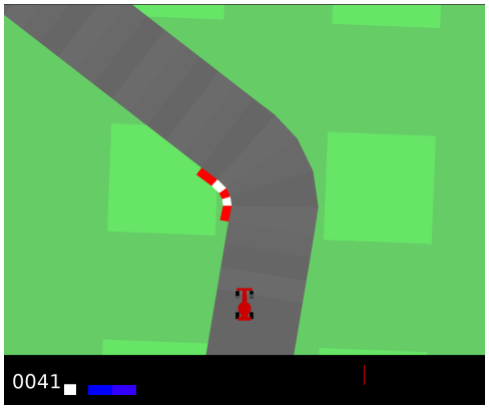


Figure 5.5: Image of Bird Eye-View

5.2.6 Training Algorithm Used

5.2.7 PPO (Proximal Policy Optimization)

We have used the PPO (Proximal Policy Optimization) algorithm for training our agent. The choice of the PPO (Proximal Policy Optimization) algorithm was mainly due to its robustness and stability. Compared to another algorithm we tested, DDPG, PPO showed more consistent and reliable results during training.

One of the key advantages of PPO is its ability to handle continuous action spaces, which is particularly important for car racing tasks where the agent needs to control the acceleration, steering, and braking of the car in a continuous manner. Additionally, PPO has a strong theoretical foundation and has been shown to be effective in a variety of reinforcement learning tasks.

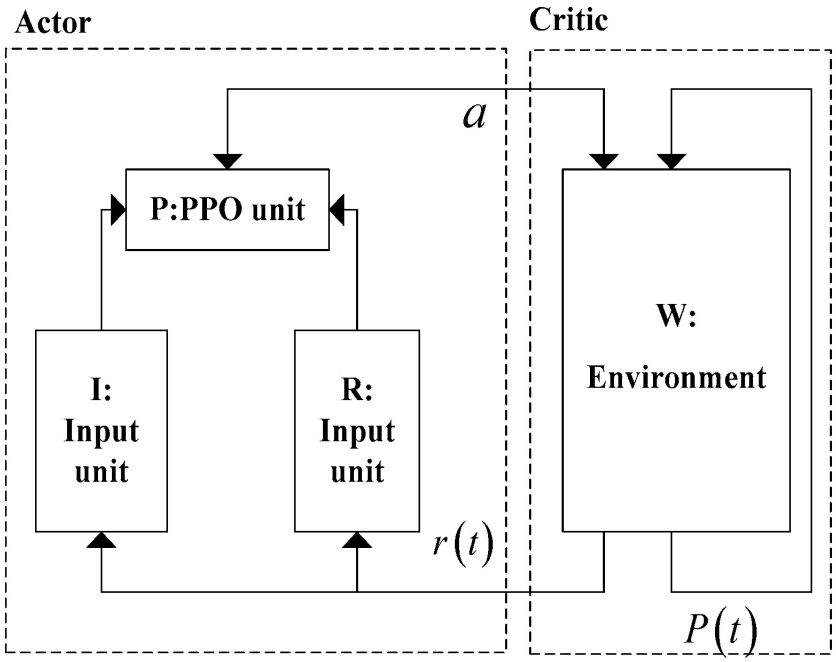


Figure 5.6: PPO-Algorithm

5.2.8 DDPG (Deep Deterministic Policy Gradient)

We initially considered using the DDPG (Deep Deterministic Policy Gradient) algorithm for training our agent in the car game environment. DDPG is a model-free, off-policy actor-critic algorithm that is specifically designed to handle continuous action spaces. It has been shown to be effective in a wide range of applications, including robotics control and game playing.

However, after experimenting with DDPG, we found that it was not as stable as we had hoped. The agent's performance was inconsistent, and it often got stuck in local optima. We also found that it was more difficult to tune the hyperparameters for DDPG than for PPO.

Therefore, we decided to switch to the PPO (Proximal Policy Optimiza-

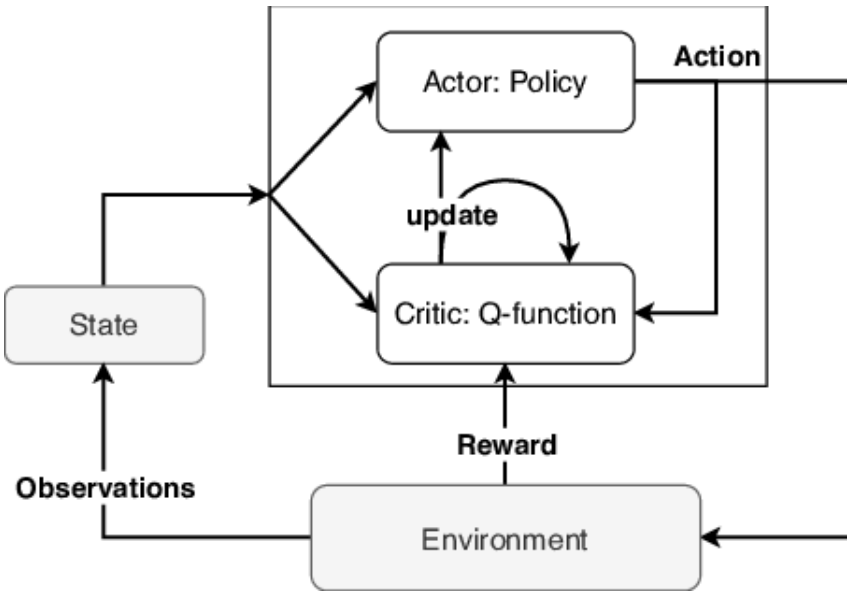


Figure 5.7: DDPG-Algorithm

tion) algorithm, which is known to be more stable and easier to use. Nonetheless, we believe that DDPG has potential for certain applications, and we plan to explore it further in future work.

5.2.9 Algorithm Policy Used for Training

CNN(Convolutional Neural Network)

We used a Convolutional Neural Network (CNN) as the algorithm policy for training our agent. A CNN is a deep learning algorithm that can learn features and patterns from images, making it an ideal choice for image-based environments like Car Racing v0.

The CNN takes the top-down 96x96 RGB image of the car and race track as input and applies a series of convolutional and pooling layers to extract important features from the image. The output of the CNN is a probability

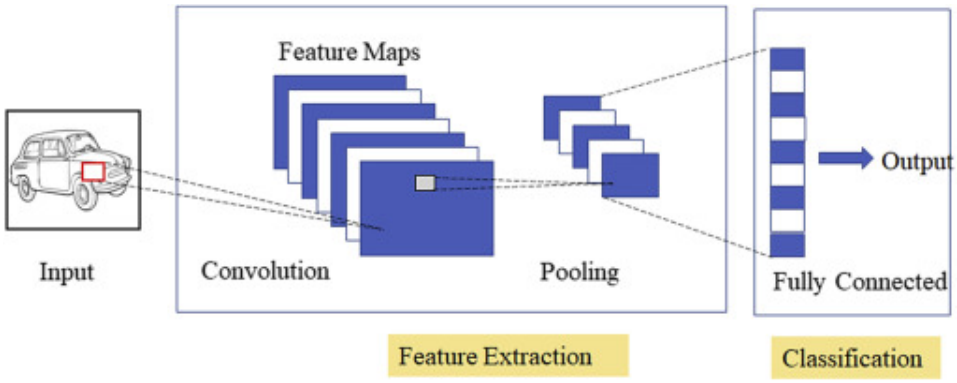


Figure 5.8: CNN Data Input Image

distribution over the available actions, which allows the agent to choose the action with the highest probability.

By using a CNN as the algorithm policy, we were able to train our agent to learn from visual inputs and make decisions based on the current state of the environment. This helped the agent to navigate the complex environment of Car Racing v0 and improve its performance over time.

Chapter 6

Experimental Results

6.1 Results

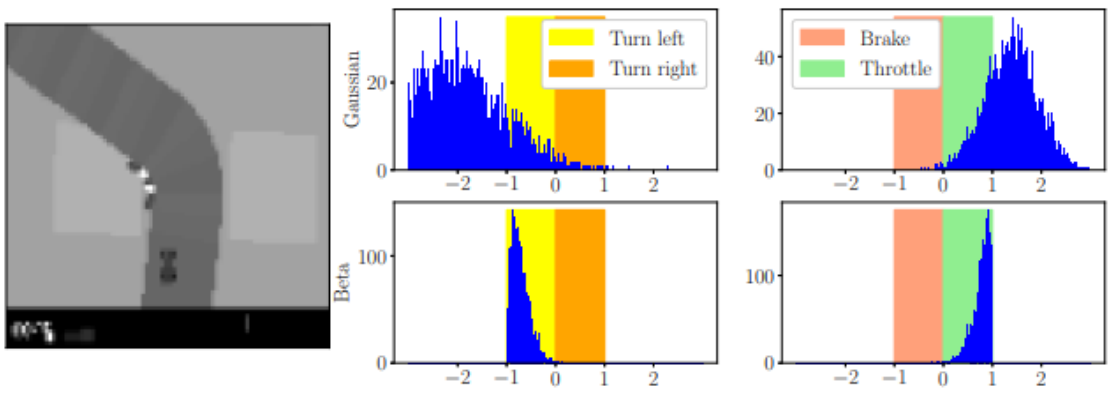


Figure 6.1: Result of 2D Model

The 2D Car Racing v0 agent was trained for a total of 20 lakhs iterations using the PPO algorithm and the CNN policy. After the training was complete, the agent was able to achieve a high score of 900. This score indicates that the agent was able to effectively navigate through the race track. The

results suggest that this approach can be effective for training agents to play similar racing games or tasks that require continuous control.

Our experiments showed that the PPO algorithm with a CNN policy was able to successfully learn to play the Car Racing v0 game, achieving a high score of 900 after being trained for 2 million iterations

Through our experimentation, we demonstrated that the PPO algorithm with a CNN policy was able to successfully learn to play the Car Racing v0 game, achieving a high score of 900. Our results also suggest that further research can be done to improve the performance of agents in similar game environments or tasks that require continuous control.

Chapter 7

Conclusion

After training our artificial agent using reinforcement learning on the Car Racing v0 from OpenAI gym, we achieved impressive results, obtaining a high score of 900 after 2 million iterations of training. We used a PPO algorithm with a CNN policy to train the agent, which proved to be robust and stable for the task, as compared to the DDPG algorithm which did not perform as well. The use of deep learning and computer vision for autonomous driving applications has great potential, as demonstrated by the high accuracy achieved by the self-driving car agent trained in the Udacity simulator. The multi-step process we followed involved collecting and preprocessing data, building and training the agent, and monitoring its performance. Overall, our work showcases the potential of reinforcement learning for developing intelligent agents capable of complex tasks in simulated environments, with important implications for real-world scenarios in transportation and robotics.

References

[1] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning", nature, vol. 521, no. 7553, pp. 436, 2015.

[2] B. Wymann, E. Espié, C. Guionneau, C. Dimitrakakis, R. Coulom, and A. Sumner, "Torcs the open racing car simulator", vol. 4, no. 6, 2000

[3] R. S. Sutton and A. G. Barto, Reinforcement learning: An introduction, MIT Press, 2018.

[4] M. Hessel, J. Modayil, H. Van Hasselt, T. Schaul, G. Ostrovski, W. Dabney, et al., "Rainbow: Combining improvements in deepreinforcement learning", Thirty-Second AAAI Conference on Artificial Intelligence, 2018.

[5] M. Jaritz, R. De Charette, M. Toromanoff, E. Perot and F. Nashashibi, "End-to-end race driving with deep reinforcement learning", 2018 IEEE International Conference on Robotics and Automation (ICRA), pp. 2070-2075, 2018.

[6] B. Renukuntla, S. Sharma, S. Gadiyaram, V. Elango, and V. Sakaray, "The road to being taken a deep reinforcement learning approach towards autonomous navigation", 2017.

[7] V. Mnih, A.P. Badia, M. Mirza, A. Graves, T.P. Lillicrap, T. Harley, et al., "Asynchronous methods for deep reinforcement learning", Proceedings of ICML, 2016.

[8] Sen Wang, Daoyuan Jia, and Xinshuo Weng. Deep Reinforcement Learning for Autonomous Driving, 2018.

[9] Ahmad El Sallab, Mohammed Abdou, Etienne Perot and Senthil Yogamani. Deep Reinforcement Learning Framework for Autonomous Driving, 2017.

[10] Sacha Lange, Matron Riedmiller and Arne Voigtlander. Autonomous reinforcement learning on raw visual input data in a real-world application, 2012.

[11] Xinlei Pan, Yurong You, Ziyan Wang and Cewu Lu. Virtual to Real Reinforcement Learning for Autonomous Driving.

[12] Matt Vitelli and Aran Nayebi. CARMA: A Deep Reinforcement Learning Approach to Autonomous Driving.

[13] B Ravi Kiran, Ibrahim Sobh, Victor Talpaert, Patrick Mannion, Ahmad A. Al Sallab, Senthill Yogamani, Patrick Perez. Deep Reinforcement Learning for Autonomous Driving: A Survey.

[14] Alexey Dosvitskiy, German Ros, Felipe Codevilla, Antonio Lopez and Vladlen Koltun. CARLA: An Open Urban Driving Simulator.

[15] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Hees, Tom Erez, Yuval Tassa, David Silver and Daan Wierstra. Continuous Control with Deep Reinforcement Learning.

[16] Victor Talpaert, Ibrahim Sobh, B Ravi Kiran, Patrick Mannion, Senthil Yogamani, Ahmad El-Sallab and Patrick Perez. Exploring applications of deep reinforcement learning for real-world autonomous driving systems.

[17] Jianyu Chen, Bodi Yuan and Masayoshi Tomizuka. Model-free Deep Reinforcement Learning for Urban Autonomous Driving.

[18] Daniele Loiacono, Luigi Cardamone and Pier Luca Lanzi. Simulated Car Racing Championship Competition Software Manual.

[19] Henry Teigar, Miron Storožev and Janar Saks University of Tartu. 2D Racing game using reinforcement learning and supervised learning