**TORCS A.I. Driver**

**Blake Engelbrecht David Englemen**

Department of Computer Science Department of Computer Science

Missouri State University Missouri State University

Springfield, MO, USA Springfield, MO, USA

e[ngelbrecht95@live.missouristate.edu](mailto:Engelbrecht95@live.missouristate.edu) [englemen369@live.missouristate.edu](mailto:englemen369@live.missouristate.edu)

**Shannon Groth Khaled Hossain**

Department of Computer Science Department of Computer Science

Missouri State University Missouri State University

Springfield, MO, USA Springfield, MO, USA

c[ampbell489@live.missouristate.edu](mailto:Campbell489@live.missouristate.edu) [khaled2049@live.missouristate.edu](mailto:khaled2049@live.missouristate.edu)

**Jacob Rader**

Department of Computer Science

Missouri State University

Springfield, MO, USA

rader503@live.missouristate.edu

***Abstract***

This research paper looks at the performance of the deep deterministic policy gradient algorithm. In this project we use the DDPG algorithm to train an autonomous agent in The Open Racing Car Simulator (TORCS). In our implementation we use TensorFlow 2.0 and Keras software frameworks. 3 reward functions are used to train the agent, a function that gives positive reward when the car stays inside the track and negative reward if it goes outside, a function that penalizes when the progress of the agent is too small and lastly, a function that gives negative reward when the car’s position does not change for more than 30 episodes. To validate learning, we the projects runs the trained agent over two tracks to evaluate the performance of the autonomous agent

# Diagram Description automatically generatedIntroduction

The Open Racing Car Simulator, or TORCS for short, is a state-of-the-art application that allows the user to set up driver-bots to perform in a realistic racing setting. TORCS is equipped with a realistic physics engine with vehicle dynamics such as traction, aerodynamics, fuel consumption, steering direction, damage detection, and other functions. The TORCS software used to train a driver-bot through reinforcement learning initially utilized TensorFlow 1.0. In this paper, we focus on the software upgrade from TensorFlow 1.0 to TensorFlow 2.0 with Python and the subsequent implementation and training of a driver bot on a randomly generated racetrack. With the use of TensorFlow 2.0 and the collection of sensor data, the driver bot is capable of navigating any given terrain and track layout.

The TORCS software contains an architecture known as the *sensors and actuators model* that structures TORCS as a client/server application. Therefore, the TORCS engine utilizes a *client/server model* architecture. The sensors and actuators model gives the developer the freedom to whose which programming language they want to use for the bots and restricts access to information designed by the developer. The client/server model allows the software to communicate with the TORCS server via UDP (User Datagram Protocol) connection. The client and server communicate in real time. The server sends current sensor inputs from the vehicle to the driver-bot every game tic. After obtaining the sensor data, the training software builds actor and critic models to allow the neural network to learn. The models, weights, and loss are updated to train the driver bot to learn the racetrack.

# Literature Review

## Literature 1

“Data Classification with Deep Learning using TensorFlow” is a journal article published in 2017, detailing the use of TensorFlow’s deep learning libraries to classify an MNIST (Modified National Institute of Standards and Technology) dataset of handwritten numbers and train an image processing system. The article provides evidence of the effectiveness of TensorFlow to train an application through deep learning by using ReLu, eLu, tanH, sigmoid, softPlus, and softsign activation functions to test the accuracy of the classification of the system. Using the ReLu activation function, the TensorFlow software obtained a 98.43% classification accuracy on the test data. This classification accuracy points to the current version of TensorFlow being a good candidate for training the driver bot in TORCS. [1]

## Literature 2

“A Study on Image Classification Based on Deep Learning and TensorFlow” is a journal article published in 2019, detailing the use of TensorFlow’s deep learning framework and its effectiveness. The deep neural network using framework TensorFlow categorizes flowers by five different types. The authors chose DNN (Deep Neural Network) for the training process due to the high rate of accuracy for the image classification, the average result percentage being up to 90% and above. [2]

## Literature 3

“Large-Scale Machine Learning on Heterogeneous Distributed Systems” is a research paper published in 2016, which details how TensorFlow is an interface for machine learning algorithms on heterogeneous systems such as mobile phones and GPU cards. The researchers draw the conclusion that the TensorFlow system is flexible and can be used to implement different training and inference algorithms for deep neural network models. TensorFlow has been used for speech recognition, robotics, language processing, and other areas of study. [3]

## Literature 4

“State of the Art Reinforcement Learning Algorithms” is a research paper published in 2020, providing information about the aspects of reinforcement learning algorithms and their effectiveness. The researchers explain that Deep Deterministic Policy Gradient (DDPG) is a result of a modified DQN (Deep Q-Network) that allows continuous actions or action spaces. Deep deterministic policy gradient is a combination of DQN and policy gradient (PG). The gradient of the loss is calculated, and the future rewards are maximized, and losses minimized. However, DDPG is slower than the actor-critic algorithm but faster than DQN. [4]

## Literature 5

“Continuous Control with Deep Reinforcement Learning” is a research paper published in 2016, detailing the effectiveness of an actor-critic, model-free algorithm based on deterministic policy gradient than can operate over continuous action spaces (DDPG). The researchers use this learning algorithm to solve more than 20 physics tasks such as cartpole and car driving. The researchers concluded that DDPG requires many training episodes to learn and find a solution, but difficult problems can be solved if the model is given enough time. [5]

# Proposed Solution

In the past, there were two main approaches in training agents that are autonomously driven in the TORCS environment. The first one being a *DeepDriving* model that implemented the deep supervised learning and Convolutional Neural Networks (CNN) to learn the driving policy. Secondly, there was research done by DeepMind, a well-known group of scientists, engineers, and deep learning experts at the forefront of Artificial Intelligence (A.I.) advancement, that showed the Actor Critic structures can be used for a variety of different projects. In this paper we can see that continuous control problems can be solved by using neural networks. DDPG uses an experience replay buffer for each training step in the episodes, the critic and actor are updated.

The main objective of this project was to implement the Deep Deterministic Policy Gradient algorithm to train a model and compare the autonomous agent’s performance as it navigates the track. We used reinforcement learning and the DDPG algorithm to create a program that teaches the agent to drive around the track by taking the proper actions and maximizing the reward, improper actions such as going off-course result in negative rewards, or punishments, to dissuade reoccurrence of such actions. In the paper “Continuous Control with Deep Reinforcement Learning” we are introduced to the DDPG algorithm. The authors explain that this algorithm is a model-free, off-policy actor-critic that uses deep function approximators to learn policies in high dimensional continuous action spaces.

# Implementation

Text, letter

Description automatically generated

The DDPG algorithm mostly follows the deep policy gradient algorithm. The only difference is that the function approximation for both actor and critic acted upon deep neural networks

In our TORCS environment, each state observation considers component wise velocities, distance to the track center and edges, the angle that is relative to the track, and the wheel rotation speed. The action that the agent takes specifies the steering angle, acceleration and braking. The agent is trained off-policy with samples, forming a replay buffer to minimize correlations between the samples. We initialize the buffer size to 100,000. We chose this number because we wanted the agent to have stable behavior. We chose not to make the replay buffer too large as it would require a lot of memory and the training would be very slow. A value of 100,000 was good enough for this experiment as it contained enough experiences for the agent to learn without taking too much time or requiring too many resources.

During each episode, with “episode” referring to a single instance of training, there were 3 reward functions in action which calculated the total reward while our autonomous agent was being trained. the first one decreased the reward if the car went outside of the track boundaries. The second reward function was responsible for the termination of the episode if the progress of the agent is too small. This helps to make the training process more efficient by minimizing training time by preventing episodes of lower progression from continuing. Finally, the last reward function terminated the entire training process if the agent was stuck in one position for more than 30 episodes.

Diagram

Description automatically generated

The table shows the types of sensor data we used as inputs for our autonomous agent, it also displays the units and the range for each sensor data type. During the training of the autonomous agent, we can see the car shake but each episode it gets better and tries to reduce the angle and get to the center of the track

|  |  |
| --- | --- |
| Name | Description |
| Ob.angle | Angle between the car direction and direction of the track axis |
| Ob.track | Vector of 19 range sensors: each returns the distance between the track edge and the car |
| Ob.rackPos | Distance between the car and the track axis |
| Ob.speedX | Speed of the car in the x-axis |
| Ob.speedY | Speed of the car in the y-axis |
| Ob.speedZ | Speed of the car along the z-axis |
| Ob.wheelSpinVel | Rotation speed of wheels |
| Ob.rpm | Number of rotations per minute of the car |

# Performance Evaluation

We used TensorFlow 2.0 and Keras API (Application Programming Interface) in our implementation. The batch size for the program was initialized to 64 and we used a replay buffer size of 100,000 state-action pairs. In our implementation we used two hidden nodes and set the values to 300 and 600. The hidden nodes have no connection with the outside world. Since our problem was complex, we used a multi-layer neural network, and this allowed us to find more complex decision boundaries. Additionally, in our implementation we used the *Ornstein-Uhlenbeck* process to add noise to our program. We added noise to our program because it allowed us to better explore the range of actions and their potential outcomes.

Text

Description automatically generated

The car controller incorporates a Feed Forward Neural Network that uses the 8 inputs and 3 actions. We used three activation functions in our program, the sigmoid, tanh and rectified linear unit. The sigmoid takes a real-valued input and squashes it to range between 0 and 1. Tanh takes a real valued input and squashes it in the range of -1 to 1, lastly, the rectified linear unit replaces negative values with zero. Additionally, we assign weights to our agent based on its relative importance to other inputs. Additionally, we set the learning rate for our actor to 0.0001 and the learning rate for the critic to 0.001. These parameters allowed us to train the autonomous agent and we were able to test it on several tracks. In the graph below we can see the episodic reward over 100 episodes.

Chart, histogram

Description automatically generated

*Figure 1: - reward vs episode*

In the graph we can see the reward values of the agent increase and this shows us during this project we were only able to train the agent for 100 episodes which took us 8 hours and 35 minutes. However, just from training our agent for less than 10 hours we were able to create an autonomous agent that was capable of navigating the track and finishing a lap on the CG-speed way the track shown in figure-1.

Shape

Description automatically generatedA picture containing hanger, light

Description automatically generated

Figure CG-speedway

Figure G-speedway

Figure C-speedway

A picture containing mirror

Description automatically generated

By observing the performance of the autonomous agent we can conclude that if we trained our autonomous agent for a longer period of time it will continue to learn and improve, lowering the time it takes for the bot to successfully complete a lap and perform even better. We tested our trained model on 2 tracks, they were: “G-speedway” and “C-speedway”. In figures 5 and 6 we can see the episodic reward of the autonomous agent as it navigated the track

In the image we can see the car is on lap 2, has a best time of 59.95 sec on the CG-speedway track, and that it is travelling at a speed of 146 km/h. This picture allows us to show that the autonomous driver can navigate along the track at high speeds and finish the lap in under 1 minute:

A picture containing text, sport, athletic game, outdoor

Description automatically generated

Figure 4 TORCS race

Chart, line chart, histogram

Description automatically generated

Figure 5 Testing on G-speedway

Chart, line chart

Description automatically generated

Figure 6 Testing on C-speedway

# Conclusion and Future Work

In this paper, we proposed that continuous control problems can be solved using neural networks through implementing the Deep Deterministic Policy Gradient algorithm to train a model and compare the agent’s performance while it navigates around a racetrack. We upgraded a TensorFlow 1.0 program to TensorFlow 2.0. The learning agent used DDPG and reinforcement learning to learn to drive around a track while maximizing reward and without making collisions. This project will give researchers the ability to perform tests without sacrificing hardware. It will be more cost efficient and beneficial for engineers to build models and apply the knowledge to real life scenarios.

An area of improvement for the track learning would be the amount of time we spent teaching the agent to drive around a racetrack (8 hours and 35 minutes) due to equipment access issues. If the agent were able to learn a racetrack for 24+ hours, the driver would be quicker and more successful at navigating a racetrack.

In the future, we would like to apply the ability for the software to detect if the vehicle is low on fuel or has a high amount of damage dealt, and if either is true, the vehicle will make a pit stop to refuel or repair. We also plan on the driver to be able to race against human drivers in real-time in a competitive environment. More ambitious plans include the autonomous driver bot being involved with multiplayer, virtual reality games or real-life self-driving cars.

##### **References**

[1] [online] Ertam, F., & Aydin, G., “Data classification with deep learning using Tensorflow,” 2017. Last accessed: November, 2020. Retrieved from: https://ieeexplore.ieee.org/document/8093521

[2] [online] Abu, M. A., Indra, N. H., Ahmad1, I., Rahman, A. H., & Sapiee, N. A., “A study on Image Classification based on Deep Learning and Tensorflow,” 2019. Last accessed: November, 2020. Retrieved from: https://www.ripublication.com/irph/ijert19/ijertv12n4\_16.pdf

[3] [online] Various Authors, “TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems,” 2015. Last accessed: November, 2020. Retrieved from: https://arxiv.org/pdf/1603.04467.pdf%20http://arxiv.org/abs/1603.04467.pdf

[4] [online] Mehta, D., “State-of-the-Art Reinforcement Learning Algorithms,” 2020. Last accessed: November, 2020. Retrieved from: https://www.researchgate.net/publication/338396174\_State-of-the-Art\_Reinforcement\_Learning\_Algorithms

[5] [online] Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., . . . Wierstra, D., “Continuous Control With Deep Reinforcement Learning,” 2016. Last accessed: November, 2020. Retrieved from: https://arxiv.org/pdf/1509.02971.pdf