Journal of Advanced Research in Applied Sciences and Engineering Technology

Journal homepage:
https://semarakilmu.com.my/journals/index.php/applied_sciences_eng_tech/index
ISSN: 2462-1943

Implementing Deep Reinforcement Learning in Autonomous Control Systems

Noureldin Ragheb[1], Mervat M. A. Mahmoud[1,2,*]

[1] Department of Electrical Engineering, The British University in Egypt, Cairo, Egypt
[2] Microelectronics Department, Electronics Research Institute, Cairo, Egypt

ARTICLE INFO

Article history:
Received 15 October 2023
Received in revised form 20 December 2023
Accepted 11 January 2024
Available online 4 March 2024

Keywords:

Deep reinforcement learning; Autonomous vehicle; CARLA simulator

ABSTRACT

Developing a safe and reliable autonomous vehicle has been a significant focus in recent years. Supervised learning methods require large amounts of labelled data for training, making it expensive. The performance of these agents is limited to the data provided in training and the inability to generalize performance in different environments. In addition, some driving situations, such as near-accident scenarios, are difficult to cover in the training data. As a result, the autonomous driving agent may behave unexpectedly in safety-critical situations, making it unreliable for safe transportation. Reinforcement learning is a potential solution for these issues. This research paper explores the potential of applying deep reinforcement learning techniques to autonomous driving, with a spotlight on comparing two popular deep reinforcement learning algorithms: Deep Q-Network (DQN) and Deep Deterministic Policy Gradient (DDPG). The study uses the CARLA simulator, which provides a realistic environment and conditions for testing autonomous driving algorithms. The study finds that DDPG outperforms DQN regarding average reward, but DQN performs better regarding collision rate.

1. Introduction

Autonomous vehicles have been spurred primarily by their potential to resolve various transportation issues, such as traffic congestion, enhance the travel experience for mobility-impaired passengers, and reduce automobile accidents. However, these technologies require constant human intervention to achieve safe driving capabilities. Autonomous vehicles are innovative sensing devices with sensors such as radar, LIDAR, and cameras capable of storing and gathering immense quantities of data and information, necessitating efficient processing techniques to ensure continued effectiveness throughout the operational lifespan [1,2]. Due to the abundance of data, machine learning models may now be trained to carry out complex tasks such as driving. As a result, instead of using classical methods for controlling a vehicle, such as Proportional Integral Derivative (PID)

[*] *Corresponding author.*
E-mail address: mervat-m@eri.sci.eg

https://doi.org/10.37934/araset.41.1.168178

controllers or Model Predictive Control (MPC) [3], some alternative machine learning methods could be used.

Nonetheless, machine learning researchers have made encouraging progress, as evidenced by completing challenges such as the DARPA Urban Challenge, suggesting that innovative machine learning-based solutions are conceivable [4-6]. Deep learning is one such technique that can effectively convert high-dimensional input data into task-specific representations. Due to the advancement of deep learning, the modular pipeline is replacing the conventional pipeline with the end-to-end pipeline. Rather than having multiple modules designed by humans, an end-to-end system directly converts the sensory input to control signals [7-9]. However, large quantities of labelled data, obtained at considerable expense using expert human drivers, are required to train such systems properly. Besides, the policies learned through supervised learning methods lack interpretability [10]. Moreover, the collected data may only encompass a subset of the conceivable scenarios autonomous vehicles can handle. This may indicate instability in situations where safety is paramount. Because it provides a paradigm that requires few or no expert demonstrations, reinforcement learning (RL) has received considerable attention in the context of control tasks.

RL algorithms can be grouped into model-free and model-based algorithms. The model-based algorithm depends on an environment model for learning the optimal policy. Specifically, modelling the environment as a Markov Decision Process (MDP). Knowing the probability distribution and rewards will allow solving the problem by optimization. On the other hand, model-free algorithms such as Q-learning and Deterministic Policy Gradient (DPG) learn the optimal policy without any previous knowledge of the environment. In a complex task such as autonomous driving, it is not possible to model the environment [11]. In RL, the agent attempts various actions and observes the resulting environmental feedback. The agent modifies its internal parameters based on this data to determine the optimal action. The main objective of RL is to find an optimal policy that maximizes the expected cumulative reward over time [12]. The policy is a function that maps each state to an action. The optimal policy is the one that maximizes the expected cumulative reward over time. The optimal policy can be found using multiple RL algorithms, such as Q-learning, SARSA (state-action-reward-state-action), and policy gradient methods. These algorithms learn the optimal policy by iteratively updating their estimates of the value function or the policy. The value function represents the expected cumulative reward under a given policy, while the policy represents the mapping from states to actions [13,14]. The Q-learning algorithm starts with the Q-values set to zero. During training, the agent learns to update these Q-values from interacting with the environment by selecting actions, observing their resulting rewards and next states, and adjusting its estimate of the value of certain actions in specific states. Deep reinforcement learning (DRL) combines the scalability of deep learning and RL's sample efficiency, producing more accurate predictions. This is particularly important for problems with extensive or continuous state and action spaces [15-17]. SARSA is an on-policy algorithm designed to teach a machine learning model a new Markov decision process policy in order to solve RL challenges. It’s an algorithm where, in the current state (S), an action (A) is taken and the agent gets a reward (R), and ends up in the next state (S1), and takes action (A1) in S1. Therefore, the tuple (S, A, R, S1, A1) stands for the acronym SARSA. In the SARSA algorithm, the Q-value is updated taking into account the action, A1, performed in the state, S1. In Q-learning, the action with the highest Q-value in the next state, S1, is used to update the Q-table. So, Q-learning directly learns the optimal policy while SARSA learns a near optimal policy and may not find the optimal policy in certain environments. Moreover, Deep Deterministic Policy Gradient (DDPG) is a reinforcement learning algorithm that belongs to the actor-critic family of reinforcement learning methods. This type of algorithm is particularly useful when working with continuous action spaces and can provide high-quality policies for control in complex environments. DDPG is a model-free algorithm, which means it does not know about the underlying model of the environment and learns from interaction with the environment. The DDPG is also an off-policy algorithm, which means that the agent has a network for learning and another network for acting [18,19].

The selection of an algorithm for autonomous vehicle navigation control holds significant importance. We chose two promising and recent deep reinforcement learning algorithms, DQN and DDPG to investigate. This research aims to experiment and compare the results of the algorithms simulated on a high-fidelity simulator, CARLA [20,21]. It is the fundamental step of our project work, to choose a good algorithm and enhance it in the future work. The rest of the paper is structured as follows: Section 2 provides a literature review. The practical implementations of the two deep reinforcement learning algorithms, DQN and DDPG, are presented in section 3. Section 4 provides our results and discussion; and section 5 concludes the paper.

2. Literature Review

One of the approaches to autonomous vehicles is Behavioral Cloning [22], which was presented by Wael *et al.,* in 2018. The core idea is that a convolutional neural network takes data provided by a human expert driver and learns from it. The input to the CNN is an image which was taken from the vehicle front camera. The CNN then uses the input to obtain steering commands in an end-to-end fashion. By simply using the provided human steering angle as the training signal, the CNN automatically picks up internal representations of the necessary processing pipeline steps, such as spotting useful road features. The instability of behaviour and inability to generalize to diverse situations are some of the drawbacks of this strategy. Another method based on supervised learning was proposed in 2019 by Jianyu *et al.,* [23]. The adopted approach was using imitation learning for handling complex driving scenarios in urban areas with safety enhancements using deep learning. A bird eye view representation for the environment was also designed to simplify the sample complexity for imitation learning. Afterwards, a safety controller is added, which creates control orders to follow the intended trajectory while ensuring safety. The model obtained promising results, however, there were some failure cases. The system depended on ground truth information which was provided by the simulator. Consequently, the system needs a perception module which will affect the performance.

To solve problems such as generalization, over fitting, and significant reliance on labelled data that occur in supervised learning environments, reinforcement learning approaches could be used. A reinforcement learning framework based on AlphaGo Zero algorithm [24] was presented by Carl-Johan *et al.,* [6]. This framework basic idea is combining planning and decision-making for autonomous driving. A neural network is trained for the Monte Carlo Tree Search to be guided to the relevant regions of the tree search. This framework is promising; however, it is limited to high level actions for autonomous driving such as staying on a specific lane or changing lanes. The AlphaGo Zero algorithm is a model-based reinforcement learning algorithm which can experience performance limitations in a diverse action space environment such as autonomous driving.

Newly, in late 2022, Elallid *et al.,* used the Deep Q-Networks to control the autonomous vehicle in a complex scenario involving vehicles and pedestrians. They test and validate their approach using the CARLA simulator [25]. However, Park *et al.,* apply the deep deterministic policy gradient (DDPG) path-planning method for mobile robots using Gazebo simulator [26].

3. Methodology
3.1 The Problem

The RL problem could be described as a Markov Decision Process (MDP). The MDP consists of three components: the state, s_t, action, a_t, and reward, r_t, generated every timestep according to the agent's interaction with the environment. The main goal of an MDP is to find an optimal policy for interacting with the environment given a specific state. In the conducted experiments, the MDP is represented as a tuple (S, A, P, R).

i. S (State Space): The state space is the set of available states that could be perceived from the environment during the interaction. In this case, the state space consists mainly of visual features, vf_t, and driving features, df_t. The visual features are represented as a vector that contains a set of waypoints, w_t, obtained from the global path planner provided by the CARLA simulator. The driving features are the vehicle's state in the environment relative to the waypoints provided. The driving features are described as a vector, $df_t = (v_t, d_t, \emptyset_t)$, with multiple components, including the vehicle velocity, v_t, the vehicle's distance to the center of the lane, d_t, and the angle between the vehicle and the center of the lane, $\emptyset_t$.
ii. A (Action Space): CARLA simulator offers some commands to control the vehicle in the environment. At each timestep, the agent (vehicle) should perform an action that includes values of different types: throttle, brake, and steering. The accepted values for throttle and brake are in the range [0,1], while the steering angle is in the range [-1,1]. So, the agent should perform an action at each time step that includes a value for *a_t* = (throttle, brake, steering).
iii. P (Transition Probability): The transition probability is the probability of being in the state, s_{t+1}, due to acting, a_t, from being in the state: s_t: $P(s_{t+1}|s_t, a_t)$.
iv. R (Reward Function): The reward results from taking a specific action from being in a specific state. The main goal is to maximize the reward over time.

3.2 State Representation
3.2.1 CARLA waypoints

In order to control the vehicle, we need to obtain the waypoints or trajectories that the agent should follow. The experiment will use the CARLA waypoints generated using the global planner using the Python API, which means the vehicle is not perceiving the environment using images and convolutional neural networks. The global planner gives two random points, which are the starting and ending points. The agent is required to follow the trajectory specified by the global planner. Since the two points are random, the path distance could vary in each episode.

The global planner outputs the waypoints relative to the map starting position, which will result in unwanted values. The RL learning algorithm requires the vehicle's state relative to the environment. Therefore, a transformation matrix, T, is applied to the provided waypoints to obtain their positions relative to the vehicle. The following matrix performs translation and rotation. The vector $[X_c, Y_c, Z_c]$ is the global position of the vehicle on the map.

$$T = \begin{bmatrix} cos\alpha_c & -sin\emptyset_c & 0 & X_c \\ sin\alpha_c & cos\emptyset_c & 0 & Y_c \\ 0 & 0 & 1 & Z_c \\ 0 & 0 & 1 & 1 \end{bmatrix} \tag{1}$$

The waypoints are an array of points updated every timestep to keep enough information about potential obstacles, turns, or intersections.

3.3 Experiment 1 : DQN Algorithm
3.3.1 Experiment setup

To specifically use DQN to solve the Markov Decision Process, we need to redefine the action and reward functions.

i. Reward Function: Assuming the goal of the vehicle is to move as fast as possible while being in the centre of the lane, the reward function should give high rewards in proportion to the velocity and being as close as possible to the centre. On the other hand, the reward function should penalize the deviation from the centre of the lane. This framework has three terminal states: crossing the lane line, collision, and reaching the goal position. The reward function is given as follows:

$$R = \begin{cases} 100, & goal\ reached \\ -200, & on\ collision\ or\ lane\ crossing \\ \sum_t^{\infty} |v_t \cos\emptyset_t| - |v_t \sin\emptyset_t| - |v_t||d_t|, & (every\ episode) \end{cases} \tag{2}$$

ii. Action Space: The DQN algorithm does not handle continuous action spaces. Therefore, we need to discretize the action space to fit the algorithm. CARLA simulator accepts control commands for steering and braking in the range [0,1] and for steering in the range [-1,1]. To discretize the action space, we need to take some samples of the action space. The values for acceleration are [0.5], and the values for steering are [-0.4, -0.2, 0, 0.2, 0.4]. Instead of having many values in the original ranges, the total number of actions possible is 5.

3.3.2 Training

Due to limited resources, the DQN agent used for this experiment is a pre-trained model. It was trained for 9000 episodes. The vehicle used in training is a regular sedan (Tesla Model 3), the most common type and size of an autonomous vehicle, with multiple sensors, including collision and lane invasion sensors. The collision sensor and lane invasion are attached to the vehicle to detect collisions and lane crossings. The target network of this neural network is updated every five episodes. The replay memory size for this agent is 5000. The steps for training the DQN algorithm are as follows:

i. Initialize the size of replay memory to 5000.
ii. Actions are selected using the epsilon greedy strategy (epsilon decays over the episodes)
iii. Set the number of episodes to 9000 and start the training to iterate over the episodes
iv. When starting the episode, call the global planner to generate a random route between two points in the map

v. The state is fed to the neural network at each step to predict the actions.
vi. The actions are used to control the vehicle in the simulator
vii. Check the collision sensor and lane invasion sensor at every step to continue the episode
viii. Repeat until the training episodes end.

Figure 1, Figure 2 and Figure 3 show the average distance, average reward, and maximum reward, respectively, over the 9000 training episodes. These metrics are calculated over the most recent twenty episodes of training. The values increase as the episodes increase.

3.3.3 Validation

The agent was observed while running for 20 episodes. The testing occurred in two different towns in the CARLA simulator, town01 (training town) and town02. The agent is tested in multiple towns to test the agent's ability to generalize the policy learned over different environments. Figure 7 and Figure 8 shows the driving distance of the agent and the reward for every episode over 20 episodes. The episodes where the agent failed to reach the destination is due to a collision or lane crossing. The DQN agent has shown the ability to generalize to different environments. Figure 9 and Figure 10 show the agent's results in town 02.

3.4 Experiment 2 : DDPG Algorithm
3.4.1 Experiment setup

In contrast to DQN, DDPG does not require any modifications to the action space. The DDPG algorithm can handle continuous action spaces. As a result, the action space in this experiment is the original ranges accepted by CARLA which are [0,1] for brake and throttle and [-1,1] for steering. The reward function is the same as in the DQN experiment.

3.4.2 Training

The setup of this experiment is similar to the DQN in terms of the vehicle type and the sensors attached to the vehicle. The replay memory for this experiment is 100,000 experiences. It is higher than the DQN because DDPG algorithms deal with continuous state and action spaces, which means many values of states and actions are considered. So, we need a large experience memory for storing these continuous states and actions resulting from interacting with the environment. Figure 4 to Figure 6 shows that the DDPG agent reached the highest driving distance, average reward, and maximum reward in the early stages of the training. The values are not correlated with the training time. The DDPG algorithm can reach a suitable model in less training time.

3.4.3 Validation

The same testing strategy is used for this experiment. Figure 11 and Figure 14 show this test's progress regarding driving distance and rewards. It needs to improve its ability to generalize in different environments.

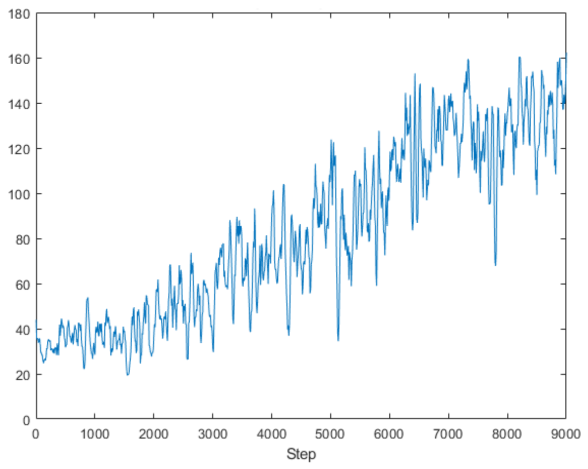

Fig. 1. DQN agent average driving distance

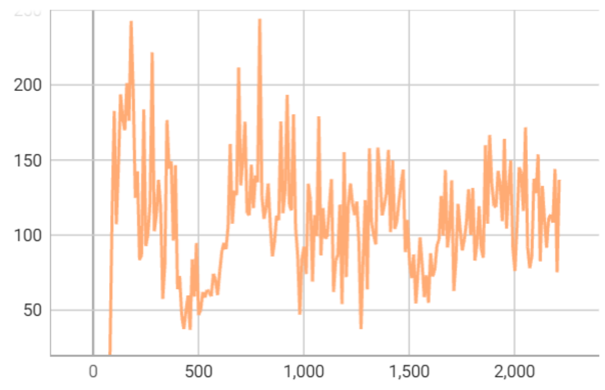

Fig. 4. DDPG agent average distance

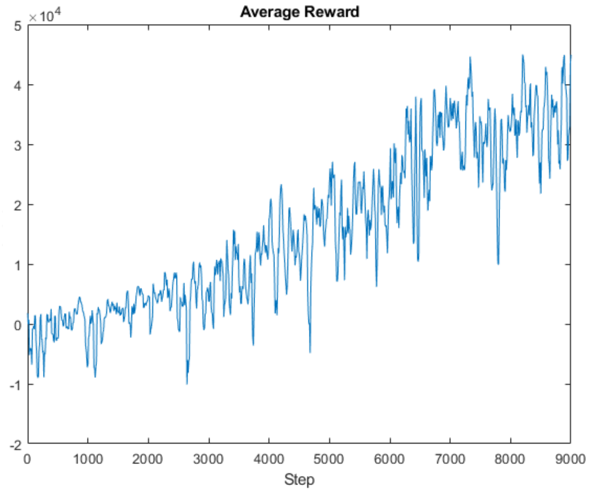

Fig. 2. DQN agent average reward

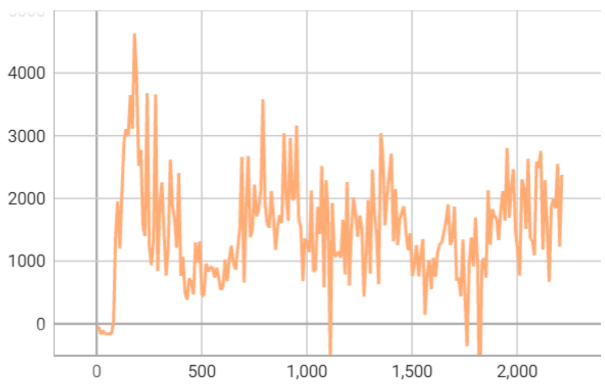

Fig. 5. DDPG agent average reward

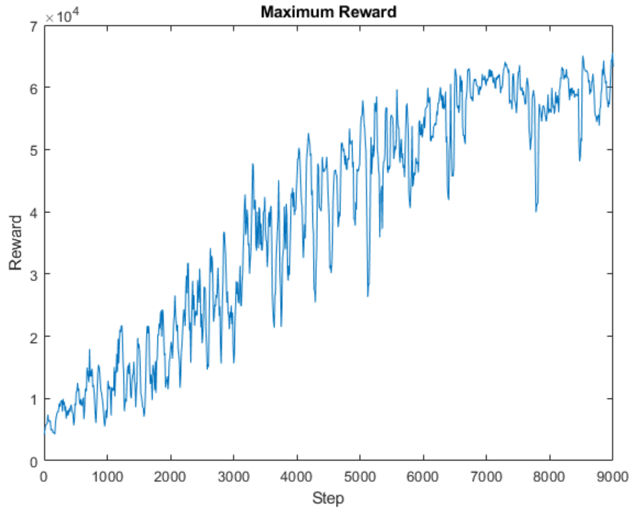

Fig. 3. DQN agent maximum reward

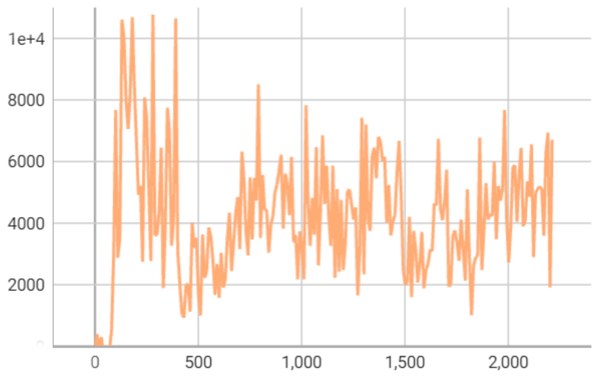

Fig. 6. DDPG agent maximum reward

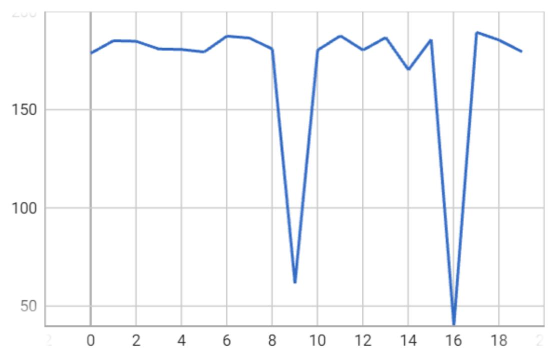

Fig. 7. DQN agent driving distance in town 01

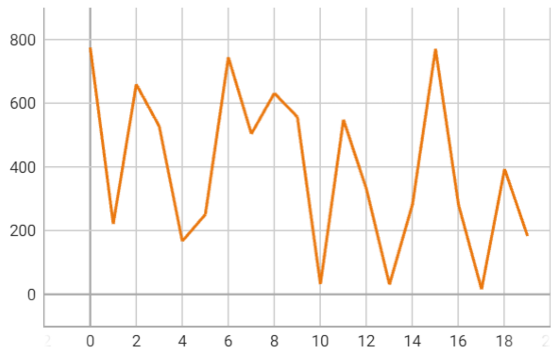

Fig. 11. DDPG agent driving distance in town 01

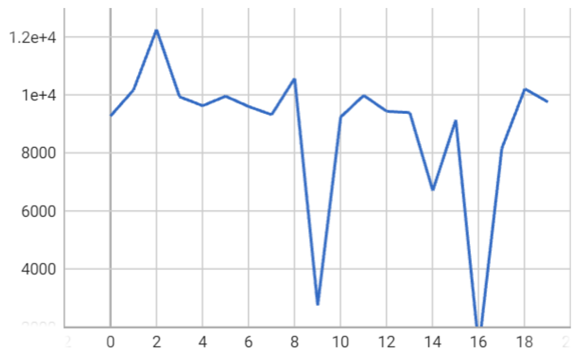

Fig. 8. DQN agent rewards in town 01

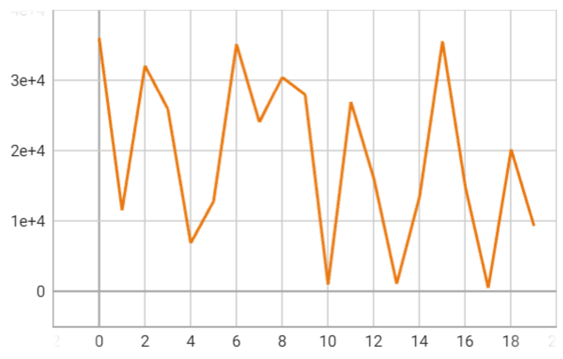

Fig. 12. DDPG agent reward in town 01

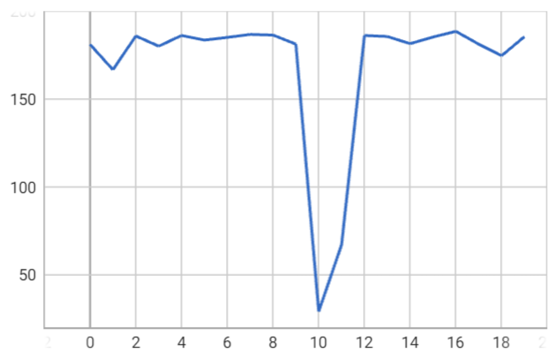

Fig. 9. DQN agent driving distance in town 02

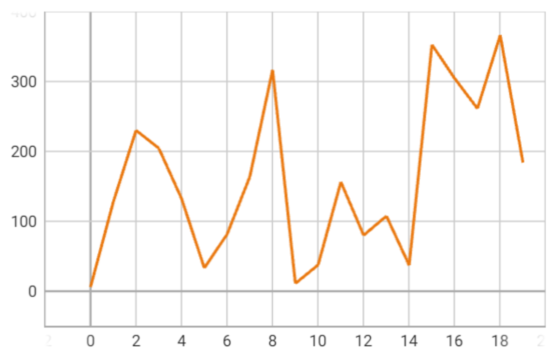

Fig. 13. DDPG agent driving distance in town 02

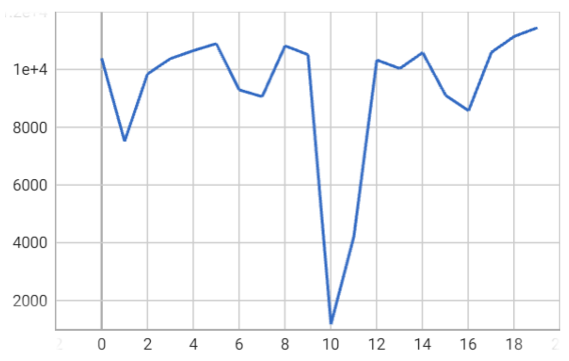

Fig. 10. DDPG agent reward in town 02

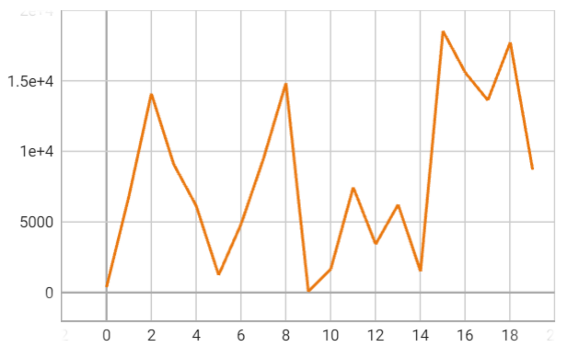

Fig. 14. DDPG agent reward in town 02

4. Results & Discussion

The experiments were conducted on a machine with the following specs: Intel(R) Xeon(R) CPU E5-2670 0 @ 2.60 GHz (2 processors), 32GB RAM, and NVIDIA Quadro K2200. The results were plotted using the Tensorboard Python library. The DQN agent was trained for 9000 episodes, while the DDPG agent was trained for only 2200 episodes. The DDPG algorithm collects more rewards during the episodes from testing the agents than the DQN agent. This may result from varying distances of the path planned by the global planner. However, the DDPG performs better actions in the environment as the agent stays at the centre of the lane and reaches high speeds. This results in a higher reward per step and, consequently, a higher reward per episode. On the other hand, the DQN algorithm achieved a higher success rate than the DDPG algorithm. Table 1 shows the success rate over two different 20 episode. Also, As observed during testing, the DDPG has overall better performance except for a specific state in the map which is a left turn at the corner of a street. The main reason for this problem is the sampling from the experience replay memory.

Table 1
Success rate over two different 20 episodes

Training Town1		Town2	
DQN	DDPG	DQN	DDPG
90%	75%	90%	55%
100%	75%	75%	60%

The replay memory may not contain enough samples of this state. This problem could be solved by fine-tuning the agent with the desired situations after training. In addition, different hyperparameters and hardware specifications may lead to a better model.

Table 2
Time comparison in (minutes: seconds) between DQN and DDPG algorithms

	Training Town1				Town2			
	DQN	Failed	DDPG	Failed	DQN	Failed	DDPG	Failed
Total Time	14:32		15:38		12:44		8:05	
Episode 0	00:48		01:21		00:39		00:04	F
Episode 1	00:29		00:31	F	00:43		00:22	
Episode 2	00:44		01:12		00:44		00:31	
Episode 3	00:44		00:59		00:44		00:28	
Episode 4	00:46		00:27	F	00:46		00:24	
Episode 5	00:45		00:34		00:42		00:12	F
Episode 6	00:46		01:17		00:39		00:17	F
Episode 7	00:43		01:03		00:41		00:24	
Episode 8	00:44		01:06		00:43		01:03	F
Episode 9	00:42		01:02		00:21	F	00:08	F
Episode 10	00:32		00:14	F	00:43		00:15	F
Episode 11	00:45		00:58		00:30	F	00:24	
Episode 12	00:44		00:43		00:41		00:15	
Episode 13	00:47		00:09	F	00:21	F	00:19	
Episode 14	00:44		00:38		00:38		00:13	F
Episode 15	00:43		01:24		00:30	F	00:42	
Episode 16	00:46		00:36		00:43		00:39	
Episode 17	00:47		00:08	F	00:24	F	00:35	
Episode 18	00:48		00:47		00:41		00:43	F
Episode 19	00:45		00:29		00:45		00:26	

5. Conclusions

The DDPG algorithm is better at handling continuous action spaces and has better transitions between actions while driving. The DQN is mainly aimed at tasks with discrete state and action spaces. The performance of the DQN is unacceptable in the real world, as the agent can only perform a limited number of actions while driving. In the simulation, the agent alternates between steering left and steering right, trying to find equilibrium to stay straight. In terms of generalization, both algorithms have their strengths and weaknesses. DQN has been shown to generalize well to new environments and tasks, as it learns a more general Q-value function that can be used to make decisions in new situations. However, DQN can struggle with continuous action spaces and be sensitive to hyperparameters. DDPG, on the other hand, can learn a more precise policy function that can generalize well to new situations, especially in continuous action spaces. However, DDPG can be more environmentally sensitive and may require more data to learn a good policy. With the appropriate training, DDPG is the better choice for autonomous driving.

Acknowledgement
This research was not funded by any grant. However, we are deeply thankful to all those who played a role in the success of this work. We are very grateful to Dr. Azza El-Sayed Ibrahim, the head of the central robotic lab in the Electronics Research Institute. Her experience was instrumental in shaping the direction of this work.

References
[1] Kiran, B. Ravi, Ibrahim Sobh, Victor Talpaert, Patrick Mannion, Ahmad A. Al Sallab, Senthil Yogamani, and Patrick Pérez. "Deep reinforcement learning for autonomous driving: A survey." *IEEE Transactions on Intelligent Transportation Systems* 23, no. 6 (2021): 4909-4926. https://doi.org/10.1109/TITS.2021.3054625

[2] Pendleton, Scott Drew, Hans Andersen, Xinxin Du, Xiaotong Shen, Malika Meghjani, You Hong Eng, Daniela Rus, and Marcelo H. Ang. "Perception, planning, control, and coordination for autonomous vehicles." *Machines* 5, no. 1 (2017): 6. https://doi.org/10.3390/machines5010006

[3] Samak, Chinmay Vilas, Tanmay Vilas Samak, and Sivanathan Kandhasamy. "Control strategies for autonomous vehicles." In *Autonomous driving and advanced driver-assistance systems (ADAS)*, pp. 37-86. CRC Press, 2021. https://doi.org/10.1201/9781003048381-3

[4] Wang, Fei-Yue. "AI and intelligent vehicles future challenge (IVFC) in China: From cognitive intelligence to parallel intelligence." In *2017 ITU Kaleidoscope: Challenges for a Data-Driven Society (ITU K)*, pp. 1-2. IEEE, 2017. https://doi.org/10.23919/ITU-WT.2017.8246841

[5] Urmson, Chris, Joshua Anhalt, Drew Bagnell, Christopher Baker, Robert Bittner, M. N. Clark, John Dolan *et al.,* "Autonomous driving in urban environments: Boss and the urban challenge." *Journal of field Robotics* 25, no. 8 (2008): 425-466. https://doi.org/10.1002/rob.20255

[6] Hoel, Carl-Johan, Katherine Driggs-Campbell, Krister Wolff, Leo Laine, and Mykel J. Kochenderfer. "Combining planning and deep reinforcement learning in tactical decision making for autonomous driving." *IEEE transactions on intelligent vehicles* 5, no. 2 (2019): 294-305. https://doi.org/10.1109/TIV.2019.2955905

[7] Leong, Eric. "Bridging the Gap Between Modular and End-to-end Autonomous Driving Systems." (2022).

[8] Tampuu, Ardi, Tambet Matiisen, Maksym Semikin, Dmytro Fishman, and Naveed Muhammad. "A survey of end-to-end driving: Architectures and training methods." *IEEE Transactions on Neural Networks and Learning Systems* 33, no. 4 (2020): 1364-1384. https://doi.org/10.1109/TNNLS.2020.3043505

[9] Jamil, Amirah Hanani, Fitri Yakub, Azizul Azizan, Shairatul Akma Roslan, Sheikh Ahmad Zaki, and Syafiq Asyraff Ahmad. "A Review on Deep Learning Application for Detection of Archaeological Structures." *Journal of Advanced Research in Applied Sciences and Engineering Technology* 26, no. 1 (2022): 7-14. https://doi.org/10.37934/araset.26.1.714

[10] Paleja, Rohan, Yaru Niu, Andrew Silva, Chace Ritchie, Sugju Choi, and Matthew Gombolay. "Learning Interpretable, High-Performing Policies for Autonomous Driving." *arXiv preprint arXiv:2202.02352* (2022). https://doi.org/10.15607/RSS.2022.XVIII.068

[11] Carton, Florence. "Exploration of reinforcement learning algorithms for autonomous vehicle visual perception and control." PhD diss., Institut polytechnique de Paris, 2021.

[12] Szewczyk, Alexander Stensland Iversen. "AI-agents Trained Using Deep Reinforcement Learning in the CARLA Simulator." Master's thesis, NTNU, 2022.

[13] Yang, Caipei, Yingqi Zhao, Xuan Cai, Wei Wei, Xingxing Feng, and Kaibo Zhou. "Path Planning Algorithm for Unmanned Surface Vessel Based on Multiobjective Reinforcement Learning." *Computational Intelligence and Neuroscience* 2023 (2023). https://doi.org/10.1155/2023/2146314

[14] Wang, Sung-Jung, and SK Jason Chang. "Autonomous bus fleet control using multiagent reinforcement learning." *Journal of Advanced Transportation* 2021 (2021): 1-14. https://doi.org/10.1155/2021/6654254

[15] Vergara, Marcus Loo. "Accelerating training of deep reinforcement learning-based autonomous driving agents through comparative study of agent and environment designs." Master's thesis, NTNU, 2019.

[16] Jang, Beakcheol, Myeonghwi Kim, Gaspard Harerimana, and Jong Wook Kim. "Q-learning algorithms: A comprehensive classification and applications." *IEEE access* 7 (2019): 133653-133667. https://doi.org/10.1109/ACCESS.2019.2941229

[17] Arulkumaran, Kai, Marc Peter Deisenroth, Miles Brundage, and Anil Anthony Bharath. "A brief survey of deep reinforcement learning." *arXiv preprint arXiv:1708.05866* (2017). https://doi.org/10.1109/MSP.2017.2743240

[18] Guo, Siyu, Xiuguo Zhang, Yisong Zheng, and Yiquan Du. "An autonomous path planning model for unmanned ships based on deep reinforcement learning." *Sensors* 20, no. 2 (2020): 426. https://doi.org/10.3390/s20020426

[19] Lillicrap, Timothy P., Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. "Continuous control with deep reinforcement learning." *arXiv preprint arXiv:1509.02971* (2015).

[20] Malik, Sumbal, Manzoor Ahmed Khan, and Hesham El-Sayed. "Carla: Car learning to act—an inside out." *Procedia Computer Science* 198 (2022): 742-749. https://doi.org/10.1016/j.procs.2021.12.316

[21] Dosovitskiy, Alexey, German Ros, Felipe Codevilla, Antonio Lopez, and Vladlen Koltun. "CARLA: An open urban driving simulator." In *Conference on robot learning*, pp. 1-16. PMLR, 2017.

[22] Farag, Wael, and Zakaria Saleh. "Behavior cloning for autonomous driving using convolutional neural networks." In *2018 International Conference on Innovation and Intelligence for Informatics, Computing, and Technologies (3ICT)*, pp. 1-7. IEEE, 2018. https://doi.org/10.1109/3ICT.2018.8855753

[23] Chen, Jianyu, Bodi Yuan, and Masayoshi Tomizuka. "Deep imitation learning for autonomous driving in generic urban scenarios with enhanced safety." In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 2884-2890. IEEE, 2019. https://doi.org/10.1109/IROS40897.2019.8968225

[24] Silver, David, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert *et al.,* "Mastering the game of go without human knowledge." *nature* 550, no. 7676 (2017): 354-359. https://doi.org/10.1038/nature24270

[25] Elallid, Badr Ben, Nabil Benamar, Nabil Mrani, and Tajjeeddine Rachidi. "DQN-based Reinforcement Learning for Vehicle Control of Autonomous Vehicles Interacting With Pedestrians." In *2022 International Conference on Innovation and Intelligence for Informatics, Computing, and Technologies (3ICT)*, pp. 489-493. IEEE, 2022. https://doi.org/10.1109/3ICT56508.2022.9990801

[26] Park, Minjae, Seok Young Lee, Jin Seok Hong, and Nam Kyu Kwon. "Deep Deterministic Policy Gradient-Based Autonomous Driving for Mobile Robots in Sparse Reward Environments." *Sensors* 22, no. 24 (2022): 9574. https://doi.org/10.3390/s22249574