A Novel Approach to Autonomous Driving Using DDQN-Based Deep Reinforcement Learning

Ahmed Khlifi *, Mohamed Othmani *, Monji Kherallah *

Posted Date: 8 January 2025

doi: 10.20944/preprints202501.0617.v1

Keywords: deep reinforcement learning; neural networks; Autonomous driving; DQN; DDQN; CARLA simulator

Article

A Novel Approach to Autonomous Driving Using DDQN-Based Deep Reinforcement Learning

Ahmed Khlifi [1,2,*], Mohamed Othmani [3,4] and Monji Kherallah [5,6]

[1] National Engineering School of Sfax, University of Sfax, Sfax, Tunisia
[2] Advanced Technologies for the Environment and Smart Cities ATES, Faculty of Sciences of Sfax, University of Sfax, Sfax, Tunisia
[3] Faculty of Sciences of Gafsa, University of Gafsa, Gafsa, Tunisia
[4] Advanced Technologies for the Environment and Smart Cities ATES, Faculty of Sciences of Sfax, University of Sfax, Sfax, Tunisia
[5] Faculty of Sciences of Sfax, University of Sfax, Sfax, Tunisia
[6] Advanced Technologies for the Environment and Smart Cities ATES, Faculty of Sciences of Sfax, University of Sfax, Sfax, Tunisia
* Correspondence: ahmedkhlifi@ymail.com (A.K.); mohamed.othmani@yahoo.fr (M.O.); Monji.kherallah@fss.usf.tn (M.K.)

Abstract: Deep reinforcement learning (DRL) trains agents to make decisions by learning from rewards and penalties, using trial and error. It combines reinforcement learning with deep neural networks, enabling agents to process large datasets and learn from complex environments. DRL has achieved notable success in gaming, robotics, decision-making, etc. However, real-world applications, such as self-driving cars, face challenges due to complex state and action spaces, requiring precise control. Researchers continue to develop new algorithms to improve performance in dynamic settings. A key algorithm, Deep Q-Network (DQN), uses neural networks to approximate the Q-value function, but suffers from overestimation, leading to suboptimal outcomes. To address this, Double Deep Q-Network (DDQN) was introduced to reduce bias by separating action selection from evaluation, resulting in more stable learning. This work examines the effectiveness of DQN and DDQN in autonomous driving using the CARLA simulator, highlighting DDQN's benefits in reducing bias and enhancing policy performance.

Keywords: deep reinforcement learning; neural networks; Autonomous driving; DQN; DDQN; CARLA simulator

1. Introduction

Autonomous Driving (AV) [1] has been a topic of interest for many years, with various advancements being made to improve its safety and efficiency. However, developing a fully autonomous system presents several challenges, such as the need for real-time decision-making and adaptability to changing environments. One of the most promising technologies used in this field is Deep Reinforcement Learning, a type of machine earning that uses neural networks and reward-based training to enable decision-making [2]. Despite the promising advancements, deep reinforcement learning also presents challenges and ethical implications [3]. For example, it raises questions about responsibility if an autonomous vehicle causes. an accident. Who is at fault? The manufacturer, the software developer, or the vehicle owner? Moreover, there are concerns about the impact of autonomous driving on employment and the economy. Nonetheless, the potential benefits of DRL in this field are vast, and the technology continues to evolve. For instance, developing more advanced neural networks and integrating other technologies, such as computer vision and natural language processing, can further enhance the decision-making process of autonomous systems.

 doi:10.20944/preprints202501.0617.v1

Deep reinforcement learning enhances autonomous driving systems by allowing agents to learn from their mistakes and improve decision-making over time [4]. For example, an agent can receive a reward for staying within the speed limit and a penalty for breaking it [5]. This reward-based training helps the agent learn from past experiences, making adapting to different road and weather conditions possible [6]. As a result, DRL has the potential to revolutionize autonomous driving by enhancing the system's ability to respond effectively to diverse situations.

This work aims to assess autonomous driving tasks in urban environments using DQN agents. To achieve this, several approaches based on DQN agents will be investigated. The DQN agent learns a policy, set of behaviors, for lane-following tasks by applying visual and driving characteristics obtained from in-vehicle sensors and a trajectory planner based on a model [7, 8, 9]. This method comprehensively analyzes how deep reinforcement learning is transforming autonomous driving technology.

An end-to-end autonomous system based on the Deep Q-Learning algorithm [10] offers several advantages over traditional approaches. Its simplicity lies in the seamless integration of perception, prediction, and planning into a unified model that can be trained together. Our study proposes an RL-based autonomous driving system emphasizing more informative exploration and improved reward signaling. We will evaluate the performance of this system in urban environments using the DDQN approach combined with Long Short-Term Memory (LSTM) [11].

We will develop an intelligent driving agent capable of navigating complex environments along predetermined routes [12, 13], such as those in the CARLA simulator [14] to validate our approach. We will also analyze various design decisions to determine the best configurations for training autonomous driving agents using reinforcement learning. Additionally, we will demonstrate the training methods that can significantly impact the performance of a deep RL agent. Finally, we will validate our approach in various challenging traffic scenarios and show that our method outperforms previous state-of-the-art techniques.

2. Overview of Deep Reinforcement Learning in Autonomous Driving

2.1. Reinforcement Learning

RL [15] is a key artificial intelligence technique for tackling complex problems like robotics, industry automation, Natural Language Processing, and autonomous driving. In autonomous driving, RL trains vehicles to make decisions in dynamic environments. The vehicle, the agent, interacts with its surroundings, including roads, traffic, and pedestrians, by taking actions such as steering, accelerating, or braking. Each action earns rewards or penalties based on its safety and effectiveness, guiding the vehicle to improve over time.

Autonomous driving with RL involves balancing exploration, i.e. trying new maneuvers, and exploitation, i.e. using proven strategies, to maximize rewards [16]. Simulations play a vital role in training RL models safely before real-world deployment, addressing the challenges of complex, dynamic environments. Unlike supervised learning, where explicit examples are provided, RL relies on trial and error to optimize actions, using feedback to refine strategies and ensure safe, efficient performance. The general agent-environment interaction process in RL is shown in Figure 2.

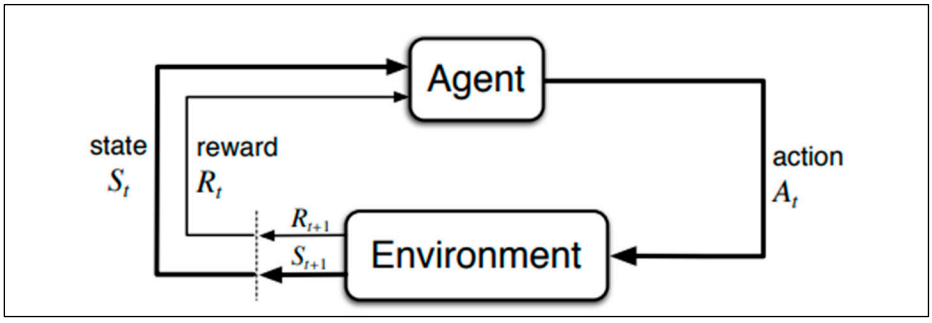

Figure 1. The agent–environment interaction in reinforcement learning [17].

 doi:10.20944/preprints202501.0617.v1

The learning and decision-making in RL critically involve an interaction between the agent and the environment. A_t a given time step t, the environment is in a certain state, S_t, from which the agent takes an action, A_t, according to its policy, which has assigned a strategy to map states to actions. The execution of action A_t causes the environment to transition into the next state, S_{t+1}, and gives the agent a reward R_t. The agent then observes this reward and new state, S_{t+1}. On the next time step, t+1, an updated action, A_{t+1} is taken by the agent in response to its updated state. It goes on repeatedly that an agent selects an action, receives a reward, and observes the next state. It tries to maximize rewards over time, which can be measured regarding a sum of rewards, discounted rewards, or another metric concerning long-term benefit. This then creates an iterative feedback loop through which the agent can improve its policy and further develop its strategy in making decisions.

2.2. Deep Reinforcement Learning

DRL [18] has become a key approach for training autonomous vehicles to make complex decisions in dynamic environments. By combining reinforcement learning and deep learning, DRL allows vehicles to navigate, plan, and control effectively while handling tasks like lane changes, intersections, and obstacle avoidance. Training often occurs in simulated environments, enabling safe exploration and refinement of strategies. Using reward systems, DRL optimizes decision-making by encouraging safe behaviors and penalizing unsafe actions [19]. Techniques like convolutional neural networks process sensory data to enhance perception and planning. Despite its promise, challenges remain, such as ensuring generalization to new scenarios, improving safety mechanisms, and addressing computational efficiency. DRL's hierarchical structures help tackle complex traffic scenarios, improving speed, trajectory, and collision avoidance, while algorithms like DQN are evaluated for maneuver planning. Simulations significantly reduce real-world testing needs, minimizing risks and enhancing algorithm performance.

2.3. Deep Q-Networks (DQN)

DQNs [20] combine Q-learning with deep neural networks to enable reinforcement learning agents to handle high-dimensional state spaces. By approximating the Q-value function using a neural network, DQNs allow agents to learn optimal policies directly from raw inputs, such as images or sensor data. This approach has been successfully applied in various domains, including playing video games, robotics, and autonomous driving, showcasing its ability to solve complex decision-making problems.

2.4. Double Deep Q-Networks (DDQN)

DDQN is an enhancement to the original DQN algorithm designed to address the overestimation bias in Q-learning, which arises when the same network is used to select and evaluate actions, leading to inflated Q-values [21]. Unlike DQN, DDQN employs two separate networks: an online network for action selection and a target network for Q-value evaluation. This decoupling reduces bias, improves learning stability, and enhances performance in complex environments by ensuring more accurate Q-value updates.

3. Benchmarking - Urban Driving Simulator

Testing autonomous systems in real-world conditions is not only expensive but also risky. For this reason, simulation represents an effective alternative to these practical tests. With simulators, developers can quickly create and evaluate prototypes, productively iterate on their designs, and explore a wide range of scenarios—all at a fraction of the cost of real-world testing. In addition, simulators offer precise tools for measuring performance, enabling in-depth analysis and optimization of autonomous driving algorithms. This method not only accelerates development but also improves safety by facilitating comprehensive testing in controlled virtual environments.

CARLA, (Car Learning to Act) [22, 23, 24] is an open-source simulator for urban driving specifically designed for the study of autonomous vehicles. Built from the ground up to simplify the

 doi:10.20944/preprints202501.0617.v1

creation, training, and evaluation of autonomous driving systems in urban environments. CARLA provides tools to perform detailed simulations and evaluate system performance. In addition to its open-source code and protocols, CARLA offers open digital assets, such as urban layouts, buildings, and vehicles, that users can use independently. Deep customization of sensor sets, and environmental conditions is possible through the platform, enabling accurate simulation.

CARLA provides a powerful API that allows the users to control all aspects related to the simulation and permits them to configure diverse sensors for environment perception (cameras, LiDAR, GPS, etc). This way, CARLA provides environment and
ego-vehicle data. With this data, monocular camera image, and vehicle position and speed, it is possible to obtain visual and driving features. These two feature sets can be integrated into a deep reinforcement learning agent to generate actions in the form of control commands.

Each town in CARLA possesses its own unique characteristics. Towns (Figure 2) was utilized as the primary platform for training our agent, and the performance evaluation.

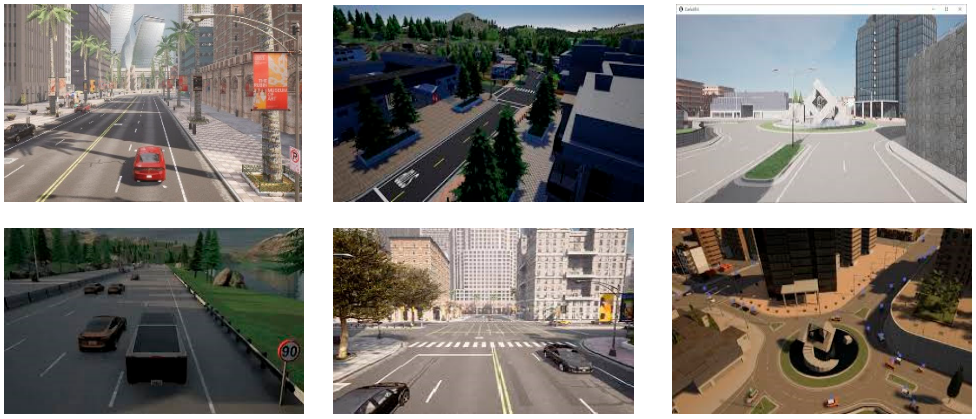

Figure 2. CARLA driving Simulator Screenshots.

4. Related work

In recent years, Deep Reinforcement Learning (DRL) has achieved great success in the field of autonomous vehicles. Due to this great success, several researchers have chosen to use it in their research. Their main challenge is to develop an intelligent agent that can on the one hand avoid obstacles and mitigate collisions of autonomous vehicles, and on the other hand correct errors from autonomous driving pipeline tasks such as decision-making and motion planning. In this context, several approaches have used the DQN-based DRL algorithm [25] that has demonstrated great effectiveness in ensuring safe navigation in various simulated dynamic environments, namely CARLA.

Elallid et al. 2022 [26] present an approach that uses the CARLA simulator that is designed to imitate real-world streets to train and validate autonomous vehicles. This employed method for controlling an AV in complicated environments is based on a DQN model. The car has been equipped with a front-facing camera to take real-time pictures. The captured photos, which were originally 640 x 480 pixels in RGB, are first converted to grayscale levels and then resized to 192 x 256 pixels. These processed images are then passed through two dense couches with 512 and 256 neurons, respectively, enabling the model to generate 190 alternative actions. In the CARLA environment, a 389-meter course with right turns and intersections has been designed. Ten pedestrians and five vehicles were added to make the environment more dynamic and realistic. The model was trained for 5000 episodes, with a mini-batch size of 16, using a repetition memory system to learn from past experiences. The results show that the model learns effectively across episodes: average rewards increase as the success rate of actions improves, and the collision rate gradually decreases until reaching almost zero. This demonstrates that the AV learns to avoid accidents with other vehicles and pedestrians present in the environment, ensuring safe driving.

 doi:10.20944/preprints202501.0617.v1

Hossain et al (2023) [27] have proposed a model based on a deep neural network to implement the DQN algorithm, used to approximate the Q-value function. This function evaluates the quality of each possible action in each state. The agent, representing the autonomous car, interacts with a simulated environment, where it receives observations, chooses actions, and learns to maximize cumulative rewards over time. The model architecture consists of several layers designed to process observations of the environment and produce the Q-values associated with each possible action. The observations, constituting the input space, are represented by a 5x5 array describing the vehicles in the vicinity of the autonomous car. Each row of the table corresponds to a vehicle, with columns indicating the following characteristics: position (x, y) and speed (Vx, Vy). This information is processed by the neural network to determine the optimal action. The neural network is a Multi-Layer Perceptron (MLP) comprising several fully connected layers. These layers learn to identify complex relationships between vehicles, such as neighborhood and relative speed, to assess the quality of possible actions. Activation functions such as ReLU (Rectified Linear Unit) are used to introduce non-linearity and enable the model to better approximate the Q function. As an output, the network produces a vector of Q values for each possible action in the environment. The action space comprises 5 possible actions: change lanes to the left (LANELEFT), stay put (IDLE), change lanes to the right (LANERIGHT), accelerate (FASTER), and slow down (SLOWER). Each Q value represents the quality of the associated action in the current state of the environment. The agent then selects the action with the highest Q value, corresponding to that which maximizes the expected cumulative reward. The reward function is designed to encourage fast and safe driving behavior. Thus, the agent receives a reward of 0.1 points when staying in the right lane, and a reward of 0.4 points when maintaining a high speed. On the other hand, there is no specific penalty or incentive for lane changes, which therefore do not directly affect the reward (0 points). For each action performed, the agent receives these rewards, which incentivizes it to adopt a behavior that maximizes speed while avoiding collisions.

Tammewar et al. 2023 [28] studied the improvement of autonomous driving performance using DRL. The approach involves training a simulated vehicle to navigate autonomously on a racing track using the DQN algorithm. The system uses the CarRacing-v2 simulator which provides a top view of a randomly generated track. The vehicle receives visual information from a front camera and interacts with the environment via actions. The built model receives input images from the front camera, represented as RGB pixels (96 x 96 pixels in this case). The images are subsequently converted to grayscale to reduce computational complexity and focus the model's attention on important structural aspects of the image, such as the contours of the track. These images are used to capture information about the vehicle’s environment (speed, direction, etc.). The inputs are then processed CNNs to extract relevant features. To capture temporal dependencies and understand the dynamics of vehicle movement across images, a recurrent neural network (LSTM) is used after the CNN layers. The goal of this part is to allow the model to retain information over multiple time steps, and to adjust its actions based on past trajectories. Rewards are assigned based on the coverage of track tiles, while penalties are applied when the vehicle goes off track. As an output, the model chooses among possible actions (acceleration, braking, steering) based on the extracted features. In the continuous version, the actions are represented by three parameters: direction (from -1 to 1), acceleration (from 0 to 1), and braking (from 0 to 1). These actions aim to help the vehicle navigate the track while maximizing the reward obtained. The results show that the DQN algorithm with epsilon decay (ε-decay) performed well and provided excellent stability and efficiency as well as cumulative scores over episodes for the autonomous navigation task.

In these approaches, although the agents have been tested in various simulation environments, their performance may not generalize to other real-world environments with very different driving scenarios. Likewise, the policies learned may be too specialized for the specific conditions of the simulation (traffic, weather, road infrastructure), which limits their applicability in varied real-life situations.

 doi:10.20944/preprints202501.0617.v1

5. Methodology

The main methodology of this work is to introduce a novel approach based on deep reinforcement learning, which will enable a car to drive autonomously in a virtual environment. Since the system is based on a computer-generated environment, the CARLA_0.9.13 simulator for autonomous cars is the environment used. Our research focused on the impact of various hyperparameters. The effects on convergence and robustness of learning were studied using learning rates for each model. Our main performance measures were the consistency of training each model over a given number of episodes, with episode reward and learning stability as the primary measures. A controlled and reproducible comparison between models was facilitated by this approach, ensuring that any observed performance disparities were related to the intrinsic characteristics of the models and the chosen hyperparameters.

5.1. Proposed Model

In this work, we propose a novel architecture that combines CNN [29], LSTM [30,31], and a Deep Q-learning with a DDQN to tackle reinforcement learning tasks. This hybrid model leverages CNNs to extract spatial features from image data and LSTMs to capture the temporal dependencies in sequential data, making it particularly well-suited for environments where inputs are image sequences (e.g., video frames).

The integration of DDQN enhances the reinforcement learning component by addressing the overestimation bias common in standard Q-learning [32]. This allows the model to make more stable and accurate decisions in dynamic and complex environments. To prevent overfitting, Dropout is applied within the convolutional layers, which is crucial when working with high-dimensional input data and limited training samples.

The architecture is modular and flexible, enabling configuration of key parameters such as the number of LSTM layers, hidden units, and CNN filters to adapt to various task complexities. By combining the strengths of CNNs, LSTMs, and DDQN, this approach presents a robust and efficient solution for reinforcement learning tasks involving sequential image data. The model's structure is shown in Figure 3.

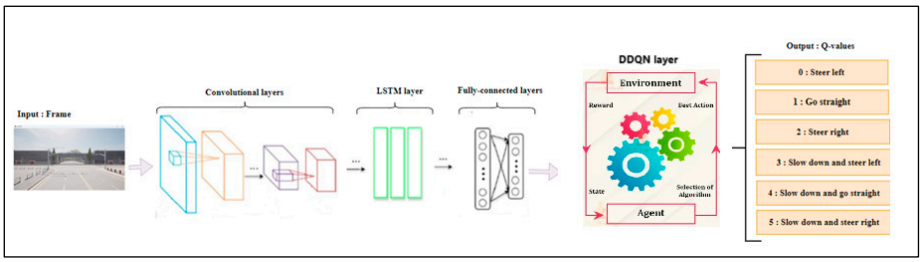

Figure 3. Proposed model architecture.

5.2. Model Architecture and State Space

In our scenario, we process a stack of four RGB images captured by the front camera of the autonomous vehicle (AV). Initially, each images have 640 × 480 × 3 pixels. We resize them to 84 × 84 × 3 pixels, then convert them to grayscale. This transformation yields a new state, denoted as S_t, with dimensions of 84 × 84 × 1, which are fed into the input of the neural network. The model combines convolutional layers with an LSTM, followed by fully connected layers for final predictions. By correcting the overestimation of action values that can occur in the original algorithm, DDQN enhances the conventional Q-Learning algorithm.

The Q-values of each action in each state, which stand for the anticipated future benefit of performing that action in that state, are learned by the algorithm in typical
Q-Learning. However, when function approximation is used, which occurs frequently in large state spaces, these Q-values might become overstated. As a result, less-than-ideal decisions may be made.

 doi:10.20944/preprints202501.0617.v1

By creating a second network that is used to choose the actions to be executed, DDQN resolves this problem. While the secondary network is used to assess the Q-value of the selected action, the primary network is utilized to estimate the Q-values of each action in each state. To reduce the overestimation of Q-values, the idea is to decouple the selection and evaluation of activities.

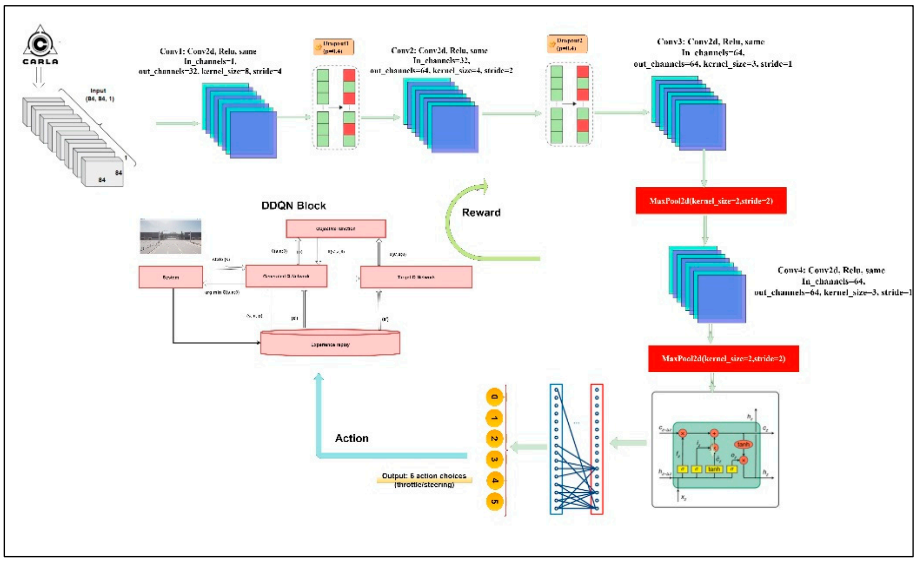

Figure 4. The proposed CNN-LSTM-DDQN architecture.

Our architecture is mainly based on multiple layers such as convolutional layers. The number of convolutional layers is four as shown in the architecture figure. The first convolutional layer applies 32 convolutional filters with a kernel size of 8x8 and a stride size of 4. It reduces the spatial dimensions of the input while extracting the initial feature maps. A dropout with a probability of 0.4 is applied after the first convolutional layer to mitigate overfitting. The second convolutional layer uses 64 convolutional filters with a kernel size of 4x4 and a stride size of 2 to extract more features and reduce the spatial dimensions. A dropout with a probability of 0.4 is applied after the second convolutional layer. For the third convolutional layer, 64 convolutional filters with a kernel size of 3x3 and a stride size of 1 are used to add complexity to feature extraction. Two Maxpooling layers, each with a kernel size and stride of 2, are used to further downsample the feature maps and capture the most salient features. A fourth convolutional layer with 64 filters, a kernel size of 3x3, and a stride of 1 is used to refine the features.

After the convolution and pooling operations, the spatial dimensions are flattened to prepare the data for the LSTM layer. The tensor is reshaped from (batch_size * seq_len, c, h, w) to (batch_size, seq_len, -1), where -1 automatically calculates the size of the flattened feature.Afterward, an LSTM layer captures the temporal dependencies in the sequence of image frames. It consists of hidden units lstm_hidden_size and layers num_lstm_layers. The input_size is the flattened size of the output of the convolutional layers. Next, we find the first fully connected layer that transforms the LSTM output into a 512-dimensional vector. Finally, a last fully connected layer maps the 512-dimensional vector to a vector of size num_actions, representing the Q values for each possible action in the reinforcement learning task.

For the forward pass, the input x is processed through the convolutional layers, followed by the LSTM layer to capture temporal dependencies, and finally through the fully connected layers to produce the Q values. In this model, we aim to design a robust architecture for reinforcement learning tasks involving image sequences, exploiting the strengths of convolutional and recurrent neural networks to process and learn from complex temporal data. The architecture of our system is a DDQN with the following inputs and outputs shown in the following table:

 doi:10.20944/preprints202501.0617.v1

Table 1. Architecture of proposed system.

Layer	Input Dimensions	Output Dimensions	Activation	Dropout	Notes
Convolutional 1	84x84x1	32x42x32	ReLU	Yes (0.4)	8x8 kernel, stride 4
Convolutional 2	32x42x32	64x21x64	ReLU	Yes (0.4)	4x4 kernel, stride 2
Convolutional 3	64x21x64	64x10x64	ReLU	No	3x3 kernel, stride 1
Max Pooling 1	64x10x64	64x5x64	N/A	No	2x2 kernel, stride 2
Convolutional 4	64x5x64	64x4x64	ReLU	No	3x3 kernel, stride 1
Max Pooling 2	64x4x64	64x2x64	N/A	No	2x2 kernel, stride 2
LSTM	64x2x64 (flattened)	256 hidden units	N/A	No	1 layer
Fully Connected 1	256	512	ReLU	No	Dense layer
Fully Connected 2	512	6	N/A	No	Action output layer

5.3. Experiments

The reward function in the provided code is structured to guide the agent's behavior based on its interactions with the environment. The primary factors influencing the reward are collisions, the duration of the episode, and whether the agent avoids obstacles during its driving task. The reward function in this method is based on several specific scenarios, described in our approach. Here are the details of the different cases in this driving simulation: Firstly, in the event of a collision, detected by the presence of items in the collision_hist list, the reward is set to -20, reflecting a severe penalty for this incident. The episode immediately ends with done equal to True, marking the end of the episode following an accident. Then if no collision occurs and the car continues to run smoothly, a+5 reward is awarded. This encourages collision-free driving, and the episode continues with done equal to False, allowing the agent to continue the episode without interruption. Finally, if the episode lasts longer than 30 seconds, a significant reward of 250 is awarded. The episode ends with a done equal to True, reflecting a bonus for driving through the entire episode without major incident. This scenario offers an additional incentive e to maintain prolonged, safe driving. In summary, this reward structure strongly penalizes collisions while promoting continuous, accident-free driving, enabling the agent to learn to avoid collisions and drive stably. Here is a summary of the reward function in pseudo-code form:

$$reward = \begin{cases} -20 & if\ collision \\ 250 & if\ maximum\ time\ reached \\ +5 & otherwise \end{cases} \quad (1)$$

This encourages the agent to avoid collisions and continue driving until the episode ends.

5.4. Mounted Sensors and Hyperparameters

Sensors mounted on autonomous vehicles play a crucial role in their ability to perceive and understand their environment in real-time, to make safe and efficient decisions. To achieve this, several types of sensors are integrated into autonomous vehicles, each providing specific data to analyze different aspects of the environment. The following table introduces the types of data commonly captured by these sensors and their respective functions:

 doi:10.20944/preprints202501.0617.v1

Table 2. Data for Sensors mounted on the Autonomous Vehicle.

Sensor	Usage	Specific Attributes	Location of the Vehicle	Function
RGB Camera	Images are resized to 84x84 pixels and converted to grayscale to be processed by the CNN-LSTM model. Used for autonomous decision-making.	Resolution: 640x480 pixels (modifiable). Field of View (FOV): 110 degrees	Mounted at the front, coordinates: x=2.5, z=0.7	Captures color images for visual perception of the environment.
Collision Sensor	Detects vehicle collisions with the environment and logs these events.	Logs collisions in a collision_hist list. Negative reward assigned in case of collision, with a penalty of -20 points.	Attached to the vehicle (exact position unspecified).	Used to evaluate safety during training, penalizing unsafe behavior and preventing crashes.

This table outlines the two sensors used in our simulation, with details on their attributes, functions, and roles in controlling the autonomous vehicle.

It is also necessary to define the set of various hyperparameters that that serve as outside configuration variables utilized to control model training. Those hyperparameters determine important model properties like design, learning rate, and complexity.

Here's a detailed table of the hyperparameters:

Table 3. Data for Sensors mounted on the Autonomous Vehicle.

Hyperparameters	Value	Description
SHOW_PREVIEW	False	Controls whether the front camera preview is shown during the simulation.
IM_WIDTH	640	Width of the camera image.
IM_HEIGHT	480	Height of the camera image.
SECONDS_PER_EPISODE	30	Maximum time (in seconds) for each episode in the environment.
MIN_REWARD	-200	Minimum reward threshold to consider for episode termination.
STEER_AMT	1.0	Steering amount (how much the vehicle turns when an action is taken).
num_frames	8	Number of image frames to stack for input to the neural network.
Gamma	0.99	The discount factor is used in Q-learning to calculate the future expected rewards.

 doi:10.20944/preprints202501.0617.v1

Batch_Size	32	Number of transitions to sample from the replay buffer in each training iteration.
Buffer_Size	5,000,000	Maximum size of the replay buffer (number of stored transitions).
Min_Replay_Size	100,000	Minimum number of transitions to collect before starting training.
Episodes	2000	Total number of episodes to run the training loop.
Epsilon	1.0	Initial exploration rate for ε-greedy policy (controls how often random actions are taken).
min_epsilon	0.0001	The minimum value to which epsilon can decay.
Decay	(min_epsilon/epsilon)**(1/episodes)	Decay factor for reducing epsilon after each episode.
lstm_hidden_size	256	Number of hidden units in the LSTM layer.
num_lstm_layers	1	Number of layers in the LSTM.
Optimizer	Adam	Optimization algorithm used for training the neural network.
learning_rate	5e-4	The learning rate for the Adam optimizer.
Dropout	0.4	Dropout rate used in the CNN layers to prevent overfitting.
target_net_update_freq	Every 4 episodes	Frequency of updating the target network with the weights from the online network.
max_pool_kernel_size	2	Kernel size for max-pooling layers in the CNN.
reward_success	250	The reward is assigned when the vehicle completes an episode without collision within the time limit.
reward_collision	-20	A reward penalty is assigned when a collision occurs.
reward_step	5	The reward for each successful step taken without a collision.

 doi:10.20944/preprints202501.0617.v1

These hyperparameters control various aspects of the neural network architecture, reinforcement learning algorithm, and training dynamics.

5.5. Action Space

Our model inputs forward-facing RGB camera images from the CARLA simulator. Each image is converted to grayscale and resized to 84x84 for processing. The output of our system is actions: 6 possible actions (combinations of steering and throttle). In the Carla simulator environment, the AV interacts with its environment using four main control commands: Steer left, Steer right, Go straight, and Slow down. These commands are represented as integers values in the range 0 to 5. DDQN is a discrete DRL algorithm, the agent must make discrete action choices as per Table 4. The action is to be taken by car. The possible actions are:

Table 4. Actions and their Corresponding Values.

Actions	Control Commands
0	Steer left
1	Go straight
2	Steer right
3	Slow down and steer left
4	Slow down and go straight
5	Slow down and steer right

Algorithm

The provided algorithm presents a clear and concise implementation of the DDQN for autonomous navigation:

Initialize CARLA environment and sensors
For each episode:
    Reset environment and variables (collisions, camera, etc.)
    As long as the episode has not ended:
        Choose an action (epsilon-greedy)
        Execute action in environment
        Obtain new observation, reward, collision status, etc.
        Add transition to buffer
        If buffer is sufficient:
            Sample a batch of transitions
            Calculate Q-targets
            Calculate loss between current Q-values and targets
            Update model weights by minimizing loss
        If episode is over or time is up, stop
    Every N episodes:
        Update target network with online network weights
    Record model weights
    Track rewards and display results

 doi:10.20944/preprints202501.0617.v1

5.6. Results

The suggested approach has been validated using the free autonomous driving simulation program CALRA [3], [9], [36]. Several training simulations were carried out in the TOWN05 environment using CARLA 0.9.13. The simulation environment is shown in Figure 5.

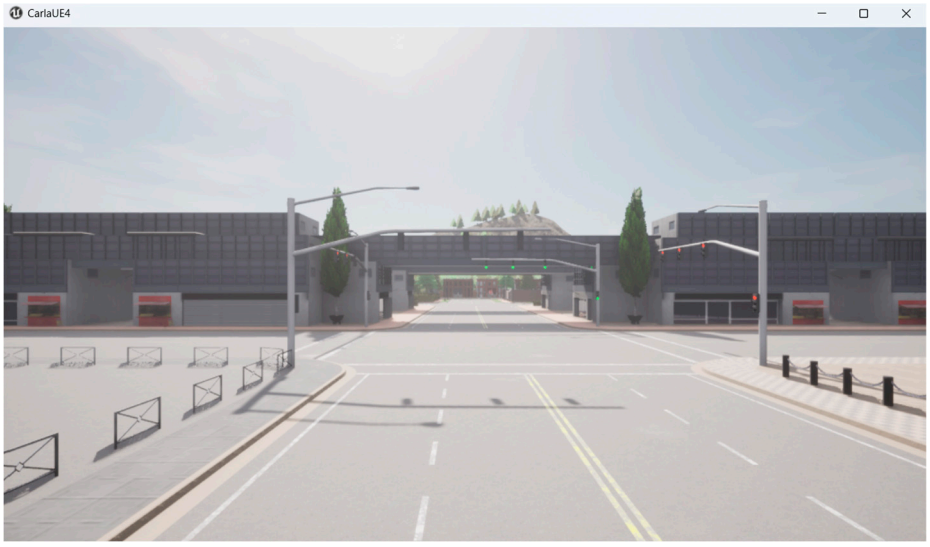

Figure 5. The CARLA Highway Environment (Town 05).

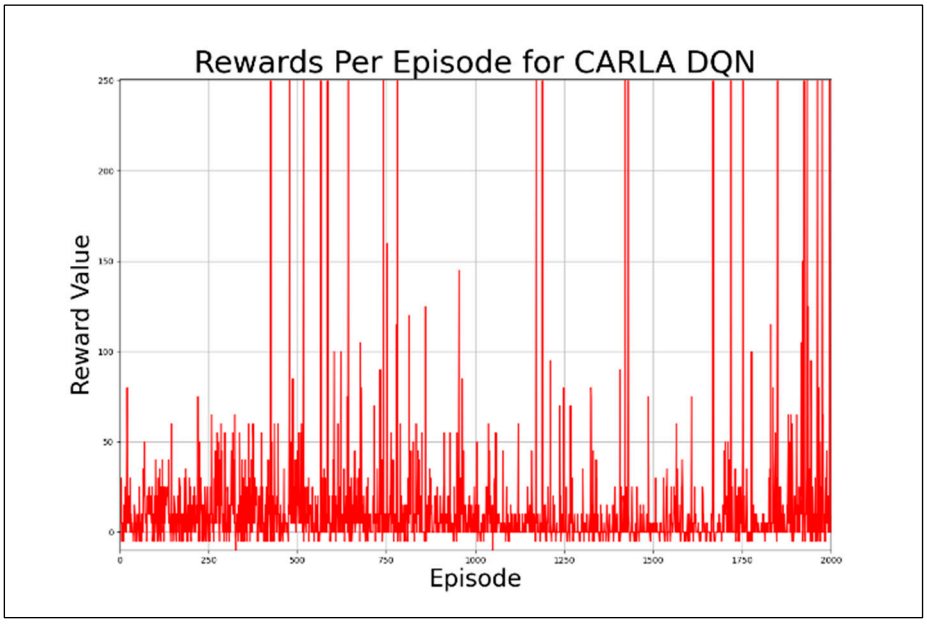

Figure 6. Representation of Reward vs. Episode of DQN algorithm with 2000 episodes.

 doi:10.20944/preprints202501.0617.v1

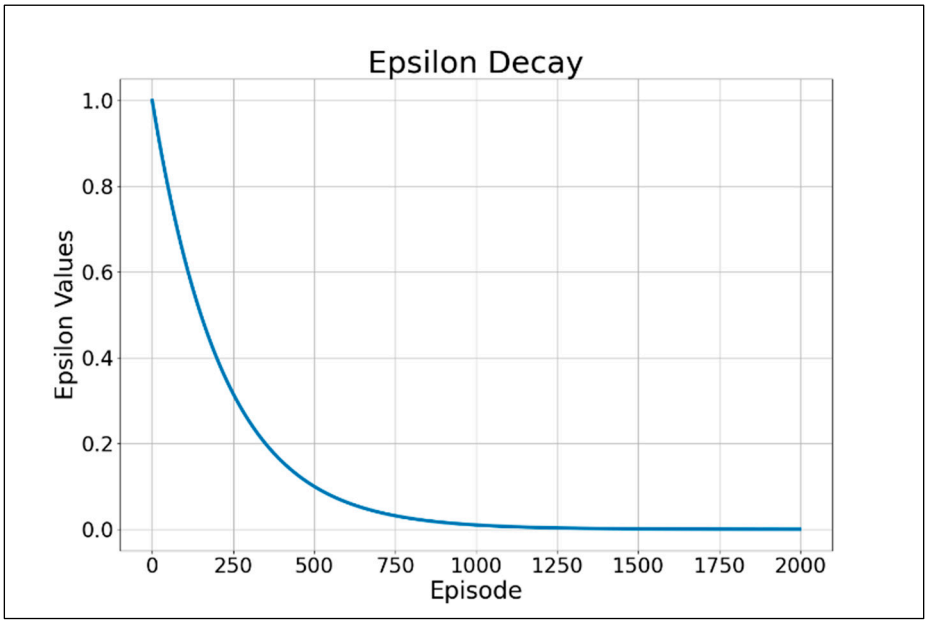

Figure 7. Representation of Epsilon Decay values of DQN algorithm with 2000 episodes.

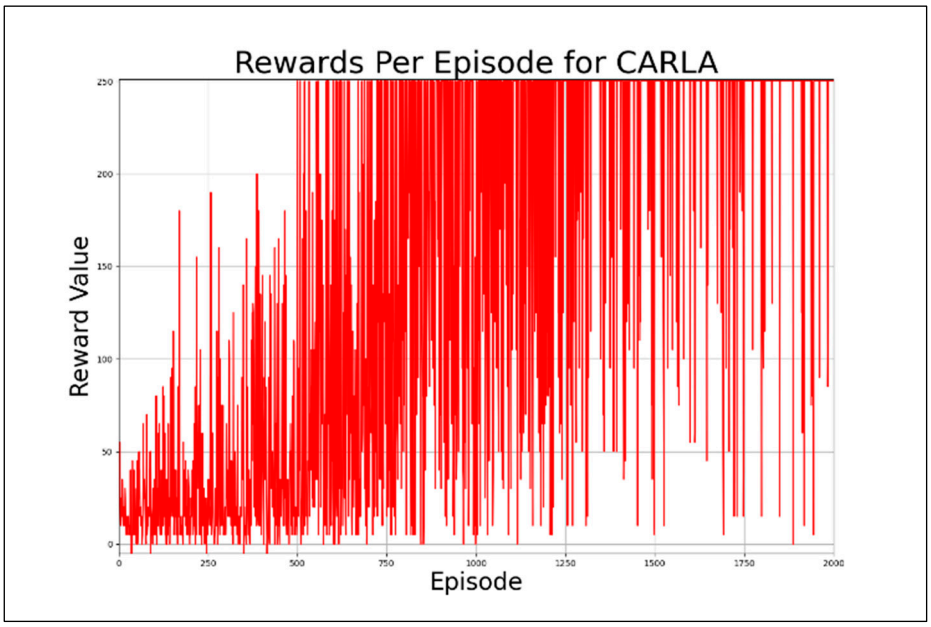

Figure 8. Representation of Reward vs. Episode of proposed DDQN algorithm with 2000 episodes.

 doi:10.20944/preprints202501.0617.v1

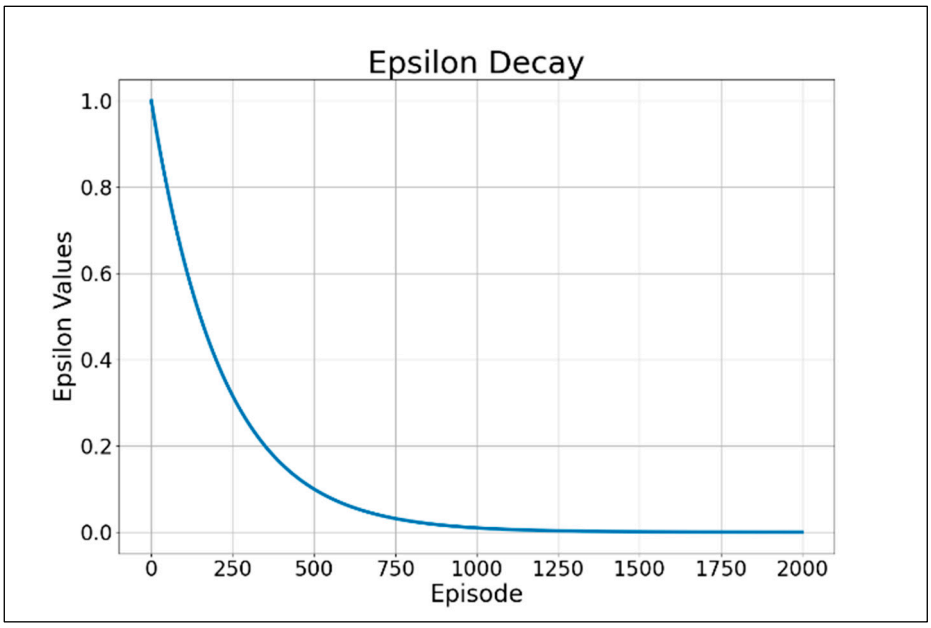

Figure 8. Representation of Epsilon Decay values of proposed DDQN algorithm with 2000 episodes.

With the CARLA simulator, we can observe that the suggested double-deep Q-Network method performs excellently. The trend on the graph is rising. As the number of episodes increases, the agent gains the ability to learn how to drive itself correctly, staying in its lane and taking the right turns to prevent collisions.

6. Conclusion

In this work, we developed an autonomous driving system using reinforcement learning in the CARLA simulator. A DDQN model was implemented, leveraging both CNN and LSTM layers to process image data from the vehicle's front-facing camera and capture temporal dependencies between frames. The model was designed to control a Tesla Model 3 in a complex urban environment, Town05, by learning to make decisions based on visual input, with actions like turning, accelerating, and slowing down. The system was trained using a reward function that penalized collisions and rewarded smooth, collision-free driving over time. The use of a replay buffer and target network improved training stability, and the epsilon-greedy policy ensured a balance between exploration and exploitation. Over time, the model demonstrated improved performance, as seen through increased reward accumulation and fewer collisions per episode. This work highlights the potential of reinforcement learning for autonomous driving tasks. Combining deep learning techniques with simulated environments makes it possible to create systems that learn complex driving behaviors without human intervention. While the results are promising, future work could focus on fine-tuning the model's hyperparameters, increasing the complexity of the driving scenarios, and extending the system to handle additional tasks such as lane changes or interactions with other road users. Further experimentation with real-world data and scenarios would also be necessary to bring such systems closer to practical applications.

References

1. Shalev-Shwartz, S., Shammah, S., and Shashua, A. (2016). Safe, Multi-Agent, Reinforcement Learning for Autonomous Driving. ArXiv, abs/1610.03295.
2. Haklidir, M., and Temeltas, H. (2022). Autonomous Driving Systems for Decision-Making Under Uncertainty Using Deep Reinforcement Learning. *2022 30th Signal Processing and Communications Applications Conference (SIU)*, 1-4.

3. Qian, Z., Guo, P., Wang, Y., and Xiao, F. (2023). Ethical and moral decision-making for self-driving cars based on deep reinforcement learning. Journal of Intelligent and Fuzzy Systems, 45(4), 5523–5540. https://doi.org/10.3233/jifs-224553
4. Sallab, A.E., Abdou, M., Perot, E., and Yogamani, S.K. (2017). Deep Reinforcement Learning framework for Autonomous Driving. *Autonomous Vehicles and Machines*.
5. Kuutti, S., Bowden, R., Jin, Y., Barber, P., and Fallah, S. (2020). A Survey of Deep Learning Applications to Autonomous Vehicle Control. *IEEE Transactions on Intelligent Transportation Systems, 22*, 712-733.
6. Liu, Z., Cai, Y., Wang, H., Chen, L., Gao, H., Jia, Y., and Li, Y. (2021). Robust target recognition and tracking of Self-Driving cars with radar and camera information fusion under severe weather conditions. IEEE Transactions on Intelligent Transportation Systems, 23(7), 6640–6653. https://doi.org/10.1109/tits.2021.3059674.
7. Hoel, C., Wolff, K., and Laine, L. (2018). Automated Speed and Lane Change Decision Making using Deep Reinforcement Learning. 2018 21st International Conference on Intelligent Transportation Systems (ITSC), 2148-2155.
8. Ronecker, M.P., and Zhu, Y. (2019). Deep Q-Network Based Decision Making for Autonomous Driving. 2019 3rd International Conference on Robotics and Automation Sciences (ICRAS), 154-160.
9. Elallid, B.B., Benamar, N., Mrani, N., and Rachidi, T. (2022). DQN-based Reinforcement Learning for Vehicle Control of Autonomous Vehicles Interacting With Pedestrians. 2022 International Conference on Innovation and Intelligence for Informatics, Computing, and Technologies (3ICT), 489-493.
10. Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A.A., Veness, J., Bellemare, M.G., Graves, A., Riedmiller, M.A., Fidjeland, A.K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., and Hassabis, D. (2015). Human-level control through deep reinforcement learning. Nature, 518, 529-533.
11. Elallid, B.B., Benamar, N., Bagaa, M., and Aoul, Y.H. (2024). Enhancing Autonomous Driving Navigation Using Soft Actor-Critic. Future Internet, 16, 238.
12. Katrakazas, C., Quddus, M.A., Chen, W., and Deka, L. (2015). Real-time motion planning methods for autonomous on-road driving: State-of-the-art and future research directions. Transportation Research Part C-emerging Technologies, 60, 416-442.
13. Giannaros, A., Karras, A., Theodorakopoulos, L., Karras, C.N., Kranias, P., Schizas, N., Kalogeratos, G., and Tsolis, D. (2023). Autonomous Vehicles: Sophisticated Attacks, Safety Issues, Challenges, Open Topics, Blockchain, and Future Directions. J. Cybersecur. Priv., 3, 493-543.
14. Pérez-Gil, Ó., Barea, R., López-Guillén, E., Bergasa, L.M., Gómez-Huélamo, C., Gutiérrez, R., and Diaz-Diaz, A. (2022). Deep reinforcement learning based control for Autonomous Vehicles in CARLA. Multimedia Tools and Applications, 81, 3553 - 3576.
15. Thompson, C.R., Talla, R.R., Gummadi, J.C., and Kamisetty, A. (2019). Reinforcement Learning Techniques for Autonomous Robotics. Asian Journal of Applied Science and Engineering.
16. Hu, D., Huang, C., Wu, J., and Gao, H. (2024). Pre-trained Transformer-Enabled Strategies with Human-Guided Fine-Tuning for End-to-end Navigation of Autonomous Vehicles. ArXiv, abs/2402.12666.
17. Georgeon, O.L., Casado, R.C., and Matignon, L. (2015). Modeling Biological Agents Beyond the Reinforcement-learning Paradigm. Biologically Inspired Cognitive Architectures.
18. Rizehvandi, A., Azadi, S., and Eichberger, A. (2024). Decision-Making Policy for Autonomous Vehicles on Highways Using Deep Reinforcement Learning (DRL) Method. Automation.
19. Chen, Y., Ji, C., Cai, Y., Yan, T., and Su, B. (2024). Deep Reinforcement Learning in Autonomous Car Path Planning and Control: A Survey. ArXiv, abs/2404.00340.
20. Ronecker, M.P., and Zhu, Y. (2019). Deep Q-Network Based Decision Making for Autonomous Driving. 2019 3rd International Conference on Robotics and Automation Sciences (ICRAS), 154-160.
21. Zhang, Y., Sun, P., Yin, Y., Lin, L., and Wang, X. (2018). Human-like Autonomous Vehicle Speed Control by Deep Reinforcement Learning with Double Q-Learning. 2018 IEEE Intelligent Vehicles Symposium (IV), 1251-1256.
22. Dosovitskiy, A., Ros, G., Codevilla, F., López, A.M., and Koltun, V. (2017). CARLA: An Open Urban Driving Simulator. Conference on Robot Learning.

23. Li, P.X., Kusari, A., and Leblanc, D. (2021). A Novel Traffic Simulation Framework for Testing Autonomous Vehicles Using SUMO and CARLA. ArXiv, abs/2110.07111.
24. Papadakis, A., Theodorou, T., Mamatas, L., and Petridou, S.G. (2021). An experimentation environment for SDN-based autonomous vehicles in smart cities. 2021 17th International Conference on Network and Service Management (CNSM), 391-393.
25. Elallid, B.B., Benamar, N., Hafid, A.S., Rachidi, T., and Mrani, N. (2022). A Comprehensive Survey on the Application of Deep and Reinforcement Learning Approaches in Autonomous Driving. J. King Saud Univ. Comput. Inf. Sci., 34, 7366-7390.
26. Elallid, B.B., Benamar, N., Mrani, N., and Rachidi, T. (2022). DQN-based Reinforcement Learning for Vehicle Control of Autonomous Vehicles Interacting With Pedestrians. 2022 International Conference on Innovation and Intelligence for Informatics, Computing, and Technologies (3ICT), 489-493.
27. Hossain, J. (2023). Autonomous Driving with Deep Reinforcement Learning in CARLA Simulation. ArXiv, abs/2306.11217.
28. Tammewar, A., Chaudhari, N., Saini, B., Venkatesh, D., Dharahas, G., Vora, D.R., Patil, S.A., Kotecha, K.V., and Alfarhood, S. (2023). Improving the Performance of Autonomous Driving through Deep Reinforcement Learning. Sustainability.
29. Bojarski, M., Testa, D.W., Dworakowski, D., Firner, B., Flepp, B., Goyal, P., Jackel, L.D., Monfort, M., Muller, U., Zhang, J., Zhang, X., Zhao, J., and Zieba, K. (2016). End to End Learning for Self-Driving Cars. ArXiv, abs/1604.07316.
30. Chen, Y., Palanisamy, P., Mudalige, P.W., Muelling, K., and Dolan, J.M. (2018). Learning On-Road Visual Control for Self-Driving Vehicles With Auxiliary Tasks. 2019 IEEE Winter Conference on Applications of Computer Vision (WACV), 331-338.
31. Chen, L., Hu, X., Tang, B., and Cheng, Y. (2022). Conditional DQN-Based Motion Planning with Fuzzy Logic for Autonomous Driving. IEEE Transactions on Intelligent Transportation Systems, 23, 2966-2977.
32. Pérez-Gil, Ó., Barea, R., López-Guillén, E., Bergasa, L.M., Gómez-Huélamo, C., Gutiérrez, R., and Diaz-Diaz, A. (2022). Deep reinforcement learning based control for Autonomous Vehicles in CARLA. Multimedia Tools and Applications, 81, 3553 - 3576.