**Academic Journal of Science and Technology**
ISSN: 2771-3032 | Vol. 11, No. 3, 2024

# Intelligent Decision and Path Planning Algorithm of AGV Vehicle based on Deep Learning

**Hongwei Huang [1], Xiangda Wang [2], Xinyu Song [1], Renyu Yang [3], Run Guo [4], Shuyang Liu [1, *]**

[1] College of Information Engineering, Shanghai Maritime University, Shanghai, China
[2] School of Mechatronic Engineering and Automation, Shanghai University, Shanghai, China
[3] Business School, University of Shanghai for Science and Technology, China
[4] Shanghai University of Political Science and Law, Language and Culture College (International Exchange College), China

*** Corresponding author:** Shuyang Liu (Email: lsy201922@163.com)

**Abstract:** With the widespread application of intelligent unmanned vehicles, intelligent navigation, path planning, and obstacle avoidance technologies have become important research topics. This paper proposes an AGV (Automated Guided Vehicle) intelligent decision-making and path planning algorithm based on deep learning. This algorithm uses environmental information to trace a route to the target point while avoiding both static and dynamic obstacles, making it adaptable to different environments. By employing a combination of global planning and local obstacle avoidance decision-making, this method addresses the path planning problem with better globality and robustness, and the obstacle avoidance problem with improved dynamism and generalizability, while also reducing iteration time. During the network training phase, traditional algorithms such as PID and A* are integrated to enhance the convergence speed and stability of the proposed method. Finally, various experimental scenarios for navigation and obstacle avoidance were designed in the Robot Operating System (ROS) and simulation program Gazebo. The simulation results verified that the proposed method, which takes into account both global and dynamic aspects, is reliable and that the generated paths and time efficiency have been optimized.

**Keywords:** Intelligent Navigation; Path Planning; AGV; Network Training Phase.

## 1. Introduction

As a type of mobile robot, intelligent vehicles are becoming increasingly important in today's world and have become essential tools across various industries. They can be widely applied for flexible handling and transportation in sectors such as machinery and electronics, and can also be used as transport tools for cargo handling at stations and airports.[1] Additionally, they can be extensively utilized in locations such as hotels, hospitals, and banks to provide business guidance functions. Furthermore, they can replace humans in completing dangerous tasks such as emergency rescue operations and lunar exploration. Since autonomous mobile robots operate in unpredictable and partially unknown environments, navigation in dynamic and dense settings presents numerous challenges. This means that the robot must have the ability to navigate without interference and avoid any obstacles within its movement constraints[2].

Traditional path planning techniques can be used to solve path planning problems in known environments and are easy to implement. However, they have poor exploratory capabilities in dynamic or complex environments, slow convergence speeds, long execution times, and are prone to getting stuck in local optima. Additionally, when the environment changes, the algorithm needs to be adjusted. In contrast, deep learning methods, based on heuristic strategies obtained through data training, can be applied to many path planning tasks[3].

Currently, there are not many solutions using reinforcement learning for global planning, with some works utilizing the Q-learning algorithm. Gao et al. divided the state space using a grid map, where the coordinates of grid cells without obstacles were considered states in the Q-table, and the actions involved moving one cell left, right, up, or down[4]. The Q-learning algorithm was then used for exploration and planning, which achieved global path planning in a discrete environment. Sichkar attempted to randomly sample points in a continuous state map, use these points as states in the Q-table, connect all points into multiple line segments, and eliminate segments that crossed obstacles[5]. Thus, for a given state, the action was to move to an adjacent point. This improved method is similar to Q-learning based on grid maps, converting the continuous environment into a discrete one solvable by Q-learning. However, this algorithm is essentially still based on discrete environments and actions[6]. Panov et al. proposed using the DQN algorithm to solve the global planning problem on grid maps, effectively addressing the curse of dimensionality in state space, though the action space remained discrete[6].

The aforementioned works using reinforcement learning for global planning are all limited to discrete spaces. However, in the future, one direction is to use PPO, DDPG, SAC, and other algorithms based on continuous actions to attempt to solve the global path planning problem. This can improve the reliability of global planning.

## 2. PATH PLANNING BASED ON DEEP REINFORCEMENT LEARNING

Global planning addresses the problem of how an unmanned vehicle navigates to its destination, allowing the optimal path to be calculated from the global map. However, many uncertainties arise during actual movement, such as the appearance of additional obstacles when the global map does not match local conditions. In such cases, local obstacle avoidance becomes crucial, as it can control the vehicle to avoid obstacles at specific locations. Without global planning,

local obstacle avoidance cannot complete planning tasks in larger scenarios. Therefore, global planning and local obstacle avoidance complement each other, and combining both algorithms can solve most path planning problems for unmanned vehicles. The algorithm first plans the driving route using a global planning algorithm, and then uses local obstacle avoidance for continuous local control as the robot moves.

Figure 1 shows the overall framework for path planning of an unmanned vehicle. This framework illustrates how global planning and local obstacle avoidance are combined to accomplish the path planning and navigation tasks. First, the coordinates of the target point are input into the global planning module. The global planning uses the SAC algorithm to explore the path based on the known global map and the current position coordinates, outputting nnn intermediate exploration points $[x_i,y_i]$, where $i \in \{0,1,...,n\}$, and stores them.

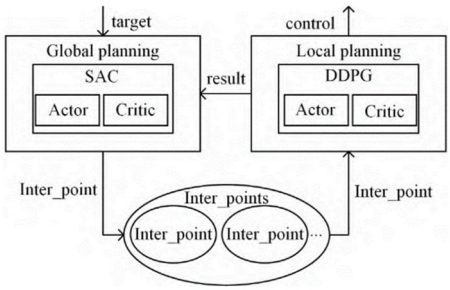



**Figure 1.** Overall path planning framework

For these $n$ intermediate exploration points, they are input into the local planning module in the order from 0 to $n$. This means that the local planning is expected to navigate to each exploration point in sequence and finally reach the target point. Each exploration point from the global planning serves as a target point for the local planning. The local planning calculates continuous control quantities $\_v_t$ and $w_t$ for the vehicle over a period of time through the Actor network of DDPG by inputting a specific point $[x_i,y_i]$ and some perception information, and navigates the vehicle to $[x_i,y_i]$. Once it is confirmed that the vehicle has approached this point, $[x_{i+1},y_{i+1}]$ is then input into the local controller to navigate the vehicle to the next exploration point. This process is repeated for each navigation to the final target point, requiring $n$ calls to the local obstacle avoidance and generating a large number of control quantities. This way, the vehicle can make real-time obstacle avoidance actions while ensuring it does not deviate from the guidance of the global path. If the local obstacle avoidance control encounters an obstacle, it will return the result to the global planning. The global planning, after obtaining the planning status from the local obstacle avoidance, will make a judgment and take re-planning measures, clearing the exploration points and re-initiating the global planning.

## 2.1. Local Obstacle Avoidance based on DDPG

Local path planning focuses on the robot's current local environmental information in situations where the environment is either completely unknown or partially known, ensuring that the robot has good obstacle avoidance capabilities. The robot uses sensors to detect the working environment, acquiring geometric and positional information of obstacles, and can dynamically update and correct its understanding of the environment at any time. The local planning method integrates environment perception and search, requiring the robot system to have high robustness to environmental errors, as well as high-speed information processing and computing capabilities, allowing it to implement feedback and correction of the planning results[7].

Due to the need for adjusting numerous parameters in traditional obstacle avoidance algorithms such as the DWA algorithm, which cannot directly benefit from continuous large-scale datasets, a novel learning-based approach is proposed to achieve faster speed and more stable learning performance. By directly mapping sensor data to motion control quantities, this approach solves the problem of traditional local planning requiring extensive computation of cost maps[8]. Additionally, this reactive obstacle avoidance can handle uncertain and dynamic environments, for which deep reinforcement learning is suitable. It improves the network by exploring the environment and obtaining rewards. To enable unmanned vehicles to navigate obstacles in different dynamic environments, model-free deep reinforcement learning algorithms are needed. They directly learn action policies by collecting data from simulated or real environments without the need for pre-modeling the environment. If model-based deep reinforcement learning models are unstable, their learning performance will be greatly reduced, and their asymptotic performance will not be as good as model-free methods. Especially for dynamic environment models, modeling them is very challenging. Model-free experience-driven deep reinforcement learning algorithms do not need to make assumptions about the environment and can change strategies during runtime by accumulating experience to adapt to environmental changes[9].

Local obstacle avoidance requires collecting sensor information, thus requiring improved sampling efficiency. As a deterministic policy gradient algorithm for offline learning, DDPG introduces the Actor-Critic framework, as well as concepts like experience replay buffers and target networks. Its algorithm exhibits high sampling efficiency and stability, preventing the correlation of different events during the training stage, avoiding convergence oscillations caused by directly copying parameters, and preventing overfitting phenomena caused by solely using new data, thus enhancing the algorithm's robustness and generalization. Therefore, the problem of obstacle avoidance for mobile robots can be addressed using deep reinforcement learning with DDPG, resolving issues related to the large exploration space, high number of steps required for network convergence, and high variance. The research focus of this method is unmanned vehicles equipped with laser rangefinder sensors, capable of perceiving obstacles within 0 to 10 meters. When encountering obstacles within this range, the laser beams are obstructed, and each beam returns a distance value.

State $S$ in DDPG algorithm includes the following:

(1) The observation data from the laser rangefinder sensor consists of laser scans within a 180° range in front of the vehicle, stored in an array called laser_scan with dimensions of 360, representing the laser values observed within every 0.5° range ($d_1, d_2, ..., d_{360}$). As the laser sensor cannot guarantee capturing all laser data, parts where $d_i$ = None are changed to $d_i = d_{max} = 10$ during laser data preprocessing. To reduce observation data errors and computational costs, the 360-dimensional data is reduced to 40 dimensions, where every 9 distance values are averaged to obtain ($d_1, d_2, ..., d_{40}$). After obtaining the distribution of obstacles at each angle,

normalization is performed by $d_i = d_i / d_{max} = d_i / 10$ to obtain new normalized data ($d_1$, $d_2$, ..., $d_{40}$). Subsequently, the normalized data collected from the past two times and the most recent normalized data are merged to form a data set ($d_1$, $d_2$, ..., $d_{120}$), which serves as the input for the laser rangefinder observation data.

(2) The driving state of the unmanned vehicle is determined by the current linear velocity $v$ and angular velocity $w$ of the vehicle, which are input into the network. The driving state can be obtained from the odometer-recorded information.

(3) The status information of the target point is input with the coordinates of the target point with the vehicle as the coordinate center. The current position coordinates and heading angle deviation of the unmanned vehicle at the current time can also be obtained from the odometer information as ($x$, $y$, $\theta$). With this information, the coordinates of the target point ($x_{target}$, $y_{target}$) in the coordinate system of the unmanned vehicle can be calculated as:

$$x^{'} = (x_{target} - x) * \cos\theta + (y_{target} - y) * \sin\theta \quad (1)$$

$$x^{'} = -(x_{target} - x) * \sin\theta + (y_{target} - y) * \cos\theta \quad (2)$$

That is, ($x'$, $y'$) represents the coordinates of the target point relative to the coordinate system centered on the vehicle. These coordinates are input into the network.

For the DDPG algorithm, the action $A$ is a continuous control value passed from the local obstacle avoidance module to the unmanned vehicle, representing linear velocity $v$ and angular velocity $w$. This control value directly guides the motion control of the vehicle.

The reward function for DDPG is as follows:

$$r(s_t, a_t) = \geq \begin{cases} r_{goal}, s_t \leq d_{goal} \\ r_{collision}, \min(d_i) \leq d_{collision} \\ k(s_t - s_{t-1}), else \end{cases} \quad (3)$$

Where $s_t$ denotes the current distance of the vehicle from the target point. When the distance of the vehicle from the target point is less than the $d_{goal}$ threshold, it can be considered as reaching the target point, and the environment provides a positive reward $r_{goal}$. When the minimum distance feedback from the laser sensor is less than the safety threshold $d_{collision}$, indicating that the vehicle is about to collide with an obstacle, in order to give the vehicle a chance to change its direction of motion, a threshold iii can be set, representing the number of times allowing $min(d_i) < d_{collision}$. If the number of times the vehicle's distance from obstacles is less than the safety threshold in multiple iterations reaches a certain limit, it is considered that the vehicle has collided with an obstacle, receiving a negative reward $r_{collision}$. If the vehicle is in other states, the algorithm aims to guide the vehicle towards the target point. If the vehicle moves away from the target point, the environment gives a small penalty. If the vehicle moves closer to the target point, a reward is given to the action from the previous time step. This also addresses the problem of sparse rewards when the vehicle explores the environment, which accelerates the convergence of the network.

Figure 2 illustrates the framework for utilizing the DDPG algorithm to solve local planning. The arrows in the figure indicate the direction of data generation or flow, and the rectangles within dashed boxes represent network layers, with the numbers inside the rectangles denoting the dimensions of the network layers. The main idea of this framework is to provide a PID controller to assist the network in policy search, thereby accelerating and stabilizing the training process.

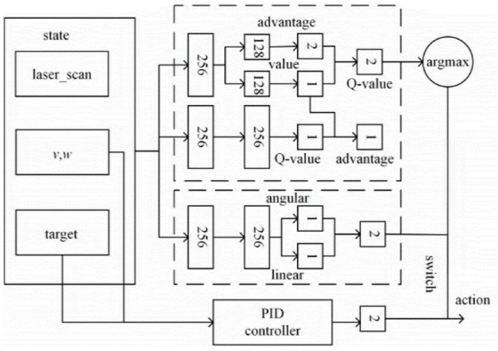



**Figure 2.** ODDPG network structure

The network also learns from the experience replay pool, regardless of the current policy. The Actor network can benefit from the learning samples generated by both the recorded policy and the PID controller policy. Initially, the PID controller is frequently chosen as the optimal policy. As the number of training iterations increases and meaningful guided experiences are collected, the learned policy will surpass the simple controller.

## 2.2. Global Planning based on SAC

Like the DDPG algorithm, the SAC algorithm is also based on model-free deep reinforcement learning, eliminating the need for environment modeling. There is no requirement for precomputing transition models, dynamics models, or reward models about the environment[10]. Instead, it directly interacts with the environment to obtain the information needed for network training, including reward values and the current state of the agent. This capability enables its applicability to different environments, without relying solely on an environment model during network training.

SAC's reward function design is as follows:

$$r(s_t, a_t) = \begin{cases} r_{target}, s_t = s_{target} \\ r_{obstacle}, C[s_t] \neq 1 \\ \alpha_{opt} r_{opt} + \alpha_{rat} r_{rat} + \alpha_{euq} r_{euq}, C[s_t] = 1 \end{cases} \quad (4)$$

Where $\sum \alpha_i = 1$, $s_t$ represents the coordinates of the current exploration point, $s_{target}$ represents the coordinates of the target point, and $r_{target}$ represents the reward obtained when the exploration point reaches the target point, i.e., $s_t = s_{target}$; if the current exploration point is not in a determined feasible area, it receives a penalty value $r_{obstacle}$.

Figure 3 depicts the framework of the SAC algorithm. After inputting the state into the network, the mean and variance of the action are obtained. Actions are then sampled using Gaussian sampling to acquire action values [Δx,Δy] along with their corresponding probabilities. The action values represent the displacement towards the next exploration point, while the probability values are used to compute the entropy of the actions. The Critic network optimizes its parameters by performing loss and gradient computations between the Q-values from the data items in the experience replay pool and the Q-values obtained from the network. The use of Double Q-network, represented by Q1Q_1Q1 and Q2Q_2Q2, eliminates the maximization bias compared to using a single Q-network, enhancing the

accuracy and stability of the network, and preventing rapid convergence or oscillations in Q-value networks.

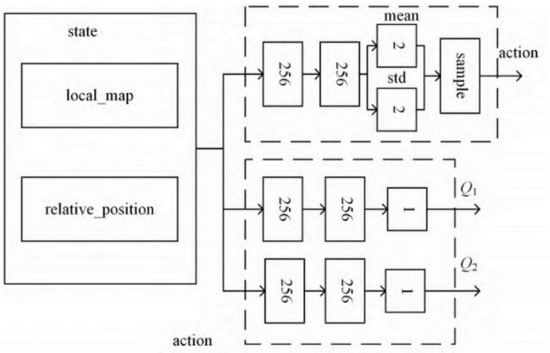



**Figure 3.** SAC network frame diagram

# 3. EXPERIMENTAL RESULTS AND ANALYSIS

The experimental environment utilizes the simulation program Gazebo under the ROS (Robot Operating System) environment. ROS is an open-source meta-operating system designed for robots, providing hardware abstraction, low-level device control, inter-process communication, and package management mechanisms, along with tools and libraries necessary for running code. ROS operates as a distributed process framework, facilitating information exchange between processes through topic subscription and publication, thus supporting robot research, development, and applications. Gazebo, on the other hand, is a powerful 3D physics simulation platform offering features such as dynamic simulation, 3D visualization environments, sensor simulation, and extensible plugin support. It is particularly suitable for research and development involving unmanned vehicles.

The experimental setup is illustrated in Figure 4, depicting an 8x8m² map enclosed by walls. The ultimate goal of the experiment is to navigate the car from the starting point to any obstacle-free location on the map, within this complex and dynamic environment. For environment construction, challenging obstacles such as concave and elongated walls were chosen to assess obstacle avoidance difficulty. Additionally, certain obstacles were made dynamic through motion plugins, aimed at validating the efficacy of the dual-layer path planning for the car.

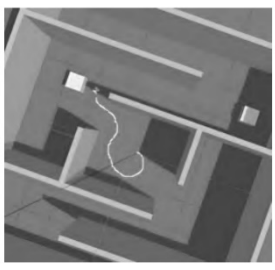


**Figure 4.** Experimental scenario

The experimental data is transmitted using the topic mechanism specific to the ROS operating system. These data are generated by plugins of the Gazebo simulation program. For example, the odometry information is transmitted via the /odom topic, including the position and orientation of the vehicle in the current simulated environment. The laser scan information of the vehicle is transmitted via the /laser_scan topic, including the maximum laser distance, scanning angle, and laser distance measurement at each angle.

For the motion control of the vehicle in the simulation program, commands for linear and angular velocities can be published to the /cmd_vel topic programmatically. As for the global cost map required by the global planning, the cost map parsed by the map resolver can be obtained through the /map topic.

## 3.1. Local Obstacle Avoidance

Training an unmanned vehicle's local obstacle avoidance using DDPG requires the utilization of sensor data such as lidar and odometry. The training environment for this algorithm is relatively simple, consisting of an 8x8 m^2 map enclosed by walls, with some obstacles placed within the walls. During the training process, the simulation system is reset at the beginning of each training iteration, placing the vehicle at the initial point (0, 0), and randomly assigning a valid target point within the wall boundaries for local planning control. This approach enhances the network's generalization capability and prevents policy overfitting. Additionally, ROS nodes are used to subscribe to topics related to environment information, such as odometry and lidar data. Then, the environmental information is inputted into the network, and the output values obtained from the policy network serve as control commands, which are published to the /cmd_vel topic by the nodes. The vehicle in the simulation system receives control information from the /cmd_vel topic and executes the control actions. When the vehicle collides with obstacles or reaches the target point, the next iteration begins.

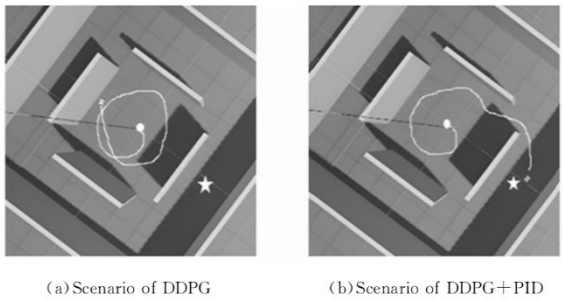
The experiment compared the use of only the DDPG algorithm with the combination of the DDPG algorithm and the PID algorithm, and tested them in multiple environments. The parameters of the DDPG algorithm are listed in Table 1.

**Table 1.** Evaluation of capacity of distributed power supply under different operation modes

| Discount factor | Actor learning rare | Critic learning rate | Soft update param | Buffer size |
|---|---|---|---|---|
| 0.99 | 0.001 | 0.001 | 0.001 | 20000 |

During the experiment, the car using only the DDPG algorithm exhibited behavior of spinning in place. The reason for this behavior is that most of the experiences learned by the car from the experience replay buffer are obstacle avoidance experiences. The car randomly explores the environment from the beginning of training, with a high probability of ending the iteration with a collision, making it difficult to reach the target point. These data are loaded into the experience replay buffer during training, allowing the car to learn obstacle avoidance ability from them. However, without controllers such as PID algorithms to assist in control, although the car can avoid collisions by spinning, it cannot trace to the target point. As a result, the reward value is almost zero, satisfying only the locally optimal condition. Moreover, the controller using only the DDPG algorithm often encounters situations of insufficient time due to accumulated time penalties. The car learned to collide to avoid time penalties instead of reaching the target. As shown in Figure 5 (a), where white obstacles represent walls, and the black areas represent shadows cast, the car (white) departs from the origin

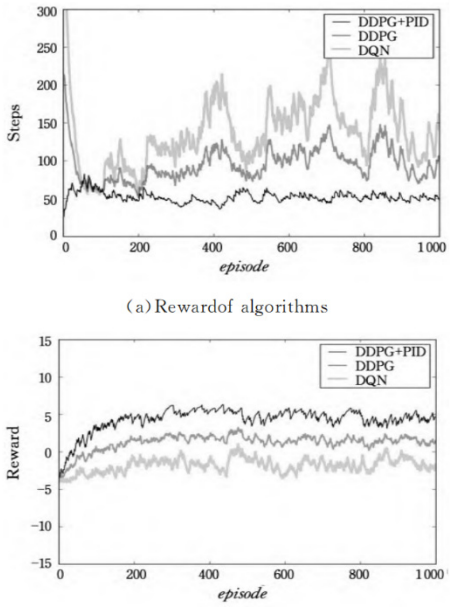
and moves around the walls to avoid obstacles, but does not trace to the target point.



(a) Scenario of DDPG (b) Scenario of DDPG+PID

**Figure 5.** The experimental comparison between DDPG method and DDPG combined with PID method in static environment

Figure 6 illustrates the variations in reward and steps under different control algorithms. Figure 6 (a) shows the changes in reward values for each algorithm, while Figure 6 (b) represents the changes in steps for each algorithm.




(a) Rewardof algorithms

**Figure 6.** Comparison of reward and steps for DQN, DDPG, DDPG combined with PID

From the graphs, it can be observed that the reward values in the combined DDPG with PID method are stabilizing at positive values, indicating the correctness of actions. Additionally, the steps are gradually decreasing, suggesting that the policy is optimizing, and the time to reach the target is reducing. In contrast, with methods using only DDPG or DQN, the networks hardly converge, with reward values difficult to stabilize and even tend towards negative rewards. Particularly, the DDPG algorithm, compared to the combination of DDPG with PID, eliminates the PID control module and relies solely on network output results. While this approach performs better than DQN, it still falls short. Based on these results from the ablation experiments, it can be inferred that PID control plays a crucial role in the convergence of the DDPG algorithm.

## 3.2. Global Planning

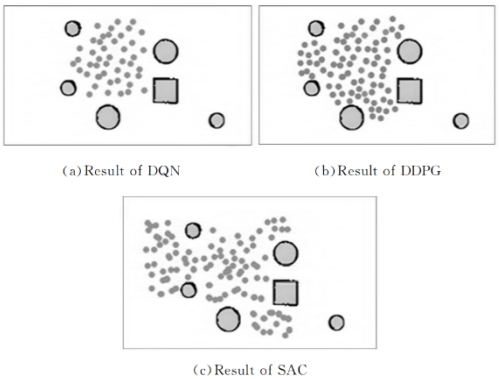
Global planning addresses the problem of path searching in static global cost maps, thus the generation of global cost maps is necessary. In the ROS environment and Gazebo simulation environment, mapping algorithms and matching image analysis programs are required to construct cost maps of the simulated environment. The experiment adopts the Gmap-ping mapping algorithm to construct images of a simulated environment measuring 8x4 square meters with fixed obstacles. The algorithm controls the robot to traverse all positions in the environment until the entire image construction process is completed and the results are saved. Subsequently, the map_server analyzes the image to generate the cost map C[*i*][*j*], which is then published through the /map topic. Global planning can access the map through the /map topic, preprocess it, and utilize it as the cost map required for global planning.

Similar to local planning, the global planning sets the initial position to the origin (0, 0) at the beginning of each iteration and randomly generates a valid target point. During the training iterations, each newly explored point is saved, and when a new exploration point reaches the target point, it signifies the success of global planning. Then, the entire set of exploration points is handed over to the local planning for processing, and the exploration point set is cleared. If an exploration point reaches an obstacle grid or if the number of exploration points becomes too large, global planning is deemed unsuccessful, and the exploration point set is cleared directly. Then, the next iteration begins.

In the process of using SAC to solve the global planning problem, we tested the global planning capabilities of algorithms such as DQN, DDPG, and SAC in various environments, and conducted experimental analysis and comparisons of the results of these algorithms. The parameters of the SAC algorithm are listed in Table 2.

**Table 2.** Sac parameter

| Discount factor | learning rart | Entropy param | Soft update param | Buffer size |
|---|---|---|---|---|
| 0.99 | 0.0003 | 0.2 | 0.005 | 107 |



(a) Result of DQN (b) Result of DDPG

(c) Result of SAC

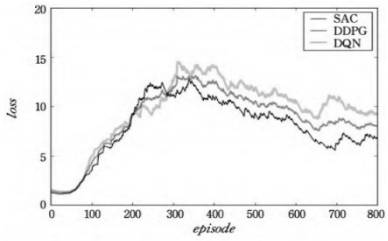
**Figure 7.** Comparison of DQN, DDPG and SAC algorithms

During the experiments, the success rate of solving the global planning task using the SAC algorithm was higher compared to the DQN algorithm. As shown in Figure 9, discrete points represent the regions where the goal points were successfully reached starting from the origin. Figure 7(a) presents the results of the DQN algorithm, Figure 7(b) shows the results of the DDPG algorithm, and Figure 7(c) illustrates the results of the SAC algorithm. Since the DQN algorithm is based on discrete actions, it requires higher accuracy in policy, thus resulting in a lower success rate. On the other hand, the DDPG algorithm may be influenced

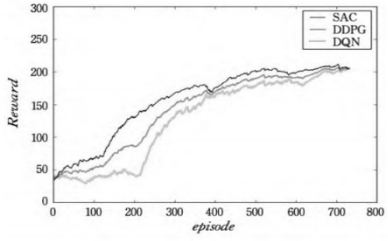
by local optima, resulting in higher success rates at certain points but being ineffective for points that are farther away.

For the SAC algorithm, it utilizes the continuous action space output by the network, which makes it more favorable for the success rate of global planning. This allows for more possibilities in the exploration direction of exploration points. Additionally, based on the maximum entropy stochastic policy, it exhibits stronger action randomness, making it easier to explore new paths, including those that bypass obstacles. Consequently, its performance is superior to that of the DQN and DDPG algorithms.

The graphs in (a) and (b) respectively depict the variations in reward and loss values under various algorithms. From the graphs, it can be observed that the loss value under the SAC algorithm is lower compared to DQN and DDPG, approaching a converged state. DQN algorithm exhibits higher loss values due to a higher number of failures, while DDPG falls between DQN and SAC in terms of loss values.




(a) Loss of algorithms




(b) Reward of algorithms

**Figure 8.** Comparison of reward and loss in DQN, DDPG and SAC algorithms

# 4. Conclusion

As the applications of robots continue to evolve, unmanned vehicles find utility across various industries, where their navigation and path planning technologies constitute the core of their intelligence. However, traditional path planning techniques are inadequate for unknown and dynamically changing environments. Hence, there arises a need to propose intelligent methods to enhance task execution efficiency and adaptability to complex environments.

In this regard, a solution based on deep reinforcement learning (DRL) has been introduced, which includes algorithms such as DDPG and SAC. By integrating global planning and local obstacle avoidance, this approach employs intelligent algorithms to address the entire path planning problem. Experimental results demonstrate that this method enhances planning and obstacle avoidance abilities in most scenarios. However, there remains a necessity to improve navigation capabilities, especially in more complex environments, particularly addressing global issues within global planning, in future endeavors.

# References


[1] T. Chu, and J. Wang, “Multi-agent deep reinforcement learning for large-scale traffic signal control,” IEEE Transactions on Intelligent Transportation Systems, vol. 21, no. 3, pp. 1086-1095, 2020.

[2] S. Dong, P. Wang, and Abbas K, et al, “A survey on deep learning and its applications,” Computer Science Review, no. 40, pp. 100378-100379, 2021.

[3] J. Gu, and Z. Wang, “Recent advances in convolutional neural networks,” Pattern Recognition, no. 77, pp. 354-377, 2020.

[4] Prashanth L A, and Bhatnagar S, “Reinforcement learning with function approximation for traffic signal control,” IEEE Transactions on Intelligent Transportation Systems, vol. 12, no. 2, pp. 412-421, 2021.

[5] Mercader P, Uwayid W, and Haddad J, “Max-pressure traffic controller based on travel times: An experimental analysis,” Transportation Research Part C: Emerging Technologies, no. 110, pp. 275-290, 2020.

[6] Somes N, and Moore J, “Design and Construction of a Regional Scale Bioret ention System,” Rainwater&Urban Design, vol. 20, no. 3, pp. 12-13, 2007.

[7] Rajeswari L P, Kannan A, and Baskaran R, “An Escalated Approach to Ant Colony Clustering Algorithm for Intrusion Detection System,” Distributed Computing & Networking, vol. 23, no. 56, pp. 23-25, 2018.

[8] Lt A, and Yh A, “Application of photovoltaic power generation in rail transit power supply system under the background of energy low carbon transformation,” Alexandria Engineering Journal , vol. 60, no. 6, pp. 5167-5174, 2021.

[9] Barbieri F, Rajakaruna S, and Ghosh A, “Very short-term photovoltaic power forecasting with cloud modeling: A review,” Renewable & Sustainable Energy Reviews, vol. 75, no. 2, pp. 242-263, 2016.

[10] Kazem H A , Yousif J , and Chaichan M T , et al, “Experimental and deep learning artificial neural network approach for evaluating grid-connected photovoltaic systems,” International Journal of Energy Research ,vol. 43, no. 14, pp. 8572-8591, 2019.