# Automatic Collision Avoidance via Deep Reinforcement Learning for Mobile Robot


1st Yongsheng Yang
*School of Systems Science and Engineering*
*Sun Yat-sen University*
Guangzhou, China
yangysh25@mail2.sysu.edu.cn

2nd Chengyu Xie
*School of Systems Science and Engineering*
*Sun Yat-sen University*
Guangzhou, China
xiechy28@mail2.sysu.edu.cn

3rd Zhiwei Hou*
*School of Systems Science and Engineering*
*Sun Yat-sen University*
Guangzhou, China
houzhw5@mail.sysu.edu.cn

4th Hongbo Chen
*School of Systems Science and Engineering*
*Sun Yat-sen University*
Guangzhou, China
chenhongbo@mail.sysu.edu.cn



***Abstract*—Collision avoidance is one of the most critical problems to be explored in the mobile robot field for finding an optimal path between source and destination. In this paper, we propose a novel DRL-based mapless collision avoidance algorithm for mobile robots, which maps the raw sensor data to control-level instructions and realizes robot navigation in an unknown environment without any complex numerical calculation. A game-like simulation environment enables the agent to interact with the surroundings and optimize its policy. The simulation environment is abstracted as walls and cylindrical obstacles, and it would be randomly initialized at the beginning of the training process to improve the generalization ability of DRL policy and obtain reliable test data for comprehensive analysis. We compare the performance of some state-of-the-art deep reinforcement learning algorithms with randomized environments, including DDPG, TD3, and SAC. The result shows that TD3 and SAC both gain a high passing rate when the obstacle density is from 0.0 to 0.10. The agent with DDPG fails to learn in the training environments and performs poorly. When obstacle density is 0.0 to 0.5, TD3 is more stable and performs better than SAC. However, SAC serves better when the obstacle density is 0.10 to 0.20. Both TD3 and SAC generate smooth trajectories in the test stage. The video and code are available: https://github.com/Vinson-sheep/turbot_rl.**



***Index Terms*—deep reinforcement learning, collision avoidance, mobile robot**


## I. Introduction

Deep reinforcement learning (DRL) is one of the essential branches of deep learning (DL), and it has grown exponentially in research and applications in recent years. It has the strong representation ability of deep learning and the excellent decision-making feature of conventional reinforcement learning, which enables an agent to autonomously learn an optimal policy to maximize accumulative rewards through trial-and-error interactions with its environment in continuous state space. In this case, we hope to apply deep reinforcement learning to robot applications.


*Corresponding author


Designing a safe and efficient local path planner for mobile robots is a longstanding research challenge. Many classic techniques have solved such obstacle avoidance problems, including Model Predictive Control (MPC), Potential Field methods, or numerical optimization approaches [1], [2]. They rely on a local map generated by some recognized approaches such as Simultaneously Localization And Mapping (SLAM) [3]–[5] or Structure from Motion (SfM) [6]. However, building and updating a high-resolution local map for local planning in real-time is time-consuming, and it can burden the robot's limited computing resources.

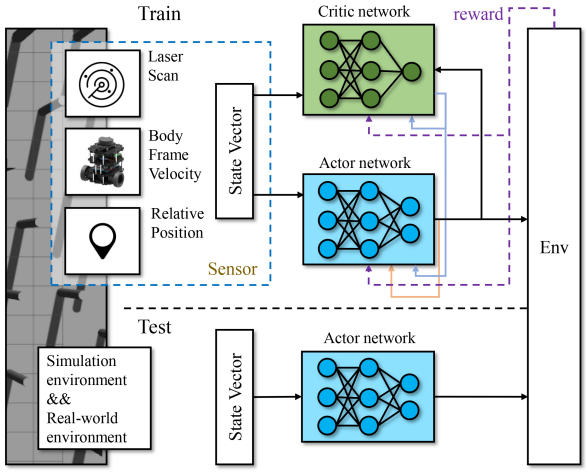


Fig. 1. The architecture of our automatic collision avoidance system in training and testing stages.

The motivation of this paper is to develop a learning-based collision avoidance framework for mobile robots to realize direct mapping from complex sensor-level data to continuous control-level instructions and drive the mobile robot to the destination without crashing, as shown in Fig. 1. For most

DRL-based applications in the industry, including game AI and recommended systems, the training process is implemented in a simulation environment that is very close to reality, and the distribution of data sent to the learning module is near to the actual application, so the policy obtained by DRL can work well in practice. However, the gap between simulation and reality cannot be ignored for a real robot. Transferring the well trained DRL-policy to the actual mobile robot remains a challenge. In this work, We design a randomized environment generation module to bridge this gap. The effectiveness of our approach is validated using a Turtlebot3 Burger with a LiDAR in a randomized simulation environment. The main contributions of this paper are as follows:

1) A DRL-based obstacle avoidance framework is proposed to realize safe and efficient local path planning for mobile robots.
2) Several representative state-of-the-art DRL algorithms are investigated, and their performance is discussed statistically in the simulation section.
3) A randomized environment module is adopted to implement the training and testing process and enhance data reliability.

## II. Related Work

DRL algorithms have grown tremendously recently, and many new optimization schemes have been proposed. Previous works focused on improving stability when training, increasing exploration early or reducing the overestimation of network updating. Lillicrap et al. [7] proposed a deep deterministic policy gradient (DDPG), an off-policy algorithm based on the deterministic policy gradient (DPG) [8] that can operate over continuous action spaces. Compared to the high variance of off-policy methods, Schulman et al. [9] designed proximal policy optimization (PPO), a popular on-policy algorithm with the benefits of trust region policy optimization (TRPO) [10]. However, it was much simpler to implement, more general and had better sample efficiency than actor-critic based on-policy methods [11]. To address the overestimation issue, Fujimoto et al. [12] proposed twin delayed deep deterministic policy gradient (TD3), an extension of DDPG. Haarnoja et al. [13] used soft actor-critic to achieve stable, similar performance across different random seeds with maximum entropy objective and to improve exploration. Despite so many improvements, the reinforcement learning method does not exist as a winner-take-all, and the choice of the actual algorithm needs to be determined according to the practical scene.

Due to the self-learning characteristic of deep reinforcement learning, many researchers attempt to apply this technology to social robots and promote robotic intelligence. Prianto et al. [14] presented a SAC-based path planner to handle multi-arm manipulators. Given arbitrary start and target positions, their agent could generate the shortest path, and its arms cooperate very well. The real-time application also verified the effectiveness of their algorithm. Haarnoja et al. [15] combined the automatic temperature mechanism with the original SAC and made a comprehensive analysis in the simulation section. Also, they trained a real-world Minitaur robot to learn walking gaits, and the resulting policy could successfully guide the robot to cross different terrain. Silva et al. [16] proposed a learning-based method to train humanoid robot-soccer players via a dueling DDQN algorithm. The well-trained robots could operate simple actions according to the camera image and complete a game. Then, they transferred the policy to a real robot and showed great potential for DRL-based soccer-robot application.

Since the policy network parameterized in the deep reinforcement learning algorithm is tiny, which is different from traditional deep learning applications, the DRL-based local path planner has great potential to reduce computational costs compared to conventional path planning algorithms. Research on DRL-based obstacle avoidance has been growing fast in recent years. [17] suggested using the human demonstration data and prioritized experience to improve convergence ability. They trained and tested via a Turtlebot 3 mobile robot. The result showed that their method obtained greater generalization ability than the original DDPG algorithm in real-world testing. Wang al et. [18] solved the navigation problem of unmanned aerial vehicles (UAVs) based on GPS signal and sensory information of the local environment via fast-RDPG. They constructed a virtual environment with deed ends for training, and the result showed that the proposed method obtained better convergence performance compared to RDPG algorithm [19]. Rubí al et. [20] solved the quadrotor path following and obstacle avoidance problem with two agents: path following and obstacle avoidance, which can be more interpretable, reducing the training time and permitting to train it safely in the virtual platform.

The works mentioned above are mainly trained in a fixed simulated environment, so it is difficult to quantify and analyze. Moreover, few of them evaluate entirely different state-of-the-art DRL algorithms. This research not only executes the proposed mapless collision avoidance algorithm in a randomized environment but also comprehensively compares DDPG, TD3, and SAC in terms of obstacle avoidance performance.

## III. Approach

### A. Problem definition

Automatic collision avoidance for mobile robots can be characterized as a spatial mapping problem. To realize safe collision avoidance and reach the goal position, the input data should include the obstacle information $\boldsymbol{obstacle}_t$, the goal information $\boldsymbol{goal}_t$ and the self status information $\boldsymbol{status}_t$ at each time step $t$. We hope to build a mapping from the input data mentioned above to desired predicted action $\boldsymbol{a}_t$, which corresponds to the actor network in Fig. 1. The mapping function is as follows:

$$\boldsymbol{a}_t = f(\boldsymbol{obstacle}_t, \boldsymbol{goal}_t, \boldsymbol{status}_t) \tag{1}$$

The goal of the DRL training process is to find out the optimal mapping $f$ to enable the mobile robot complete the

safe obstacle avoidance and reach the goal, given $\boldsymbol{obstacle}_t$, $\boldsymbol{goal}_t$ and $\boldsymbol{status}_t$ at time step $t$.

### B. Observation space and action Space specification

For safe collision avoidance, the observation space $\boldsymbol{s_t}$ should consist of $\boldsymbol{obstacle}_t$, $\boldsymbol{goal}_t$ and $\boldsymbol{status}_t$. The $\boldsymbol{obstacle}_t$ here is the laser data, denoted as $\boldsymbol{\chi} = [d_1, ..., d_n]$, where $d_i$ is the single distance information of LiDAR. The $\boldsymbol{goal}_t$ is the relative target position, a vector denoted as $\boldsymbol{\xi} = [d_{target}, \omega_{target}]$ that represents the distance and angle between robot's current position and target position in polar coordinates. The $\boldsymbol{status}_t$ is the body frame velocity $\boldsymbol{v} = [v_x, v_{yaw}]$, representing the current forward linear velocity and the yaw rate of mobile robot in body frame. Lastly, The output of the DRL policy network is a 1D vector consisting of desired forward linear velocity and the yaw rate, ranging from $0 \sim 0.21m/s$ and $-1.0 \sim 1.0rad/s$, respectively.

### C. Reward function design

The reward function is vital in the DRL algorithm convergence and the final policy performance. A well-designed reward function can guide the agent to reach the destination without collision. The total reward function $r_{total}$ is designed as:

$$r_{total} = r_{goal} + r_{crash} + r_{free} + r_{step} \tag{2}$$

where $r_{goal}$ denotes the distance reward, $r_{crash}$ represents the crash reward, $r_{free}$ is free-space reward, $r_{step}$ means step-penalty reward. The distance reward is designed as:

$$r_{goal} = \begin{cases} r_{reach} & \text{if } d_{goal} < d_{goal,min} \\ \Delta d_{goal} & otherwise \end{cases} \tag{3}$$

where $d_{goal}$ denotes the distance from the robot's current position to the target. The agent reaches the goal when $d_{goal}$ is less than $d_{goal,min}$, and then obtains a sparse reward $r_{reach}$, or it can receive a potential-based shaping reward relative to distance changing as an end of one episode. The crash reward $r_{crash}$ is designed as:

$$r_{crash} = \begin{cases} -r_{crash,i} & \text{if } d_{obs} \in [d_{min,i}, d_{max,i}], i = 1, 2, 3 \\ 0 & otherwise \end{cases} \tag{4}$$

where $d_{obs}$ is the distance between the agent and the nearest obstacle. $r_{crash}$ is obtained when $d_{obs} \in [d_{min,i}, d_{max,i}]$ to keep away from crashing. To further encourage the agent to steer from barriers, free-space reward $r_{free}$ is defined as:

$$r_{free} = -\sum_i \left(1 - d_{laser,i}/d_{laser,max}\right)^4 \tag{5}$$

where $d_{laser,i}$ means $i^{th}$ data of laser measurement, and $d_{laser,max}$ corresponds to its maximum. $r_{free}$ can get extremely large if and only if the mobile robot is very close to obstacles, while it can become small quickly as it is a bit away, which encourages the agent to explore in a narrow actively. To ensure the mobile robots arrive at the destination as much as possible, step-penalty reward $r_{step}$ is included, which increases over time, defined as $r_{step} = -T$.

### D. Network architecture

In this work, some popular DRL algorithms, such as DDPG, TD3, and SAC, are applied to realize automatic obstacle avoidance for a mobile robot. For a fair comparison between different DRL algorithms, we uniformly represent the actor and critic networks as a tiny fully-connected MLP with two hidden layers of 256 nodes, nonlinearized by the ReLU function. As shown in Fig. 2, a 40-dimensional input vector is sampled from observation space and then sent to two dense layers. Lastly, the actor network produces the 2-dimensional predicted actions using a Tanh function. For DDPG and TD3, the actor network's final dense layer (256 nodes) would give direct deterministic actions. For SAC, it would generate mean $\boldsymbol{\mu}$ and deviation $\boldsymbol{\sigma^2}$ as an intermediary, and predicted actions can be obtained by sampling via $\boldsymbol{a} \sim N(\boldsymbol{\mu}, \boldsymbol{\sigma^2})$. As shown in Fig. 3, the critic network receives a 42-dimensional input vector, which combines states and actions, and generates a 1-dimensional Q-value.

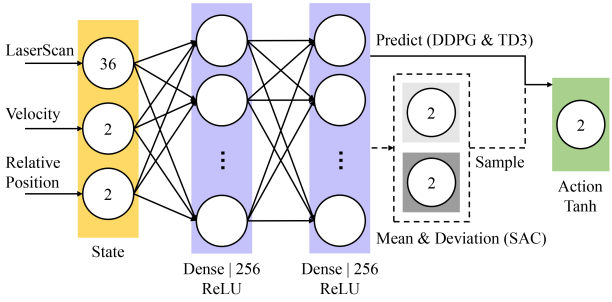


Fig. 2. Actor network of DDPG, TD3 and SAC.

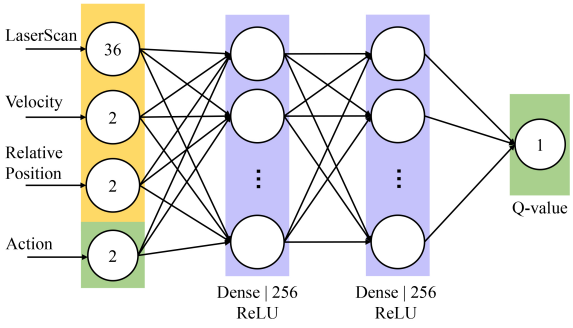


Fig. 3. Critic network for DDPG, TD3 and SAC.

### E. Randomized environment generating

In most cases, training in the maps with a randomized setup can bring better generalization ability and transfer features for DRL policies. To reproduce the scene accurately, we abstract the scene of the obstacle avoidance task into walls and cylindrical obstacles, as shown in Fig. 4.

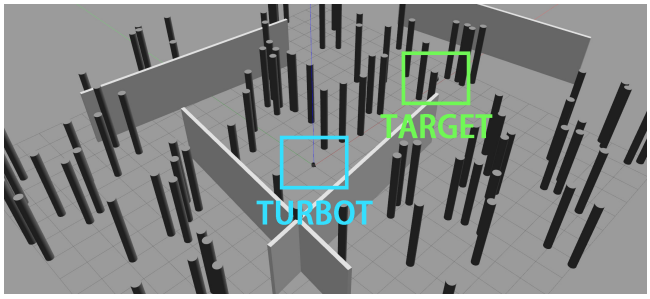


Fig. 4. Illustration of randomized generating maps.

In this case, we can define a new randomized environment generator for training and testing statistically:

$$Env_{new} = Env(d_{target}, r_{wall}, n_{max}, n_{min}) \tag{6}$$

where $d_{target}$ is the distance of target point, $r_{wall}$ is the ratio of generating wall, $n_{max}$ and $n_{min}$ are maximal and minimal number of cylindrical obstacles. The map in the simulation consists of four walls, generated by probability $r_{wall}$ and cylindrical obstacles, of which number $n \sim U[n_{min}, n_{max}]$. The walls in the map can move and rotate randomly within a certain range, thus forming corridor environments with different widths and inclination angles. Cylindrical obstacles can overlap to form obstacles of different shapes, and their positions are randomly sampled on the map. The environment we used can refer to table I. $\rho_{obs}$ is the percentage of obstacle area in total area, that is, the obstacle density.

TABLE I
PREDEFINED MAP

| Env | $d_{target}$ | $r_{wall}$ | $n_{max}$ | $n_{min}$ | $\rho_{obs}$ |
|---|---|---|---|---|---|
| TRAIN | 8 | 0.8 | 70 | 100 | 6-10% |
| TEST1 | 8 | 0.8 | 5 | 50 | 1-5% |
| TEST2 | 8 | 0.8 | 50 | 100 | 5-10% |
| TEST3 | 8 | 0.8 | 100 | 150 | 10-15% |
| TEST4 | 8 | 0.8 | 150 | 200 | 15-20% |

### F. Prioritized buffer mechanism

In the experience replay mechanism of traditional off-policy RL algorithms, transition $(\boldsymbol{s_t}, \boldsymbol{a_t}, r_t, \boldsymbol{s_{t+1}})$ is stored in replay buffer at every step as the experience of the agent. Instead of uniformly sampling data, we hope to use important historical data and sample them more frequently fully. To improve convergence speed, we define sampling probability $P_i$ of $i^{th}$ transition as:

$$P_i = \frac{p_i^\beta}{\sum_k p_k^\beta} \tag{7}$$

where $\beta$ is the regulator factor of priority. When $\beta = 0$, priority is not considered, or sampling is based entirely on priority when $\beta = 1$. The priority $p_i$ of $i^{th}$ transition data can be calculated by:

$$p_i = \delta_i^2 + \varsigma \tag{8}$$

where $\delta$ is the time difference (TD) error of critic network, $\varsigma$ represents a small positive sampling probability for each transition, ensuring each transition data can still be sampled.

## IV. Experiment and Results

### A. Experiment setup

The Turtlebot 3 burger mobile robot platform was used for training and testing in simulation, as shown in Fig. 5. Its maximal linear velocity and angular rate are 0.22 m/s and 2.84 rad/s, respectively. The robot model was trained in ROS gazebo semi-physical simulated platform. As discussed in Section III, the randomized environments for training consist of walls and cylindrical obstacles. To improve the generalization ability, we reset the posture of the obstacles and the mobile robot at the beginning of every episode. The small ball represented the target and was placed in a fixed position with a constant distance from the origin. The LDS-01 lidar sensor on the top of the mobile robot has a view of 360°, and its range is from 0.12m to 3.5m. This paper uses 36-dimensional laser range data, which can be obtained via equidistant sampling from raw laser data.

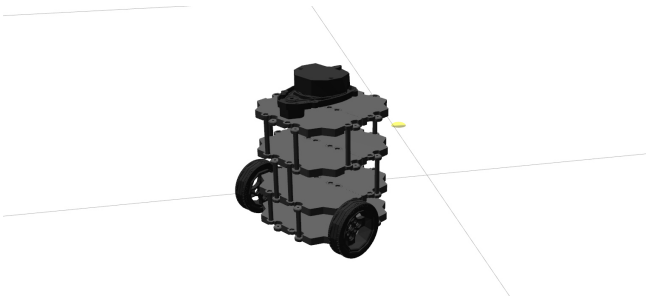

Fig. 5. Turtlebot3 Burger platform in 3D Gazebo simulator.

### B. Simulation results

We compared some state-of-the-art off-policy DRL algorithms operating over continuous action space, including DDPG, TD3, and SAC, in the TRAIN environment in Table I. To represent the policy, we used a fully-connected MLP with two hidden layers of 256 for all algorithms, and ReLU nonlinearities, outputting normalized actions via Tanh function or clipping, as Fig. 2. We did not share any parameters between the policy and Q-value network, and we used ADAM [21] with the same learning rate of 0.0003 to train all the networks and temperature parameter $\alpha$. We used SAC implementation according to the origin paper [22] with automatic temperature. For more details about hyperparameters, see Table III.

The training result of DDPG, TD3 and SAC can refer to Fig. 6(a)(b). Both TD3 and SAC algorithms converged stably after 1000 episodes. The training curve of DDPG hits a peek at the early stage and drops rapidly, far below TD3 and SAC. As TD3 and SAC can reach the platform at the end of the training stage, the training curve of TD3 always stands above

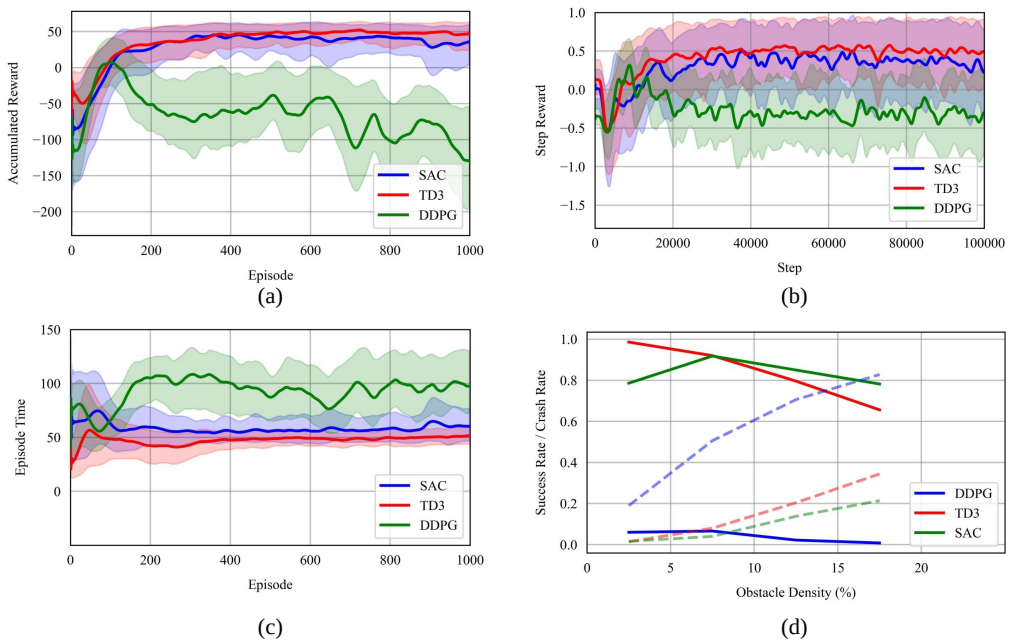


Fig. 6. Simulation results. (a)(b) the training curves for DDPG, TD3, and SAC with the increasing of episode number and step number, respectively; (c) the changing curve of episode time concerning episode number; (d) the passing rate and crash rate with different obstacle densities. The solid line denotes the average performance over three random seeds, and the shaded region corresponds to the minimal and maximal values of the seeds in (a)(b)(c). The solid line in (d) represents the passing rate, and the dashed line is the crash rate.

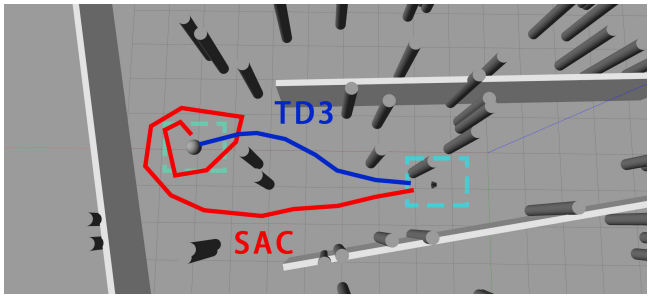


Fig. 7. Trajectory instance of TD3 and SAC in TEST3 environment.

the curve of SAC. Fig. 6(c) shows the changing curve of episode time to episode number. TD3 holds minimal episode time throughout the training stage, and SAC is next only to TD3. The episode time of DDPG is much higher than that of TD3 and SAC due to the training failure. In this case, we can draw a rough conclusion that the performance of TD3 and SAC in the context of learning-based obstacle avoidance is much better than DDPG. The TD3 algorithm might slightly outperforms SAC in this task.

We also compared the success and crash rate of three DRL algorithms, as shown in Table II. The corresponding visualization result can refer to Fig. 6(d). Noted that the success rate plus the crash rate is not equal to 1.0, and the remained part is named trap rate, which denotes the rate of agents falling into local optimum and unable to escape. The agent with DDPG maintains a high crash rate up to $0.5 \sim 0.8$ and an extremely high trap rate from $0.1 \sim 0.9$ in all test environments. One possible reason why DDPG obtains poor obstacle avoidance performance is due to its high accumulative overestimation of Q-value as the training progresses. The agent with TD3 and SAC performs well and gains a high success rate ranging from $0.92 \sim 0.98$ in TEST1 and TEST2 environments and a lower success rate in TEST3 and TEST4 environments as obstacle density increases, which outperforms the agent with DDPG with the same condition. However, TD3 and SAC have distinct inclinations in different environments. As shown in Fig. 6(d), TD3 has a 0.20 higher success rate than SAC in the TEST1 environment and the same success rate in the TEST2 environment. However, SAC outperforms TD3 in TEST3 and TEST4 environments with $0.05 \sim 0.13$ higher success rate. A reasonable explanation for the poor performance of SAC in a low obstacle density environment is its multi-modal characteristics [23]. This feature will cause agents to explore more near mainline events, as shown in Fig. 7. Adequate training can solve this problem by reducing exploration. In summary, TD3 and SAC perform far better than DDPG in obstacle avoidance tasks. The TD3 algorithm is preferred for the environment with low obstacle density and SAC for high obstacle density.

## V. Conclusion

In this paper, we propose a new DRL-based collision avoidance scheme to implement the mobile robot's safe point-to-point path planning tasks. Compulsory modules, such as a well-designed reward function, randomized environment

TABLE II
TABLE TYPE STYLES

| Environment | Algorithm | Success Rate | Crash Rate |
|---|---|---|---|
| TEST1 | DDPG | 0.06 | 0.19 |
| | TD3 | **0.986** | **0.014** |
| | SAC | 0.786 | 0.016 |
| TEST2 | DDPG | 0.066 | 0.506 |
| | TD3 | **0.920** | 0.08 |
| | SAC | 0.918 | **0.04** |
| TEST3 | DDPG | 0.022 | 0.706 |
| | TD3 | 0.796 | 0.204 |
| | SAC | **0.850** | **0.138** |
| TEST4 | DDPG | 0.008 | 0.828 |
| | TD3 | 0.656 | 0.344 |
| | SAC | **0.782** | **0.214** |

TABLE III
TABLE TYPE STYLES

| Hyperparameter | Value |
|---|---|
| Batch size | 512 |
| Replay buffer size | 15000 |
| Step per learning update | 1 |
| Optimizer | Adam |
| Loss function | Mean squared error |
| Target smooth coefficient | 0.01 |
| Initial temperature (SAC) | 0.2 |
| Target entropy (SAC) | -dim(Action) |
| Delayed factor (TD3) | 2 |
| episilon (DDPG & TD3) | 0.8 |
| epsilon_decay (DDPG & TD3) | 0.99995 |

generator, and prioritized experience replay, are adopted to improve convergence and obstacle avoidance performance. In a simulated environment, TD3 and SAC converge successfully and hit a stable platform as training processes when DDPG fails to optimize itself and obtains decreasing rewards during most training time. The comparison of success rate and crash rate shows that TD3 can bring a higher success rate in environments with obstacle density of 0.01 to 0.05, and SAC can outperform TD3 in obstacle density of 0.10 to 0.20. Both TD3 and SAC can bring smooth trajectories in short-distance tasks of simulation. In the future, memory-based deep reinforcement learning can be considered to improve the passing rate when encountering deed ends or narrow lanes. Also, the reality gap is still the main challenge, and more work should focus on transferring well-trained DRL-based policy to a real robot.

## ACKNOWLEDGMENT

This work is supported by Guangdong Basic and Applied Basic Research Foundation (No. 2020A1515110815).

## REFERENCES

[1] F. Gao, Y. Lin, and S. Shen, "Gradient-based online safe trajectory generation for quadrotor flight in complex environments," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2017, pp. 3681–3688.

[2] Z. Wang, H. Ye, C. Xu, and F. Gao, "Generating large-scale trajectories efficiently using double descriptions of polynomials," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*, 2021, pp. 7436–7442.

[3] C. Aakash and V. M. Kumar, "Path planning of an UAV with the help of lidar for slam application," *IOP Conference Series: Materials Science and Engineering*, vol. 912, no. 6, p. 062013, aug 2020. [Online]. Available: https://doi.org/10.1088/1757-899x/912/6/062013

[4] Y. Bharadwaja, S. Vaitheeswaran, and C. Ananda, "Obstacle avoidance for unmanned air vehicles using monocular-slam with chain-based path planning in gps denied environments," *Journal of Aerospace System Engineering*, vol. 14, no. 2, pp. 1–11, 2020.

[5] S. Chen, H. Chen, W. Zhou, C. Y. Wen, and B. Li, "End-to-End UAV Simulation for Visual SLAM and Navigation," *arXiv e-prints*, p. arXiv:2012.00298, Dec. 2020.

[6] C. Wu, "Towards linear-time incremental structure from motion," in *2013 International Conference on 3D Vision - 3DV 2013*. IEEE, 2013, pp. 127–134.

[7] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *arXiv e-prints*, p. arXiv:1509.02971, Sep. 2015.

[8] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller, "Deterministic policy gradient algorithms," in *International conference on machine learning*. PMLR, 2014, pp. 387–395.

[9] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal Policy Optimization Algorithms," *arXiv e-prints*, p. arXiv:1707.06347, Jul. 2017.

[10] J. Schulman, S. Levine, P. Moritz, M. I. Jordan, and P. Abbeel, "Trust Region Policy Optimization," *arXiv e-prints*, p. arXiv:1502.05477, Feb. 2015.

[11] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," in *International conference on machine learning*. PMLR, 2016, pp. 1928–1937.

[12] S. Fujimoto, H. van Hoof, and D. Meger, "Addressing Function Approximation Error in Actor-Critic Methods," *arXiv e-prints*, p. arXiv:1802.09477, Feb. 2018.

[13] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor," *arXiv e-prints*, p. arXiv:1801.01290, Jan. 2018.

[14] E. Prianto, M. Kim, J.-H. Park, J.-H. Bae, and J.-S. Kim, "Path planning for multi-arm manipulators using deep reinforcement learning: Soft actor–critic with hindsight experience replay," *Sensors*, vol. 20, no. 20, p. 5911, 2020.

[15] T. Haarnoja, S. Ha, A. Zhou, J. Tan, G. Tucker, and S. Levine, "Learning to Walk via Deep Reinforcement Learning," *arXiv e-prints*, p. arXiv:1812.11103, Dec. 2018.

[16] I. J. da Silva, D. H. Perico, T. P. D. Homem, and R. A. da Costa Bianchi, "Deep reinforcement learning for a humanoid robot soccer player," *Journal of Intelligent & Robotic Systems*, vol. 102, no. 3, pp. 1–14, 2021.

[17] H. Niu, Z. Ji, F. Arvin, B. Lennox, H. Yin, and J. Carrasco, "Accelerated sim-to-real deep reinforcement learning: Learning collision avoidance from human player," in *2021 IEEE/SICE International Symposium on System Integration (SII)*, 2021, pp. 144–149.

[18] C. Wang, J. Wang, X. Zhang, and X. Zhang, "Autonomous navigation of uav in large-scale unknown complex environment with deep reinforcement learning," in *2017 IEEE Global Conference on Signal and Information Processing (GlobalSIP)*, 2017, pp. 858–862.

[19] N. Heess, J. J. Hunt, T. P. Lillicrap, and D. Silver, "Memory-based control with recurrent neural networks," *arXiv e-prints*, p. arXiv:1512.04455, Dec. 2015.

[20] B. Rubí, B. Morcego, and R. Pérez, "Quadrotor path following and reactive obstacle avoidance with deep reinforcement learning," *Journal of Intelligent & Robotic Systems*, vol. 103, no. 4, pp. 1–17, 2021.

[21] D. P. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization," *arXiv e-prints*, p. arXiv:1412.6980, Dec. 2014.

[22] T. Haarnoja, A. Zhou, K. Hartikainen, G. Tucker, S. Ha, J. Tan, V. Kumar, H. Zhu, A. Gupta, P. Abbeel, and S. Levine, "Soft Actor-Critic Algorithms and Applications," *arXiv e-prints*, p. arXiv:1812.05905, Dec. 2018.

[23] T. Haarnoja, H. Tang, P. Abbeel, and S. Levine, "Reinforcement Learning with Deep Energy-Based Policies," *arXiv e-prints*, p. arXiv:1702.08165, Feb. 2017.