frontiers | Frontiers in Neurorobotics

ORIGINAL RESEARCH
published: 02 May 2022
doi: 10.3389/fnbot.2022.883562

Deep Reinforcement Learning Based Trajectory Planning Under Uncertain Constraints

Lienhung Chen[1], *Zhongliang Jiang*[1], *Long Cheng*[2], *Alois C. Knoll*[1] *and Mingchuan Zhou*[1,3]*

[1] Department of Computer Science, Technische Universität München, Munich, Germany, [2] College of Computer Science and Artificial Intelligence, Wenzhou University, Wenzhou, China, [3] College of Biosystems Engineering and Food Science, Zhejiang University, Hangzhou, China

With the advance in algorithms, deep reinforcement learning (DRL) offers solutions to trajectory planning under uncertain environments. Different from traditional trajectory planning which requires lots of effort to tackle complicated high-dimensional problems, the recently proposed DRL enables the robot manipulator to autonomously learn and discover optimal trajectory planning by interacting with the environment. In this article, we present state-of-the-art DRL-based collision-avoidance trajectory planning for uncertain environments such as a safe human coexistent environment. Since the robot manipulator operates in high dimensional continuous state-action spaces, model-free, policy gradient-based soft actor-critic (SAC), and deep deterministic policy gradient (DDPG) framework are adapted to our scenario for comparison. In order to assess our proposal, we simulate a 7-DOF Panda (Franka Emika) robot manipulator in the PyBullet physics engine and then evaluate its trajectory planning with reward, loss, safe rate, and accuracy. Finally, our final report shows the effectiveness of state-of-the-art DRL algorithms for trajectory planning under uncertain environments with zero collision after 5,000 episodes of training.

Keywords: reinforcement learning, neural networks, trajectory planning, collision avoidance, uncertain environment, robotics

OPEN ACCESS

Edited by:
Hang Su,
Fondazione Politecnico di Milano, Italy

Reviewed by:
Bo Lu,
Soochow University, China
Dandan Zhang,
Imperial College London, United Kingdom
Chenming Wu,
Tencent Holdings Limited, China

***Correspondence:**
Mingchuan Zhou
mczhou@zju.edu.cn

Received: *25 February 2022*
Accepted: *28 March 2022*
Published: *02 May 2022*

Citation:
Chen L, Jiang Z, Cheng L, Knoll AC and Zhou M (2022) Deep Reinforcement Learning Based Trajectory Planning Under Uncertain Constraints. Front. Neurorobot. 16:883562. doi: 10.3389/fnbot.2022.883562

1. INTRODUCTION

Multi-Degree-of-Freedom (Multi-DOF) robotic arm is widely used in a variety of automation scenarios, including the automotive industry, equipment fabrication, food industry, health care, and agriculture. In the past, Multi-DOF robotic arms usually operated in isolated, structured environments, and tasks that need to adapt to actual conditions are often done by humans. Human-Robot Collaboration (HRC) combines the flexibility of humans and the efficiency of robots, making manufacturing more flexible and productive (Vysocky and Novak, 2016). However, it is a challenge for traditional motion planning algorithms to define a safe, collision-free HRC system, since all its parameters are established based on a specific environment which makes it difficult to adapt new workspace. Probability Road Map (PRM) and Rapidly-exploring Random Tree (RRT) for instance, are not suitable for dynamics environments, since they require higher real-time performance of algorithms to deal with dynamic obstacles, i.e., they need to construct a real-time mapping of obstacles in the configuration space so as to plan a collision-free path, which is very computationally expensive (Adiyatov and Varol, 2017; Kurosu et al., 2017; Wei and Ren, 2018; Wittmann et al., 2020; Jiang et al., 2021; Liu et al., 2021). Another common approach, potential field (PF), has

less computation and better real-time control compared to PRM and RRT, however, it often gets stuck in the local minimum, and has limited performance when the obstacles are in the vicinity of the target (Flacco et al., 2012; Lu et al., 2018, 2021; Xu et al., 2018; Melchiorre et al., 2019; Zhou et al., 2019). Therefore, finding an efficient, safe, and flexible motion planning algorithm is required. Reinforcement learning (RL) paves an alternative way to solve these challenges, especially in many high-dimensional tasks or games, RL can exhibit its outperformance. In DeepMind, it is even thought to be enough to reach general AI (Shanahan et al., 2020).

Recently, deep RL, which leverages neural networks as function approximation, has been proven its effectiveness in many different kinds of high complexity of robotic control tasks. Joshi et al. (2020) shows multiple RGB images with double-deep Q-learning can reach over 80% success rate in different grasping tasks without training on a large dataset. In Gu et al. (2017), the robot learns to complete a door opening task with DDPG and Normalized Advantage Function algorithm (NAF) with only a few hours of training. Another example shown in Haarnoja et al. (2018a) with soft Q-learning a robot can learn how to stack Lego blocks together within 2 h of policy training. Therefore, we carry out an idea, using the DRL-based method to tackle complex trajectory planning under uncertain environments. However, deep RL still faces some challenges: (1) defining an appropriate reward function is not straightforward, especially dealing with high dimensional problems, it is easy to obtain the result we incentivize instead of what we intended. (2) In simple tasks, normally an RL agent can discover an optimal policy in a short period, however when encountering complex tasks it may take a few million training steps to achieve the desired result. (3) It is hard to prevent an RL agent from overfitting, to overcome this problem an agent should be trained on a large distribution of environments, but it's very computationally expensive.

In this article, collision-free trajectory planning under uncertain environments is tackled with state-of-the-art DRL algorithms. Since the robotic systems are high dimensional and the state, action space is continuous, model-free Deep Neural Networks (DNN) approaches for Q- and policy-function approximations are used, which has shown its effectiveness in Amarjyoti (2017). Moreover, we expect our approaches to be more suitable for continuous and stochastic environments as well as to have higher sample efficiency and stability, we leverage a combined version, actor-critic based network, which updates the policy network for better action choices also updates the value network for more precise evaluation on policy at each step (Sutton and Barto, 2018). The primary contributions of this paper are summarized as follows:

- Construct an appropriate dense reward function that includes distance-to-goal reward and distance between obstacles reward, and the weight between both rewards is tuned by comparing the performance across different random seeds, in order to make sure the robot manipulator can follow the goal as long as possible, while also avoid collision with dynamic obstacles.
- Build an uncertain environment in a physics engine to simulate a human coexistent environment and apply state-of-the-art DRL algorithms to 7-DOF robots. Then compare the accuracy, safe rate, and reward of two model-free, policy gradient-based algorithms, SAC, and DDPG.
- To further improve learning efficiency and stability, the state space of goal and obstacles are set to relative position and velocity instead of absolute so that the RL agent can learn the correlation between the end-effector and obstacles as well as the goal, as shown in Section 4.

The rest of the article is structured as follows. Section 2 discusses the study related to the traditional trajectory planning method. Section 3 presents our method and its workflow. Section 4 demonstrates our experiment setup and evaluation of our proposed approaches. Section 5 gives the conclusion of this article and future study.

2. RELATED STUDY

There are currently some possible solutions to trajectory planning and obstacle avoidance. With its probability completeness and exploration efficiency, the RRT has been widely applied in Multi-DOF manipulator's collision-free trajectory planning. Adiyatov and Varol (2017) introduced RRT Fixed Nodes Dynamic (RRT*FND), with the procedures of Reconnect and Regrow in the RRT*FND algorithm, the manipulator can repair the path with an average of 300 ms when encountering an invalid path caused by a dynamic obstacle. Wei and Ren (2018) proposed an improved RRT algorithm, called Smoothly RRT (S-RRT), to generate a smoother path and more stable motion when avoiding obstacles which have shown better exploring speed and exploring efficiency than Basic-RRT and Bi-RRT. In a dual-arm robot pick-and-place environment, Kurosu et al. (2017) regard one of the robot arms as a dynamic obstacle, leveraging the RRT algorithm to effectively avoid collision with another arm during pick-and-place tasks. Although RRT-based algorithm has shown its robustness in either dynamic or static obstacles avoidance, constantly, and randomly moving obstacles avoidance, still requires more research.

Another common approach for collision avoidance trajectory planning is the artificial potential field (APF). This method leverages the PF force of attraction for reaching the goal and repulsion for avoiding obstacles. Xu et al. (2018) leverage a similar algorithm to APF, called velocity potential field (VPF), to avoid collision with a static/dynamic obstacle and a collaborative robot arm. They use the velocity of the robot instead of the distance in APF to avoid suffering from local minima problems when attractive and repulsive forces/velocities confront each other on the same line. Flacco et al. (2012) leverage a simple version of APF, a repulsive vector, generated by the distance to estimate obstacles velocity for collision avoidance. Melchiorre et al. (2019) also leverage the repulsive vector with the distance calculated from the point cloud and have also shown its effectiveness in avoiding collision with static/dynamic obstacles. However, the PF has limited performance when encountering

two obstacles that are placed too close to each other. For example, if the goal is in-between or behind two close obstacles, the robot will neglect the goal and turn away.

The other approach is PRM, which takes random samples from the configuration space of the robot and finds a collision-free path between the start and goal nodes. Liu et al. (2021) proposed a Grid-Local PRM that combined a mapping model, sampling strategies, lazy collision detection, and a single local detection method. This proposed method can implement for dynamic path planning for static/dynamic obstacle avoidance. Wittmann et al. (2020) introduce the Obstacle-related Sampling Rejection Probabilistic Roadmap planner (ORSR-PRM). They leverage PRM for trajectory planning and PF for obstacle avoidance in real-time. Similar to the PF method, the probability of generating nodes in between narrow passages is very small, and hence, no path will be planned through the gap.

To overcome the above problem in traditional path planning methods, we proposed another method. Instead of finding a path in configuration space or tackling complicated optimization problems, we leverage the model-free DRL method that allowed the manipulator to autonomously learn optimal collision-free trajectory planning in an uncertain environment.

3. METHODS

The four essential parts of RL are policy, reward function, value function, and model of the environment. With the idea that an intelligent agent should learn to take a sequence of actions that will lead to maximizing cumulative rewards interacting with the environment. Hence, the agent should *exploit* what it has experienced in order to obtain rewards, but also *explore* in order to make better action decisions in the future (Sutton and Barto, 2018). The Basic RL problem is modeled as the Markov decision process (MDP) with elements S_t, A_t, $P(S_{t+1}|S_t, A_t)$, γ, $R(S_{t+1}|S_t, A_t)$, where t represents timestep, S_t and S_{t+1} represent the current state and next state, respectively, A_t stands for the current action, $P(S_{t+1}|S_t, A_t)$ stands for the transition probability of being in S_{t+1} when taking action A_t in the current state S_t, and $\gamma \in [0, 1)$ represents discount factor which determines the importance of future rewards, $R(S_{t+1}|S_t, A_t)$ represents the immediate reward received after transitioning from the current state S_t to the next state S_{t+1}, due to the taken action A_t. In MDP, we assume that the transition probability (or the probability of moving to the next state S_{t+1}) depends on the current state S_t and the decision action A_t. But given S_t and A_t, it is conditionally independent of all previous states and actions.

3.1. Deep Reinforcement Learning

In the case of complex systems such as robotic systems, the explicit model of the dynamics in the environment associated with MDP, i.e., transition probability function, is often not available or difficult to define. Therefore, a model-free-based method is required. Q-learning is one of the most important breakthroughs in RL also known as the off-policy Temporal Difference (TD) control algorithm, defined by

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)], \quad (1)$$

where S_t and A_t are state and action in timestep t, respectively. α stands for learning rate. R_{t+1} is the obtained immediate reward due to the taken action A_t. a represents the action that has a maximum Q-value from the state S_{t+1}. The optimal action-value function can be directly approximated by the learned action-value function Q, which dramatically simplified the analysis of the algorithm and has been proven for convergence (Sutton and Barto, 2018).

In traditional Q-learning, we utilize Q-table to help track states, actions, and corresponding expected rewards. However, for continuous action and state-space such as robotic systems, it is infeasible to build up a large table. Therefore, we need a function approximation for the action-value function Q and a DNN is one of the efficient and easy techniques to approximate a non-linear function. However, RL with DNN is pretty unstable, the weights of the network can oscillate or diverge due to the high correlation between actions and states. To overcome this issue, we need to leverage two important techniques, *Experience Replay*, and *Target Network*. By Experience Replay, the agent's experience at each time step will be stored in replay memory as the tuple $(S_t, A_t, R_{t+1}, S_{t+1})$, and when the replay memory size is equal to or bigger than a mini-batch size, we then uniformly sample the memory randomly for a mini-batch of experience and use this to learn off-policy, in order to break the correlation (Lin, 1992). Moreover, to make training more stable, a target network is used for calculating the estimate of optimal future value $\max_a Q(S_{t+1}, a)$ in the Bellman equation, and hence, the loss function can be defined as

$$L(\theta) = \{[R_{t+1} + \gamma \max_a Q(S_{t+1}, a; \theta_{target})] - Q(S_t, A_t; \theta_{prediction})\}^2, \quad (2)$$

where $\theta_{prediction}$ are prediction network's weights updated in every iteration, whereas θ_{target} are the target network's weights, which are not trained but periodically synchronized with the parameters of the prediction Q-network.

3.2. The Proposed DRL-Based Trajectory Planning for Uncertain Environments

In this section, we define the setup for the DRL framework, such as the state space $\mathcal{S}$, the action space $\mathcal{A}$, and the reward function.

3.2.1. State Space

In our experiment the robot manipulator, we used is 7-DOF, therefore, if we set joint positions and velocity as the observations, the learning efficiency of the agent is quite low (Henderson et al., 2018) or may even be unable to find the optimal trajectory. To overcome this issue, we instead use the end-effector position p_e and velocity $\dot{p_e}$ then calculate inverse kinematic (IK) to control joint position. Moreover, we use relative position and velocity to the end-effector instead of obstacles or

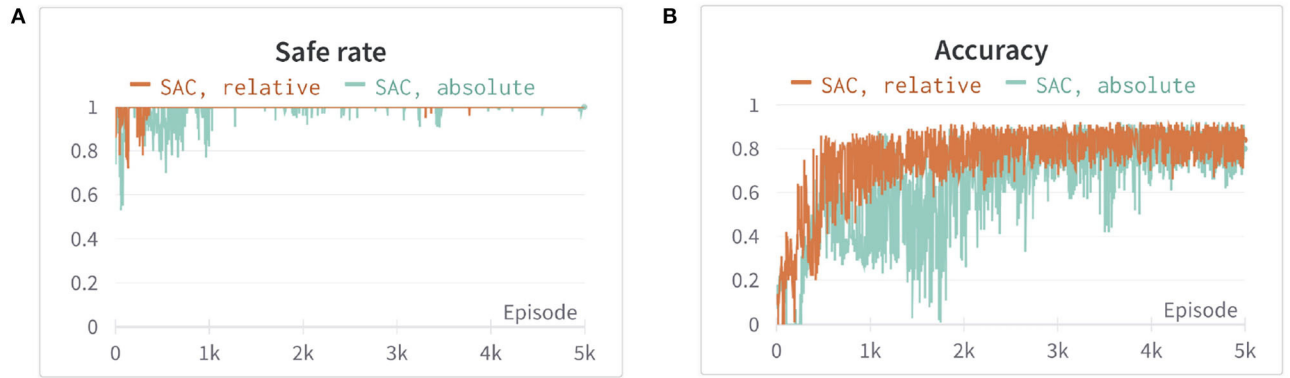

FIGURE 1 | The comparison of two different observation spaces set up in the first environment. Both are using soft actor-critic (SAC) and with the same hyperparameters setting. The orange line is the result using relative position and velocity in observation space, whereas the blue line is using position and velocity. **(A)** The safe rate (defined in Section 4.2) of different observation spaces. **(B)** The accuracy (defined in Section 4.2) of different observation spaces. Each episode corresponds to 100 time steps.

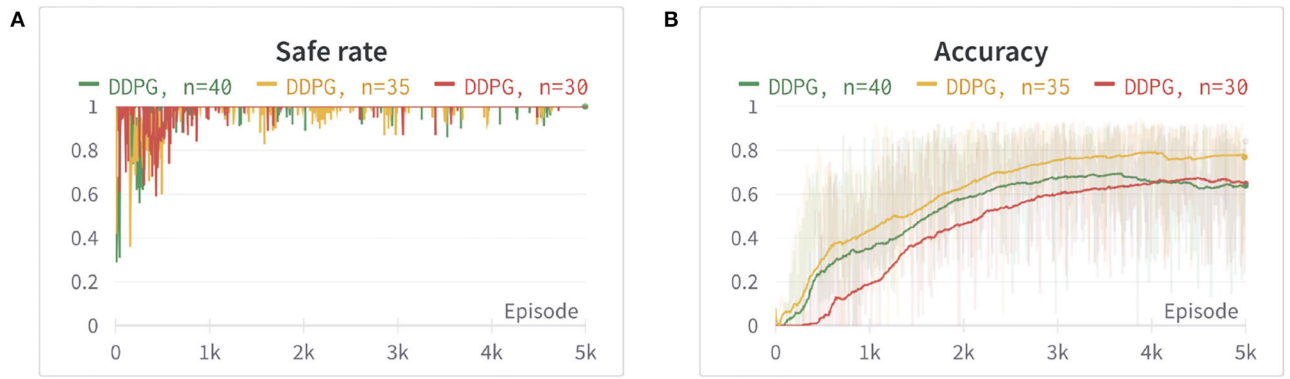

FIGURE 2 | The comparison between different power (n) of the exponential decay function. **(A)** The safe rate (defined in Section 4.2) of three different power (n) of the exponential decay function. **(B)** The accuracy (defined in Section 4.2) of three different power (n) of the exponential decay using an exponential moving average for better visualization. The two figures show that $n = 35$ has better learning efficiency, safe rate, and accuracy.

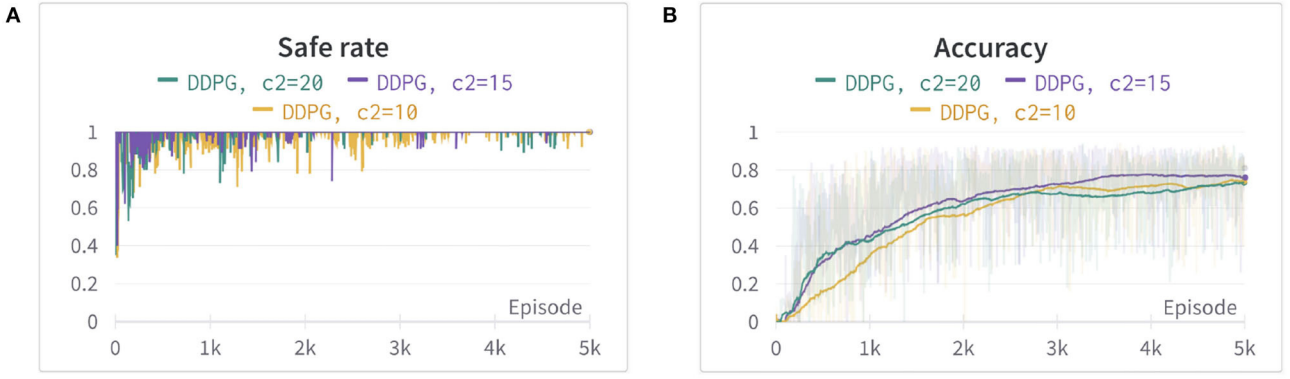

FIGURE 3 | The comparison between different weights of reward R_O. **(A)** The safe rate (defined in Section 4.2) of three different weights of reward R_O. **(B)** The accuracy (defined in Section 4.2) of three different weights of reward R_O using an exponential moving average for better visualization. The two figures show that $c2 = 15$ has better learning efficiency, safe rate, and accuracy.

the goal position $\bar{p_o}/\bar{p_t}$ and velocity $\dot{\bar{p}_o}$ / $\dot{\bar{p}_t}$, which has shown faster convergence and higher stability in **Figure 1**. The above information is assumed known and obtained from the sensor. Furthermore, to increase learning efficiency, we constrain the manipulator in a specific workspace, and hence, the robot will only explore its reachable area and the area with the goal nearby. The State-space S is hence defined as

$$\mathcal{S} = \{p_e,\ \dot{p_e},\ \bar{p_t},\ \dot{\bar{p}_t},\ \bar{p_o},\ \dot{\bar{p}_o}\}. \tag{3}$$

3.2.2. Action Space

As we mentioned in section A, we set the end-effector position as the observation for better learning efficiency. Therefore, we can reduce the dimension of actions from seven dimensions to three dimensions with fixed orientations. The action space $\mathcal{A}$ is defined as

$$\mathcal{A} = \{\Delta x,\ \Delta y,\ \Delta z\}, \tag{4}$$

where Δx,Δy,Δz are bounded between -0.1 and 0.1 such that we can avoid sudden movements of the robotic arm in every time step due to excessive output of the action.

3.2.3. Reward Function

The reward function mixes reward variables into a single output value and provides feedback for an agent to learn what we incentive. In our case, we expect the robot to follow the goal as long as possible while avoiding dynamic obstacles. Therefore, we define the reward function by a weighted sum of two terms: First, the distance between the end-effector and the goal. Second, the closest distance between the robot manipulator and

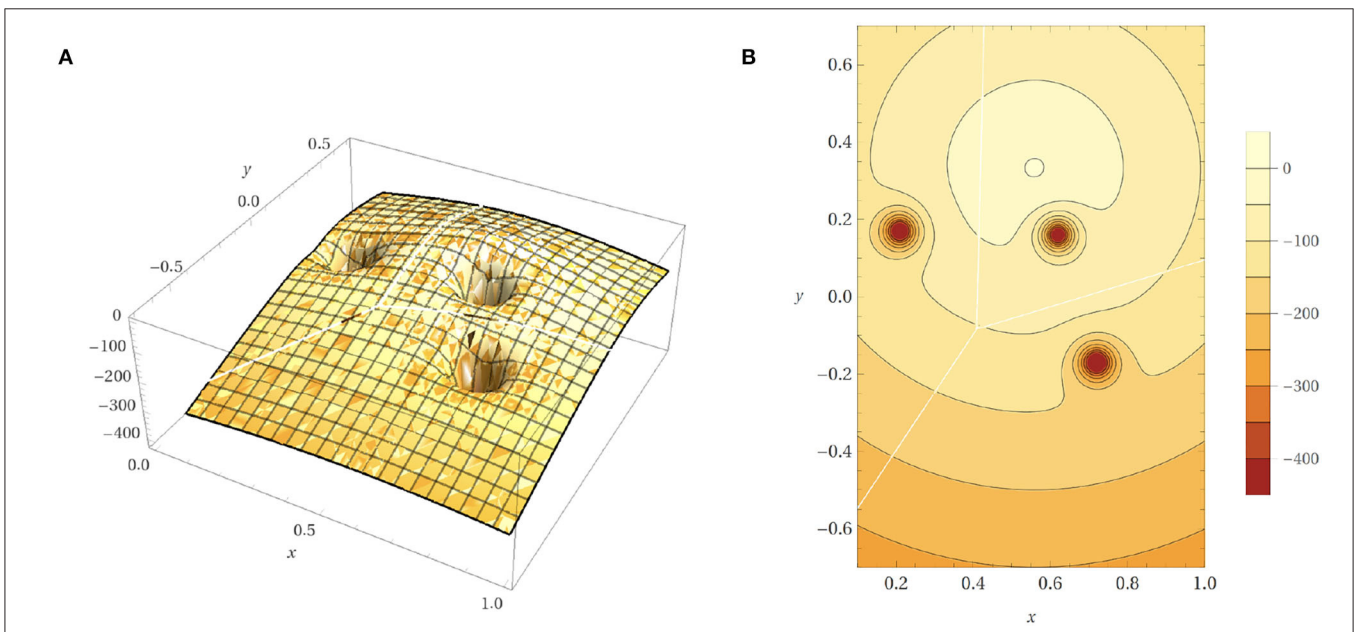

FIGURE 4 | Reward function on the planar section of the workspace. **(A)** The 3D plot of the reward function. **(B)** Contour plot of reward function.

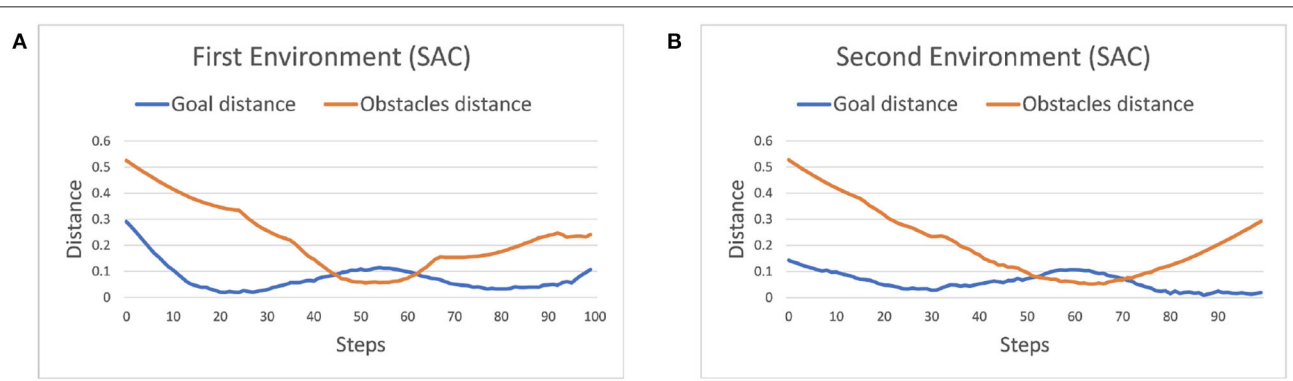

FIGURE 5 | The behavior of the manipulator with respect to the distance(m) in one of the episodes (100-time steps). **(A)** Distance between robot and obstacles as well as the target (first scenario). **(B)** Distance between robot and obstacles as well as the target (second scenario).

obstacles. Moreover, we use negative a reward over a positive so that the robot will try its best to avoid penalties and, hence, can learn as quickly as possible. The reward function is defined as

$$R = -c_1 R_T - c_2 R_O, \tag{5}$$

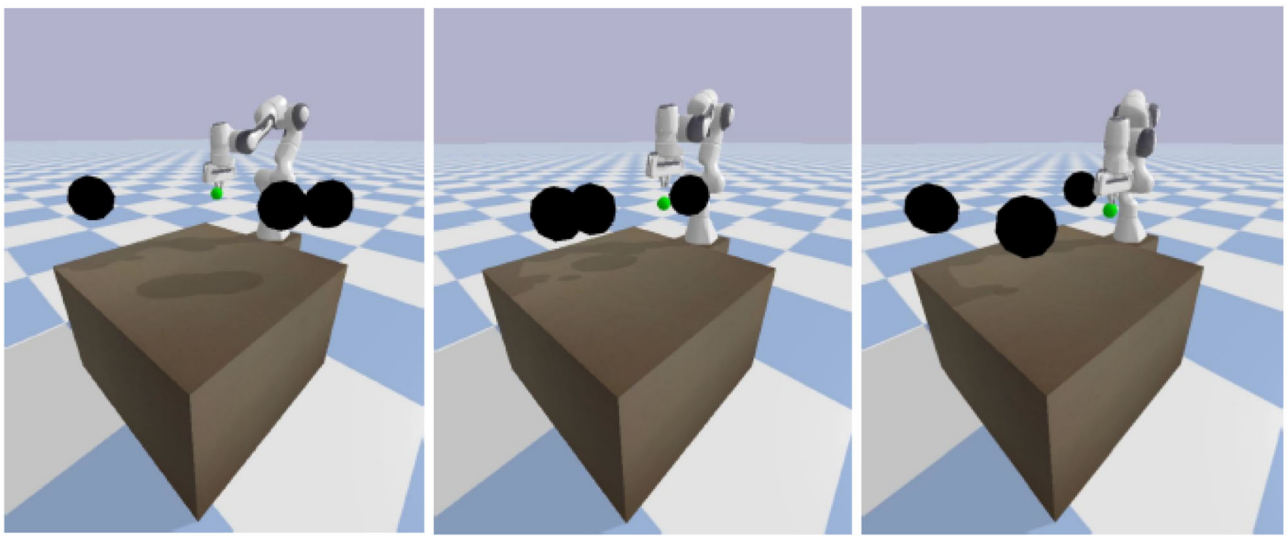

FIGURE 6 | From left to right, first, the robot learns to reach the goal. Second, avoid collisions with dynamic obstacles. Third, keep reaching the goal.

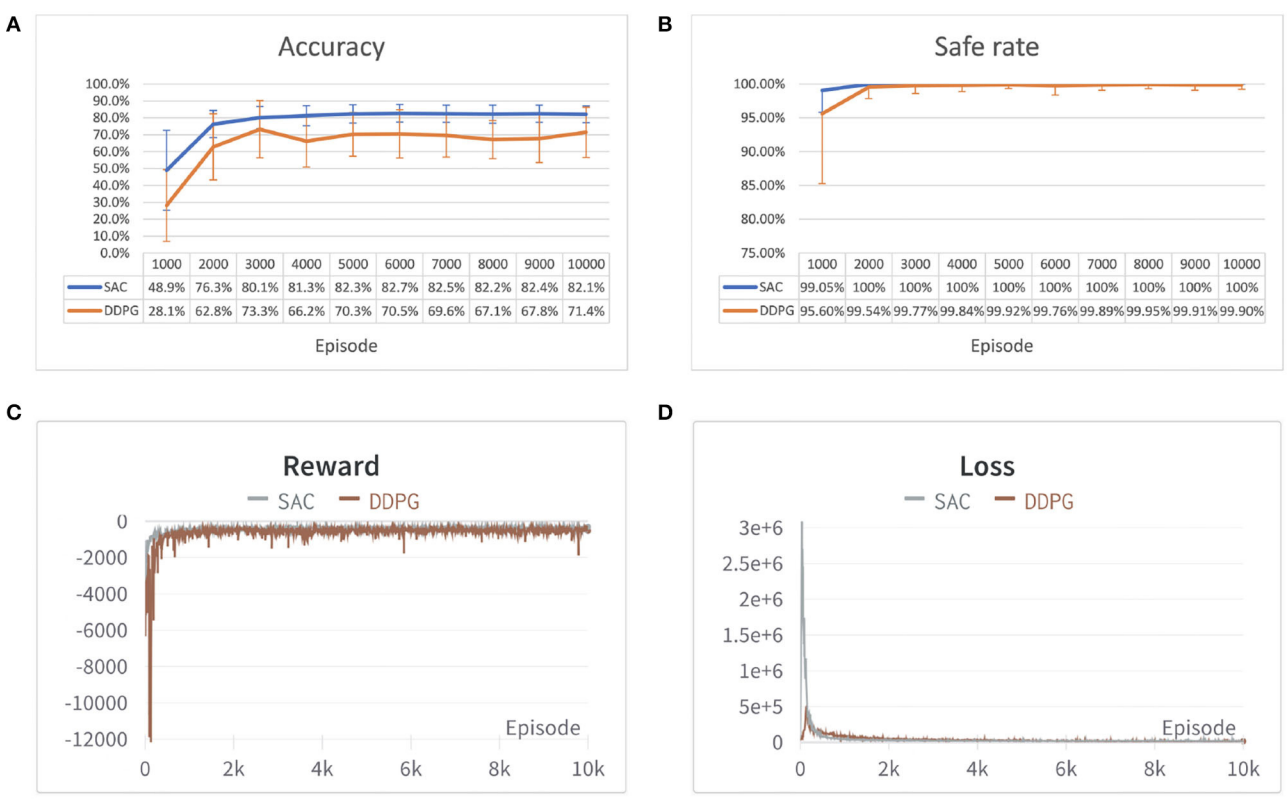

FIGURE 7 | Performance comparison of SAC and Deep Deterministic Policy Gradient (DDPG) algorithm in the first environment. **(A)** Accuracy of different algorithms shown in error bar line graph. **(B)** A safe rate of different algorithms in the error bar line graph. **(C)** The cumulative reward for each episode. **(D)** Loss for each episode. Each episode corresponds to 100 time steps.

where R_T is the reward obtained from the distance between the end-effector and the goal using Euclidean distance (m),

$$R_T = \frac{1}{2} d_T^2, \tag{6}$$

where d_T is the Euclidean distance between the end-effector and the goal. The reward R_O is obtained from the closest distance (m) between the robot manipulator and obstacles,

$$R_O = \left(\frac{a}{a + d_O}\right)^n, \tag{7}$$

where d_O is the closest distance from the obstacles computed by PyBullet. a is set to 1, in order to avoid the denominator equaling zero when a collision happens. The power of exponential decay function $n = 35$ and the weights $c2 = 15$ are determined by using trial and error. We set $c1$ as a fixed value of 500 and tune the parameters n and $c2$ by evaluating the safe rate, accuracy, and learning efficiency, as shown in **Figures 2**, **3** (Since DDPG is more sensitive to parameters, we use DDPG for comparison; Haarnoja et al., 2018b). In order to show our reward function has a maximum, we plot our reward function on the planar section of the workspace, as shown in **Figure 4**. Moreover, since the dynamic/static goal and dynamic obstacles are on the same x-y plane, it can be demonstrated in 3D space instead of 4D for better visualization. As it can be observed from **Figure 4**, the reward decreases as the robot's end-effector moves toward obstacles and increases as it moves toward the goal, and when the end-effector reaches the goal point, the reward is maximum. The behavior of the robot manipulator in two environments is also shown in **Figure 5**. The distance between the end-effector and goal diminishes as the robot approaches the goal and when obstacles are close to the body of the robot, the robot backs off until obstacles move away from the manipulator.

3.2.4. Deep Deterministic Policy Gradient

Deep deterministic policy gradient (DDPG), introduced in Lillicrap et al. (2015), is an actor-critic, model-free algorithm based on the deterministic policy gradient that can operate over continuous action spaces. Traditionally, in policy gradient-based algorithms the policy function is always stochastic, i.e., it is modeled as a probability distribution over actions given the current state. In DDPG, the policy function is instead modeled as a deterministic decision. However, this may lead to a low exploration issue, and hence, they add additive noise to the deterministic action to explore the environment, which is represented as:

$$\mu'(S_t) = \mu(S_t|\theta_t^\mu) + \mathcal{N}, \tag{8}$$

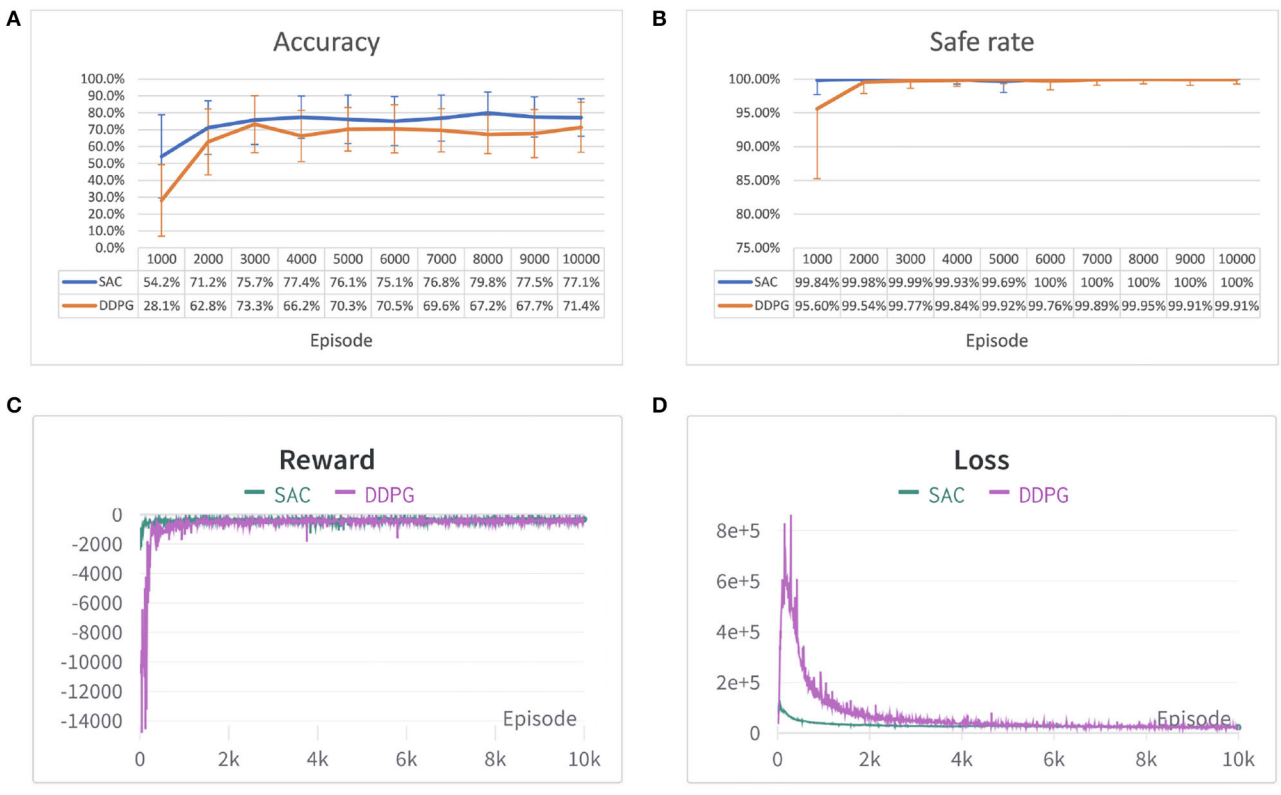

FIGURE 8 | Performance comparison of SAC and DDPG algorithm in the second environment. **(A)** Accuracy of different algorithms shown in error bar line graph. **(B)** A safe rate of different algorithms in the error bar line graph. **(C)** The cumulative reward for each episode. **(D)** Loss for each episode. Each episode corresponds to 100 time steps.

Algorithm 1 | DDPG-based trajectory planning.

Input: batch size: B, target smoothing coefficient τ, discount factor: γ, number of training episode: M, timesteps of each episode: T
Randomly initialize Q network $Q(S, A|\theta^Q)$ and policy network $\mu(S|\theta^\mu)$ with weights θ^Q and θ^μ
Initialize target Q network Q' and target policy network μ' with weights $\theta^{Q'} \leftarrow \theta^Q$, $\theta^{\mu'} \leftarrow \theta^\mu$
Initialize replay buffer $D \leftarrow \phi$
Input: A set of observation state $\mathcal{S} = \{p_e, \dot{p}_e, \bar{p}_t, \dot{\bar{p}}_t, \bar{p}_o, \dot{\bar{p}}_o\}$
Output: A set of optimal policy $\mathcal{A} = \{\Delta x, \Delta y, \Delta z\}$
for episode = 1, M **do**
  Initialize a random noise $\mathcal{N}$ for action exploration
  Receive initial observation state S_1
  for t = 1, T **do**
    Select action $A_t = \mu(S_t|\theta^\mu) + \mathcal{N}_t$ according to the current policy and exploration noise
    Execute action A_t and observe reward R_t and observe new state S_{t+1}
    Store transition (S_t, A_t, R_t, S_{t+1}) in D
    Sample a random minibatch of B transitions (S_i, A_i, R_i, S_{i+1}) from D
    Set $y_i = R_i + \gamma Q'[S_{i+1}, \mu'(S_{i+1}|\theta^{\mu'})|\theta^{Q'}]$
    Update critic by minimizing the loss: $L = \frac{1}{B}\sum_i [y_i - Q(S_i, A_i|\theta^Q)]^2$
    Update the actor policy using the sampled policy gradient: $\nabla_{\theta^\mu} J \approx \frac{1}{B}\sum_i \nabla_A Q(S, A|\theta^Q)|_{S=S_i, A=\mu(S_i)} \nabla_{\theta^\mu}\mu(S|\theta^\mu)|_{S_i}$
    Update the target networks:
    $\theta^{Q'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'}$
    $\theta^{\mu'} \leftarrow \tau\theta^\mu + (1-\tau)\theta^{\mu'}$
  end for
end for

where $\mu(S_t)'$ and $\mathcal{N}$ are the exploration policy and additive noise, here, use an Ornstein-Uhlenbeck process. $\mu(S_t|\theta_t^\mu)$ and θ_t^μ are the output action and parameters of the actor-network. Moreover, the traditional target networks are updated with the parameters of the trained networks every couple of thousand steps, which may cause big differences between the two updates. Therefore, they introduced a *soft target update*, which is actually better to make the target networks slowly track the trained networks, by updating their parameters after each update of the trained network using a sliding average for both the actor and the critic:

$$\theta' \leftarrow \tau\theta + (1-\tau)\theta', \ with\ \tau \ll 1, \tag{9}$$

where θ and θ' represent parameters for the actor- or critic-network and the target actor- or target critic-network. τ is the target smoothing coefficient. The Experience Replay mentioned in Section A is also used here to store past trajectories and provides samples of them to perform gradient updates for better learning efficiency. The detailed pseudo algorithm of DDPG-based trajectory planning is shown in **Algorithm 1**.

3.2.5. Soft Actor-Critic

Similar to DDPG, soft actor-critic (SAC) introduced in Haarnoja et al. (2018b), is also an actor-critic, model-free algorithm that can operate over continuous action spaces, but is based on the stochastic policy by maximizing the expected reward of the actor while maximizing entropy, i.e., achieve the goal while acting as randomly as possible. Hence, the general maximum entropy can be represented as:

$$J(\pi) = \sum_{t=0}^{T} \mathbb{E}_{(S_t, A_t)\sim\rho_\pi}[r(S_t, A_t) + \alpha\mathcal{H}(\pi(\cdot|S_t))], \tag{10}$$

where ρ_π and $\mathcal{H}(\pi(\cdot|s_t))$ are the policy and the entropy. α is the temperature parameter for determining the relative importance of the entropy term against the reward, and hence controls the stochasticity of the optimal policy. Since SAC is an off-policy algorithm, the Experience Replay is also used to improve the learning efficiency. Moreover, in order to decrease the changes between two updates, the soft target update technique in Equation (4) is also applied. In Haarnoja et al. (2018b), they also compare with some other on-policy DRL algorithms, such as TRPO (Schulman et al., 2015), PPO (Schulman et al., 2017), or A3C (Mnih et al., 2016), and has shown its higher sample efficiency. Compare with DDPG, it also has lower hyperparameter sensitivity and higher stability, which is also our case shown in Section 4. The detailed pseudo algorithm of SAC-based trajectory planning is shown in **Algorithm 2**.

4. EXPERIMENT AND RESULTS

In this section, we show that DDPG and SAC can learn optimal trajectory planning for dynamic obstacles collision avoidance. For the evaluation, we compare two different DRL algorithms with safe rate, accuracy, and reward in two different environments.

4.1. Environment Setup

In our experiment, we applied the proposed collision avoidance DDPG and SAC algorithm on a 7-DOF manipulator (Panda from Franka Emika) simulated in a PyBullet physics engine and leveraged the RL toolkit Gym. The environment setup contains a manipulator, a table, a green sphere goal, and three black sphere obstacles, as shown in **Figure 6**. Moreover, we constructed two environments for the evaluation, either a static goal and dynamic obstacles or a dynamic goal and dynamic obstacles. Besides, in order to make sure our model can learn under uncertainty, the starting positions of the goal and three obstacles are uniformly and randomly sampled in a specific range, and the goal moving areas are constrained in the robot's reachable area.

4.2. Evaluation

We evaluate different DRL algorithms in two defined scenarios with the safe rate, accuracy, reward, and loss. Each episode corresponds to 100 time steps, i.e., the robot has to reach the goal and avoid collision within 100 time steps. The safe rate represents

Algorithm 2 | SAC-based trajectory planning.

Input: batch size: B, target smoothing coefficient τ, discount factor: γ, number of training episode: M, timesteps of each episode: T
Randomly initialize Q network Q_1, Q_2, policy network π and value network V with weights θ_1, θ_1, ϕ and ψ.
Initialize target Q networks Q_1' and Q_2', target value network V' with weights $\theta_1' \leftarrow \theta, \theta_2' \leftarrow \theta$ and $\psi' \leftarrow \psi$
Initialize replay buffer D
Input: A set of observation state $\mathcal{S} = \{p_e, \dot{p}_e, \bar{p}_t, \dot{\bar{p}}_t, \bar{p}_o, \dot{\bar{p}}_o\}$
Output: A set of optimal policy $\mathcal{A} = \{\Delta x, \Delta y, \Delta z\}$
for episode = 1, M **do**
  Receive initial observation state S_1
  for t = 1, T **do**
    Select action $A_t \sim \pi_\phi(A_t|S_t)$
    Execute action A_t and observe reward R_t and observe new state S_{t+1}
    Store transition (S_t, A_t, R_t, S_{t+1}) in D
    Sample a random minibatch of B transitions (S_i, A_i, R_i, S_{i+1}) from D
    Update V by minimizing the mean squared error: $\nabla_\psi J_V(\psi) = \frac{1}{B}\sum_i \nabla_\psi V_\psi(S_i)[V_\psi(S_i) - \min_{j=1,2} Q_{\theta_j'}(S_i, A_i) + log\pi_\phi(A_i|S_i)]$
    Update Q by minimizing the soft Bellman residual: $\nabla_{\theta_{1,2}} J_Q(\theta_{1,2}) = \nabla_{\theta_{1,2}} \frac{1}{B}\sum_i \{[Q_{\theta_1}(S_i, A_i) - \alpha R(S_i, A_i) - \gamma V_{\psi'}(S_{i+1})]^2 - [Q_{\theta_2}(S_i, A_i) - \alpha R(S_i, A_i) - \gamma V_{\psi'}(S_{i+1})]^2\}$
    Update π by minimizing the expected KL-divergence: $\nabla_\phi J_\pi(\phi) = \nabla_\phi \frac{1}{B}\sum_i [log\pi_\phi(A_i|S_i) - \min_{j=1,2} Q_{\theta_j'}(S_i, A_i)]$
    Update the target value networks: $\psi' \leftarrow \tau\psi + (1-\tau)\psi'$
  end for
end for

the number of time steps without collision divided by 100 time steps (one episode),

$$Safe\ rate = \frac{number\ of\ timesteps\ without\ collision}{100\ timesteps}. \quad (11)$$

The accuracy stands for a success rate of keeping a distance between the target within 0.05 m while far away from obstacles, and 0.12 m while avoiding collision with obstacles (the radius of obstacles is 0.1 m), therefore, it is not possible to obtain 100% of accuracy, since it also includes time steps from the rest position to the target. The accuracy is set as,

$$Accuracy = \frac{number\ of\ timesteps\ that\ succeed}{100\ timesteps}. \quad (12)$$

The two algorithms' performances were evaluated in two different environments using the same set of parameters respectively, such as learning rate, number of hidden layers, target smoothing coefficient. The environment settings considered for the experiments are (1) dynamic goal and dynamic obstacles: the starting position of both goal and obstacles are uniformly and randomly sampled in a specific range for each episode, and moving with constant speed. (2) fixed goal and dynamic obstacles: the obstacles remain in the same setting as the first one, but with a fixed goal sampled randomly for each episode.

In each experiment, the safe rate, accuracy, reward, and loss per episode have been traced during the training process and compared after 10,000 episodes of 100 time steps each. The result of the two environments is shown in **Figures 7**, **8**. From both **Figures 7C,D**, **8C,D** it can be observed that both algorithms' cumulative reward converges to their maximum value, and the losses do not have significant reductions, i.e., the robot has learned a stable optimal trajectory planning under an uncertain environment. Moreover, both **Figures 7A**, **8A** show that SAC performs much more consistently, efficiently, and higher accuracy, whereas deterministic policy-based DDPG exhibits high variability between episodes and less stable. Furthermore, both **Figures 7B**, **8B** demonstrate that SAC can learn a collision free trajectory with 100% of safe rate within 6,000 episodes, while DDPG still cannot guarantee to reach a 100% of safe rate within 10,000 episodes. Similar results are also corroborated in Gu et al. (2016) and Haarnoja et al. (2018b). The reason for that is because the interplay between the deterministic actor-network and the Q-function makes DDPG unstable and sensitive to hyperparameters, especially for complex and high-dimensional tasks, however, DDPG still shows its effectiveness in both scenarios. Overall, the performance of the stochastic policy-based SAC is more stable and consistent when dealing with complex tasks.

5. CONCLUSION AND FUTURE STUDY

In this article, we presented two state-of-the-art off-policy DRL approaches that can be used to discover optimal trajectory planning under an uncertain environment. Especially, stochastic policy-based SAC can achieve an average of 82% of accuracy in the first scenario and 79% in the second, moreover, with lower variability between episodes and zero collision after 5,000 episodes. The results show the clear potential of the proposed approaches in the application of an uncertain environment, such as HRC scenarios. The future study will transfer the trained model from a simulation environment to real physical robotic manipulators and transfer the learning skill from simulation to the real environment with visual sensing.

DATA AVAILABILITY STATEMENT

The original contributions presented in the study are included in the article/supplementary material, further inquiries can be directed to the corresponding author/s.

AUTHOR CONTRIBUTIONS

LChen implemented the code and draft the manuscript. ZJ assisted to implement the code and discussed the manuscript. LCheng assisted to implement the code and discussed the manuscript. AK guided the research and discussed the results. MZ guided the research, implemented parts of code, and revised the manuscript.

FUNDING

This study was supported by the National Natural Science Foundation of China (Grant Numbers: 32101626 and 61902442) and ZJU 100 Young Talent Program. This study was supported in part by the German Research Foundation (DFG) and in part by the Technical University of Munich (TUM) in the framework of the Open Access Publishing Program.

REFERENCES

Adiyatov, O., and Varol, H. A. (2017). "A novel RRT*-based algorithm for motion planning in dynamic environments," in *2017 IEEE International Conference on Mechatronics and Automation (ICMA)* (Takamatsu), 1416–1421. doi: 10.1109/ICMA.2017.8016024

Amarjyoti, S. (2017). Deep reinforcement learning for robotic manipulation-the state of the art. *arXiv preprint arXiv:1701.08878*. doi: 10.48550/arXiv.1701.08878

Flacco, F., Kröger, T., De Luca, A., and Khatib, O. (2012). "A depth space approach to human-robot collision avoidance," in *2012 IEEE International Conference on Robotics and Automation* (St Paul, MN), 338–345. doi: 10.1109/ICRA.2012.6225245

Gu, S., Holly, E., Lillicrap, T., and Levine, S. (2017). "Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates," in *2017 IEEE International Conference on Robotics and Automation (ICRA)* (Singapore), 3389–3396. doi: 10.1109/ICRA.2017.7989385

Gu, S., Lillicrap, T., Ghahramani, Z., Turner, R. E., and Levine, S. (2016). Q-prop: Sample-efficient policy gradient with an off-policy critic. *arXiv preprint arXiv:1611.02247*. doi: 10.48550/arXiv.1611.02247

Haarnoja, T., Pong, V., Zhou, A., Dalal, M., Abbeel, P., and15 Levine, S. (2018a). "Composable deep reinforcement learning for robotic manipulation," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*, 6244–6251. doi: 10.1109/ICRA.2018.8460756

Haarnoja, T., Zhou, A., Abbeel, P., and Levine, S. (2018b). "Soft actor-critic: off-policy maximum entropy deep reinforcement learning with a stochastic actor," in *International Conference on Machine Learning* (Brisbane, QLD), 1861–1870.

Henderson, P., Islam, R., Bachman, P., Pineau, J., Precup, D., and Meger, D. (2018). "Deep reinforcement learning that matters," in *Proceedings of the AAAI Conference on Artificial Intelligence, Vol. 32* (Stockholm).

Jiang, Z., Li, Z., Grimm, M., Zhou, M., Esposito, M., Wein, W., et al. (2021). Autonomous robotic screening of tubular structures based only on real-time ultrasound imaging feedback. *IEEE Trans. Indus. Electron.* 69, 7064–7075. doi: 10.1109/TIE.2021.3095787

Joshi, S., Kumra, S., and Sahin, F. (2020). "Robotic grasping using deep reinforcement learning," in *2020 IEEE 16th International Conference on Automation Science and Engineering (CASE)* (Hong Kong), 1461–1466. doi: 10.1109/CASE48305.2020.9216986

Kurosu, J., Yorozu, A., and Takahashi, M. (2017). Simultaneous dual-arm motion planning for minimizing operation time. *Appl. Sci.* 7:1210. doi: 10.3390/app7121210

Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., et al. (2015). Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*. doi: 10.48550/arXiv.1509.02971

Lin, L.-J. (1992). Self-improving reactive agents based on reinforcement learning, planning and teaching. *Mach. Learn.* 8, 293–321. doi: 10.1007/BF00992699

Liu, Y., Chen, B., Zhang, X., and Li, R. (2021). Research on the dynamic path planning of manipulators based on a grid-local probability road map method. *IEEE Access* 9, 101186–101196. doi: 10.1109/ACCESS.2021.3098044

Lu, B., Chu, H. K., Huang, K., and Cheng, L. (2018). Vision-based surgical suture looping through trajectory planning for wound suturing. *IEEE Trans. Autom. Sci. Eng.* 16, 542–556. doi: 10.1109/TASE.2018.2840532

Lu, B., Li, B., Chen, W., Jin, Y., Zhao, Z., Dou, Q., et al. (2021). Toward image-guided automated suture grasping under complex environments: a learning-enabled and optimization-based holistic framework. *IEEE Trans. Autom. Sci. Eng.* 1–15. doi: 10.1109/TASE.2021.3136185

Melchiorre, M., Scimmi, L. S., Pastorelli, S. P., and Mauro, S. (2019). "Collison avoidance using point cloud data fusion from multiple depth sensors: a practical approach," in *2019 23rd International Conference on Mechatronics Technology (ICMT)* (Salerno), 1–6. doi: 10.1109/ICMECT.2019.8932143

Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T., Harley, T., et al. (2016). "Asynchronous methods for deep reinforcement learning," in *International Conference on Machine Learning* (New York, NY), 1928–1937.

Schulman, J., Levine, S., Abbeel, P., Jordan, M., and Moritz, P. (2015). "Trust region policy optimization," in *International Conference on Machine Learning* (Lille), 1889–1897.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. (2017). Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*. doi: 10.48550/arXiv.1707.06347

Shanahan, M., Crosby, M., Beyret, B., and Cheke, L. (2020). Artificial intelligence and the common sense of animals. *Trends Cogn. Sci.* 24, 862–872. doi: 10.1016/j.tics.2020.09.002

Sutton, R. S., and Barto, A. G. (2018). *Reinforcement Learning: An Introduction*. Cambridge, MA; London: MIT Press.

Vysocky, A., and Novak, P. (2016). Human-robot collaboration in industry. *MM Sci. J.* 9, 903–906. doi: 10.17973/MMSJ.2016_06_201611

Wei, K., and Ren, B. (2018). A method on dynamic path planning for robotic manipulator autonomous obstacle avoidance based on an improved RRT algorithm. *Sensors* 18:571. doi: 10.3390/s18020571

Wittmann, J., Jankowski, J., Wahrmann, D., and Rixen, D. J. (2020). "Hierarchical motion planning framework for manipulators in human-centered dynamic environments," in *2020 29th IEEE International Conference on Robot and Human Interactive Communication (RO-MAN)* (Naples), 525–532. doi: 10.1109/RO-MAN47096.2020.9223549

Xu, X., Hu, Y., Zhai, J., Li, L., and Guo, P. (2018). A novel non-collision trajectory planning algorithm based on velocity potential field for robotic manipulator. *Int. J. Adv. Robot. Syst.* 15:172988141 8787075. doi: 10.1177/1729881418787075

Zhou, M., Yu, Q., Huang, K., Mahov, S., Eslami, A., Maier, M., et al. (2019). Towards robotic-assisted subretinal injection: a hybrid parallel-serial robot system design and preliminary evaluation. *IEEE Trans. Indus. Electron.* 67, 6617–6628. doi: 10.1109/TIE.2019.2937041

Conflict of Interest: The authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

Publisher's Note: All claims expressed in this article are solely those of the authors and do not necessarily represent those of their affiliated organizations, or those of the publisher, the editors and the reviewers. Any product that may be evaluated in this article, or claim that may be made by its manufacturer, is not guaranteed or endorsed by the publisher.