Complex & Intelligent Systems (2022) 8:1899–1912
https://doi.org/10.1007/s40747-021-00366-1

ORIGINAL ARTICLE

Collision-free path planning for welding manipulator via hybrid algorithm of deep reinforcement learning and inverse kinematics

Jie Zhong[1] · Tao Wang[1] · Lianglun Cheng[1]

Received: 28 November 2020 / Accepted: 1 April 2021 / Published online: 10 April 2021

Abstract
In actual welding scenarios, an effective path planner is needed to find a collision-free path in the configuration space for the welding manipulator with obstacles around. However, as a state-of-the-art method, the sampling-based planner only satisfies the probability completeness and its computational complexity is sensitive with state dimension. In this paper, we propose a path planner for welding manipulators based on deep reinforcement learning for solving path planning problems in high-dimensional continuous state and action spaces. Compared with the sampling-based method, it is more robust and is less sensitive with state dimension. In detail, to improve the learning efficiency, we introduce the inverse kinematics module to provide prior knowledge while a gain module is also designed to avoid the local optimal policy, we integrate them into the training algorithm. To evaluate our proposed planning algorithm in multiple dimensions, we conducted multiple sets of path planning experiments for welding manipulators. The results show that our method not only improves the convergence performance but also is superior in terms of optimality and robustness of planning compared with most other planning algorithms.

Keywords Path planning · Obstacle avoidance · Welding manipulator · Deep reinforcement learning · Inverse kinematics

Introduction

Welding tasks exist in various industrial manufacturing processes. Especially in the shipbuilding process, welding workload accounts for 40% of the total workload. Shipbuilding, known as a labor-intensive industry, requires a considerable number of skilled technicians to weld in enclosed and hazardous surroundings. Thanks to the development of industrial robotics, automated welding is gradually replacing manual welding by means of equipping welding manipulators on the shipbuilding production line [1,2]. Nonetheless, path planning for a specific welding task such as spot welding or arc welding, which is often completed by skilled welding technicians with an offline programming teach pendant, is still challenging automated welding. When dealing with a complex welding task operated on different workpieces with various shapes, it is rather tedious teaching on-line or programming off-line due to the existent of multifarious pose switching of the welding torch which demands many path planning queries. In view of the demand for flexible and high-efficiency welding, a reliable, safe, and automated path planning technology serving for pose switching of the welding torch is urgently needed.

In the field of welding, the term path planning may be formulated as a constrained Traveling Salesman Problem (TSP) [3] while in this paper we study a new method to find an elegant collision-free path in configuration space of the welding manipulator which connects the initial configuration and the target configuration [4].

Most existing path planning methods for robots focus on their performance in one domain they pursue. The sampling-based method is one of the most popular path planning methods owing to its probabilistic completeness. The sampling-based planning algorithms mainly include the multi-query algorithm Probabilistic Roadmap Method(PRM) [5,6] and the single-query method Rapidly exploring Random Trees (RRT) [7,8]. The algorithms mentioned above can almost

✉ Tao Wang
wangtao_cps@gdut.edu.cn

Jie Zhong
zhongjie@mail2.gdut.edu.cn

Lianglun Cheng
llcheng@gdut.edu.cn

1 School of Automation, Guangdong University of Technology, Guangzhou, China

always find a feasible path (i.e., a sequence of valid states) when sampling in low-dimension state space like a 2-D or 3-D map for wheeled mobile robots [9,10] and Unmanned Aerial Vehicle [11]. However, when searching for a feasible path in high-dimension state space like the configuration space of a six-DOF welding manipulator, the computational complexity of the sampling-based method shows an exponential growth trend due to the dimensional explosion of sampling space. To better apply the sampling-based algorithms to the high-dimension path planning(e.g. manipulator planning), many improved methods have been proposed. In [12], a new variation path planning method based on PRM and lazy-PRM for industrial manipulators was proposed to reduce the time spent on the roadmap construction. Cao et. al. adopted the idea of target gravity to accelerate the path search speed in [13] and in [14] an improved sampling-based method using learned critical sources and local sampling is developed to go through narrow passages more quickly. Despite the fact that those methods increase the planning efficiency and inherit the superiority of traditional RRT algorithm, the sampling-based planning method is actually a probabilistic search-based offline method with uncertainty and the planning time may tend to infinity in a complex environment especially a narrow passage.

One of the alternative methods for high-dimension manipulator path planning is based on Artificial Potential Field (APF) which is an online method proposed in [15]. Compared with the sampling-based search algorithm mentioned before, the APF algorithm plans the same path each time for the same task which is meaningful in industrial scenario [16] while both of the two algorithms' performance suffers from finding a path in high-dimension state space. Furthermore, it should be noted that the APF algorithm, as a local planner, may converge to a local optimal solution and lacks completeness. In view of the aforementioned characteristics, scholars have proposed many improved APF algorithms for path planning in high-dimension state space such as [17,18]. Another path planning method treats the path planning problem as a constrained optimal control problem and solves it by a nonlinear model predictive controller (MPC) [19], but this method is model dependent and requires computing power and memory resources. What is even more unacceptable is that it is a local planning algorithm as APF does.

With the development of machine learning and deep learning, the learning-based method becomes an elegant alternative method for robot path planning [20]. For example, the Motion Planning Networks(MPNet) based on deep learning [21] aims to train a model that is able to generate an end-to-end collision-free path in the obstacle space for the given start and goal configurations of a robot, yet the representation of the whole workspace with obstacles is needed and its completeness depends on the replanning method(e.g. RRT). Nevertheless, the existing learning-based planning methods often rely on traditional search algorithms (such as RRT) or optimal control algorithms, so that they inevitably inherit the disadvantages of basic algorithms. In addition, a large amount of sample data is needed, and sampling efficiency needs to be improved urgently. There is still a need to balance exploration and utilization to obtain better performance.

In recent years, owing to the amazing performance of various robot controllers or planners based on Deep Reinforcement Learning (Deep-RL) as described in [22–26], many researchers turn to Deep-RL which shows excellent prospects for solving path planning problems in high-dimensional continuous state and action spaces. At the same time, deep reinforcement learning itself has also received more attention from researchers [27–29]. Xue et al. [30] proposed an obstacle avoidance path planning method based on a Deep-RL algorithm applied in discretized actions, they introduced sub-goals to guide the mobile robot to the final goal. In [31] a continuous action space Deep-RL algorithm was used for navigation of a mobile robot in simulated environments and the authors believe that the continuous Deep-RL algorithm is an effective collision-free decision-maker for a mobile vehicle.

Compared with the mobile robots, a welding manipulator with high-dimension continuous action space seems to be not that easy to handle by Deep-RL algorithms directly due to the tremendous exploration space and the sampling inefficiency. The exploration–exploitation trade-off is at the heart of reinforcement learning which restricts the convergence of the algorithm thus the balance of exploration and exploitation must be considered when doing path planning using Deep-RL. In [32] a dynamic obstacle avoidance approach for robot manipulator base on a continuous Deep-RL algorithm [33] was proposed and a well-designed reward function was presented to avoid the sparse reward and increase useful exploitation. In Sangiovanni et al. [34], presented a hybrid control method that combines the Deep-RL policy and the PRM policy (i.e. a sampling-based method) to achieve collision-free path planning for anthropomorphic robot manipulators. Hua et al. [35] proposed a path planner for a redundant robot in a narrow duct based on the Deep Deterministic Policy Gradient (DDPG) algorithm [36], this method reduces the exploration space by decoupling the self-motion from the motion for obstacle avoidance; however, the manipulator must be redundant. It can be seen from the related work above that most scholars try to improve the convergence performance of reinforcement learning by adopting models which is inherent in the environment. This is indeed an area worth continuing to study. Aside from reinforcement learning itself, it is possible to mine information from the environment so that the agent could have a certain innate wisdom.

In this paper, we propose an informed learning-based path planner for a 6-DOF welding manipulator in the semi-closed and narrow environment which is common in real welding scenarios, see in Fig. 1. The model-free Deep-RL algorithm DDPG is adopted to train an agent that is able to navigate the welding manipulator in configuration space so that the welding torch can reach the target position in the workspace finally without any collision on the path. For the purpose of speeding up the convergence, we provide the agent the informed knowledge by introducing the inverse kinematics of the welding manipulator. To avoid the agent's excessive exploitation due to the prior knowledge and encourage exploration for obstacle avoidance, we introduce a hybrid method of Deep-RL and inverse kinematics of robot manipulator which will be discussed later in detail. The main contributes of this paper are as follows:

– A well-designed interactive environment with high fidelity in simulation scenario is developed. Our modeling method provides a new physical modeling solution based on CoppeliaSim as a substitute for gym or mujoco for Deep-RL algorithms.
– We utilize the inverse kinematics based action as the prior knowledge to reduce the unnecessary exploration in the state-action space thus the learning speed is accelerated.
– To avoid excessive exploitation caused by the presence of prior knowledge based on inverse kinematics, the effect of the inverse kinematics based action is adjusted dynamically by introducing the gain module.
– We design a method for accelerating deep reinforcement learning training process by combining a inverse kinematics module, which provides researchers with a new idea that is different from improving the deep reinforcement learning algorithm itself.

The rest of this paper is structured as follows. The theoretical background is introduced in the next section followed by our modified method in the subsequent section. Then we demonstrate the relevant elegant performance of the proposed method with several experiment results. At last, we provides the conclusion.

Background

In this section, we give an overview of the background theories for the proposed Deep-RL-based collision-free path planner, including the kinematics modeling of a welding manipulator used in this research, the Sequential Decision-Making model, and the model-free Deep-RL algorithm: DDPG.

Kinematics modeling

The kinematics of a manipulator mainly studies the displacement relationship, velocity relationship, and acceleration relationship between different links or joints. In this research, we should model the kinematics of the used six-DOF welding manipulator as shown in Fig. 1. Denavit–Hartenberg (D–H) [37] representation has been widely used in kinematic modeling and analysis of series manipulator and has become the standard method for modeling kinematics. Therefore, the detailed standard D–H parameters are shown in Table 1. Using the D–H parameters, the Homogeneous Transformation Matrix (HTM) of the last link relative to the base link can be obtained as

$$ {}_6^0T(\boldsymbol{q}) = \prod_{i=1}^{6} {}_i^{i-1}T(q_i), \tag{1}$$

where the $\boldsymbol{q} = [q_1, q_2, \ldots, q_6] \in \mathbb{R}^6$ is the joint angle variable drawn from the Configuration Space (C-space), ${}_i^{i-1}T(q_i)$ represents the HTM of ith link relative to the one of the previous links denoted as follows:

$$ {}_i^{i-1}T(q_i) = \begin{bmatrix} \cos\theta_i & -\sin\theta_i\cos\alpha_i & \sin\theta_i\sin\alpha_i & a_i\cos\theta_i \\ \sin\theta_i & \cos\theta_i\cos\alpha_i & -\cos\theta_i\sin\alpha_i & a_i\sin\theta_i \\ 0 & \sin\alpha_i & \cos\alpha_i & d_i \\ 0 & 0 & 0 & 1. \end{bmatrix} \tag{2}$$

The last link is usually called an end flange, and an end-effector with a reference frame is fixed on it, e.g., a welding torch. In practice, the task is executed by the end-effector in Cartesian space, and the end-effector's HTM refer to the base link can be computed by ${}_e^0T(\boldsymbol{q}) = {}_6^0T(\boldsymbol{q})\,{}_e^6T$, for simplicity, both sides of the equation can be written as

$$ \boldsymbol{x}_e = \mathrm{fkine}(\boldsymbol{q}), \tag{3}$$

where $\boldsymbol{x}_e = [\boldsymbol{p}_e\ \boldsymbol{\phi}_e]^T \in \mathbb{R}^6$ is a pose, with $\boldsymbol{p}_e$ being the three-dimensional position and $\boldsymbol{\phi}_e$ the orientation (e.g. Euler angles) in the Cartesian space.

Most of the motion specified by the task is defined in Cartesian space, thus it is inevitable to map the end-effector's Cartesian motion to the motion in joints in C-space. In robotics, given the end-effector's Cartesian velocity $\dot{\boldsymbol{x}}_e$ relative to itself, the corresponding velocity of joints' angles $\dot{\boldsymbol{q}}$ is as follows:

$$ \dot{\boldsymbol{q}} = J(\boldsymbol{q})^{\dagger}\dot{\boldsymbol{x}}_e, \tag{4}$$

where $J(\boldsymbol{q})$ is a 6×6 Jacobian matrix for a 6-DOF manipulator, and it can be a known matrix once the joint position $\boldsymbol{q}$

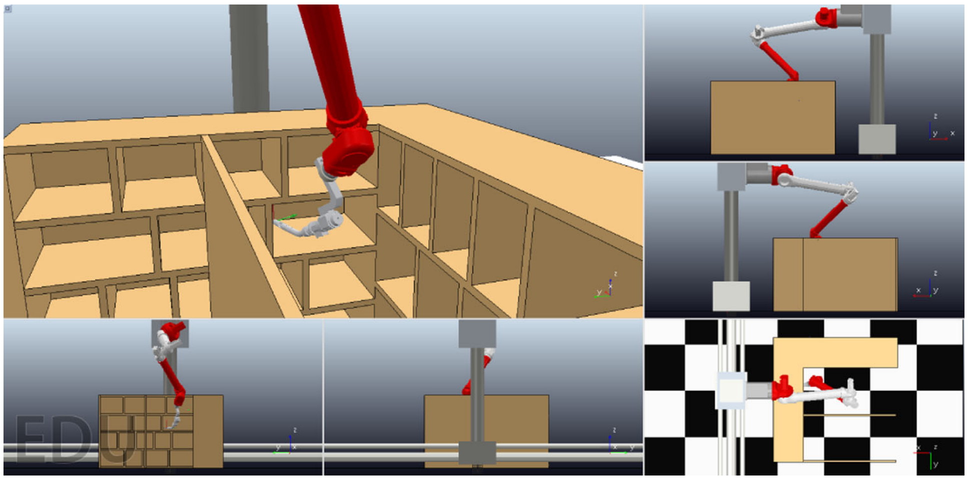


Fig. 1 The considered 6-DOF welding manipulator in the semi-closed and narrow environment simulated in a robot simulation platform CoppeliaSim

Table 1 Standard D–H parameters of the used welding manipulator

Link i	α_i (°)	a_i (m)	d_i (m)	θ_i (°)
1	0	0	0.4205	q_1
2	− 90	0	0	q_2
3	0	0.7260	− 0.1265	q_3-90
4	90	0	− 0.0960	q_4-180
5	90	0	0.6305	q_5
6	− 90	0	0.0910	q_6

is given [38]. It is worth noting that the pseudo-inverse of the matrix $J(\boldsymbol{q})^{\dagger}$ is computed due to the existence of singularity.

The inverse kinematics (IK) problem of manipulator is to find a correct and proper joint configuration $\boldsymbol{q}$ drawn from the C-space which satisfies the equation $\boldsymbol{x}_e^* = \text{fkine}(\boldsymbol{q})$, the term $\boldsymbol{x}_e^*$ represents the target pose of end-effector.

Algorithm 1 Pseudo-inverse-based method for IK problem

```
Input: $x_e^*$: target pose; $q_{init}$: initial guess; $dt$: sampling period;
Output: solution $q^*$
1: initial $q \leftarrow q_{init}$;
2: repeat
3:     compute current pose $x_e \leftarrow \text{fkine}(q)$;
4:     compute pose differential $\Delta x_e \leftarrow x_e \ominus x_e^*$;
5:     compute the joint velocity $\dot{q}_{inv} \leftarrow J(q)^{\dagger}\Delta x_e$;
6:     update the joint position $q \leftarrow q + \dot{q}_{inv}dt$;
7: until $(\|\Delta x_e\| < \epsilon)$
8: return $q^* \leftarrow q$
```

In general, the traditional solutions to the inverse kinematics problem include closed-form solutions and numerical solutions. The closed-form solutions consist of algebraic and geometric methods, they solve the problem faster than numerical solutions yet they depend on the model of manipulator and lack generalization ability. On the other hand, the numerical solutions do not rely on the model of manipulator which mainly refer to iterative methods.

Among all of the iterative methods, the algorithm based on the Pseudo Inverse of the Jacobian matrix [39] and the one based on Damped Least-Squares(DLS) [40] are most popular, both of them can converge to an optimal or approximately optimal solution $\boldsymbol{q}^*$ based on an initial guess $\boldsymbol{q}_{\text{init}}$. Unfortunately, sometimes these methods may not converge to a definite solution due to the presence of singularity or the initial guess is not proper enough. From the perspective of optimization theory, the Pseudo Inverse method is faster than the DLS method and can be implemented readily in practice.

In this research, the pseudo-inverse-based method (see in Algorithm 1) is considered to find a iteration direction in the C-space which minimizes the error between $\boldsymbol{x}_e^*$ and current pose $\boldsymbol{x}_e$. In Line 4, we use the difference of pose $\Delta\boldsymbol{x}_e$ instead of the velocity $\dot{\boldsymbol{x}}_e$ in Eq. 4, and then $\dot{\boldsymbol{q}}_{\text{inv}}$ is obtained in Line 5. The $\Delta\boldsymbol{x}_e$ is computed as follows:

$$\boldsymbol{x}_e \ominus \boldsymbol{x}_e^* \doteq \begin{bmatrix} \boldsymbol{p}_e^* - \boldsymbol{p}_e \\ \text{vex}\left(\boldsymbol{R}_e^*\boldsymbol{R}_e^T - \boldsymbol{I}_{3\times3}\right) \end{bmatrix}, \tag{5}$$

where the $\boldsymbol{R}_e \in \mathbb{R}^{3\times3}$ is the rotation matrix form of the orientation vector $\boldsymbol{\phi}_e$. The function vex$(\cdot)$ can be seen in [41] which aims to compute the increment of $\boldsymbol{R}_e^*$ relative to $\boldsymbol{R}_e$.

The Markov decision process

In a wide sense of the word, Reinforcement Learning (RL) solves the sequential decision problems by finding the decision sequence which maximizes the expected cumulative return. A Markov Decision Process (MDP) is a typical model

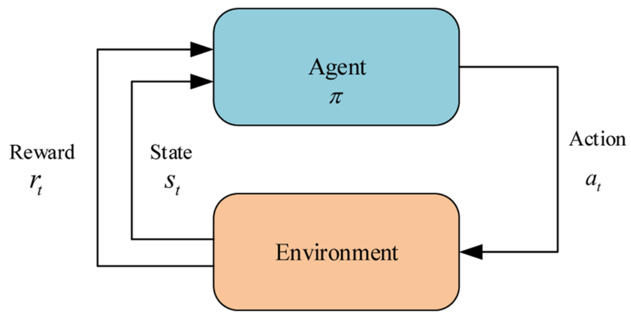


Fig. 2 The interaction between the agent and the environment [42]

of sequential decision problem which is described by a tuple $(\mathcal{S}, \mathcal{A}, \pi, \mathcal{P}, r, \gamma, H)$. In this tuple, $\mathcal{S}$ is a complete set of all existing states, $\mathcal{A}$ is a set of all actions, both of them can be discrete or continuous. The policy π can be divided into a stochastic one $a \sim \pi(\cdot|s)$ and a deterministic one $a = \mu(s)$, where $a \in \mathcal{A}, s \in \mathcal{S}$. The state transition probability function $\mathcal{P} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ denotes the dynamics of the environment: the probability of reaching s_{t+1} after taking action a_t at the environment state s_t. In some cases, the state transition function can be deterministic.

The reward function $r = R(s_t, a_t)$, which is a scalar obtained from the environment when taking action a_t at observing a state s_t, implies the goal of a RL problem. Until now, we can use the MDP and π to generate sequence as follows:

$$s_0, a_0, r_0, s_1, a_1, r_1, s_2, \ldots, s_{H-1}, a_{H-1}, r_{H-1}, s_H,$$

where $H \in \mathbb{N}^+$ is a time or step horizon of an episode. It is also called a trajectory which is generated by the interaction between the agent and the environment, see in Fig. 2. The agent mainly plays the role of a policy π while the environment define the state transition model and output the reward signal.

The discounted return at time step t is defined as follows:

$$G_t \doteq \sum_{k=0}^{H-t-1} \gamma^k R\left(s_{t+k}, a_{t+k}\right), 0 \leq t < H, \tag{6}$$

where the term $\gamma \in [0, 1]$ denotes the discount factor. To evaluate the performance of the policy π, we can compute the expectation of the total reward in one episode:

$$J_\pi = \boldsymbol{E}_\pi \left[\sum_{t=0}^{H-1} \gamma^t R\left(s_t, \pi(\cdot|s_t)\right) \right], \tag{7}$$

where the policy and the environment models are stochastic.

In general, the RL algorithms aim to find the optimal or asymptotically optimal policy $\pi_\theta^*(\cdot|s)$ with parameters θ. In recent years, scholars use deep neural networks (DNN) to represent the parameters of policy and therefore the iteration of policy is carried out by training the policy network weights. This method is called Deep Reinforcement Learning (Deep-RL) which interests more and more researchers all over the world from different domains [43] such as robotics.

Algorithm 2 Deep Deterministic policy gradient

```
 1: Initialize: The predicted actor network $\mu(s)$ and critic network $Q(s,a)$ with weights $\theta^\mu$ and $\theta^Q$
 2: Initialize: The target actor network $\bar{\mu}(s)$ and critic network $\bar{Q}(s,a)$ with weights $\bar{\theta}^\mu \leftarrow \theta^\mu$ and $\bar{\theta}^Q \leftarrow \theta^Q$
 3: Initialize: The replay buffer $\mathcal{R}$
 4: for episode=1,N do
 5:   Initialize a random process $\mathcal{N}$ for exploration
 6:   Reset the environment and observe the $s_0$
 7:   for t=0,T-1 do
 8:     Select an action $a_t = \mu(s_t) + \mathcal{N}$
 9:     Execute $a_t$ at the environment
10:     Get the reward $r_t$ and next state $s_{t+1}$
11:     Store the transition $(s_t, a_t, r_t, s_{t+1})$ in $\mathcal{R}$
12:     Sample a minibatch of M transitions in $\mathcal{R}$:
            $(s_i, a_i, r_i, s_{i+1})$ where $i = 1, \ldots, M$
13:     Set $y_i = r_i + \gamma \bar{Q}(s_{i+1}, \bar{\mu}(s_{i+1}))$
14:     Update network $Q(s,a)$ by minimize the loss:
            $L = \frac{1}{M}\sum_i (y_i - Q(s_i, a_i))^2$
15:     Update network $\mu(s)$ with the sampled policy gradient:
            $\nabla_{\theta^\mu} J = \frac{1}{M}\sum_i \nabla_{\theta^\mu} Q(s,a)|_{s=s_i, a=\mu(s_i)}$
16:     Soft update target networks
            $\bar{\theta}^\mu \leftarrow \tau\theta^\mu + (1-\tau)\bar{\theta}^\mu$
            $\bar{\theta}^Q \leftarrow \tau\theta^Q + (1-\tau)\bar{\theta}^Q$
17:   end for
18: end for
```

Deep deterministic policy gradient

In this subsection, we focus on one of the state-of-the-art Deep-RL algorithm: deep deterministic policy gradient algorithm (DDPG) which was proposed in [36] and it is shown in Algorithm 2. This algorithm is suitable for training an agent or policy to control a robot such as a 6-DOF welding manipulator to accomplish a specify task with position or torque, because its policy can output an action that is high dimensional and continuous.

The motivation of DDPG is actually to allow Deep Q-learning Network [44] to expand to continuous action domain with a deterministic actor. Like the algorithm DQN, a pioneering work that combines reinforcement learning and the deep learning, DDPG trains an action-value function network

$Q^{\pi}(s, a)$ to approximate the Bellman function:

$$Q^{\pi}(s_t, a_t) = \mathbb{E}_{\pi}\left[G_t | s_t, a_t\right], \tag{8}$$

where the deterministic policy π is $a = \mu(s)$ which maps the current state to a specific action. The action-value function network is trained by minimize the loss function:

$$L(\theta^{Q}) = \mathbb{E}\left[\left(Q\left(s_t, a_t|\theta^{Q}\right) - y_t\right)^2\right], \tag{9}$$

where the label y_t is given by

$$y_t = R(s_t, a_t) + \gamma Q\left(s_{t+1}, \mu\left(s_{t+1}\left|\bar{\theta}^{\mu}\right)\right|\bar{\theta}^{Q}\right) \tag{10}$$

in which the action-value network $Q\left(s, a|\bar{\theta}^{Q}\right)$ and the policy network $\mu\left(s|\bar{\theta}^{\mu}\right)$ represents the target network, respectively. The purpose of introducing target network is to make the training process more stable.

On the other hand, the DDPG algorithm utilize the actor-critic framework-based DPG algorithm [45] to maintain the policy $\mu(s)$, and in this framework, the action-value network is called critic network while the policy network is called actor network. The deterministic policy gradient is proved to be as follows:

$$\nabla_{\theta^{\mu}} J = \mathbb{E}\left[\nabla_a Q\left(s, a|\theta^{Q}\right)|_{s=s_t, a=\mu(s_t)} \nabla_{\theta^{\mu}} \mu\left(s|\theta^{\mu}\right)|_{s=s_t}\right] \tag{11}$$

which represents the performance of the actor network, we can update the actor network using this equation.

An experience replay buffer which is designed for off-policy updating denoted as $\mathcal{R}$ is used to store the transition tuple (s_t, a_t, r_t, s_{t+1}) see in Algorithm 2 Line 11. The predicted network θ^{μ} and θ^{Q} are updated using Eqs. 11 and 9, respectively, with the sampled minibatch experience replay transitions. To train the target network more stably, a soft update technic as Algorithm 2 Line 16 shows is employed.

Due to the determinacy, the actor may coverage to a local optimum, and this is harmful to the agent to explore for higher reward in the environment. Thus, in Algorithm 2 Line 8, a stochastic process noise, Ornstein–Uhlenbeck, is added to the actor's action to enhance exploration and protect the robot in the environment at the same time.

Methodology

In this section, we introduce in detail the hybrid collision-free path planning method of Deep-RL and inverse kinematics for the 6-DOF welding manipulator.

The interaction environment

To solve the path planning problem with the Deep-RL framework, we must convert the primal problem to a sequential decision problem, namely the Markov decision process (MDP). To this end, we need to define the interface between the agent and the environment in Fig. 2.

State space

In the real world, the state set may be extremely massive or even infinite, so enumerating all the states is unrealistic. In practice, we usually select some important characteristics as the state information. It should be noted that the state information of Deep-RL represents the environmental information perceived by the agent and implies the influence brought about by the actions of that agent. State information is the basis for the agent to make decisions and evaluate its long-term benefits, namely accumulative reward, and the quality of the state design directly determines the convergence, convergence speed, and final performance of the Deep-RL algorithm.

Task analysis is the soul of state design. In this research, the goal of the collision-free path planning task is to train an agent that drives the joints of the welding manipulator without any collision to minimize the distance between the end-effector of the welding manipulator and its target position. In the light of the given task, the state observed from the environment at time step t is designed as a stacked vector:

$$\boldsymbol{s_t} = \left[\boldsymbol{q}, \boldsymbol{p}_e, d_{\text{tar}}, \text{collision}\right] \in \mathcal{S}, \tag{12}$$

where $\boldsymbol{q} \in \mathbb{R}^6$ is the joint position of the welding manipulator, and $\boldsymbol{p}_e \in \mathbb{R}^3$ is the end-effector reference frame's Cartesian position relative to the target frame. The scalar d_{tar} denotes the Euclidean distance between the end-effector reference frame's origin and the origin of the target frame. The last element $collision$ is a Boolean that indicates whether the collision occurs between the links of the welding manipulator and obstacles in the environment, it can be obtained by the CoppeliaSim API function. It is obvious that the dimension of the state vector is

$$\mathbf{dim}(\boldsymbol{s_t}) = \mathbf{dim}(\boldsymbol{q}) + \mathbf{dim}(\boldsymbol{p}_e) + 2, \tag{13}$$

where $\mathbf{dim}(\cdot)$ means the dimension of a vector.

Action space

When applying Deep-RL algorithm to an actual problem, perhaps the most intuitive part is the definition of the action space $\mathcal{A}$. In this research, the actor in the environment is a 6-DOF welding manipulator; thus, the control method of

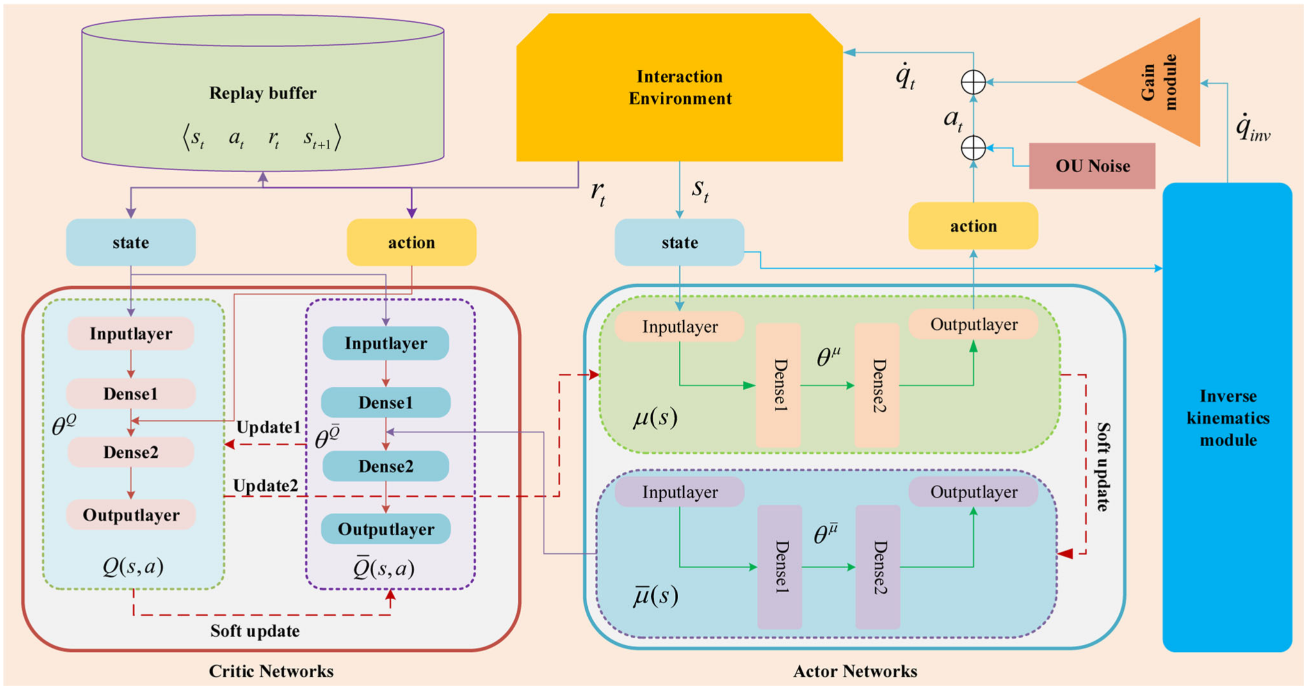


Fig. 3 This is the schematic diagram of our method. In this diagram, apart from the DDPG learning framework, we add an Inverse kinematics module in series with a gain module. The *Update1* represents the update of the network $Q(s, a)$ and the *Update2* represents the update of the network $\mu(s)$

the agent for the manipulator can be velocity control mode intuitively as follows:

$$\boldsymbol{a_t} = \dot{\boldsymbol{q}} \in \mathcal{A}, \tag{14}$$

where $\dot{\boldsymbol{q}} \in \mathbb{R}^6$ is the target angular velocity within the limitation of each rotation joint driven by a servo motor. The dimension of the action space is

$$\mathbf{dim}(\boldsymbol{a_t}) = 6 \tag{15}$$

Reward signal

Contrary to the definition of action space mentioned aforehand, the reward signal, which implies the goal of the specified problem, determines whether the agent can finally learn the desired skills, and directly affects the convergence speed and final performance of the algorithm as the state space does. After considering our problem, we define the reward signal consists three parts.

The first part is designed for the end-effector approaching a specific target position. We utilize a nonlinear piecewise function to compute this part of the reward:

$$r_1 = \begin{cases} -\frac{1}{2} d_{\text{tar}}^2 & d_{\text{tar}} \leq \delta \\ -\delta \left(|d_{\text{tar}}| - \frac{1}{2}\delta \right) & d_{\text{tar}} > \delta, \end{cases} \tag{16}$$

where d_{tar} is the Euclidean distance between the end-effector and its target position, the parameter δ is the turning point of the function. The popular explanation for r_1 is the longer the distance d_{tar}, the smaller the reward.

The second part is designed for the purpose of avoiding collision, so in this part, we give a large penalty according to the current collision condition.

$$r_2 = \begin{cases} -C_{\text{o}} & \text{if collision} \\ 0 & \text{otherwise} \end{cases}, \tag{17}$$

where $C_{\text{o}} \in \mathbb{R}^+$ is a constant by which we can regulate the penalty. The collision state collision is a Boolean as mentioned in Eq. 12.

The third part is relative to the norm of the current action $\dot{\boldsymbol{q}}$ as shown in Eq. 18, that means we expect the agent to take smaller actions for safety.

$$r_3 = -\|\dot{\boldsymbol{q}}\|^2. \tag{18}$$

Finally, we can construct the total reward signal as follows:

$$r_t(\boldsymbol{s_t}, \boldsymbol{a_t}) = \lambda_1 r_1 + \lambda_2 r_2 + \lambda_3 r_3 \in \mathbb{R}, \tag{19}$$

where $\lambda_i \in \mathbb{R}^+$ denotes the constant scaling factor which is designed to make the contribution of each reward component to the final reward at a close level, so that each target reward can be taken into account by the agent to the same extent.

The training process

Although the Deep-RL algorithm DDPG as shown in Algorithm 2 has achieved elegant results on many robotic operation tasks in Mujoco environment such as *reacherObstacle* in which the agent is required to move a 5-DOF manipulator to the target position without collision, the complexity of learning mapping relationship between state space $\mathcal{S}$ and action space $\mathcal{A}$ in the path planning problem is so high that it requires amazing training time steps and even millions of steps are commonplace.

To be honest, training a policy model for a path planning problem from scratch using the DDPG algorithm is irrational. Thus, in this paper, we aim to speed up the DDPG training process for the collision-free path planning problem via a hybrid method of DDPG and inverse kinematics of the welding manipulator as shown in Fig. 3.

The path planning problem for a welding manipulator actually contains two sub-problems, one is moving the end-effector to its target position while the other is avoiding collision during the moving process. If the collision avoidance is not in consideration, moving the end-effector to its target position can be solved by a sequential dynamic model shown in Eq. 20.

$$\boldsymbol{q}_{t+1} = \boldsymbol{q}_t + \dot{\boldsymbol{q}}_{\text{inv}} \cdot \mathrm{d}t, \tag{20}$$

where dt is the sampling period, $\dot{\boldsymbol{q}}_{\text{inv}}$ is obtained as Line 5, Algorithm 1 shows.

Due to the presence of obstacles, we need to correct the $\dot{\boldsymbol{q}}_{\text{inv}}$ generated by the inverse kinematics module to avoid collision. To this end, we might as well use the action $\boldsymbol{a_t}$ selected by the policy network of DDPG to share the task of avoiding obstacles as Eq. 21 shows.

$$\boldsymbol{q}_{t+1} = \boldsymbol{q}_t + (\dot{\boldsymbol{q}}_{inv} + \boldsymbol{a_t}) \cdot \mathrm{dt}. \tag{21}$$

In this way, there is no need for the DDPG agent to learn from scratch since the inverse kinematics module will constraint the search space between the start and goal region as well as guide the welding manipulator move to the target position so that the agent's responsibility is just to correct the output of inverse kinematics module for collision avoidance. In other words, the invalid exploration which discourage the welding torch moving to the target position is lessened, thus the search space is reduced significantly which is beneficial to speed up the training process of the agent.

However, we find that when training with dynamic model Eq. 21, the excessive exploitation of the inverse kinematics module leads the agent to overlook the objective of collision avoidance and the policy converge to local a optimum. We blame this failure on the changeless usage of $\dot{\boldsymbol{q}}_{\text{inv}}$ generated by the inverse kinematics module. It is the fixed $\dot{\boldsymbol{q}}_{\text{inv}}$ that restricts the exploration of the agent for collision avoidance. Thus we add a time-varying gain module to encourage exploration in the early time of one episode with a relatively large growth rate and increase the exploitation in the later time by applying a lower growth rate when the manipulator ought to be near the goal position. The gain module is an increasing nonlinear function $\ln(x)$ of time step t as shown in Eq. 22.

Algorithm 3 Hybrid of DDPG and inverse kinematics

```
1:  Initialize: actor networks and critic networks
2:  Initialize: replay buffer $\mathcal{R}$
3:  for episode=1,N do
4:      Initialize a random process $\mathcal{N}$ for exploration
5:      Reset the environment and observe the $\boldsymbol{s_0}$
6:      for t=0,T-1 do
7:          Select an action $\boldsymbol{a_t} = \mu(\boldsymbol{s_t}) + \mathcal{N}$
8:          Get $\mathcal{G}(t)\dot{\boldsymbol{q}}_{inv}$ as correction of $\boldsymbol{a_t}$
9:          update the environment using Equation 23
10:         Get the reward $r_t$ and next state $\boldsymbol{s_{t+1}}$
11:         Store the transition $(\boldsymbol{s_t}, \boldsymbol{a_t}, r_t, \boldsymbol{s_{t+1}})$ in $\mathcal{R}$
12:         Sample a minibatch of M transitions in $\mathcal{R}$
13:         Update actor network and critic network
14:         Soft update target networks
15:     end for
16: end for
```

$$\mathcal{G}(t) = \ln(t+1), \tag{22}$$

where $t = 0, \ldots, T-1$ and the hybrid dynamic model which balances the exploitation and exploration is as follows:

$$\boldsymbol{q}_{t+1} = \boldsymbol{q}_t + \left(\mathcal{G}(t) \cdot \dot{\boldsymbol{q}}_{\text{inv}} + \boldsymbol{a_t}\right) \cdot \mathrm{d}t. \tag{23}$$

The integral training process of our collision-free path planning method is described in Algorithm 3.

Experiments and analysis

In this section, we evaluate the performance of our proposed hybrid method through several path planning experimental results in the simulation system. We also prove the progress of our method through comparative analysis.

Experimental system description

The simulation experiment system is roughly divided into three parts. The first part is the interaction environment which mainly includes a welding manipulator with obstacle around it as shown in Fig. 1 and we model them in a well-known open-source robot simulator CoppeliaSim with high fidelity.

Table 2 Choice of key parameters for the DDPG agent

Parameter	Value
Mini-batch size M	128
Number of episodes N	5000
Max episode step T	120
Discount factor γ	0.99
Soft update factor τ	0.001
Replay buffer size	2e5
Learning rate of actor network	1e−3
Learning rate of critic network	1e−4
Size of hidden layer	128
Action dimension	6
State dimension	11

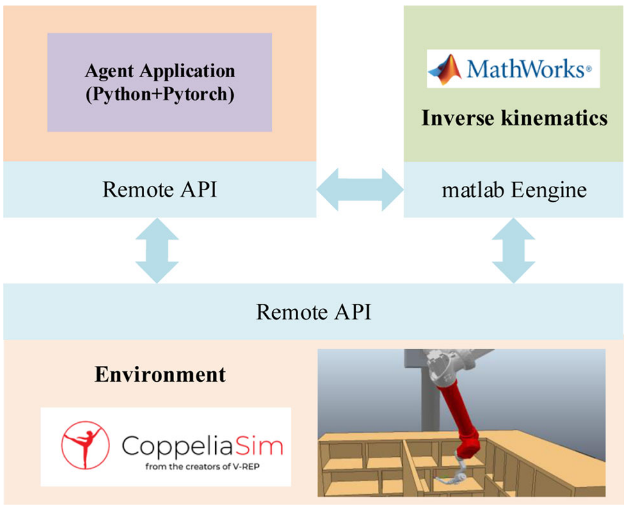


Fig. 4 The relation chart of the simulation system

The simulator CoppeliaSim provides remote interface functions for collision detection between specific robot entity and obstacles, thus we can determine the Boolean *collision* in Eq. 17 which indicates whether any collision occurs when the welding manipulator is at a certain joint angle $\boldsymbol{q}$. In general, this part, as a server application, provides an interface for the exchange of state information and action input as shown in Fig. 2. When receiving the action information, namely the joints' angular velocity $\dot{\boldsymbol{q}} \in \mathbb{R}^6$ predicted by the agent, the environment server application will update the current state $\boldsymbol{s}_t$ of the welding manipulator as described in Eq. 12 to its transition state $\boldsymbol{s}_{t+1}$, and the joints' angle is updated with Eq. 23, where dt is set to 0.05 in second. To make the simulation closer to the real situation, we must constraint the action $\boldsymbol{a}_t$ or $\dot{\boldsymbol{q}}$ in a bounded range: $\dot{\boldsymbol{q}}_{\min} \le \dot{\boldsymbol{q}} \le \dot{\boldsymbol{q}}_{\max}$. In the meanwhile the joint angle $\boldsymbol{q}$ should meet the joint limit requirements which are very crucial to the safety of the welding manipulator, otherwise, the links of the welding manipulator may collide with each other.

After the update of the state, the acquisition of the instant reward signal instead of a sparse one is pivotal. The constant parameters which formulate the reward signal are instantiated as follows: the δ in Eq. 16 is set to 0.05, the C_o in Eq. 17 is set to 200, the λ_i in Eq. 19 set to $\lambda_1 = 2000$, $\lambda_2 = 1$, $\lambda_3 = 1$, respectively. It needs to be pointed out that these parameters are carefully picked up through a large number of experiments. Different parameters will directly affect the convergence of the algorithm.

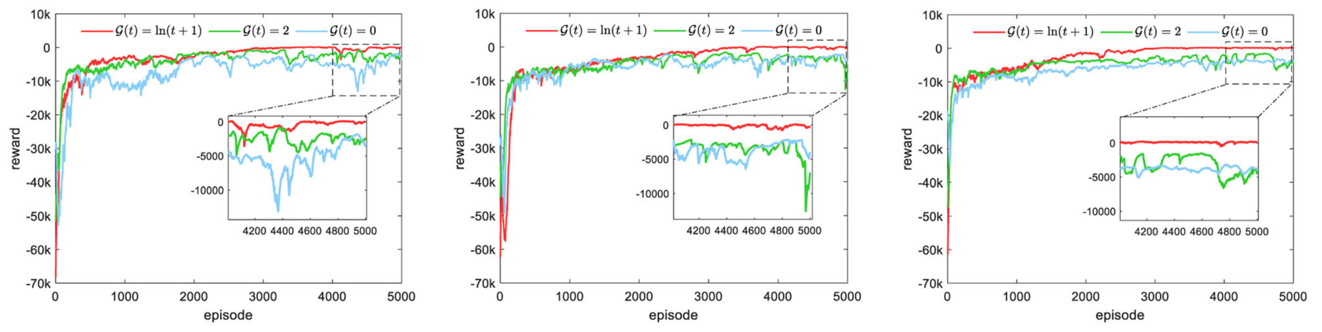
The second part is the DDPG agent represented by four fully connected deep neural networks constructed by a famous deep learning framework Pytorch and their respective update method for internal neural network weights. All of those networks are trained by an Adam optimizer using a certain number (Mini-batch size M) of transitions data sampled from the replay buffer $\mathcal{R}$ at each time step. All of the aforementioned four networks including two actor networks and two critic networks have two hidden layers respectively. The actor networks approximate the deterministic policy function $\boldsymbol{a}_t = \mu(\boldsymbol{s}_t)$; thus, the dimension of the input layer equals to the state dimension $\mathbf{dim}(\boldsymbol{s}_t)$ while the output layer equals to $\mathbf{dim}(\boldsymbol{a}_t)$. On the other hand, the critic networks approximate the action-value function $Q(\boldsymbol{s}_t, \boldsymbol{a}_t)$ thus the dimension of the input layer equals to the sum of the action dimension $\mathbf{dim}(\boldsymbol{a}_t)$ and the state dimension $\mathbf{dim}(\boldsymbol{s}_t)$ while the dimension of the output layer equals to one because the action value is a real scalar. The choice of key parameters for the DDPG agent is shown in Table 2.

The third part is the inverse kinematics module. We perform the inverse kinematics calculation according to the current position of joints $\boldsymbol{q}$ to obtain $\dot{\boldsymbol{q}}_{\text{inv}}$ at each time step. This part will be executed in Matlab. More specifically, we set up a pure kinematic model of the welding manipulator synchronize with the one in simulator CoppeliaSim, and the instantaneous Jacobian matrix as shown in Eq. 4 is computed at each time step which is used to obtain $\dot{\boldsymbol{q}}_{\text{inv}}$.

These three parts communicate through remote Application Programming Interface (API) or engine as shown in Fig. 4. The whole training phase is implemented on a PC (CPU 3.6 GHz, RAM 16 GB).

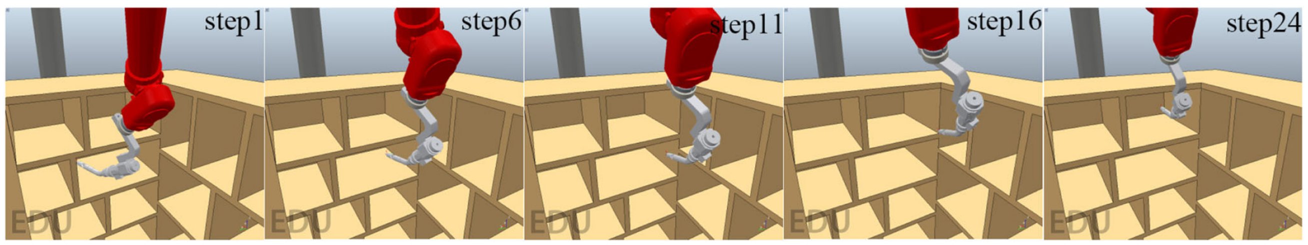
Experimental results and analysis

In various algorithms of path planning, the proposed learning-based path planning algorithm can be classified as a global and single query path planning method. To validate the effectiveness of the proposed method, we test the learning-based planner with three different path planning tasks. Each task was specified with an initial position $\boldsymbol{q}_s \in \mathbb{R}^6$ drawn from

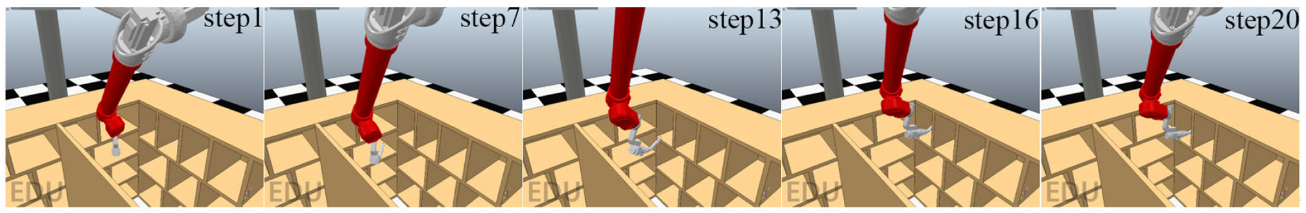


(a)Task#1 (b)Task#2 (c)Task#3

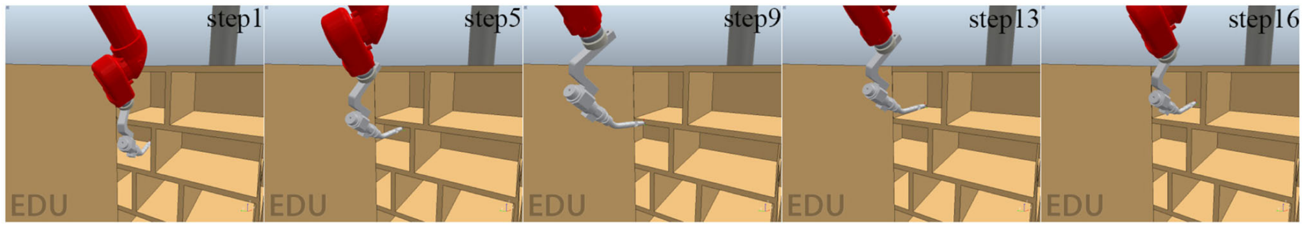
Fig. 5 The learning curves of the training process for three tasks which different gain module $\mathcal{G}(t)$. These curves represent the convergence of different training strategies



(a) Task1: initial configuration q_s=(29.97,11.15,30.45,-65.09,-37.99,70.12) in degs, target position $p_e^* = (0.87, 0.18, 0.40)$ in meters



(b) Task2: initial configuration q_s=(17.57,13.24,31.23,-87.83,9.65,118.41) in degs, target position $p_e^* = (0.92, 0.48, 0.70)$ in meters



(c) Task3: initial configuration q_s=(11.12,10.10,27.83,-46.0,-19.68,45.12) in degs, target position $p_e^* = (0.87, -0.17, 0.39)$ in meters

Fig. 6 The path planning solutions conducted by our method for three single-query tasks rendered in CoppeliaSim. For each solution, we select some pivotal way points to express the complete safe path

the configuration space and a target end-effector's Cartesian position $\boldsymbol{p}_e^* \in \mathbb{R}^3$ relative to the welding manipulator's reference frame.

Performance evaluation

We evaluate the performance of our method on the three single-query test tasks from different domains. First of all, it is known that the objective of the Deep-RL algorithm is to maximize the cumulative reward in one episode with finite time steps and, therefore, it is necessary to analyze the trend of the reward curve or the learning curve vs training episode number which reflects whether the target deterministic policy model has converged as well as the learning efficiency. We show the learning curves of the three test tasks, respectively, in Fig. 5. For each specified path planning task, we train the policy model using different gain module noted as $\mathcal{G}(t)$. When $\mathcal{G}(t) = 0$, it means that the inverse kinematics module is removed and the agent should learn from scratch which is abandoned in our research. To exploit the inverse

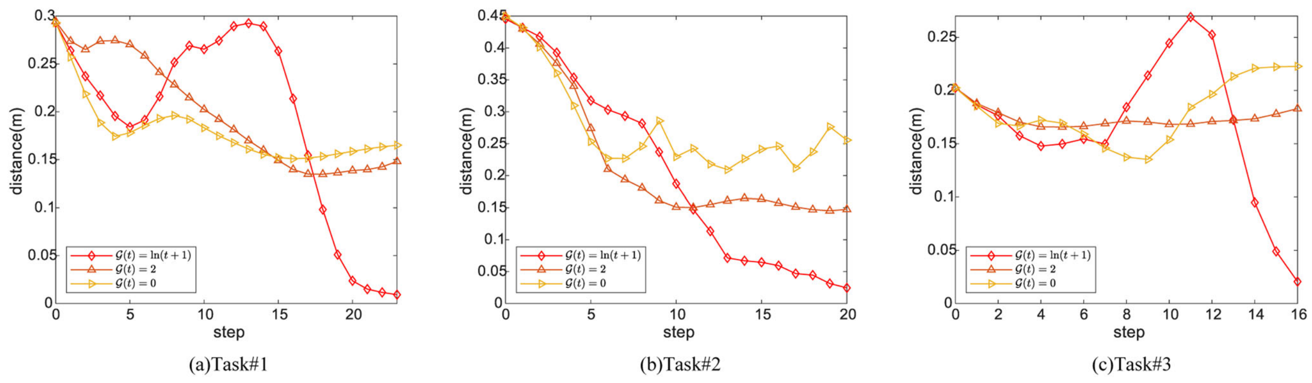


Fig. 7 Distance between the current end-effector's Cartesian position and the target Cartesian position

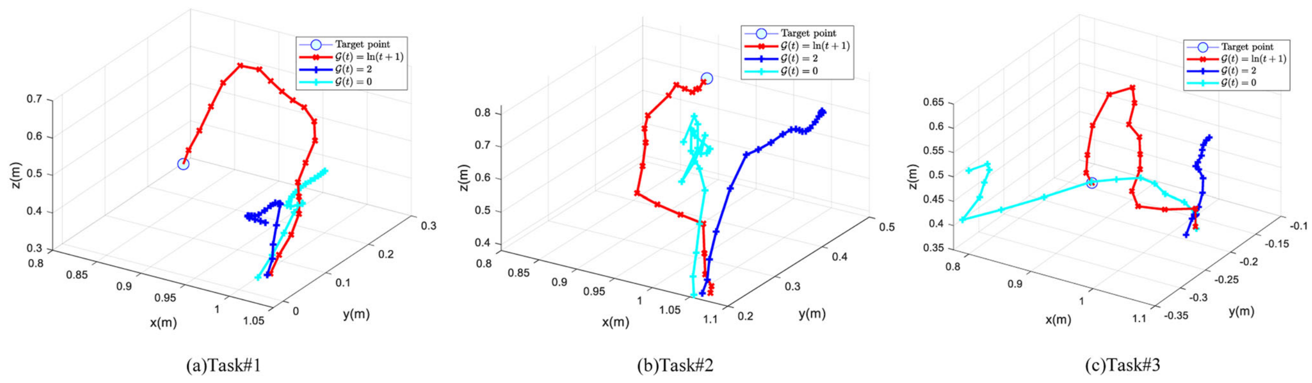


Fig. 8 Cartesian position of the end-effector of the welding manipulator along the path in the workspace of the welding manipulator

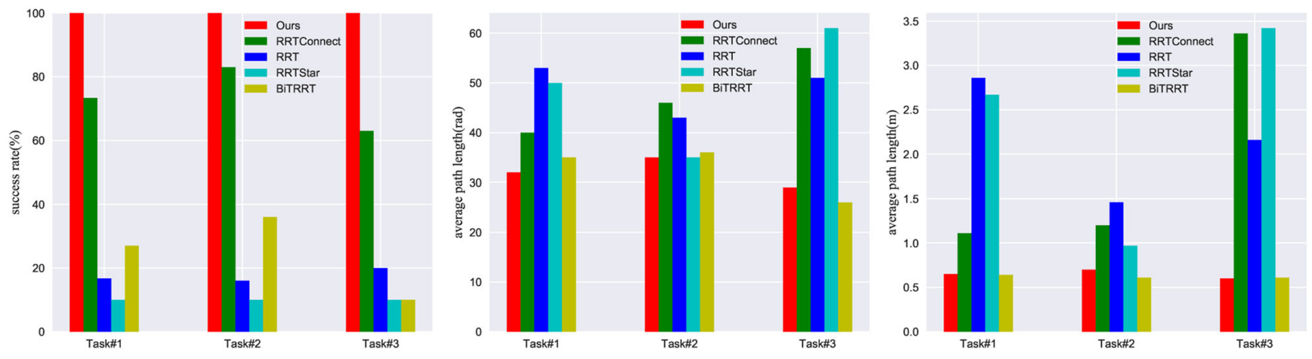


Fig. 9 Performance comparison of our algorithm and sampling-based algorithms by three histograms, where **a** illustrates the success rate while **b** and **c** illustrates average path length in C-space and Cartesian space, respectively

kinematics module during the training phase, we need a non-zero $\mathcal{G}(t)$; thus, we adopt $\mathcal{G}(t) = \ln(t+1)$ as shown in Eq. 22. To reveal the progress of the proposed time-varying gain module, we also adopt a fixed gain module $\mathcal{G}(t) = 2$ as contradistinction. After comparison, we find that different selection of gain module $\mathcal{G}(t)$ leads to significantly different learning curve when dealing with the same task. Our method which adopts a time-varying gain module $\mathcal{G}(t) = \ln(t+1)$ achieved the best performance in terms of the convergence as the red curves in Fig. 5 show. The red curves in Fig. 5a–c finally converge without exception to a near-zero level from which we can infer that the learned agents have accomplished the specified tasks in the light of our reward design mechanism. On the other hand, when $\mathcal{G}(t) = 0$ or $\mathcal{G}(t) = 2$, the learning curves fail to converge like the red curves do in the same range of training episodes; therefore, it can be seen that our method has significantly improved convergence performance and convergence speed which is a credit to the introduction of the proposed inverse kinematics module and gain module.

Second, we need to confirm whether the final deterministic policy model mentioned above can be used to generate a collision-free path that is a solution to the respective specified task. The collision-free path in configuration space of the welding manipulator can be obtained indirectly by way of sampling a trajectory $\boldsymbol{s}_0, \boldsymbol{a}_0, \boldsymbol{s}_1, \ldots, \boldsymbol{a}_T, \boldsymbol{s}_T$ using the final policy model and then construct the solution $\left\{\boldsymbol{q}_i\right\}$, $i = 0, 1, \ldots, T$ from $\{\boldsymbol{s}_i\}$ where $\boldsymbol{s}_0$ is equivalent to the state when the welding manipulator is at the configuration of $\boldsymbol{q}_s$. Consistent with the previous assumption, the policy model trained by our method described in Algorithm 3 which employs both the gain module $\mathcal{G}(t) = \ln(t+1)$ and inverse kinematic module can generate a unique collision-free path in the configuration space which fulfills respective task requirements. We render the path intuitively and succinctly in the simulator CoppeliaSim as Fig. 6 shows.

In addition to collision avoidance, the path should ensure that the end-effector of the welding manipulator is at its target position $\boldsymbol{p}_e^*$ finally. Although all of the three models trained by different $\mathcal{G}(t)$ can generate a collision-free path for respective task, only the model trained by $\mathcal{G}(t) = \ln(t+1)$ is able to move the end-effector to its target position. To intuitively reflect the performance of the models in the control of end-effector's position, we plot the distance between the end-effector and its target Cartesian position vs the path steps as Fig. 7 shows. It is clear that all of the red curves that represent our method eventually drop to zero or near-zero while other curves fail to drop as the red ones do, the second half of these failed curves fluctuates around a number significantly greater than zero, which shows that those learned policy fails to convergence and is trapped in local optima. On the other hand, as shown in Fig. 8, obviously, only our proposed training method can move the end-effector to its target position. However, the policy learned by other training methods (e.g. $\mathcal{G}(t) = 2$ or $\mathcal{G}(t) = 0$) controls the manipulator arm to gradually move toward the non-target direction and the movement step is gradually reduced, which can also indicate that these paths fall into the local optimum. The above description indicates that only the policy trained by our method can successfully complete all given tasks including collision avoidance and moving the end-effector to its target position.

Comparison with other methods

In this paper, we introduce an alternative collision-free path planning algorithm for industry manipulators. As we all know, for the path planning problem in high-dimensional sampling space, e.g. the configuration space of industry manipulators, the sampling-based method is one of the most efficient algorithms. To demonstrate the progress of our algorithm, the three experimental path planning tasks which have been used to test our algorithm are now used to query several staple sampling-based single-query path planning algorithm, i.e. RRT-Connect [46], RRT [7], RRTStar [47], and BiTRRT [48]. We utilize the open-source motion planning library "*The Open Motion Planning Library*"(OMPL) [49] to implement the above planning algorithm on the three test tasks.

We select three performance indicators to evaluate the performance of different algorithms from different dimensions. The first performance indicator is the success rate for a single-query path planning task in a specified number of times and the other two indicators aim to measure path length in configuration space and workspace, namely Cartesian space. Generally speaking, the success rate of path planning means the robustness of the algorithm while the length of the planned path represents the optimality of the path. In Fig. 9, we reveal our algorithm's performance through comparison with other algorithms.

First, since the path generated by the converged model is deterministic and successful each time, the success rate of our path planner based on the policy model is 100%. However, in line with expectations, the success rate of sampling-based planning algorithms is relatively low, and the highest success rate is only just about 70% which is got by the RRT-Connect algorithm. This is obviously not allowed in industrial scenarios, so our algorithm has unique advantages over those sampling-based planning algorithms. Second, as we all know, from the perspective of path length, only optimal RRT algorithms such as RRTStar [47] can find the optimal or asymptotical optimal path, but they may require a huge number of samples to achieve such optimality. our algorithm achieves the same performance as BiTRRT does which is better than any other algorithms, In other words, the path generated by a well-trained policy using our method can be comparable to an optimal sampling-based path planning algorithm.

Conclusion and future work

In this research, we propose a hybrid algorithm of DDPG and Inverse kinematics as an alternative global and single-query path planning algorithm for industry welding manipulator. Simulation results in high-fidelity virtual scenario show that our improvements to the original algorithm DDPG have greatly increased its convergence performance and convergence speed and the well-trained policy model is able to generate a collision-free path for a specified task. In industry robot systems like automatic welding robot systems that require extremely high stability and safety, our algorithm may be competitive comparing with other sampling-based algorithms because the sampling-based algorithm can not guarantee 100% success rate of planning. Furthermore, our algorithm can also produce path planning results comparable to those of optimality or progressive optimality algorithms in terms of path length. In summary, our algorithm provides an alternative solution for teaching-free automatic path planning of actual welding robots or other industrial robots.

Despite the progress of the proposed algorithm, we still have lots of future work to do. We always remember that the model-free deep reinforcement learning algorithm such as DDPG in this paper is born with the characteristics of low sample efficiency. Although we promote the sample efficiency greatly in this research, we will still go on to develop a more elegant method to accelerate the convergence of the Deep-RL algorithm for path planning. On the other hand, we also wish our algorithm can be applied to real-world robots to complete sim-to-real tasks.

Acknowledgements Our work is supported by multiple funds in china, including the Key Program of NSFC-Guangdong Joint Funds (U1801263, U1701262, U2001201), Major projects of science and technology plan of Guangdong Province (2015B090922013, 2017B090901019), Major projects of science and technology plan of Foshan City (1920001001367). Our work is also supported by Guangdong Provincial Key Laboratory of Cyber-Physical System (2016B030301008).

Declarations

Conflict of interest The authors declare that they have no conflict of interest.

References

1. Lee D, Ku N, Kim TW, Kim J, Lee KY, Son YS (2011) Development and application of an intelligent welding robot system for shipbuilding. Robot Comput-Integr Manufact 27(2):377–388
2. Lee D (2014) Robots in the shipbuilding industry. Robot Comput-Integr Manufact 30(5):442–450
3. Budak G, Chen X (2020) Evaluation of the size of time windows for the travelling salesman problem in delivery operations. Compl Intell Syst 6(3):681–695
4. Hsu D, Latombe J.C, Motwani R (1997) Path planning in expansive configuration spaces. In: Proceedings of International Conference on Robotics and Automation, IEEE, vol 3, pp 2719–2726
5. Amato N.M, Wu Y (1996) A randomized roadmap method for path and manipulation planning. In: Proceedings of IEEE international conference on robotics and automation, IEEE, vol 1, pp 113–120
6. Kavraki LE, Svestka P, Latombe JC, Overmars MH (1996) Probabilistic roadmaps for path planning in high-dimensional configuration spaces. IEEE Trans Robot Autom 12(4):566–580
7. LaValle SM, Kuffner JJ, Donald BR (2001) Rapidly-exploring random trees: progress and prospects[J]. In: Algorithmic and computational robotics: new directions, vol 5. pp 293–308
8. Qureshi AH, Ayaz Y (2015) Intelligent bidirectional rapidly-exploring random trees for optimal motion planning in complex cluttered environments. Robot Autonom Syst 68:1–11
9. Pérez-Higueras N, Jardón A, Rodríguez Á, Balaguer C (2020) 3d exploration and navigation with optimal-rrt planners for ground robots in indoor incidents. Sensors 20(1):220
10. Connell D, La HM (2017) Dynamic path planning and replanning for mobile robots using rrt. In: 2017 IEEE international conference on systems, man, and cybernetics (SMC), IEEE, pp 1429–1434
11. Yang K, Keat Gan S, Sukkarieh S (2013) A gaussian process-based rrt planner for the exploration of an unknown and cluttered environment with a uav. Adv Robot 27(6):431–443
12. Akbaripour H, Masehian E (2016) Semi-lazy probabilistic roadmap: a parameter-tuned, resilient and robust path planning method for manipulator robots. Int J Adv Manuf Technol 89(5–8):1401–1430
13. Cao X, Zou X, Jia C, Chen M, Zeng Z (2019) Rrt-based path planning for an intelligent litchi-picking manipulator. Comput Electron Agric 156:105–118
14. Jenamani R.K, Kumar R, Mall P, Kedia K (2020) Robotic motion planning using learned critical sources and local sampling. arXiv:2006.04194
15. Khatib O (1986) Real-time obstacle avoidance for manipulators and mobile robots. In: Autonomous robot vehicles, pp 396–404. Springer, Berlin
16. Xie L, Liu S (2018) Dynamic obstacle-avoiding motion planning for manipulator based on improved artificial potential filed. Control Theory Appl 35(9):27–37
17. Li H, Wang Z, Ou Y (2019) Obstacle avoidance of manipulators based on improved artificial potential field method. In: 2019 IEEE international conference on robotics and biomimetics (ROBIO), IEEE, pp 564–569
18. Liu S, Zhang Q, Zhou D (2014) Obstacle avoidance path planning of space manipulator based on improved artificial potential field method. J Inst Engineers (India): Ser C 95(1):31–39
19. Sathya A.S, Gillis J, Pipeleers G, Swevers J (2020) Real-time robot arm motion planning and control with nonlinear model predictive control using augmented lagrangian on a first-order solver. In: 2020 European Control Conference (ECC), IEEE, pp 507–512
20. Chen B, Dai B, Lin Q, Ye G, Liu H, Song L (2020) Learning to plan in high dimensions via neural exploration-exploitation trees. arXiv:1903.00070
21. Qureshi A.H, Simeonov A, Bency M.J, Yip M.C (2019) Motion planning networks. In: 2019 international conference on robotics and automation (ICRA), IEEE, pp 2118–2124
22. Nguyen H, La H (2019) Review of deep reinforcement learning for robot manipulation. In: 2019 Third IEEE international conference on robotic computing (IRC), IEEE, pp. 590–595
23. Kober J, Bagnell JA, Peters J (2013) Reinforcement learning in robotics: a survey. Int J Robot Res 32(11):1238–1274
24. Li Y, Xia H, Zhao B (2018) Policy iteration algorithm based fault tolerant tracking control: an implementation on reconfigurable manipulators. J Electr Eng Technol 13(4):1740–1751
25. Zhang Y, Zhao B, Liu D (2020) Deterministic policy gradient adaptive dynamic programming for model-free optimal control. Neurocomputing 2020:5
26. Yang Y, Guo Z, Xiong H, Ding DW, Yin Y, Wunsch DC (2019) Data-driven robust control of discrete-time uncertain linear systems via off-policy reinforcement learning. IEEE Trans Neural Netw Learn Syst 30(12):3735–3747

27. Yang Y, Vamvoudakis KG, Modares H, Yin Y, Wunsch DC (2020) Hamiltonian-driven hybrid adaptive dynamic programming. IEEE Trans Syst Man Cybernet Syst 2020:1–12
28. Yang Y, Ding DW, Xiong H, Yin Y, Wunsch DC (2019) Online barrier-actor-critic learning for h ∞ control with full-state constraints and input saturation. J Franklin Inst 357(6):3316–3344
29. Yang Y, Vamvoudakis KG, Modares H, Yin Y, Wunsch DC (2020) Safe intermittent reinforcement learning with static and dynamic event generators. IEEE Trans Neural Netw Learn Syst PP(99):1–15
30. Xue X, Li Z, Zhang D, Yan Y (2019) A deep reinforcement learning method for mobile robot collision avoidance based on double dqn. In: 2019 IEEE 28th international symposium on industrial electronics (ISIE), IEEE, pp 2131–2136
31. Jesus J.C, Bottega J.A, Cuadros M.A, Gamarra D.F (2019) Deep deterministic policy gradient for navigation of mobile robots in simulated environments. In: 2019 19th international conference on advanced robotics (ICAR), IEEE, pp 362–367
32. Sangiovanni B, Rendiniello A, Incremona G.P, Ferrara A, Piastra M (2018) Deep reinforcement learning for collision avoidance of robotic manipulators. In: 2018 European Control Conference (ECC), IEEE, pp 2063–2068
33. Gu S, Lillicrap T, Sutskever I, Levine S (2016) Continuous deep q-learning with model-based acceleration. In: International conference on machine learning, pp 2829–2838
34. Sangiovanni B, Incremona GP, Piastra M, Ferrara A (2020) Self-configuring robot path planning with obstacle avoidance via deep reinforcement learning. IEEE Control Syst Lett 5(2):397–402
35. Hua X, Wang G, Xu J, Chen K et al (2019) Reinforcement learning-based collision-free path planner for redundant robot in narrow duct. J Intell Manufact 2019:1–12
36. Lillicrap T.P, Hunt J.J, Pritzel A, Heess N, Erez T, Tassa Y, Silver D, Wierstra D (2015) Continuous control with deep reinforcement learning. arXiv:1509.02971
37. Paredis CJ, Khosla PK (1993) Kinematic design of serial link manipulators from task specifications. Int J Robot Res 12(3):274–287
38. Siciliano B, Khatib O (2016) Springer handbook of robotics. Springer, Berlin
39. Thomopoulos SC, Tam RY (1991) An iterative solution to the inverse kinematics of robotic manipulators. Mech Mach Theory 26(4):359–373
40. Wampler CW (1986) Manipulator inverse kinematic solutions based on vector formulations and damped least-squares methods. IEEE Trans Syst Man Cybern 16(1):93–101
41. Corke P (2017) Robotics, vision and control: fundamental algorithms in MATLAB® second, completely revised, vol 118. Springer, Berlin
42. Sutton RS, Barto AG (2018) Reinforcement learning: an introduction. MIT Press, Hoboken
43. Henderson P, Islam R, Bachman P, Pineau J, Precup D, Meger D (2017) Deep reinforcement learning that matters. arXiv:1709.06560
44. Mnih V, Kavukcuoglu K, Silver D, Rusu AA, Veness J, Bellemare MG, Graves A, Riedmiller M, Fidjeland AK, Ostrovski G et al (2015) Human-level control through deep reinforcement learning. Nature 518(7540):529–533
45. Silver D, Lever G, Heess N, Degris T, Wierstra D, Riedmiller M (2014) Deterministic policy gradient algorithms[C]. In: International conference on machine learning. PMLR, pp 387–395
46. Kuffner J.J, LaValle S.M (2000) Rrt-connect: an efficient approach to single-query path planning. In: Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No. 00CH37065), IEEE, vol 2, pp 995–1001
47. Karaman S, Frazzoli E (2011) Sampling-based algorithms for optimal motion planning. Int J Robot Res 30(7):846–894
48. Gammell JD, Barfoot TD, Srinivasa SS (2020) Batch informed trees (bit*): informed asymptotically optimal anytime search. Int J Robot Res 39(5):543–567
49. Sucan IA, Moll M, Kavraki LE (2012) The open motion planning library. IEEE Robot Autom Mag 19(4):72–82