ASME Journal of Manufacturing Science and Engineering
Online journal at:
https://asmedigitalcollection.asme.org/manufacturingscience

Qi Zhou
University of Michigan–Shanghai Jiao Tong University Joint Institute,
Shanghai Jiao Tong University,
800 Dong Chuan Road, Minhang District,
Shanghai 200240, China
e-mail: zhouqi1998@sjtu.edu.cn

Jin Wu
Laboratory for Computational Sensing and Robotics,
Johns Hopkins University,
3400 North, Charles Street,
Baltimore, MD 21218
e-mail: jwu220@jh.edu

Boyan Li
University of Michigan–Shanghai Jiao Tong University Joint Institute,
Shanghai Jiao Tong University,
800 Dong Chuan Road, Minhang District,
Shanghai 200240, China
e-mail: liboyan@sjtu.edu.cn

Sikai Li
University of Michigan–Shanghai Jiao Tong University Joint Institute,
Shanghai Jiao Tong University,
800 Dong Chuan Road, Minhang District,
Shanghai 200240, China
e-mail: skevinci@sjtu.edu.cn

Bohan Feng
University of Michigan–Shanghai Jiao Tong University Joint Institute,
Shanghai Jiao Tong University,
800 Dong Chuan Road, Minhang District,
Shanghai 200240, China
e-mail: bohan.feng@sjtu.edu.cn

Jiangshan Liu
University of Michigan–Shanghai Jiao Tong University Joint Institute,
Shanghai Jiao Tong University,
800 Dong Chuan Road, Minhang District,
Shanghai 200240, China
e-mail: jiangshan.liu@sjtu.edu.cn

Adaptive Robot Motion Planning for Smart Manufacturing Based on Digital Twin and Bayesian Optimization-Enhanced Reinforcement Learning

Advanced motion planning is crucial for safe and efficient robotic operations in various scenarios of smart manufacturing, such as assembling, packaging, and palletizing. Compared to traditional motion planning methods, Reinforcement Learning (RL) shows better adaptability to complex and dynamic working environments. However, the training of RL models is often time-consuming, and the determination of well-behaved reward function parameters is challenging. To tackle these issues, an adaptive robot motion planning approach is proposed based on digital twin and reinforcement learning. The core idea is to adaptively select geometry-based or RL-based methods for robot motion planning through a real-time distance detection mechanism, which can reduce the complexity of RL model training and accelerate the training process. In addition, Bayesian Optimization is integrated within RL training to refine the reward function parameters. The approach is validated with a Digital Twin-enabled robot system through five kinds of tasks (Pick and Place, Drawer Open, Light Switch, Button Press, and Cube Push) in dynamic environments. Experiment results show that our approach outperforms the traditional RL-based method with improved training speed and guaranteed task performance. This work contributes to the practical deployment of adaptive robot motion planning in smart manufacturing.
[DOI: 10.1115/1.4067616]

Keywords: robot motion planning, reinforcement learning, digital twin, Bayesian optimization, smart manufacturing, assembly, control and automation, robotics and flexible tooling

[1]Corresponding author.
Manuscript received August 7, 2024; final manuscript received December 15, 2024; published online January 24, 2025. Assoc. Editor: Ran Jin.

Youyi Bi[1]
University of Michigan–Shanghai Jiao Tong
University Joint Institute,
Shanghai Jiao Tong University,
800 Dong Chuan Road, Minhang District,
Shanghai 200240, China
e-mail: youyi.bi@sjtu.edu.cn

1 Introduction

Robots are playing an increasingly significant role in manufacturing owing to their efficiency and accuracy in executing repetitive tasks, such as stacking, welding, and assembly [1]. In the context of smart manufacturing, robotic operations are frequently involved with human-robot collaboration and multi-robot cooperation [2]. In these environments, robots face various challenges, such as avoiding moving objects, interacting with other robots, and adapting to human interventions [3,4]. However, traditional robot motion control based on manual instruction may not adapt well to frequently changing working scenarios. Therefore, it has become highly challenging for robots to operate safely and efficiently in dynamic environments, and the research for advanced motion planning methods is crucial.

The core of robot motion planning is path planning, which aims to generate safe paths from start positions to target positions. Conventional path planning methods (e.g., geometry-based method [5], potential field method [6], and bio-inspired heuristic method [7]) are often limited in adapting to complex working environments [8]. Sampling-based methods, such as Rapidly Exploring Random Trees (RRT) [9] and its variations [10], generate viable paths by sampling the configuration space and have been used widely in motion planning. However, these methods suffer from long computation times when dealing with constantly changing surroundings [11]. In contrast, learning-based methods, especially Reinforcement Learning (RL), are gaining more attention recently. RL treats the robot as an intelligent agent that learns to take actions through interactions with environment [12]. With strong self-learning capabilities, RL can enhance the adaptability and efficiency of robotic systems in complex and dynamic settings [13].

Despite RL-based motion planners having shown favorable performance in dynamic scenarios, they still confront several challenges in a practical industrial context. One challenge is the time-consuming training of RL models. Training a workable RL model for robot motion planning in a laboratory environment often takes several hours. In real industrial scenarios, it may require even more time, limiting the practical use of such methods [14]. In addition, the performance of RL models is sensitive to the choice of reward functions, especially under complex tasks and environments. Slight adjustments to these parameters may significantly impact the model performance. Therefore, there is a need to quickly select refined reward function parameters of RL models [15].

To address these issues, a robot motion planning approach is proposed for the dynamic manufacturing environment. It can adaptively select geometry-based or RL-based methods with a real-time distance detection mechanism. This adaptive switching of methods can reduce the complexity of the training RL model and accelerate the training speed. In addition, a mechanism is proposed for refining the reward function parameters of the RL model based on Bayesian Optimization (BO). To demonstrate the approach, a Digital Twin (DT)-enabled robot system with a high-fidelity training environment is established. With the support of DT, RL-based methods allow robots to train within the virtual environment with highly realistic working spaces in a parallel way, facilitating the learning of optimal motion trajectories. We validate the approach through five kinds of tasks (Pick and Place, Drawer Open, Light Switch, Button Press, and Cube Push) in dynamic environments. Experiment results show that our approach outperforms the traditional RL-based method with improved training speed and guaranteed task performance. The main contributions of this study include the following:

- A comprehensive robot motion planning framework for dynamic working scenarios based on digital twin and reinforcement learning is proposed. Our framework can be applied in multiple robotic operations in smart manufacturing and exhibits strong adaptability to different dynamic environments.
- An adaptive robot motion planning method integrating geometry-based methods and reinforcement learning is developed. This method can improve the model performance and its robustness to various reward magnitudes with faster convergence speed.
- A Bayesian Optimization–based mechanism is designed for determining reward function parameters of the RL model. This mechanism facilitates automatic optimization of reward function parameters in relatively few iterations.

The rest of this paper is organized as follows. Section 2 reviews related work in robot path planning, methods for accelerating RL training, and optimization of reward function parameters. Section 3 introduces the architecture of the proposed approach and explains its key components. Section 4 presents the simulation and real-world experimental setups used to validate our approach, and provides a detailed discussion of the experimental results. Section 5 summarizes our work and highlights potential future research directions.

2 Related Work

2.1 Robot Motion Planning in Dynamic Environment. In the robot motion planning area, sampling-based methods are commonly used and have achieved success in certain dynamic environments. For example, Adiyatov and Varol [16] proposed RRT*FN-Dynamic algorithm for dynamic scenarios, which retains tree parts, uses heuristics for repair, and achieves shorter pathfinding times. Chen et al. [17] introduced a framework that combines generalized velocity obstacles (GVO) with RRT* for nonholonomic robots and validated for feasibility in environments with moving obstacles. Qi et al. [18] presented a multi-objective dynamic rapidly exploring random (MOD-RRT*) algorithm for robot navigation in an unknown dynamic environment. This approach consists of a path generation stage and a path replanning stage, demonstrating improved performance in path cost, smoothness, and stability. However, sampling-based methods often remain slow and inefficient, especially in complex scenarios [19].

To tackle this issue, researchers explore to use of reinforcement learning (RL) for motion planning in dynamic scenarios due to its strong self-learning capabilities [20]. For instance, Zhang and Chen [21] proposed an improved Soft Actor–Critic Long Short-Term Memory (SAC-LSTM) algorithm for fast path planning of mobile robots in dynamic environments. By incorporating historical and current states, the method exhibits a better performance in higher path planning success rate and shorter path length. Schmitt et al. [22] presented a feedback planner for manipulation tasks

and achieved increased robustness in dynamic environments using a switching-control scheme guided by a reinforcement learning agent. Besides, Chai et al. [23] developed a hierarchical deep learning-based control framework for mobile robots operating in uncertain environments, employing a recurrent deep neural network (RDNN) for motion planning and a deep reinforcement learning (DRL) algorithm for collision-free waypoint tracking. However, the training for RL models is time-consuming for complex robot tasks due to the low utilization of samples and low learning efficiency of agents, which limits the practical application of RL-based methods.

2.2 Accelerate the Training of Reinforcement Learning Models. In order to speed up the training process of RL models, researchers have made several attempts. For example, Chen et al. [24] integrated the RL algorithm with Priority Experience Replay (PER) to replay training experiences with higher learning values at a higher frequency. This method improves the convergence speed and stability of the training process. Zhou et al. [25] introduced curriculum learning to divide the Pick and Place task into three stages, which helps the RL agent learn the task faster and achieves better task performance in a static environment.

In the context of robot motion planning, traditional planning methods have been integrated to reduce the training workload of RL models. For example, Luipers et al. [26] leveraged RL to plan waypoints first and then invoked the Inverse Kinematics (IK) solver to calculate the trajectory between waypoints. This method greatly accelerates the convergence rate of RL models. Zhong et al. [27] added an IK module into the RL algorithm to provide prior knowledge and reduce unnecessary exploration in the state-action space. Li et al. [28] combined traditional path planners and DRL and demonstrated a more powerful performance in length-optimal path planning problems. Faust et al. [29] integrated RL with a sampling-based method, Probabilistic Roadmaps (PRMs). The RL agents learn short-range, point-to-point navigation policies while the PRMs planners provide roadmaps that connect robot configurations.

In earlier studies, RL is usually employed to plan the entire motion path for collision avoidance, which can be inefficient when obstacles are confined to certain areas. In other words, the various safety conditions of different working areas are not treated targetedly. Thus, we expect to develop a method that can adaptively select an RL-based method or alternative planning method for planning areas with different safety conditions, which can further reduce the difficulty of RL model training and improve its adaptability to diverse dynamic environments.

2.3 Optimization of Reward Function Parameters in Reinforcement Learning Models. Reward function is a core design of RL models and the reward function parameters (e.g., the proportional coefficient to adjust the strength of reward signal, or the threshold coefficient controlling critical decision points during the training process) can influence the model performance greatly [15]. Due to the inherent trade-off between exploration and exploitation in RL model training, there is no straightforward mapping between reward functions and the model performance. Thus, traditional optimization methods face challenges when applied to optimize the reward function parameters of RL models. Inverse reinforcement learning (IRL) is a versatile approach and has been used to learn reward function of an agent from human demonstrations [30], such as learning a socially adaptive control manner of robotic wheelchair and playing video games [31,32]. When applied in optimizing reward function parameters, IRL requires a large amount of high-quality human demonstration data, which is especially difficult in complex and dynamic industrial manufacturing environments.

Recently, researchers have started to utilize BO to refine the hyperparameters of RL models because of their probabilistic nature and balance between exploration and exploitation [33]. For example, Wilson and Fern [34] demonstrated an application of BO in RL to optimize parametric policies by making use of the sequential trajectory data generated by RL agents. Young et al. [35] presented "HyperSpace," a distributed BO technique that leverages statistical correlations among hyperparameters to discover optimal configurations which has shown good performance in delivering superior results for DRL tasks. Gong et al. [36] enhanced the efficiency of RL by utilizing BO to predict the decisions made by Unmanned Aerial Vehicles using historical trajectory data, which helps prevent inefficient exploration of actions and leads to improved convergence speed. These mentioned works focus on optimizing RL policies or hyperparameters, while few of them concentrate on optimizing the reward function parameters. Thus, in this study, we expect to develop a BO-based mechanism to refine the parameters of reward functions in RL models for robot motion planning.

3 Methods

3.1 Overall Structure of the Proposed Approach. In this study, the motion planning problem can be defined as follows. Let be the configuration space and $\mathcal{C}_{\mathbf{obs}}$ be the obstacle region. Denote the obstacle-free space as $\mathcal{C}_{\mathbf{free}} = \mathcal{C} \setminus \mathcal{C}_{\mathbf{obs}}$. The goal of motion planning is to find a collision-free path $\boldsymbol{\sigma}:[0, \mathbf{T}] \rightarrow \mathcal{C}_{\mathbf{free}}$ that connects the initial state $\mathbf{q}(\mathbf{x}_{\mathbf{init}})$ to the goal state $\mathbf{q}(\mathbf{x}_{\mathbf{goal}})$. Figure 1 presents the overall structure of the proposed adaptive robot motion planning approach enabled by digital twin (DT). Here, DT provides a high-fidelity virtual environment for the visual representation of physical entities which also benefits the offline training of RL models. The whole structure comprises a physical layer and a virtual layer. The former contains hardware components such as the multi-joint robot arm, electric gripper, RGB-D camera, physical robot controller, and workspace. The latter includes the high-fidelity digital twin model of the physical components, virtual robot controller, object detection, robot path planning, and reward function parameters refining with Bayesian Optimization. The communication between the virtual and physical layers is achieved through socket communication with TCP protocol. The working space information in physical layer can be mapped to virtual layer and the motion strategy generated by the virtual layer can be sent to the physical robot instantly.

The flow of our approach works as follows. Initially, the path planning model with refined RL reward function parameters is obtained by the Bayesian Optimization module. Then RGB-D camera captures the workspace information, which is transmitted to the virtual layer to update the digital twin model. Then, the path planning module adaptively chooses RL-based or geometry-based methods to generate path points according to the safety condition of the current state. These path points are sent to the Inverse Kinematics module to generate an action sequence in robot joint space, which are then provided to both the virtual robot for simulation and the physical robot for execution. To ensure synchronized movement between physical robot and virtual robot, the virtual layer generates the subsequent action sequence only after the real robot has executed the previous command. The key techniques involved in our approach are explained in detail in the following subsections.

3.2 Object Detection and Location. The first step for robot motion planning in a dynamic environment is to obtain real-time information about the environment (e.g., the position of obstacles and target objects). In this study, we develop an object detection mechanism based on OVE6D [37], a 6D pose estimation algorithm where the six dimensions involve the position (x, y, z) and orientation (R_x, R_y, R_z) of the object. OVE6D demonstrates outstanding generalizability in object detection applications and can handle unseen objects without a fine-tuning process. Additionally, OVE6D's fast response speed can benefit real-time detection in dynamic scenarios. Figure 2 presents the workflow of the object

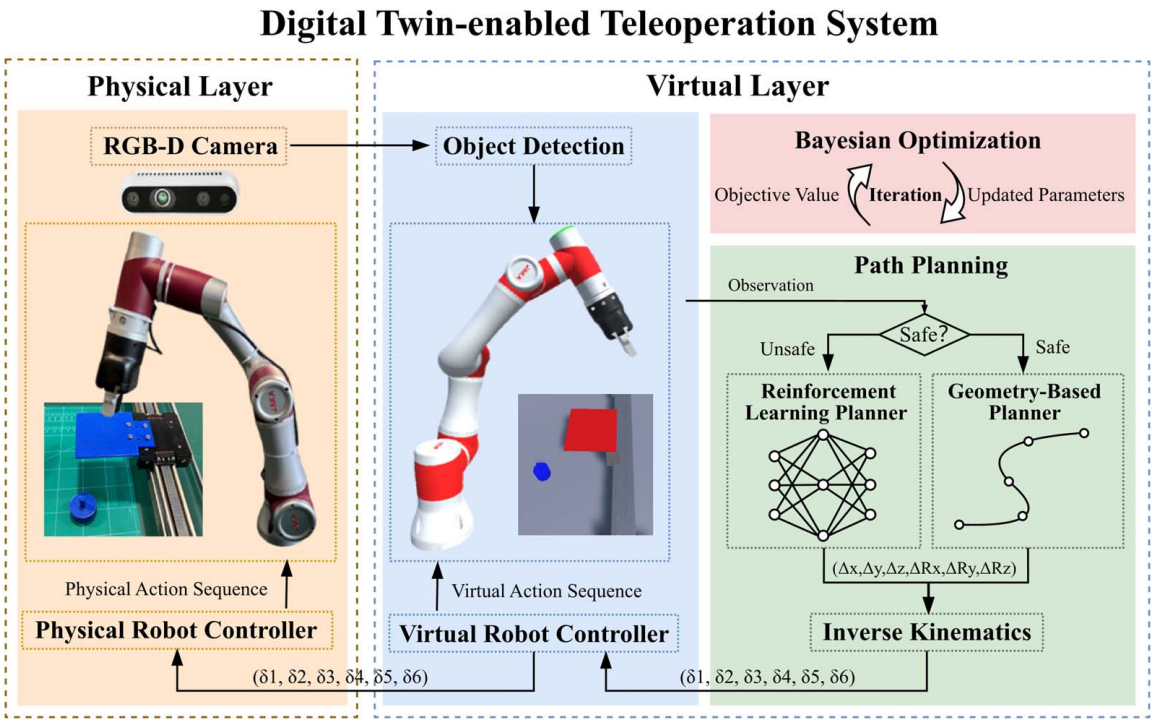

Fig. 1 Overall structure of the proposed adaptive robot motion planning approach

detection mechanism which consists of four steps. First, preprocessing is executed using the provided 3D models of objects to generate codebooks. With uniformly distributed views of the CAD model, rendered images from various viewpoints are encoded into high-dimensional feature vectors, which are then stored in a database for pose matching. The codebooks serve as a reference for the subsequent pose estimation. Then, an RGB-D camera is employed to capture the environmental image and the mask image is generated using RGB color thresholding and edge detection techniques. After that, the captured image, mask image, and the corresponding codebooks undergo processing via the OVE6D algorithm, providing the prediction of the label and pose for each object. Finally, the label and pose are transmitted to Unity3D to update the digital models' shape and pose simultaneously. This process is iteratively executed to track environmental changes in real time. Additionally, if an unexpected object (out of codebooks) suddenly appears, it is considered an interruption. The robot will halt until the object is removed, ensuring the safety and robustness of the robotic system during task execution.

In order to improve the adaptability and efficiency of object detection, we designed a new mechanism to eliminate the necessity of prior semantic information in the pose estimation neural network. Specifically, we calculate similarity scores between generated masks and views of each object during the viewpoint prediction process. Then, we determine the average of the top 10 scores for each object, representing the probability that the mask corresponds to the respective object. This mechanism is applied to each mask, attributing them to specific object types with the highest probability.

3.3 Adaptive Motion Planning

3.3.1 Workflow of the Adaptive Motion Planning Method. Figure 3 illustrates the workflow of the adaptive motion planning method. In the beginning, the RGB-D camera captures images of the environment. Then, the object detection mechanism computes real-time locations of target objects and obstacles from the images and transmits this location data to Unity3D for model update. Simultaneously, the safety distance detection mechanism monitors the minimum distance D_{ro} between the robot and obstacles. If D_{ro}

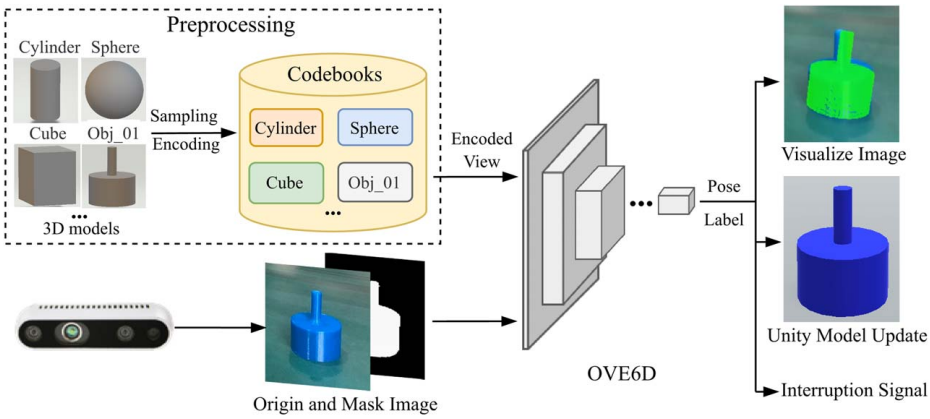

Fig. 2 The workflow of the object detection

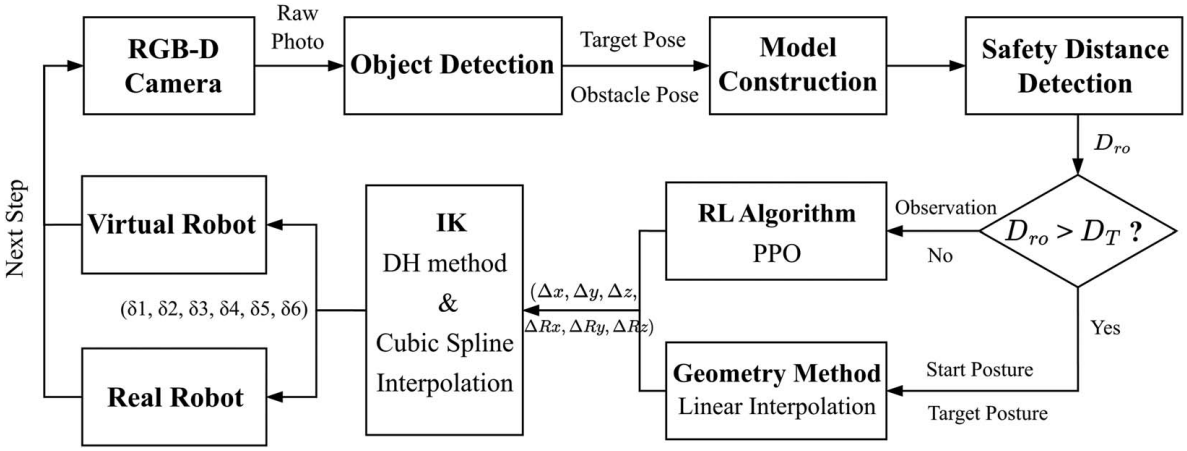

Fig. 3 The workflow of adaptive motion planning method

is greater than the switch distance D_T (determined through Bayesian Optimization, as detailed in Sec. 3.5), indicating that the robot is sufficiently away from obstacles, the geometry-based method is triggered without considering obstacles. Otherwise, the RL method will engage motion planning and generate action (Δx, Δy, Δz, ΔRx, ΔRy, ΔRz) for robot to avoid collision.

Subsequently, the IK module calculates the joint angles of the robot arm based on its current and target poses with cubic spline interpolation. The generated joint angles will be transmitted to both the virtual robot and the real robot to execute the corresponding motions. This motion planning procedure continues each time when the camera captures the change of the environmental status (i.e., one step) until the robot successfully accomplishes the task.

3.3.2 Safety Distance Detection. In order to detect the minimum distance (D_{ro}) between the robot and the obstacle in each step, a vertex-based distance detection mechanism is established, as Fig. 4 shows. This mechanism relies on the collision detection and coordinate transformation capabilities of the Unity3D engine. First, select several key vertices from the mesh model of the robot and obstacle which are prone to collision (e.g., the sixth joint and end effector of the robot, and the upper surface of the obstacle). Then, in each RL training step, the spatial distances between these selected vertices on the robot and obstacles are calculated. Finally, the shortest distance between the robot and the obstacle is set as D_{ro}. Usually, more vertices can improve the detection accuracy but may slow down the detection efficiency. Thus, the number of key vertices is determined through empirical tests to strike a balance between detection precision and computational cost. When observing a sudden slowdown of the detection speed as the number of vertices increases, the corresponding vertex number at this inflection point can be chosen. In addition, Gaussian noise D_n is added to D_{ro} to reflect the distance measurement inaccuracies caused by the sparse vertices of the robot and obstacle, making the RL model more robust to potential measurement errors during practical deployment. The proposed vertex-based distance detection offers both computational efficiency and adaptability compared to traditional convex hull detection techniques.

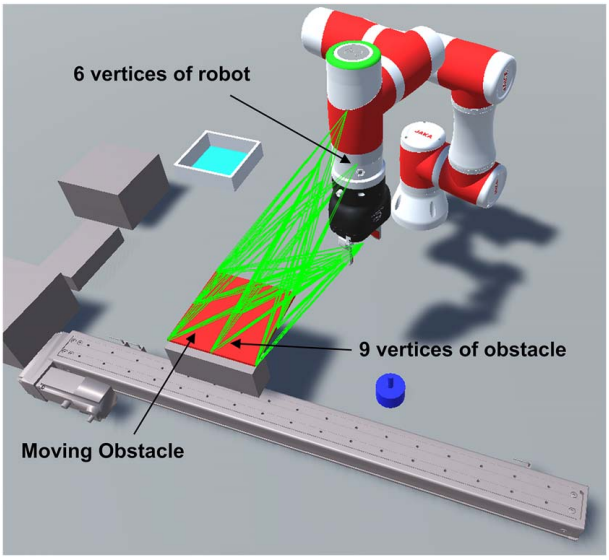

Fig. 4 Illustration of vertex-based distance detection. Here, six and nine vertices are selected from the robot and the surface of the obstacle, respectively. The lines between robot and obstacle indicate the detection lines between vertices.

3.3.3 Geometry-Based Path Planning Method. When the minimum distance between robot and obstacle (D_{ro}) is greater than switch distance (D_T), indicating the robot is far from obstacles, the geometry-based path planning method will be triggered. Geometry-based methods are selected for their simplicity and efficiency when the working space is clear and path planning is straightforward, in contrast to RL-based methods that require extensive training and computational resources to handle dynamic environments. Separating the use of geometry-based and RL-based methods can reduce the complexity of the RL model's training and accelerate the overall training process. Commonly used geometry-based methods include linear interpolation, spline interpolation, and Bezier curve interpolation. In view of the need for fast computation and collisions being not taken into consideration, linear interpolation is selected as the geometry-based method in our case. Specifically, given two waypoints representing the current and target positions of the robot's end-effector, linear interpolation calculates the intermediate positions along a straight line between these points in the Cartesian space.

After the RL or linear interpolation generates robot action in each step, IK method is utilized to convert the Cartesian action space into joint angle space for direct control of the robot. IK has been widely used in robotics to determine the needed rotation angles for each joint of a multi-joint robot arm given its initial and target poses. In this study, the Denavit–Hartenberg (DH) method [38], a commonly used method for describing the kinematics of robotic manipulators, is employed to standardize the kinematic model. Using the DH model, the joint angles are then calculated through an analytical IK approach, which ensures rapid and stable solutions compared to numerical IK methods. After obtaining the joint angles, cubic spline interpolation is applied to generate a smooth trajectory as the final output.

3.4 Reinforcement Learning. As mentioned earlier, if the distance between the robot and the obstacle is equal to or below the switch distance, the RL algorithm will be invoked to avoid collision. The core of reinforcement learning is a sequential decision-making problem referred to as Markov Decision Process (MDP) [39]. At each time-step t, the agent observes the environment state $S_t \in \mathcal{S}$ and chooses an action $A_t \in \mathcal{A}(s)$ based on the strategy $\pi(a|s)$. The agent then receives a reward $R_{t+1} \in \mathcal{R}$ and enters a new state S_{t+1}. In MDP, the occurring probability of reward R_t and environment state S_t depends only on the preceding state and action, as described in Eq. (1):

$$p(s', r|s, a) = \Pr\{S_t = s', R_t = r|S_{t-1} = s,\ A_{t-1} = a\} \tag{1}$$

where $p{:}\mathcal{S} \times \mathcal{R} \times \mathcal{S} \times \mathcal{A} \to [0, 1]$ is the probability of R_t and S_t occurring at time t. The goal of the agent is to maximize the expected return G_t:

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots = \sum_{i=0}^{\infty} \gamma^i R_{t+i+1} \tag{2}$$

where $\gamma \in [0, 1]$ is a discount factor to help maximize future expected returns rather than the current reward. In the context of robot arm motion planning, the agent of the robot arm can learn to find optimal motion strategy by iteratively interacting with the environment to achieve maximum reward, i.e., arriving at the target position without collision. In the proposed approach, we employ the Proximal Policy Optimization (PPO) algorithm [40] as the RL model. Its basic idea, the setting of the environment, states and action space, and the design of reward functions are explained as follows.

3.4.1 Proximal Policy Optimization. PPO is a deep reinforcement learning (DRL) method based on policy gradient. Compared with other DRL methods, PPO is more stable with high sampling efficiency for continuous action space and high-dimension state space problems. It utilizes a new objective function known as the clipped surrogate objective, which maximizes the expected return of the new policy while keeping the difference between the old and new policies small as defined in Eqs. (3) and (4):

$$L^{CLIP}(\theta) = \hat{\mathbb{E}}_t[\min(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t)] \tag{3}$$

$$r_t(\theta) = \frac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta\text{old}}(a_t \mid s_t)} \tag{4}$$

where $\hat{\mathbb{E}}_t$ is the expected value over all samples at time-step t, θ denotes the policy parameters, and r_t is the ratio between the new and old policies. $\hat{A}_t$ represents the advantage function, and ϵ is a hyperparameter that controls the range of the ratio.

3.4.2 Environment States and Action Space:. The state parameters $\boldsymbol{s_t}$ of the robot arm agent are set as follows:

$$\boldsymbol{s_t} = \{\boldsymbol{P_g}, \boldsymbol{P_t}, \boldsymbol{P_o}, \boldsymbol{P_p}, D_{\text{ro}}, D_{\text{gt}}, D_{\text{gp}}\} \tag{5}$$

where

$\boldsymbol{P_g} = (P_g^x, P_g^y, P_g^z, Rx_g, Ry_g, Rz_g)$ Pose of gripper
$\boldsymbol{P_t} = (P_t^x, P_t^y, P_t^z, Rx_t, Ry_t, Rz_t)$ Pose of target object
$\boldsymbol{P_o} = (P_o^x, P_o^y, P_o^z, Rx_o, Ry_o, Rz_o)$ Pose of obstacle
$\boldsymbol{P_p} = (P_p^x, P_p^y, P_p^z)$ Position of placing area
D_{ro} Minimum distance between robot and obstacle, float
D_{gt} Distance between gripper and target object, float
D_{gp} Distance between gripper and placing area, float

The pose vectors of gripper, target object, and obstacle mentioned above contain both position (e.g., P_g^x, P_g^y, P_g^z) and orientation (e.g., Rx_g, Ry_g, Rz_g) information. The action space of robot arm agent is continuous for more flexible control as shown below:

$$a = (\Delta x, \Delta y, \Delta z, \Delta Rx, \Delta Ry, \Delta Rz, \alpha) \tag{6}$$

$$\Delta x, \Delta y, \Delta z \in [-D_S, D_S]m \tag{7}$$

$$\Delta Rx, \Delta Ry, \Delta Rz \in [-D_S, D_S]\text{rad} \tag{8}$$

where $(\Delta x, \Delta y, \Delta z)$, $(\Delta Rx, \Delta Ry, \Delta Rz)$ are the translational displacement and rotational displacement of gripper center in Cartesian space, respectively. α is a binary-value variable controlling the opening and closing of the gripper. An action step size factor D_S is introduced to control the amplitude of robot action.

3.4.3 Design of Reward Functions. In order to reduce the difficulty of each task (Pick and Place, Drawer Open, Light Switch, Button Press, Cube Push), we divide the robot manipulation process into three stages: (1) Approaching, (2) Manipulating, and (3) Executing. The details of the three stages of each task are illustrated in Table 1.

The reward functions are designed as follows:

$$r(s) = R_A + R_O + R_E + R_{S0} + R_{S1} + R_{S2} + R_{S3} \tag{9}$$

where

$$R_A = \begin{cases} \lambda_1 - D_{\text{gt}} & \text{if Stage} = 1 \text{ or } 2 \\ 0 & \text{else} \end{cases} \tag{10}$$

$$R_O = \begin{cases} -\max\left(0, 1 - \dfrac{D_{\text{ro}}}{D_F}\right) & \text{if } D_{\text{ro}} \in \left[\dfrac{1}{2}D_F, D_F\right] \\ -2 \cdot \max\left(0, 1 - \dfrac{D_{\text{ro}}}{D_F}\right) & \text{if } D_{\text{ro}} \in \left[0, \dfrac{1}{2}D_F\right) \end{cases} \tag{11}$$

$$R_E = \begin{cases} \lambda_2 + \lambda_3 \cdot (\lambda_4 - D_{\text{gp}}) & \text{if Stage} = 3 \\ 0 & \text{else} \end{cases} \tag{12}$$

$$R_{S0} = \begin{cases} -1 & \text{if } P_{\text{Collide}} = 1 \\ 0 & \text{if } P_{\text{Collide}} = 0 \end{cases} \tag{13}$$

$$R_{S1} = \begin{cases} 1 & \text{if } P_{\text{Arrive}} = 1 \\ 0 & \text{if } P_{\text{Arrive}} = 0 \end{cases} \tag{14}$$

$$R_{S2} = \begin{cases} 5 & \text{if } P_{\text{Manipulate}} = 1 \\ 0 & \text{if } P_{\text{Manipulate}} = 0 \end{cases} \tag{15}$$

$$R_{S3} = \begin{cases} 10 & \text{if } P_{\text{Finish}} = 1 \\ 0 & \text{if } P_{\text{Finish}} = 0 \end{cases} \tag{16}$$

These reward functions include progressive rewards (i.e., R_A, R_O, and R_E) and phase rewards (i.e., R_{S0}, R_{S1}, R_{S2}, and R_{S3}) to avoid sparse reward and local optimum issues, which are commonly

Table 1 Illustration of the three stages for each task

Task	Stage 1: Approaching	Stage 2: Manipulating	Stage 3: Executing
Pick and place	Approach object	Grab object	Move to place object
Drawer open	Approach handle	Grab handle	Move to pull drawer
Light switch	Approach switch	Grab switch	Move to turn on/off switch
Button press	Approach button	Touch button	Move to press button
Cube push	Approach cube	Touch cube	Move to push cube

utilized during RL training [41]. The progressive reward is dense reward which could provide gradient information of reward to help agent learn better during each stage. The phase rewards are sparse rewards and are only added when the agent achieves certain states, such as collision, gripper arriving at manipulation area, target object being gripped or touched, and robot's finishing task. The phase reward could help the agent avoid falling into local optimum. The λ_1 to λ_4 are positive coefficient parameters to adjust the weight of each reward.

In Stage 1 (Approaching), the main goal is to approach the target object. R_A will increase if the distance between gripper and target (D_{gt}) is reducing. P_{Arrive} means whether the gripper has arrived at the gripping area (e.g., $D_{\text{gt}} < 3$ mm). A phase reward R_{S1} will be added if the gripper arrives at manipulation area. R_O indicates the safety condition and will punish potential collision conditions (i.e., too close to the obstacle). If the minimum distance between robot and obstacle (D_{ro}) is among $\left[\frac{1}{2}D_F, D_F\right]$ (D_F is the safety distance threshold), the agent will get a punishment. If D_{ro} is smaller than $\frac{1}{2}D_F$, which means the probability of collision at the current state is high, a larger punishment will be added. R_{S0} is a collision reward penalizing collision state and will be invoked during all stages.

In Stage 2 (Manipulating), the gripper needs to grab target object (for Pick and Place, Drawer Open, Light Switch) or touch target object (for Button Press and Cube Push). A phase reward R_{S2} will be added if Stage 2 is finished. R_A and R_O are still invoked at this stage.

In Stage 3 (Executing), the agent is expected to arrive at the designated placing area to finish task. R_E will increase if the distance between gripper and placing area (D_{gp}) is reduced. A phase reward R_{S3} will be added if the task is finished. R_O is still invoked during this stage to avoid collision and R_A will be 0.

3.5 Refine Reward Function Parameters Based on Bayesian Optimization. To obtain better-performing RL models, we propose a method to refine the reward function parameters based on Bayesian Optimization. We adopt Gaussian Process Regression (GPR) and Expectation Improvement (EI) as the probabilistic model and acquisition function, respectively. GPR has high flexibility and scalability and is suitable for optimization problems without a large number of samples. Given samples X, Y and a new sample point x^*, the corresponding predicted distribution of the output y^* can be calculated:

$$p(y^*|X, Y, x^*) = N(m^*, \sigma^*) \tag{17}$$

where m^* and σ^* refer to mean value and variance, respectively. EI explores the regions that have the potential to enhance performance by calculating the expected increase in the objective function's value, as Eq. (18) [42] shows:

$$EI_{y^*}(x) = \int_{-\infty}^{+\infty} \max(y^* - y, 0) p_M(y|x) dy \tag{18}$$

where $EI_{y^*}(x)$ is an expectation of sample point x and y^* is a threshold value.

In this study, the RL reward function parameters to be optimized include D_T (switch distance for geometry-based method and RL-based method), D_S (the step size factor for RL action), and D_F (safety distance threshold). The optimization goal is to minimize the objective function Y_{BO} as follows:

$$Y_{\text{BO}} = \omega_1 R_{\text{success}} + \omega_2 R_{\text{overtime}} + \omega_3 R_{\text{collide}} \tag{19}$$

where R_{success} represents the success rate of task, R_{overtime} is task timeout rate, which refers to the rate of tasks not completed within the specified number of steps, and R_{collide} is the collision rate. These metrics serve as indicators of task performance. ω_1, ω_2, and ω_3 are weighting coefficients. Y_{BO} is the overall task performance metric, and a lower Y_{BO} indicates better model performance.

Figure 5 depicts the optimization process for refining RL reward function parameters. We first collect an initial BO set ($\boldsymbol{D_{T\_init}}, \boldsymbol{D_{S\_init}}, \boldsymbol{D_{F\_init}}, \boldsymbol{Y_{BO\_init}}$) that contains the evenly distributed reward function parameters and the corresponding BO objective function values obtained by training the RL model. This initial set is passed to the BO algorithm, which then generates the next set of reward function parameters ($D_{T\_\text{next}}, D_{S\_\text{next}}, D_{F\_\text{next}}$) that are more likely to produce better objective function value. The path planning model updates the reward function parameters and re-trains to acquire task performance metric values ($R_{\text{success}}, R_{\text{overtime}}, R_{\text{collide}}$). Subsequently, the new objective function value Y_{BO} is calculated and convergence is evaluated. The BO iteration terminates if the objective function value converges.

4 Experiment Settings and Results

To demonstrate and validate our approach, we built a high-fidelity digital twin model of a real robot arm and the physical working environment. Five commonly seen tasks in manufacturing scenarios (Pick and Place, Drawer Open, Light Switch, Button Press, and Cube Push) were constructed to evaluate the performance of the proposed robot motion planning approach. The experiment settings and results are presented in the following subsections.

4.1 Experiment Settings. Figure 6 shows the setting of the physical equipment and working space. A robot arm (JAKA Zu3, JAKA Robotics, Shanghai, China) equipped with an electric gripper (CTEK CTP2F50, Zhixing Robotics Technology (Suzhou) Co., Ltd., Suzhou, Jiangsu, China) is fixed to the table. The plane of the table is set as the plane with $z = 0$ in world coordinate system. The center of the robot arm base is set as the origin point (0, 0). An RGB-D camera (RealSense D435i, Intel Corporation, Santa Clara, CA) is deployed at (0.80, −0.50, and 0.50) to detect the obstacle and target object. The obstacle is a 3D-printed plate made of PLA material with a size of 10 cm × 10 cm × 0.3 cm, which is fixed to a linear rail and moves linearly with

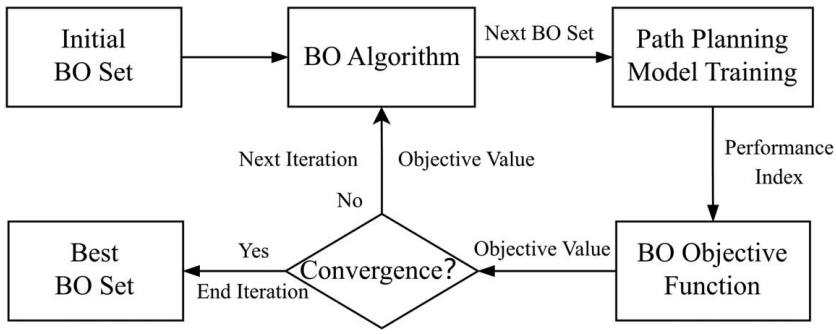

Fig. 5 Workflow of the Bayesian Optimization–based reward function parameter optimization method

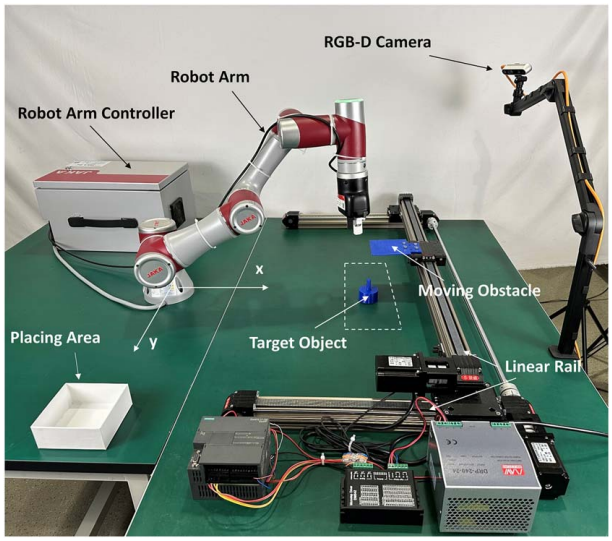

Fig. 6 Overall experiment setting

random initial positions and speeds (ranging from 5 to 10 cm/s). Its movement range is 30 cm along the x-axis and 80 cm along the y-axis. The target object of each task is placed randomly in a rectangular area (i.e., the white dashed box in Fig. 6) defined by the four vertices (0.35, 0.20), (0.35, −0.20), (0.45, 0.20), and (0.45, −0.20). In this area, the robot is highly likely to collide with obstacles for each task.

A high-fidelity digital twin model of the robot and the working environment is established in Unity3D as shown in Fig. 7. Besides visualizing these physical entities, DT provides the real-time status of the environment and robot (e.g., the poses of object and obstacle, the joint angle, and end effector pose of robot) for offline RL model training. Meanwhile, high-fidelity DT model is the foundation for collision detection which enables the adaptive switch of geometry-based or RL-based method in path planning. In addition, DT bridges the virtual environment and real entity, and the action sequence generated in virtual space could be transmitted to the physical robot controller for execution. The training of the proposed approach runs on a workstation with Intel Core i7-12700F CPU, 16 GB RAM, and RTX3080 GPU.

The hyperparameters of the RL model are chosen based on preliminary experiments. The batch size and buffer size of PPO are set as 512 and 10,240, respectively. The learning rate controls the training speed, which is set to 0.0003 to avoid reward divergence. The discount factor is $\gamma = 0.95$, and the linear learning rate schedule is adopted.

The conducted experiments are organized as follows. We first performed Bayesian Optimization to get a refined parameter configuration of RL reward functions in Sec. 4.2.1. Then, we compared the task performance of our approach with benchmark methods in a virtual environment, with a detailed analysis of each performance index presented in Sec. 4.2.2. In Sec. 4.2.3, our approach's robustness on reward magnitude was evaluated. Besides, the adaptability of our approach to more complex dynamic environments was further investigated in Sec. 4.2.4. Finally, physical experiments were conducted to validate the sim-to-real capability of the proposed approach in Sec. 4.2.5.

4.2 Experiment Results

4.2.1 Optimization of Reward Function Parameters. We first performed the Bayesian Optimization in a virtual environment with a one-dimensional moving obstacle (along the y-axis) to get refined reward function parameters for a better motion planning model. The weights of the BO objective function ω_1, ω_2, and ω_3 are set to −1, 0.5, and 5, respectively. The range of the parameters to be optimized is set as $D_T \in [0.1, 0.4]$ m, $D_F \in [0.1, 0.4]$ m, $D_S \in [0.02, 0.05]$. As for the RL model, $\lambda_1 = 0.3$, $\lambda_2 = 0.3$, $\lambda_3 = 0.1$, $\lambda_4 = 1$. These settings are designed based on our preliminary experimental results, which show that their influence on task performance is smaller compared to D_T, D_F, and D_S. Each round of training takes 5 million steps, and the episode length is set to 1500. The RL agent is trained for 5 million steps to enable the robot to complete five tasks (Pick and Place, Drawer Open, Light Switch, Button Press, and Cube Push) without collision. We first selected seven groups of parameters evenly distributed in the predefined ranges and got corresponding

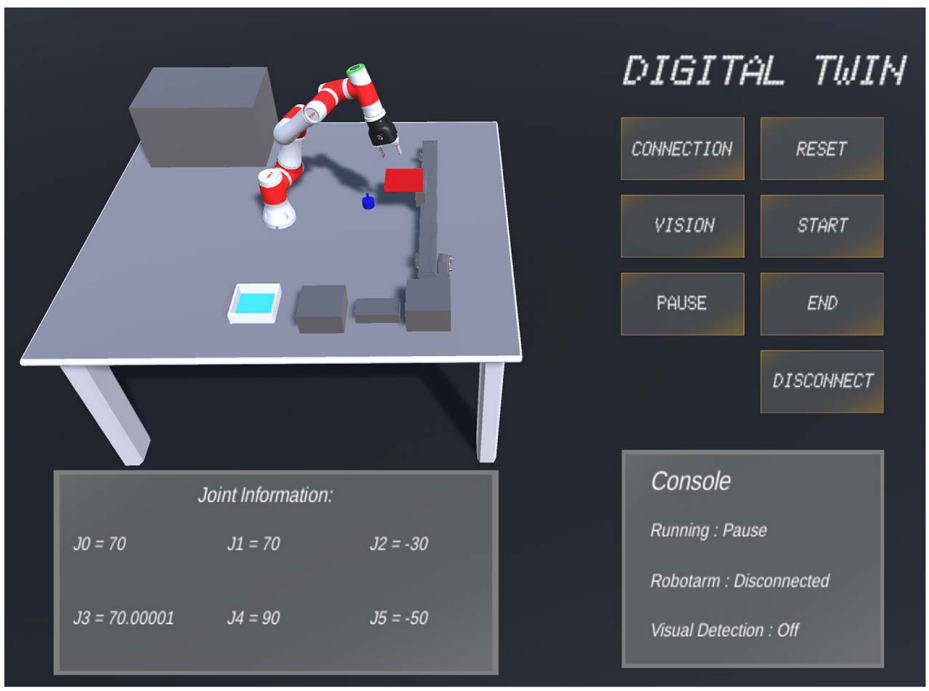

Fig. 7 The digital twin model of the robot and working environment with a user interface

BO objective function values. These initial samples were then fed into the BO algorithm to generate the next set of parameters and the training would be conducted subsequently. The optimization iteration would continue until the BO objective function value converges. The convergence criterion is defined such that the fluctuation of the BO objective function value in three successive iterations is within a range of 5% of the difference between the maximum and minimum objective function values. In order to evaluate the comprehensive performance of the RL model, the success rate, overtime rate, and collision rate in each BO iteration are averaged over five tasks.

The results of the BO iterations are shown in Fig. 8. Figure 8(*a*) shows the obtained parameter sets in each BO iteration, Fig. 8(*b*) exhibits the corresponding motion planning model performance of each model, and Fig. 8(*c*) presents the BO objective function values. From Fig. 8(*c*), we can see that the BO objective function values have large fluctuations in the initial 7 iterations. As the optimization process goes on, this value gradually gets smoother and converges at the 17th round. The 16th round shows the lowest BO objective function value, and the corresponding reward function parameters ($D_T = 0.29$ m, $D_F = 0.18$ m, $D_S = 0.037$) are selected as the final ones, leading to a model performance of $R_{\text{success}} = 98.2\%$, $R_{\text{overtime}} = 1.0\%$, and $R_{\text{collide}} = 0.8\%$. From Figs. 8(*a*) and 8(*b*), we can see that D_T (switch distance for geometry-based method and RL-based method) mainly influences the success rate and collision rate, and a smaller D_T may lead to lower success rate and higher collision rate. This is potentially because when the RL is invoked, the robot is too close to avoid collision in time with a small value of D_T. While the overtime rate mainly relates to D_F (safety distance threshold), a larger D_F will cause higher R_{overtime}. A possible reason is that the robot controlled by RL will stay too far away from the obstacle to get higher reward; thus, longer time is required to finish the task. Additionally, D_S (the step size factor for RL action) affects both the collision rate and motion smoothness of robot. A larger D_S provides the robot with broader action range to navigate around moving obstacle but may introduce jitter. Conversely, a smaller D_S enhances motion smoothness while potentially resulting in collisions if obstacle moves quickly for the robot to evade effectively. In summary, the proposed optimization process could generate a refined set of reward function parameters in relatively few iterations, providing an efficient paradigm to optimize the RL model for robot motion planning.

Based on the experimental results, we suggest the following guidelines for parameter tuning. To begin with, it is essential to get the reasonable testing ranges of these parameters based on prior knowledge, previous research, or preliminary tests. Then, experiments should be conducted to evaluate the impact of each parameter on the model performance. Parameters with insignificant impact on model performance can be determined through straightforward searching techniques such as grid search [43] or random search [44], while those key parameters can be efficiently fine-tuned using Bayesian Optimization to achieve better results.

4.2.2 Comparison Experiments to Evaluate the Performance of the Proposed Approach. To evaluate the performance of the proposed approach, we compare our approach (named Adaptive PPO) with other reinforcement learning-based methods, including PPO, Adaptive SAC, SAC, and previous work (Improved SAC [24], Improved PPO [26]). All of these methods were trained for 5 million steps on the five tasks with one-dimensional moving obstacles, which took approximately 2 h. After that, the model converged and could be effectively used for motion planning. Here, the PPO method solely relies on the PPO algorithm to control the robot throughout all tasks. The Adaptive SAC method employs the SAC algorithm [45], a type of commonly used actor-critic RL algorithm for continuous action space, and serves as a benchmark in many RL tasks. The SAC method only relies on the SAC algorithm without adaptively integrating the geometry-based method.

4.2.2.1 Comparison of convergence speed. The cumulative reward during the training process of different methods is illustrated in Fig. 9. We can see that the proposed adaptive PPO method achieves the highest cumulative reward, with the smoothest curve and fast convergence speed. Besides, adaptive methods (adaptive PPO, adaptive SAC) exhibit much faster convergence speed and significantly higher cumulative rewards than pure RL methods (PPO, SAC). This is possible as in the adaptive methods, the geometry-based method reduces the task difficulty by replacing RL in certain relatively safe areas (e.g., far away from the moving obstacles). Thus, RL only needs to handle part of the task which helps to learn faster and achieve better performance.

Furthermore, the adaptive methods and previous methods (improved SAC, improved PPO) achieve significant rewards, indicating that they all perform well in this experimental scenario. While the adaptive methods outperform the previous methods in convergence speed. We can also find there are some differences among adaptive methods. The reward curve of adaptive PPO is smoother, while adaptive SAC exhibits greater fluctuations. This discrepancy may arise from the SAC algorithm's utilization of a random sampling exploration strategy. When reaching a local optimal solution, it may lead to collisions with obstacles due to larger exploration ranges. Conversely, the deterministic strategy employed by the PPO algorithm facilitates a more stable exploration and utilization of known optimization strategies, resulting in smoother reward variation trends.

4.2.2.2 Comparison of task success rate and episode length. To further evaluate the model performance, the six methods mentioned earlier were tested 200 times for each task in a virtual environment. As shown in Table 2, the performance metrics of different methods include task success rate and the average task completion steps of the successful attempts. We can see that the proposed adaptive PPO method achieves the highest success rate in five tasks. For the relatively simple tasks (Drawer Open, Light Switch, Button Press, and Cube Push), our proposed method achieves a nearly 100% success rate. For the most challenging Pick and Place task, a success rate of 94.5% is also guaranteed, which outperforms other methods. Regarding task completion steps, adaptive PPO exhibits the lowest number of steps in the Pick and Place and Light Switch among all methods, and for the remaining three tasks, it is within 15% of the best-performing method.

In comparison, the performance of pure RL methods in terms of success rate and task completion steps is significantly worse without the adaptive combination of the geometry-based method. This is mainly because relying solely on RL necessitates more exploration and trials to search for a collision-free path, resulting in a higher collision rate under the same training steps. Moreover, with the use of pure RL-based methods, the robot agent is prone to entering a local optimum state (e.g., waiting above obstacles to avoid collision) to obtain a higher cumulative reward, thereby requiring more steps to complete the task.

4.2.2.3 Comparison of collision rate and overtime rate. To further explore the impact of the combination of geometry-based methods, we compared the collision rate and overtime rate across different methods, as shown in Fig. 10. It can be seen that the primary cause of task failure is overtime, meaning the robot failed to complete the task within the limited steps without colliding with obstacle. This outcome is predictable, as collisions result in a large penalty for the RL agent. Consequently, when the robot is in potentially dangerous areas (e.g., close to obstacle or when obstacle moves quickly), it tends to wait or evade to achieve a higher reward. Besides, the left figure shows that the collision rates for adaptive methods are much lower than those for pure RL methods. This may be attributed to the geometry-based method, which automatically plans the robot's path within safe areas, thereby reducing the burden on the RL algorithm. With the same number of training steps, the RL algorithm can effectively focus on learning obstacle avoidance strategies. In contrast, SAC and

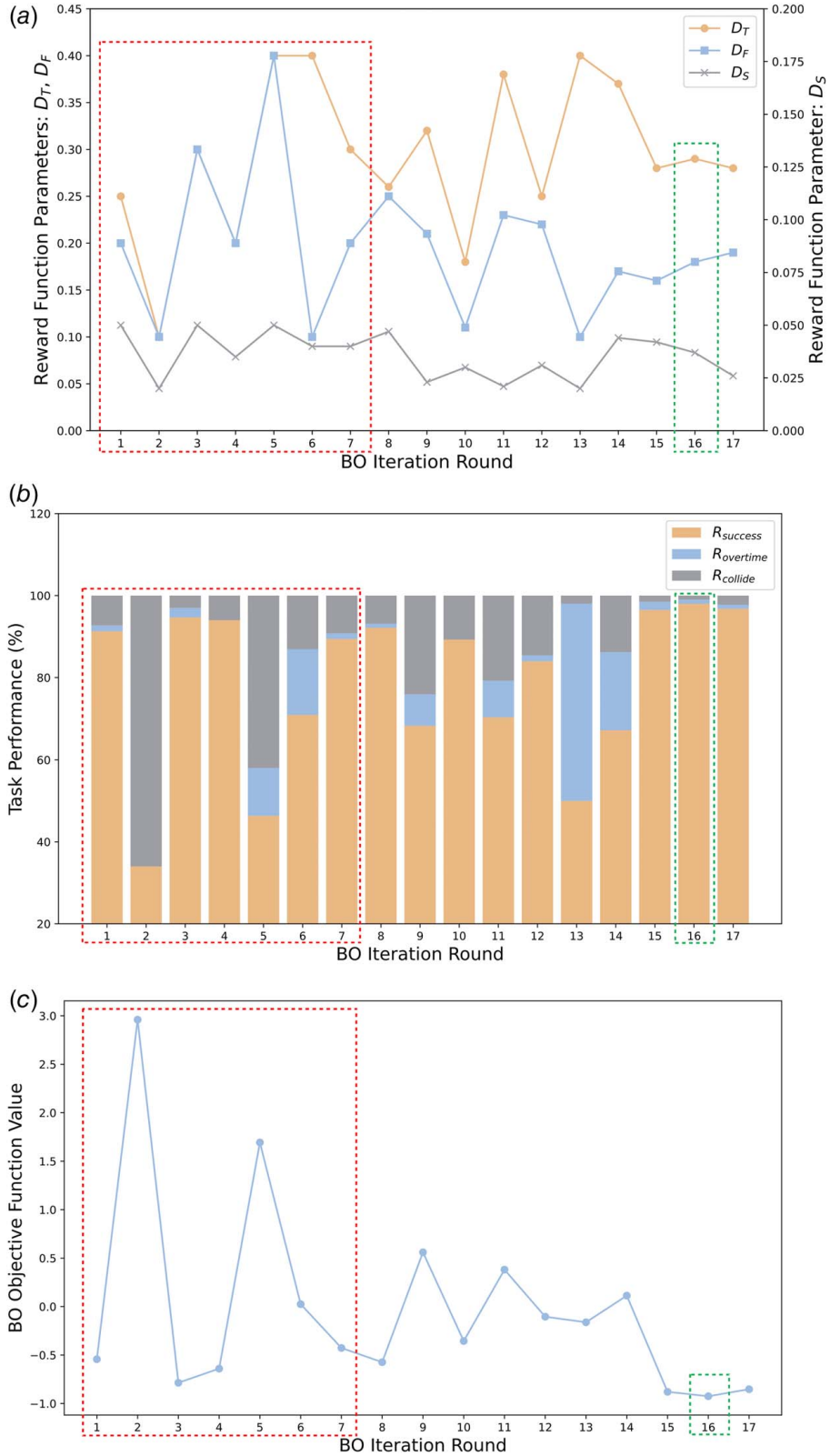

Fig. 8 Bayesian Optimization iteration results: (*a*) Reward function parameters of each iteration, (*b*) task performance of each iteration. Here, the results are the average values over five tasks. (*c*) Objective function value of each iteration. The dashed boxes on the left highlight the first seven iterations trained with initially selected parameters. The dashed boxes on the right show the 16th iteration with the best performance.

PPO are involved in the entire path-planning process, which may weaken their obstacle avoidance capabilities. From Fig. 10(*b*), we can also observe that the overtime rates for adaptive methods are lower than those for pure RL methods. There are two potential reasons for this: First, the geometry-based method plans straight paths within safe areas, which shorten the path length compared

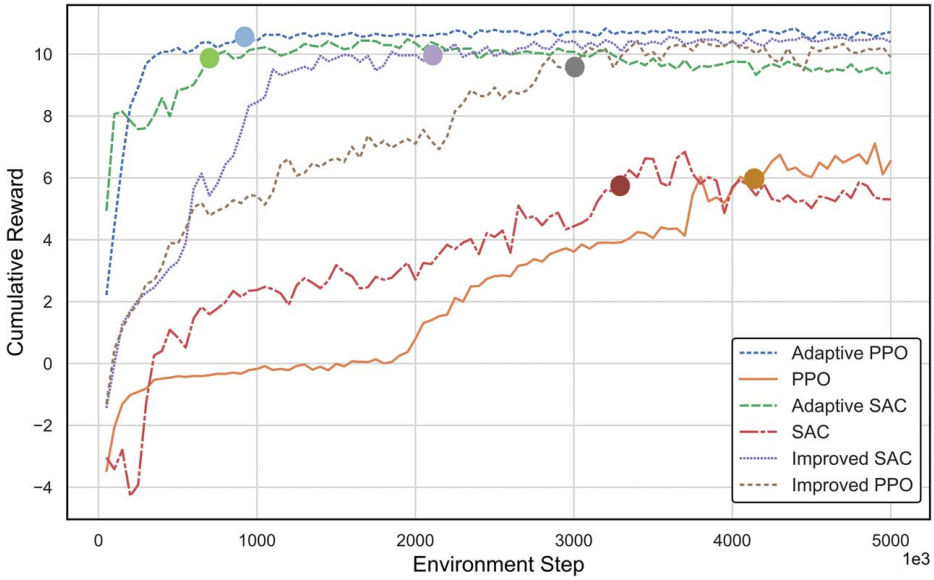

Fig. 9 Learning curve of the training process for different methods. The convergence step of each method is marked with circle dots.

Table 2 Success rate and average task completion steps with standard errors of different methods in comparison experiments

Task	Pick and place	Drawer open	Light switch	Button press	Cube push
Adaptive PPO	**94.5% ± 1.6%** **(586.7 ± 16.3)**	**99.5% ± 0.5%** (298.5 ± 8.6)	**97.5% ± 1.1%** **(323.5 ± 11.2)**	**98.0% ± 1.0%** (383.2 ± 12.1)	**98.5% ± 0.9%** (323.7 ± 9.7)
PPO	66.0% ± 3.4% (895.4 ± 25.2)	90.5% ± 2.1% (387.8 ± 13.8)	74.5% ± 3.1% (532.4 ± 14.9)	77.0% ± 3.0% (566.5 ± 18.4)	94.5% ± 1.6% (390.2 ± 11.3)
Adaptive SAC	87.5% ± 2.3% (643.5 ± 20.1)	96.0% ± 1.4% (360.7 ± 13.2)	94.5% ± 1.6% (348.6 ± 10.5)	94.0% ± 1.7% (356.5 ± 13.5)	92.0% ± 1.9% (382.8 ± 12.2)
SAC	53.0% ± 3.5% (950.4 ± 32.9)	80.5% ± 2.8% (495.2 ± 16.5)	66.5% ± 3.3% (618.7 ± 18.3)	70.5% ± 3.2% (586.0 ± 15.8)	79.5% ± 2.9% (481.9 ± 15.6)
Improved SAC (Chen et al. [24])	81.5% ± 2.8% (683.5 ± 23.0)	99.0% ± 0.7% **(267.3 ± 7.7)**	88.5% ± 2.3% (461.8 ± 15.7)	96.5% ± 1.3% (498.8 ± 14.9)	92.5% ± 1.9% (319.5 ± 13.4)
Improved PPO (Luipers et al. [26])	70.5% ± 3.2% (672.8 ± 22.4)	92.5% ± 1.9% (333.5 ± 10.2)	95.5% ± 1.5% (512.2 ± 15.4)	91.0% ± 2.0% **(343.6 ± 10.0)**	96.0% ± 1.4% **(299.4 ± 9.7)**

Note: The bolded numbers represent the best performance.

to the autonomous exploration by RL algorithms. Second, the adaptive methods keep the robot stay at the boundary of safe zones when it is close to the obstacle, while pure RL methods may adopt overly cautious strategies to evade obstacle, thereby increasing the path length. These results indicate that the proposed adaptive methods outperform traditional RL algorithms in reducing task overtime rate and collision rate. By combining the advantages of geometric methods and RL algorithms, the efficiency and safety of robot path planning are enhanced.

4.2.3 Comparison Experiment to Evaluate the Robustness on Reward Magnitude of the Proposed Approach. In order to

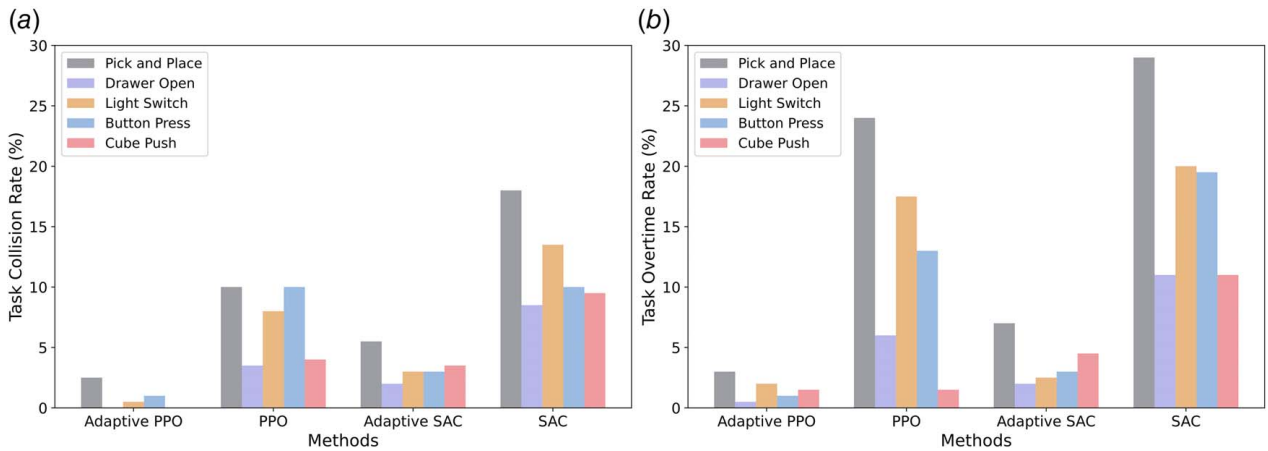

Fig. 10 Task performance comparison on (*a*) collision rate and overtime rate

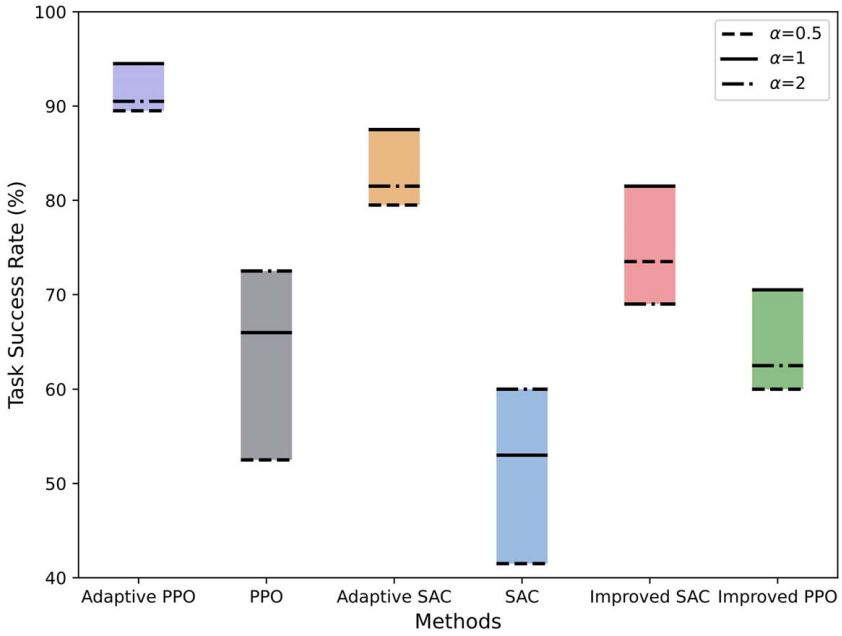

Fig. 11 Comparison of success rate for Pick and Place across different reward factors

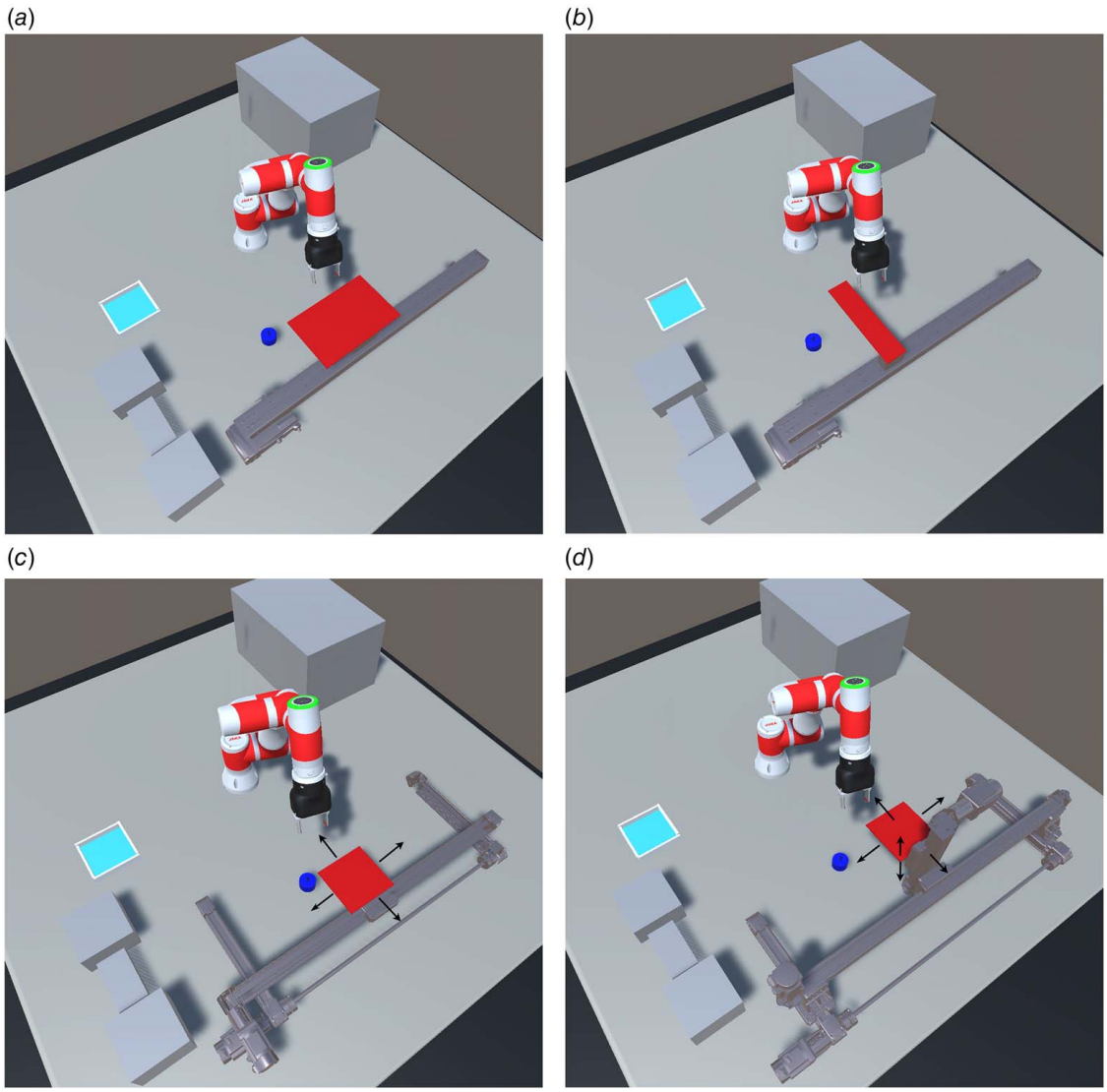

Fig. 12 Experiment settings on (*a*) obstacle plate, (*b*) obstacle bar, (c) 2D obstacle and (*d*) 3D obstacle

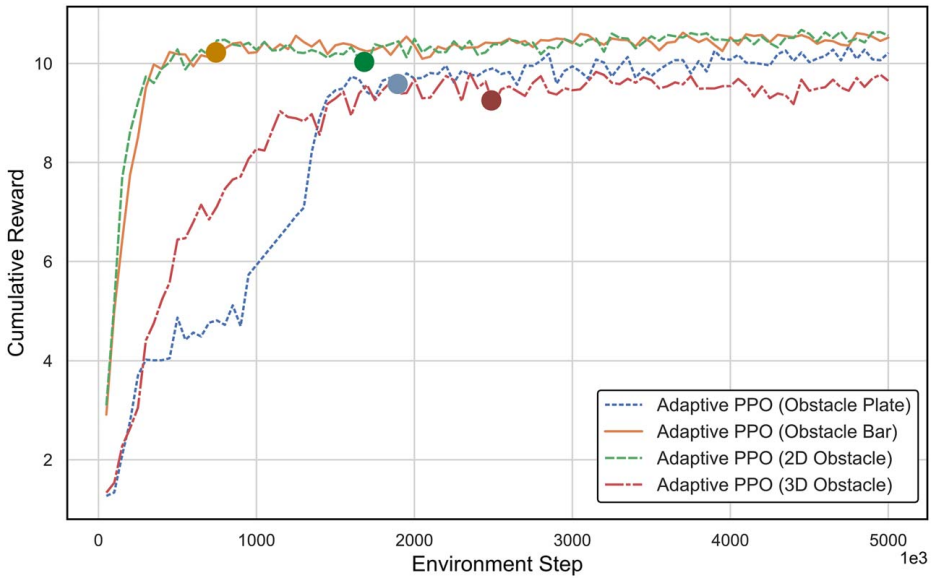

Fig. 13 Learning curve of the training process in obstacle plate, obstacle bar, 2D and 3D obstacle scenarios. The convergence step of each method is marked with circle dots.

Table 3 Success rate and average task completion steps with standard errors of adaptive PPO method on different scenarios

Task	Pick and place	Drawer open	Light switch	Button press	Cube push
Obstacle plate	89.5% ± 2.2% (640.5 ± 19.2)	99.0% ± 0.7% (363.7 ± 11.3)	95.0% ± 1.5% (353.3 ± 12.9)	93.5% ± 1.7% (361.3 ± 10.2)	96.5% ± 1.3% (433.1 ± 14.4)
Obstacle bar	93.0% ± 1.8% (618.4 ± 17.6)	96.5% ± 1.3% (398.9 ± 11.6)	94.0% ± 1.7% (362.3 ± 15.2)	96.5% ± 1.3% (377.3 ± 13.3)	97.0% ± 1.2% (412.9 ± 14.9)
2D Obstacle	91.5% ± 2.0% (622.7 ± 18.5)	96.0% ± 1.4% (406.3 ± 12.7)	93.5% ± 1.7% (345.0 ± 13.1)	95.0% ± 1.5% (417.9 ± 12.9)	97.5% ± 1.1% (423.5 ± 15.1)
3D Obstacle	89.5% ± 2.2% (640.2 ± 18.7)	92.5% ± 1.9% (428.1 ± 13.2)	90.5% ± 2.1% (334.4 ± 14.8)	95.5% ± 1.5% (445.7 ± 13.4)	93.0% ± 1.8% (459.6 ± 15.0)

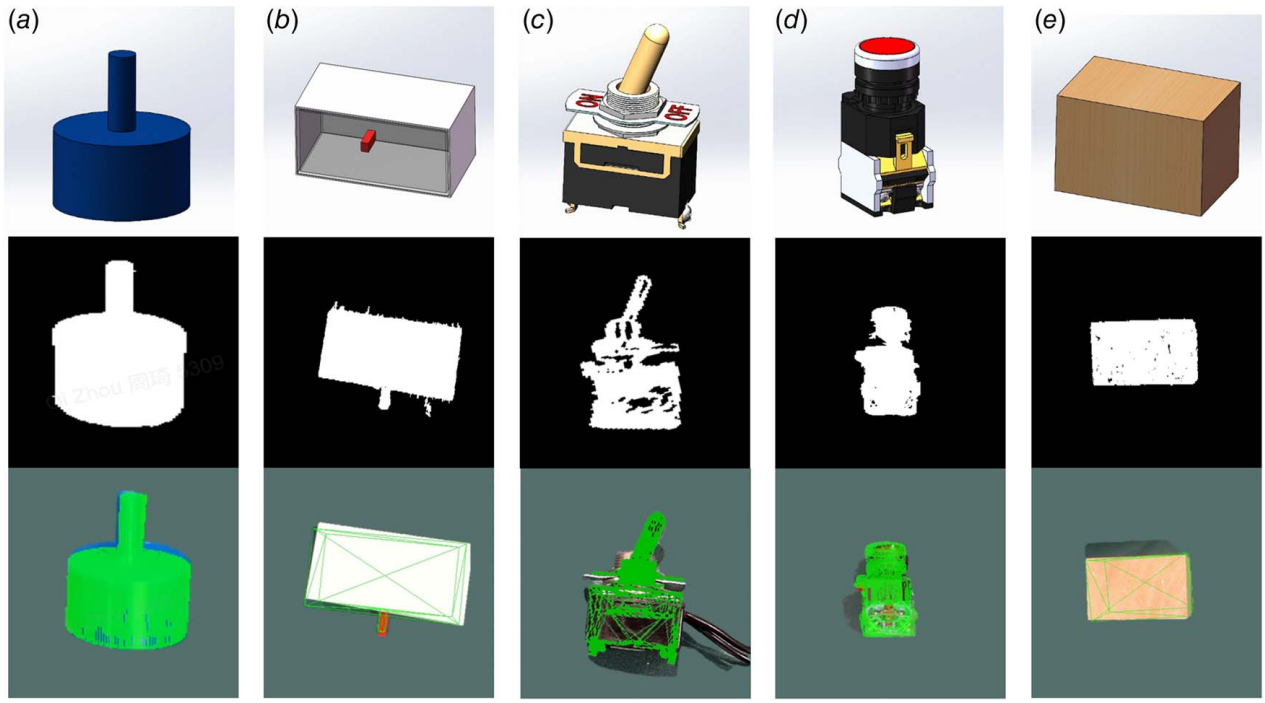

Fig. 14 Object detection results. The first row shows the 3D model of target object for each task (Pick and Place, Drawer Open, Light Switch, Button Switch, and Cube Push). The second and third rows refer to their mask images and calculated poses of these objects. The bicylinder is a 3D-printed object made of PLA material.

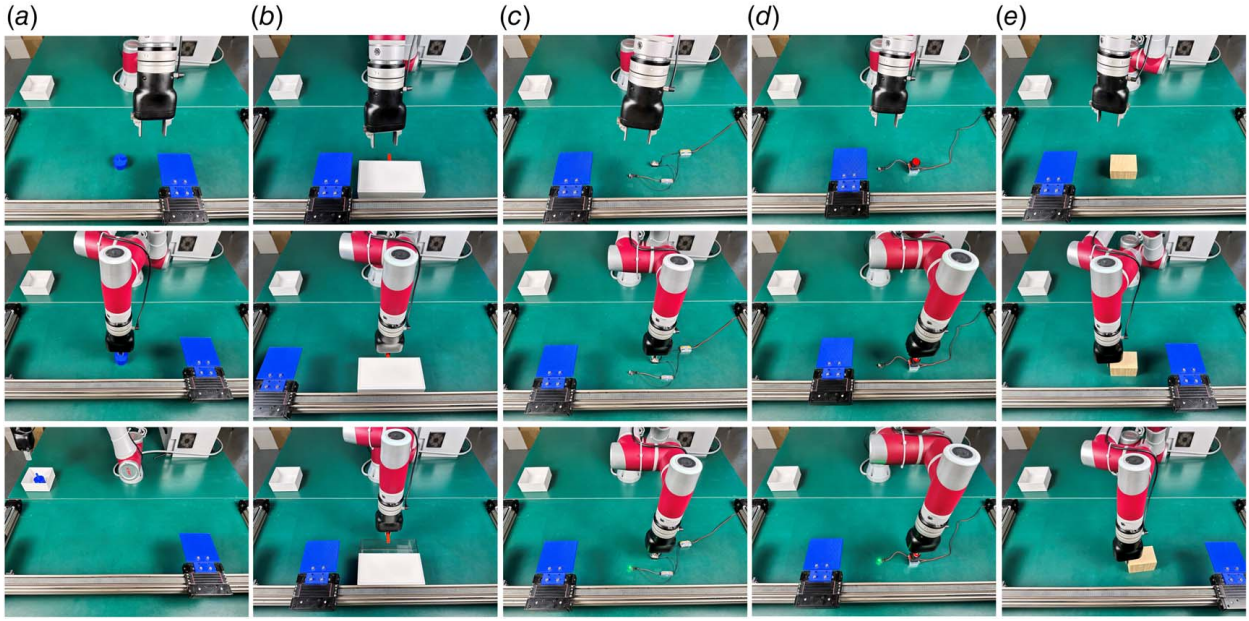

Fig. 15 Physical scenes of five tasks. Each column shows initial, intermediate, and completion states of each task.

explore the impact of reward magnitude on the performance of the proposed method, a reward factor α is introduced to control the magnitude of reward. In each training step, reward $r(s)$ was multiplied by α. We applied different values of α ([0.5, 1.2]) to the six methods mentioned in Table 2 and retrained the RL model. To visually illustrate the impact of reward factors on task performance, we tested different methods on the most challenging Pick and Place task. The success rates of each method are shown in Fig. 11.

We can see that the success rate of the adaptive PPO method remains consistently high (ranging from 89.5% to 94.5%) across different reward factors and exhibits minimal fluctuations. Conversely, the performance of PPO method and SAC method exhibits significant fluctuations, with success rates ranging from 52.5% to 72.5% and 41.5% to 60.0%, respectively. This could be possibly attributed to the stable nature of the geometry-based method, whose integration with RL-based method can enhance the robustness of the proposed approach to reward magnitude, consequently improving the stability and reliability of motion planning performance. Additionally, we find that the fluctuation range of Adaptive SAC (ranging from 79.5% to 87.5%) is slightly larger than that of Adaptive PPO. This may be due to the SAC algorithm's training objective, which not only maximizes the reward but also takes the policy's entropy into account. Therefore, a precise and stable reward function is required to guide policy learning. In contrast, PPO maintains stability by limiting the extent of policy update, making it less sensitive to the reward magnitude.

4.2.4 Comparison Experiments to Evaluate the Performance on Different Dynamic Scenarios. To further explore the adaptability of the proposed method in more complex scenarios, we added tests with two different obstacle shapes, the obstacle plate and obstacle bar, as shown in Figs. 12(*a*) and 12(*b*). Both shapes are commonly encountered in manufacturing settings. For example, in power battery assembly scenarios, the brackets used to support battery modules are typically plates, while the mounting parts providing structural stability or serving as alignment guides are usually rods. Besides, the movement dimensions of obstacle were also extended to 2D and 3D, as shown in Figs. 12(*c*) and 12(*d*). The obstacle plate and bar are planes with sizes of $20\,\text{cm} \times 10\,\text{cm} \times 0.3\,\text{cm}$ and $5\,\text{cm} \times 20\,\text{cm} \times 0.3\,\text{cm}$, respectively. The 2D moving obstacles had travel distances of 30 cm and 80 cm along the X and Y axes, respectively, at moving speeds ranging from 5 to 10 cm/s. In the 3D scenario, an additional 20-cm travel along the Z-axis was introduced. We trained the adaptive PPO approach in these environments, keeping the reward function parameters consistent with those of the 1D obstacle environment. The training reward curves are depicted in Fig. 13. We can find that the proposed method achieved high cumulative rewards in all scenarios. However, the model's performance for the obstacle plate experiences a slight decline. This is primarily due to the larger surface area of the obstacle plate, which increases the difficulty for the robot to complete the task during repetitive movements. Besides, as the dimensionality of obstacle movement increases, the convergence speed decreases. This is likely attributed to the expanded state space resulting from higher movement dimensions, requiring the RL agent to explore more state-action combinations.

Each RL model was tested 200 times on five tasks, and the results are summarized in Table 3. We can see that the proposed method maintains good performance in more complex dynamic environments, indicating its adaptability to various dynamic environments.

Table 4 Task performance of adaptive PPO method in real environment

	Pick and place		Drawer open		Light switch		Button press		Cube push	
Task	1D	2D	1D	2D	1D	2D	1D	2D	1D	2D
Success rate	27/30	26/30	29/30	29/30	29/30	27/30	28/30	28/30	30/30	29/30
Overtime rate	3/30	3/30	1/30	1/30	0/30	2/30	1/30	2/30	0/30	1/30
Collision rate	0/30	1/30	0/30	0/30	1/30	1/30	1/30	0/30	0/30	0/30
Completion time (s)	11.7	12.5	6.0	8.1	6.5	6.9	7.7	8.4	6.4	8.5

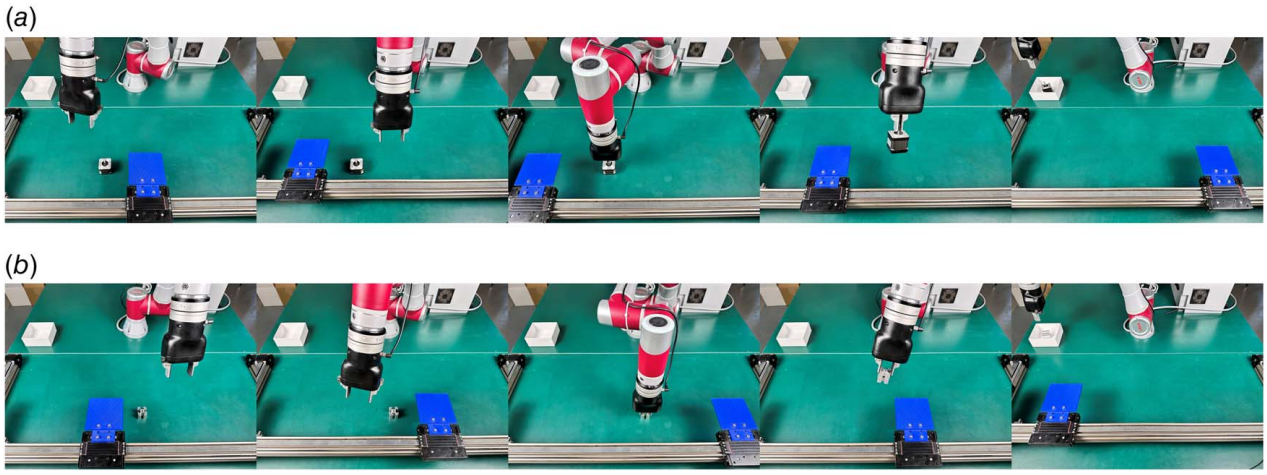

Fig. 16 Physical scenes of Pick and Place tasks for stepping motor and aluminum profile. The figure from left to right shows the obstacle avoidance process of the proposed method.

4.2.5 Generalization Performance in Physical Environment. In order to verify the practicality of the proposed adaptive motion planning approach, we also conducted physical experiments. Figure 14 depicts the 3D models, mask images, and calculated poses of target objects for each task. Figure 15 shows the physical scenes of five tasks in a dynamic environment. The visual positioning accuracy of the obstacle and target object is within ±2 mm, and the environmental update frequency in the digital twin is 12 Hz, while the real-time control rate of the robot's actions in our experiments is 20 Hz. We conducted 30 physical tests for each task in 1D and 2D obstacle environments. The task performance is shown in Table 4. We can see that the proposed adaptive motion planning method exhibits a good performance in real environment.

To validate the generalizability of the proposed method when facing different target objects, the original target object is replaced with a stepping motor and aluminum profile in the Pick and Place task, and the task process is illustrated in Fig. 16. The proposed method achieved good success rates of 25/30 and 26/30, respectively, in the 2D obstacle environment. The failures mainly resulted from task overtime, which occurred when the robot hesitated to grab the target due to the fast-moving obstacles.

5 Conclusion

In this study, an adaptive robot motion planning approach for smart manufacturing is based on digital twin and reinforcement learning. This approach can adaptively select a geometry-based method or RL-based method according to the real-time distance detection result. Our approach integrates the fast calculation speed and high stability of the geometry-based method, and the flexible collision avoidance ability of RL-based method. Bayesian Optimization is leveraged to enable fast acquisition of optimal reward function parameters for RL model training. In addition, a high-fidelity digital twin model of the robot arm and dynamic environment is built to support the model training in five typical scenarios in smart material transportation and assembly operations.

Compared to traditional reinforcement learning-based methods, our approach reduces the complexity of RL model training with enhanced robustness to reward magnitude. Moreover, it ensures reliable collision avoidance performance in different dynamic manufacturing environments. Experiments show that the proposed approach exhibits well adaptivity to various RL algorithms, tasks, and dynamic scenarios. Our work has the potential to facilitate the practical implementation of reinforcement learning-based robot motion planning in various smart manufacturing scenarios, such as robot-assisted label attaching, screwing, polishing, painting, welding, and gluing operations. For example, in the assembly of power battery packs, due to the demand for diverse power battery products, the working environments for robots are frequently changed. Our approach has the potential to handle changeovers between different battery modules and configurations. During the alignment and placement of battery modules, the robot can navigate around other moving robots or assembly parts to achieve smooth transporting operations. In screwing operations, the robot can adjust its motion trajectory to avoid collisions with different fixtures or tools entering the workspace. By providing an efficient and adaptable motion planning framework for robotic operations in varying production environments, our approach can further enhance the operational efficiency and minimize the production downtime for advanced flexible manufacturing.

One limitation of this work is that the obstacle in experiments may not fully reflect the complex dynamic scenarios in real world. Future research will broaden the scope to include more complex obstacles, such as other robots and human bodies. Dynamic working scenarios with multiple obstacles will be introduced to further improve the practicality of the proposed approach.

Acknowledgment

The authors would like to acknowledge the financial support from the National Key R&D Program of China (2022YFB4702400).

Conflict of Interest

There are no conflicts of interest.

Data Availability Statement

The datasets generated and supporting the findings of this article are obtainable from the corresponding author upon reasonable request.

Nomenclature

$\boldsymbol{a}$ = RL action
$\mathcal{C}$ = configuration space
$\boldsymbol{s_t}$ = RL state
D_{gp} = distance between gripper and placing area
D_{gt} = distance between gripper and target object
D_n = Gaussian noise
D_{ro} = minimum distance between robot and obstacles
D_F = safety distance threshold

D_S = action step size factor
D_T = switch distance
R_{collide} = collision rate
R_{overtime} = timeout rate
R_{success} = success rate
Y_{BO} = objective function value of BO
$\mathcal{C}_{\text{free}}$ = obstacle-free space
$\boldsymbol{P}_g$ = pose of gripper
$\boldsymbol{P}_o$ = pose of obstacle
$\boldsymbol{P}_p$ = position of placing area
$\boldsymbol{P}_t$ = pose of target object
$\boldsymbol{q}(x_{\text{goal}})$ = goal state
$\boldsymbol{q}(x_{\text{init}})$ = initial state
$r(s)$ = RL reward
α = reward factor
$\lambda_1, \lambda_2, \lambda_3, \lambda_4$ = coefficient of reward function
$\boldsymbol{\sigma}$ = collision-free path
$\omega_1, \omega_2, \omega_3$ = weighting coefficients for BO objective function

References

[1] Flowers, J., and Wiens, G., 2023, "A Spatio-Temporal Prediction and Planning Framework for Proactive Human–Robot Collaboration," ASME J. Manuf. Sci. Eng., **145**(12), p. 121011.
[2] Park, J., Han, C., Jun, M. B. G., and Yun, H., 2023, "Autonomous Robotic Bin Picking Platform Generated From Human Demonstration and YOLOv5," ASME J. Manuf. Sci. Eng., **145**(12), p. 121006.
[3] Fan, J., Zheng, P., and Lee, C. K. M., 2023, "A Vision-Based Human Digital Twin Modeling Approach for Adaptive Human–Robot Collaboration," ASME J. Manuf. Sci. Eng., **145**(12), p. 121002.
[4] Ma, X., Qi, Q., and Tao, F., 2024, "A Digital Twin–Based Environment-Adaptive Assignment Method for Human–Robot Collaboration," ASME J. Manuf. Sci. Eng., **146**(3), p. 031004.
[5] Jafarzadeh, H., and Fleming, C. H., 2018, "An Exact Geometry–Based Algorithm for Path Planning," Int. J. Appl. Math. Comput. Sci., **28**(3), pp. 493–504.
[6] Rasekhipour, Y., Khajepour, A., Chen, S.-K., and Litkouhi, B., 2017, "A Potential Field-Based Model Predictive Path-Planning Controller for Autonomous Road Vehicles," IEEE Trans. Intell. Transp. Syst., **18**(5), pp. 1255–1267.
[7] Gul, F., Mir, I., Alarabiat, D., Alabool, H. M., Abualigah, L., and Mir, S., 2022, "Implementation of Bio-Inspired Hybrid Algorithm with Mutation Operator for Robotic Path Planning," J. Parallel Distrib. Comput., **169**, pp. 171–184.
[8] Karur, K., Sharma, N., Dharmatti, C., and Siegel, J. E., 2021, "A Survey of Path Planning Algorithms for Mobile Robots," Vehicles, **3**(3), pp. 448–468.
[9] LaValle, S. M., and Kuffner Jr, J. J., 2001, "Randomized Kinodynamic Planning," Int. J. Rob. Res., **20**(5), pp. 378–400.
[10] Elbanhawi, M., and Simic, M., 2014, "Sampling-Based Robot Motion Planning: A Review," IEEE Access, **2**, pp. 56–77.
[11] Wang, B., Liu, Z., Li, Q., and Prorok, A., 2020, "Mobile Robot Path Planning in Dynamic Environments Through Globally Guided Reinforcement Learning," IEEE Robot. Autom. Lett., **5**(4), pp. 6932–6939.
[12] Waseem, M., and Chang, Q., 2023, "Adaptive Mobile Robot Scheduling in Multiproduct Flexible Manufacturing Systems Using Reinforcement Learning," ASME J. Manuf. Sci. Eng., **145**(12), p. 121005.
[13] Xiao, J., Gao, J., Anwer, N., and Eynard, B., 2023, "Multi-Agent Reinforcement Learning Method for Disassembly Sequential Task Optimization Based on Human–Robot Collaborative Disassembly in Electric Vehicle Battery Recycling," ASME J. Manuf. Sci. Eng., **145**(12), p. 121001.
[14] Matulis, M., and Harvey, C., 2021, "A Robot Arm Digital Twin Utilising Reinforcement Learning," Comput. Graph., **95**, pp. 106–114.
[15] Li, T., Lambert, N., Calandra, R., Meier, F., and Rai, A., 2020, "Learning Generalizable Locomotion Skills With Hierarchical Reinforcement Learning," Proceedings of the 2020 IEEE International Conference on Robotics and Automation (ICRA), Paris, France, May 31–Aug. 31, pp. 413–419.
[16] Adiyatov, O., and Varol, H. A., 2017, "A Novel RRT*-Based Algorithm for Motion Planning in Dynamic Environments," Proceedings of the 2017 IEEE International Conference on Mechatronics and Automation (ICMA), Takamatsu, Japan, Aug. 6–9, pp. 1416–1421.
[17] Chen, Y., Liu, M., and Wang, L., 2018, "RRT* Combined With GVO for Real-Time Nonholonomic Robot Navigation in Dynamic Environment," Proceedings of the 2018 IEEE International Conference on Real-Time Computing and Robotics (RCAR), Kandima, Maldives, Aug. 1–5, pp. 479–484.
[18] Qi, J., Yang, H., and Sun, H., 2021, "MOD-RRT*: A Sampling-Based Algorithm for Robot Path Planning in Dynamic Environment," IEEE Trans. Ind. Electron., **68**(8), pp. 7244–7251.
[19] Tamizi, M. G., Yaghoubi, M., and Najjaran, H., 2023, "A Review of Recent Trend in Motion Planning of Industrial Robots," Int. J. Intell. Robot. Appl., **7**(2), pp. 253–274.
[20] Wang, X., Fu, H., Deng, G., Liu, C., Tang, K., and Chen, C., 2023, "Hierarchical Free Gait Motion Planning for Hexapod Robots Using Deep Reinforcement Learning," IEEE Trans. Ind. Inform., **19**(11), pp. 10901–10912.
[21] Zhang, Y., and Chen, P., 2023, "Path Planning of a Mobile Robot for a Dynamic Indoor Environment Based on an SAC-LSTM Algorithm," Sensors, **23**(24), p. 9802.
[22] Schmitt, P. S., Wirnshofer, F., Wurm, K. M., Wichert, G. V., and Burgard, W., 2019, "Planning Reactive Manipulation in Dynamic Environments," Proceedings of the 2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), IEEE, Macau, China, Nov. 4–8, IEEE, pp. 136–143.
[23] Chai, R., Niu, H., Carrasco, J., Arvin, F., Yin, H., and Lennox, B., 2024, "Design and Experimental Validation of Deep Reinforcement Learning-Based Fast Trajectory Planning and Control for Mobile Robot in Unknown Environment," IEEE Trans. Neural Netw. Learn. Syst., **35**(4), pp. 5778–5792.
[24] Chen, P., Pei, J., Lu, W., and Li, M., 2022, "A Deep Reinforcement Learning Based Method for Real-Time Path Planning and Dynamic Obstacle Avoidance," Neurocomputing, **497**, pp. 64–75.
[25] Zhou, Q., Li, S., Qu, J., Wu, J., Xu, H., and Bi, Y., 2023, "An Adaptive Path Planning Approach for Digital Twin-Enabled Robot Arm Based on Inverse Kinematics and Deep Reinforcement Learning," Proceedings of the 2023 ASME International Mechanical Engineering Congress and Exposition, New Orleans, LA, Oct. 29–Nov. 2, p. V003T03A079.
[26] Luipers, D., Kaulen, N., Chojnowski, O., Schneider, S., Richert, A., and Jeschke, S., 2022, "Robot Control Using Model-Based Reinforcement Learning With Inverse Kinematics," Proceedings of the 2022 IEEE International Conference on Development and Learning (ICDL), London, UK, Sept. 12–15, IEEE, pp. 244–249.
[27] Zhong, J., Wang, T., and Cheng, L., 2022, "Collision-Free Path Planning for Welding Manipulator via Hybrid Algorithm of Deep Reinforcement Learning and Inverse Kinematics," Complex Intell. Syst., **8**(3), pp. 1899–1912.
[28] Li, X., Liu, H., and Dong, M., 2022, "A General Framework of Motion Planning for Redundant Robot Manipulator Based on Deep Reinforcement Learning," IEEE Trans. Ind. Inform., **18**(8), pp. 5253–5263.
[29] Faust, A., Ramirez, O., Fiser, M., Oslund, K., Francis, A., Davidson, J., and Tapia, L., 2018, "PRM-RL: Long-Range Robotic Navigation Tasks by Combining Reinforcement Learning and Sampling-Based Planning," Proceedings of the 2018 IEEE International Conference on Robotics and Automation (ICRA), Brisbane, QLD, Australia, May 21–25, pp. 5113–5120.
[30] Arora, S., and Doshi, P., 2021, "A Survey of Inverse Reinforcement Learning: Challenges, Methods and Progress," Artif. Intell., **297**, p. 103500.
[31] Kim, B., and Pineau, J., 2016, "Socially Adaptive Path Planning in Human Environments Using Inverse Reinforcement Learning," Int. J. Soc. Robot., **8**(1), pp. 51–66.
[32] Tucker, A., Gleave, A., and Russell, S., 2018, "Inverse Reinforcement Learning for Video Games." arXiv preprint arXiv:1810.10593, 2018.
[33] Frazier, P. I., 2018, "A Tutorial on Bayesian Optimization." arXiv preprint arXiv:1807.02811.
[34] Wilson, A., Fern, A., and Tadepalli, P., 2014, "Using Trajectory Data to Improve Bayesian Optimization for Reinforcement Learning," J. Mach. Learn. Res., **15**(1), pp. 253–282.
[35] Young, M. T., Hinkle, J. D., Kannan, R., and Ramanathan, A., 2020, "Distributed Bayesian Optimization of Deep Reinforcement Learning Algorithms," J. Parallel Distrib. Comput., **139**, pp. 43–52.
[36] Gong, S., Wang, M., Gu, B., Zhang, W., Hoang, D. T., and Niyato, D., 2023, "Bayesian Optimization Enhanced Deep Reinforcement Learning for Trajectory Planning and Network Formation in Multi-UAV Networks," IEEE Trans. Veh. Technol., **72**(8), pp. 10933–10948.
[37] Cai, D., Heikkia, J., and Rahtu, E., 2022, "OVE6D: Object Viewpoint Encoding for Depth-Based 6D Object Pose Estimation," Proceedings of the 2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), New Orleans, LA, June 19–24, IEEE, pp. 6793–6803.
[38] Denavit, J., and Hartenberg, R. S., 1955, "A Kinematic Notation for Lower-Pair Mechanisms Based on Matrices," J. Appl. Mech., **22**(2), pp. 215–221.
[39] Littman, M. L., 1994, "Markov Games as a Framework for Multi-Agent Reinforcement Learning," Proceedings of the 1994 Machine Learning Proceedings, San Francisco, CA, July 10–13, pp. 157–163.
[40] Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O., 2017, "Proximal Policy Optimization Algorithms." arXiv preprint arXiv:1707.06347.
[41] Li, J., Pang, D., Zheng, Y., Guan, X., and Le, X., 2022, "A Flexible Manufacturing Assembly System With Deep Reinforcement Learning," Control Eng. Pract., **118**, p. 104957.
[42] Shahriari, B., Swersky, K., Wang, Z., Adams, R. P., and De Freitas, N., 2016, "Taking the Human Out of the Loop: A Review of Bayesian Optimization," Proc. IEEE, **104**(1), pp. 148–175.
[43] Belete, D. M., and Huchaiah, M. D., 2022, "Grid Search in Hyperparameter Optimization of Machine Learning Models for Prediction of HIV/AIDS Test Results," Int. J. Comput. Appl., **44**(9), pp. 875–886.
[44] Bergstra, J., and Bengio, Y., 2012, "Random Search for Hyper-Parameter Optimization," J. Mach. Learn. Res., **13**(2), pp. 281–305.
[45] Haarnoja, T., Zhou, A., Abbeel, P., and Levine, S., 2018, "Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning With a Stochastic Actor," Proceedings of the 2018 International Conference on Machine Learning (PMLR), Stockholm, Sweden, July 10–15, pp. 1861–1870.