frontiers | Frontiers in Neurorobotics

TYPE Original Research
PUBLISHED 28 March 2024
DOI 10.3389/fnbot.2024.1375309



OPEN ACCESS

EDITED BY
Rui Li,
Chinese Academy of Sciences (CAS), China

REVIEWED BY
Lin Hong,
Technical University of Munich, Germany
Hu Cao,
Technical University of Munich, Germany

*CORRESPONDENCE
Kai Huang
✉ huangk36@mail.sysu.edu.cn

RECEIVED 23 January 2024
ACCEPTED 11 March 2024
PUBLISHED 28 March 2024

CITATION
Hong T, Li W and Huang K (2024) A reinforcement learning enhanced pseudo-inverse approach to self-collision avoidance of redundant robots. *Front. Neurorobot.* 18:1375309. doi: 10.3389/fnbot.2024.1375309



# A reinforcement learning enhanced pseudo-inverse approach to self-collision avoidance of redundant robots

Tinghe Hong, Weibing Li and Kai Huang*

School of Computer Science and Engineering, Sun Yat-sen University, Guangzhou, China

**Introduction:** Redundant robots offer greater flexibility compared to non-redundant ones but are susceptible to increased collision risks when the end-effector approaches the robot's own links. Redundant degrees of freedom (DoFs) present an opportunity for collision avoidance; however, selecting an appropriate inverse kinematics (IK) solution remains challenging due to the infinite possible solutions.

**Methods:** This study proposes a reinforcement learning (RL) enhanced pseudo-inverse approach to address self-collision avoidance in redundant robots. The RL agent is integrated into the redundancy resolution process of a pseudo-inverse method to determine a suitable IK solution for avoiding self-collisions during task execution. Additionally, an improved replay buffer is implemented to enhance the performance of the RL algorithm.

**Results:** Simulations and experiments validate the effectiveness of the proposed method in reducing the risk of self-collision in redundant robots.

**Conclusion:** The RL enhanced pseudo-inverse approach presented in this study demonstrates promising results in mitigating self-collision risks in redundant robots, highlighting its potential for enhancing safety and performance in robotic systems.

KEYWORDS

**reinforcement learning, inverse kinematics, redundant robots, self-collision avoidance, sim to real**

## 1 Introduction

Kinematically redundant robots possess more degrees of freedom (DoFs) than required to perform a user-specified task. Therefore, redundant robots deliver more advantages in human-robot interactions. However, greater flexibility originated from redundant DoFs increases the risk of self-collision, especially when the human operator forces the end-effector to move close to the robot's own links. Moreover, the robot links are constantly moving, which makes self-collision avoidance more difficult.

The crux in the self-collision problem of redundant robots lies in the difficulty of determining appropriate inverse kinematics (IK) (Paul and Shimano, 1978) solution. For non-redundant robotic arms, providing a specific position and orientation for the end effector typically results in no more than 32 IK solutions (Tsai and Morgan, 1985). However, in the case of redundant robotic arms, there often correspond an infinite number of solutions. This makes it challenging to determine an appropriate IK solution to avoid self-collision.

Firstly, it is difficult to obtain the closed form solution of redundant robots directly (Zaplana and Basanez, 2018) and it is also burdensome to meet the constraints of self-collision avoidance. Secondly, the IK solution should be consistent with the mechanical

properties of the robot arm, i.e., the joint variables need to be bounded to ensure the smoothness of the motion. Finally, the solution should have some generality to migrate on multiple robots.

There are some limitations in the existing IK solutions. Some researchers have proposed numerical methods for solving inverse kinematics problems, such as the Jacobian matrix inversion-based solution (Colomé and Torras, 2015), the transposed Jacobian matrix-based solution (Wolovich and Elliott, 1984), and the damped least squares (DLS) solution (Nakamura and Hanafusa, 1986). Based on the current joints' status and the Denavit-Hartenberg (D-H) parameters (Denavit and Hartenberg, 1955) of a given robot, the end-effector can be continuously controlled under the guidance of one predetermined path. Unfortunately, numerical methods only provide a feasible set of IK solutions, which cannot be selected in complex cases. In contrast, Heuristic-based methods transform the IK problem into an optimization problem and select adaptive solutions. Momani et. al used a genetic algorithm to solve the IK problem and obtains a continuous smooth solution (Momani et al., 2016). In Rokbani and Alimi (2013) and Dereli and Köker (2018), different variants of the particle swarm optimization were presented to solve the IK problem for redundant robots. In Dereli and Köker (2020), a quantum particle swarm was proposed to compute the IK solutions of a 7-DoF robot. Koeker and Cakar improve the control accuracy of a robotic arm by combining neural networks, simulated annealing, and genetic algorithms (Köker and Çakar, 2016). Although heuristic methods can address IK problems under constraints, the dynamic nature of the robotic arm's motion creates a constantly changing environment, making it challenging for heuristic methods to converge. Furthermore, heuristic approaches often focus on solving for a specific state of the environment, overlooking the temporal and spatial continuity of the robotic arm's movement.

In contrast to traditional methods, machine learning approaches offer greater adaptability to complex tasks in controlling robots. In Cao et al. (2022, 2023a,b), authors have employed deep learning techniques to control robotic arms for grasping tasks. In a variety of machine learning approaches, Reinforcement learning (RL) (Sutton and Barto, 2018) involves a method that an agent explores the environment to achieve a maximum reward. Therefore, RL is suitable for finding an appropriate IK solution for a redundant robot under the constraint of collision avoidance. In Bing et al. (2023a,b), the authors employed meta-RL to control robots in simulated environments to achieve specific objectives. In Bing et al. (2022b), the authors utilized RL to control the locomotion of a snake-like robot. However, in self-collision avoidance situations, there are challenges in sample acquisition. Direct control of robot joints by reinforcement learning agents may cause difficulties in obtaining successful samples. In contrast, when controlling robot joints using the traditional IK method, collision samples are rare, which makes it difficult for the agent to learn how to avoid collisions.

To address the issues mentioned above an RL-enhanced pseudo-inverse approach is proposed in this paper. The RL solver does not directly control the robot, but imposes an interference to the pseudo-inverse solver to avoid self-collision of the robot. The main contributions of this paper are listed as follows:

- Firstly, an RL-enhanced pseudo-inverse solution method is proposed. In this approach, the RL agent outputs interference. The pseudo-inverse solver incorporates these interference into the computation to obtain an IK solution for robotic positioning with self-collision avoidance.
- A novel replay buffer is designed to adjust sample proportions under self-collision avoidance scenarios. This enhances the learning efficiency of the agent by elevating the diversity of samples in the buffer.
- Finally, a simulated training and testing environment was established in CoppeliaSim, and the effectiveness of the proposed method is validated through simulations and experiments using the Frank Emika Panda robotic arm.

## 2 Related work

Collision avoidance is always one of the critical issues to be solved in robotic arm control. The method in Guo and Hsia (1990) and Cheng et al. (1998) maximizes the distance between the robot arm and the obstacle to avoid collisions, but it is unnecessary to always maximize the distance when the robot is far away from the obstacle. In Duguleana et al. (2012), the authors propose an improved quadratic programming(QP) problem formulation, representation of the collision-free scheme as a dynamically updated inequality constraint. Haviland and Corke (2021) present a motion controller which is wrapped into QP. The controller can avoid static and dynamic obstacles while moving to the desired end-effector pose.

As an intelligent learning method, RL does not require an accurate model or prior knowledge, thus providing a new solution to the complex redundant robot control problem. The authors of Al-Hafez and Steil (2021) take the concept of redundancy resolution and propose a policy search with redundant action bias to control the motion of the robotic arm and avoid collisions by maximizing the distance between the linkage and the obstacle. In Li et al. (2022) the authors propose a framework that employs deep reinforcement learning (DRL) to find the most efficient path in Cartesian space and to compute the most energy-efficient solution for robot IK. In Bing et al. (2022a, 2023c), the authors trained robotic arms to evade obstacles and grasp targets using a method based on Hindsight Goal Generation.

In Martin and Millán (1997) the authors employ proximity sensors and reinforcement learning methods to solve the self-collision problem of redundant robotic arms. However, this approach is specific to two-dimensional robotic arms and difficult to extend to three-dimensional space. In Agarwal et al. (2016) and Schappler and Ortmaier (2021), the authors employed the null space projection to address singularity avoidance in three-axis planar robots and six-axis serial robots, respectively. In comparison to singularity avoidance, self-collision avoidance requires more consideration of the robotic arm's structure, as arms with different structures have varying joint positions during motion, making it more challenging to predict the location of the links.

In summary, the issue of self-collision in robotic arms becomes increasingly significant with the growth of joint complexity,

and there is currently limited attention given to this problem. Traditional collision avoidance methods typically focus on obstacles with fixed or uniform motion or impose restrictions on the number of joint angles of the robotic arm. The proposed method in this paper aims to avoid irregularly moving robotic arm links and is insensitive to both the number and structure of joints in redundant robotic arms.

## 3 Mathematical model

Generally, the control of a robotic arm is associated with a mapping from the work space to the joint space. However, it is difficult to directly calculate the relationship between the change in end-effector's pose and the change in joints' states. Therefore, a common approach is to map the change in pose to the change of joint velocity, and the work space and joint space are related by a Jacobian matrix in the mapping.

Set the desired velocity of the end-effector of the robot arm to be $\dot{x}$, which is a 6-dimensional vector (three translations and three rotations), and let $J$ denote the Jacobian matrix. The joint velocity of the robot arm is an $n$-dimensional vector, referred to as $\dot{q}$, where $n$ represents the number of DoFs of the robot. The velocity of the end-effector $\dot{x}$ could be obtained from $\dot{q}$ and $J$ in Equation (1):

$$\dot{x} = J\dot{q}. \tag{1}$$

Then, based on the pseudo-inverse method, it yields Equation (2):

$$\begin{aligned} &\min \parallel \dot{q} \parallel^2 \\ &\text{subject to } \dot{x} = J\dot{q}. \end{aligned} \tag{2}$$

The pseudo-inverse method uses the minimum joint velocity as the optimization objective to improve the motion efficiency. However, this optimization objective cannot satisfy the need for self-collision avoidance. As a result, a controllable interference $\dot{i}$ is added, where $\dot{i}$ represents a vector with the same dimension as $\dot{q}$ in Equation (3):

$$\begin{aligned} &\min \parallel \dot{q} + \dot{i} \parallel^2 \\ &\text{subject to } \dot{x} = J\dot{q}. \end{aligned} \tag{3}$$

According to the Lagrange multiplier method (Boyd and Vandenberghe, 2004), it leads to

$$\dot{q} = J^T(JJ^T)^{-1}(\dot{x} + J\dot{i}) - \dot{i}. \tag{4}$$

Define $J^\dagger = J^T(JJ^T)^{-1}$. Since $JJ^\dagger = I$, $J^\dagger$ is the right pseudo-inverse of the Jacobian matrix $J$. Therefore, Equation (4) is equivalent to

$$\dot{q} = J^\dagger(\dot{x} + J\dot{i}) - \dot{i}. \tag{5}$$

The right pseudo-inverse of the Jacobian matrix can be solved with any of the optimization solvers without affecting the computation of $\dot{q}$. The existence of the pseudo-inverse solution with interference is not affected since the inverse matrix is replaced by the pseudo-inverse and $J^\dagger$ necessarily exists.

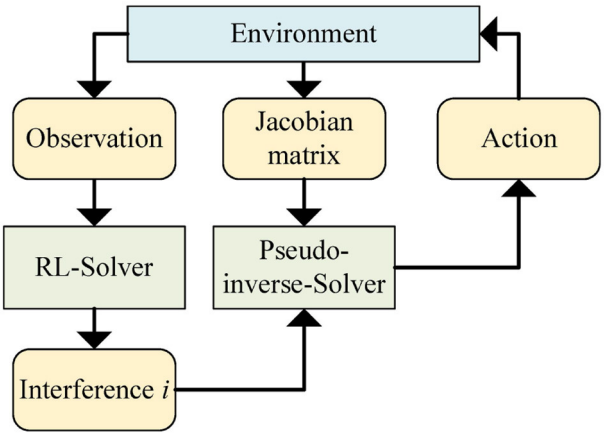


FIGURE 1
Structure of RL enhanced pseudo-inverse controller.

Lemma 1. After adding the interference, the resulting $\dot{q}$ still allows the end-effector to move with the desired velocity $\dot{x}$.

*Proof.* Multiplying $J$ left on both sides of Equation (5), then get Equation (6):

$$\begin{aligned} J\dot{q} &= JJ^\dagger(\dot{x} + J\dot{i}) - J\dot{i} \\ &= \dot{x} + J\dot{i} - J\dot{i} \\ &= \dot{x}. \end{aligned} \tag{6}$$

This proves that $\dot{q}$ with interference $\dot{i}$ can generate the desired velocity for robot motion control.

## 4 Proposed RL enhanced controller

The proposed controller contains two solvers, the RL solver and the pseudo-inverse solver. The RL solver accepts observation from the environment and returns interference $\dot{i}$. The pseudo-inverse solver accepts the Jacobian matrix $J$ from the environment and calculates the velocity $\dot{q}$ of robot joints in the current state by combining interference $\dot{i}$ according to the method in Equation (5). A series of actions will be obtained through the controller. The structure diagram of the controller is shown in Figure 1. The pseudo-inverse solver has been explained in Section 3. Next, we will present the key elements of the RL solver and the RL network structure.

### 4.1 Observation space

The RL agent receives the environment information through the observation space at each step. In RL, the correct choice of observation space parameters is crucial as the agent needs the correct set of information to understand the causality of a given reward based on the behavior.

The observation space $o^t$ is given in Table 1. $\phi$ is an n-dimensional vector to present the current robot joints' angle. Define $\dot{p}$ as the difference of coordinates between the target point and the

TABLE 1 The observation space $o^t$ of the RL controller.

| Symbols | Description |
|---|---|
| $\phi$ | Current joints angle |
| $\dot{p}$ | Difference between target and current position |
| $\dot{r}$ | Difference between initial and current rotation |

current robot end-effector. The target point position coordinates is denoted as $\dot{p}_{tar}$, and the end-effector position is denoted as $\dot{p}_{ee}$. $\dot{p}$, $\dot{p}_{tar}$, and $\dot{p}_{ee}$, are 3-dimensional vectors. The 3 dimensions correspond to the coordinates of the Cartesian coordinate system in the $x$, $y$ and $z$ directions. $\dot{p}$ can be obtained from

$$\dot{p} = \dot{p}_{tar} - \dot{p}_{ee}. \tag{7}$$

Define $\dot{r}$ as the angle difference between the end-effector at the beginning and the current position. The initial end-effector rotation is denoted as $\dot{r}_{init}$, and the current rotation is denoted as $\dot{r}_{cur}$, then obtain the rotation difference from Equation (8):

$$\dot{r} = \dot{r}_{init} - \dot{r}_{cur}. \tag{8}$$

Similarly, $\dot{r}$, $\dot{r}_{init}$, and $\dot{r}_{cur}$ are all 3-dimensional vectors that represent the rotation about the x, y, z axes in the Cartesian coordinate system.

In summary, an overall $(n + 6)$-DoFs observation space is used in this work. In the simulation of this paper, the position and rotation information of the end-effector are obtained from the simulator directly. For the robotic arm in a physical environment, this information can be estimated using forward kinematics.

## 4.2 Action space

At time $t$, the final output of the controller is an $n$-dimensional vector $a^t$, denoted as $\dot{q}$ in the Equation (5), where $n$ represents the DoF of the robotic arm. The values of $a^t$ fall within the range of [-1, 1] and are transformed linearly to correspond to the velocities of the respective joints. The RL solver produces the interference vector $i^t$, which is also an $n$-dimensional vector, representing the interference $\dot{i}$ in the Equation (5) at time $t$. It is worth noting that this paper primarily focuses on the robot's kinematics, and joint torques are beyond the scope of this study.

## 4.3 Single-step reward

When the RL solver introduces excessive disturbance to the Pseudo-inverse solver, it can result in deviations of the end effector's motion from the target trajectory or unexpected rotations. For instance, when the joint motion of the robotic arm exceeds limits, it may cause a significant deviation of the end effector from the designated motion trajectory.

To prevent the occurrence of the aforementioned phenomena, it is essential to calculate rewards for each step of the RL solver's output. The reward function for the single-step reward comprises translation and rotation components.

For the translation component, we take the current velocity vector of the end-effector, the angle $\theta$ with the target direction vector, and compute $\cos(\theta)$. In the following steps, we take the single-step translation reward as $\cos(\theta) - 1$ to ensure it is negative. When the end-effector moves exactly in the specified direction, the reward is set as the maximum value of 0. The translation component is denoted as $R_l$, it could be found that,

$$R_l = \frac{\dot{p} \cdot v_{ee}}{|\dot{p}||v_{ee}|} - 1, \tag{9}$$

where $v_{ee}$ is a 3-dimensional vector to represent the current translation velocity of the end-effector.

For the rotation component, as this paper primarily focuses on position inverse kinematics issues, we take the default rotation of the end-effector to make a difference with the current rotation. The 2-norm is taken for the resulting vector. Finally, the obtained value is divided by a factor $k$ to balance the value with the translation component. With the translation component, a non-positive value is taken for the obtained value, which is rewarded as a maximum value of 0, when the end-effector remains theinitial rotation. Let the rotation component be $R_r$, it can be computed as follows,

$$R_r = -\dot{r}/k, \tag{10}$$

where $\dot{r}$ be computed from Equation (7). In this paper, $k$ is set as 100. The coefficient $k$ balances the translation reward and rotation reward, avoiding that one reward is too large and agent ignores the other one.

Combining Equations (9, 10), yields the reward function for each step

$$R_s = R_l + R_r. \tag{11}$$

When the end effector moves in the vicinity of the robotic arm according to the specified trajectory, the single-step reward approaches zero. However, if the robotic arm's joints exceed limits or other geometric structural issues impede its normal motion, the single-step reward significantly decreases. This encourages the RL solver to avoid situations where the robotic arm becomes stuck.

It is noteworthy that the rewards returned by Equation (11) are for each step and do not encompass the rewards for each episode. Episode rewards will be explained in the next subsection.

## 4.4 Episode rewards

The single-step reward calculation involves the rewards received by the agent for each action taken. Episode rewards, on the other hand, are computed based on the outcomes after the completion of an episode. Since the position relationship between adjacent state-action pairs is lost during retraining after replays are placed into the buffer, it is necessary to finalize the reward allocation for each episode before adding the replay to the buffer. In this paper, a Monte Carlo-like method is employed, where the final reward accumulated at the end of an episode is propagated backward and assigned to all steps within that episode.

Assume there are $k$ replays in one episode, meaning that this episode takes $k$ steps. $R_j$ is the $j$th step reward in episode buffer, then adjusted the rewards in episode buffer as

$$R_a = R_j + \gamma^{k-j} R_{end}, \tag{12}$$

where $R_a$ is the adjusted reward. $R_{end}$ is the end reward of an episode, based on if the episode results in success or failure, its value is represented by Equation (13).

$$R_e = \begin{cases} R_{pos} & when\ reach\ the\ target \\ R_{neg} & other, \end{cases} \tag{13}$$

where $R_{pos} > R_{neg}$, it indicates that the reward obtained upon successfully reaching the target position exceeds the reward obtained upon failure.

The intuition behind Equation (12) is to reinforce the correlation between adjacent replays. If an action results in the robotic arm moving toward a collision direction, the reward for the corresponding state-action pair is correspondingly reduced.

While it is possible to enhance training performance by magnifying the reward/punishment values at the conclusion of each episode and transmitting this value through the Bellman equation, it is observed that directly amplifying these values during training often led to frequent training failures. The approach outlined in Equation (12), however, facilitates more successful training. We hypothesize that the use of Equation (12) disperses the reward/punishment values across multiple records, preventing issues associated with excessively large gradients.

## 4.5 Dynamic balancing replay buffer

The role of the replay buffer in RL is to improve the utilization of samples and to enable the agent to train offline. The rewards generated by the interaction between the agent and the environment are stored in the replay buffer.

The usual approach is to add the rewards returned by the environment directly to the replay buffer, then randomly select some replays from the buffer to train the agent. However, there are two challenges in this paper.

One of the challenges is that the movement of the robot is continuous, and collisions occur not triggered by only one action. The usual approach destroys the cause-and-effect relationship between adjacent actions.

The other challenge is that during the training of the combined controller, the domination of any solver causes reward/penalty sparsity. For example, when the RL solver dominates, the untrained RL solver makes the robot always collide and causes it difficult to converge. On the other hand, when the pseudo-inverse solver dominates, the robot rarely collides, which leads the RL solver difficult to get trained because of the lack of collision samples.

The first challenge was addressed in the preceding subsection through an episode-based approach. As for the second challenge, the proposed solution in this paper involves the introduction of a dynamic balancing mechanism for the replay buffer.

Let *info* return at the end of an episode. Return *True* when the end-effector successfully reaches the target, otherwise return *False*.

Let the total number of steps from successful episodes in the current replay buffer to $n_s$ and the total number of steps from failed episodes to $n_f$.

The replays in episode buffer will be added to the replay buffer only when:

1). $info = True\ and\ n_s \leq n_f$, or
2). $info = False\ and\ n_s > n_f$

This dynamic balancing mechanism ensures that the number of successful and failed steps in the replay buffer is similar, thus avoiding the problem of difficult convergence due to the lack of samples.

## 4.6 Network and training

Given the input (observation $o^t$) and the output (action $a^t$), the details of the network structure are introduced. We train our agent using the TD3 (Fujimoto et al., 2018) based on the Actor-Critic architecture, thus requiring two Critic networks and one Actor network. Each Critic network contains two fully connected hidden layers that act as non-linear function approximators of Q value. The dimension of the input layer is the sum of the dimensions of $o^t$ and $a^t$. Both hidden layers have 128 PReLU (He et al., 2015) units. The last layer outputs a Q value. The Actor network also contains two fully connected hidden layers. The input layer has the same dimensions as $o^t$. Both two hidden layers have 128 ReLU units, with the last layer outputting the interference $\dot{i}$.

Theoretically, any deterministic policy RL algorithm can be applied to our method, but the RL algorithm with a stochastic policy may cause the robot arm to jitter, such as the SAC (Haarnoja et al., 2018) we have tested which shown in Section 5.

# 5 Simulation and experiment

In this section, simulative and experimental validations were conducted to assess the effectiveness of the proposed approach in avoiding self-collision for redundant robotic arms. In both simulation and experimentation, the end effector's motion trajectory was defined to compel its movement in proximity to the robotic arm's own structure. The experimental scenarios simulated situations that might occur when an operator directly manipulates the end effector of the robotic arm.

Coppeliasim is a kind of mainstream robot simulator, which can simulate the motion and collision detection of robot arms. In the simulation, a built-in Franka Panda robotic arm with 7-DoFs are employed. The initial joint angles are set to [0°, −17°, 0°, −126°, 0°, 114°, 45°]. The initial coordinate position of the end-effector is [0.499 $m$, 0 $m$, 1.189 $m$].

The goal of the simulation and experiment is to guide the end-effector motion to the target point. The target points are generated in a hemispherical space of 0.5 $m$ radius around the first joint of the robot arm. A series of path points are generated at 0.01 m intervals along a straight line between the initial position and the target point position to force the end-effector to move near the

TABLE 2 Convergence verification parameter.

| Parameters | Value |
|---|---|
| Episodes | 1,000 |
| Critic learning rate | $2 \times 10^{-4}$ |
| Actor learning rate | $1 \times 10^{-4}$ |

robot arm itself. When the distance between the end effector and the target is less than 0.001 meters, the end effector is considered to have reached its destination. Millimeter-level accuracy is deemed sufficiently precise given the range of motion of the robotic arm in this experiment. Additionally, the methods employed in this study are iterative, and demanding excessive precision would result in the control algorithm consuming an impractical number of steps during the final convergence process. In the simulation, the built-in collision detector in Coppeliasim is employed to detect whether the robotic arm has collided. PyRep (James et al., 2019) is employed to set up the reinforcement learning environment.

The platform configuration of training and simulation is listed as follows: CPU: i9-9900K; GPU: 2080ti; Memory: 64G. Coppeliasim version is 4.1 EDU and the operating system is Ubuntu 18.04 LTS.

## 5.1 Convergence validation simulation

This simulation compares the convergence with and without the use of our improved replay buffer. The algorithm has been trained for 1000 episodes, which is about 200,000 steps. The parameters employed during training are outlined in Table 2. For the sake of stability during training, Stochastic Gradient Descent (SGD) was utilized instead of momentum-based optimizers. The results shown in Figure 2 represent the upper and lower bounds and average values for 25 replicate simulations.

As can be seen from the Critic convergence curve, the improved replay buffer makes the algorithm converge more stably. On the other hand, the unimproved replay buffer affected Critic's convergence and even led to failure to converge. Since Critic estimates Q values of state-action pairs, it is verified that our improved replay buffer improves the accuracy of the agent's estimation of the environment.

While the lower Actor loss is better, the unimproved replay buffer leads to an inaccurate estimate of Critic. Thus the lower loss under the unimproved replay buffer is not convincing. In contrast, our improved replay buffer exhibits more stability compared to the unimproved replay buffer.

Benefiting from better stability, our improved replay buffer allows for higher rewards per step and lower volatility during multiple training sessions.

## 5.2 Single target positions arrival experiment

In this simulation, 1071 final target points are evenly distributed in the hemispherical space around the robot arm. In each episode,

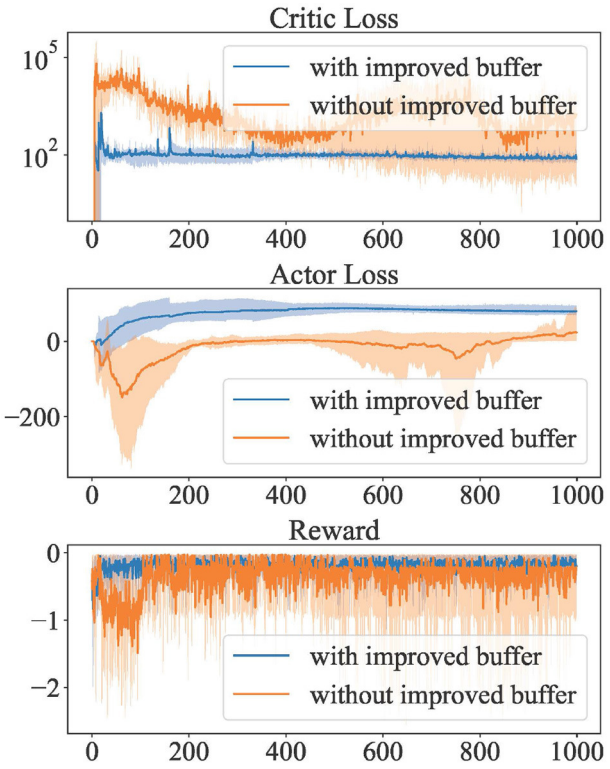


FIGURE 2
Convergence verification under initialization conditions dominated by RL solver.

the end-effector of the robotic arm starts from the same initial position and follows a moving target to reach the final target point. The moving target moves in a straight line, forcing the end-effector to move close to the arm's own structure.

Six different methods are compared in this simulation. To compare methods without collision avoidance policies, PI (pseudo-inverse) and TJ (Jacobian Transpose) methods are introduced in this simulation as comparisons. NEO (Haviland and Corke, 2021) is introduced into the comparison as a non-learning-based obstacle avoidance method. The RL method uses only the TD3 algorithm as a comparison of not improving on the replay buffer. HER (Andrychowicz et al., 2017) is introduced as a method that improved the replay buffer in this comparison. Replay buffer in HER method uses future choosing strategy. It is difficult to predict the timing of collision occurrence in the simulation, thus the update of the replay buffer in HER method is to encourage the tracking of target points by the agent. OURS method is the enhanced pseudo-inverse method of reinforcement learning with an improved replay buffer proposed in this paper.

The max number of steps for each episode is 1000. There are three outcomes for each episode, successful arrival at the target position (Success), exceeding the step limit but no collision (Run out), and Collision. The results of this simulation are summarized in Figure 3 and Table 3. The number of the three results in a single simulation is counted in the Success, Run out, and Collision columns respectively. Avg steps column counts the average number of steps spent in the episode of Success. The avg reward column

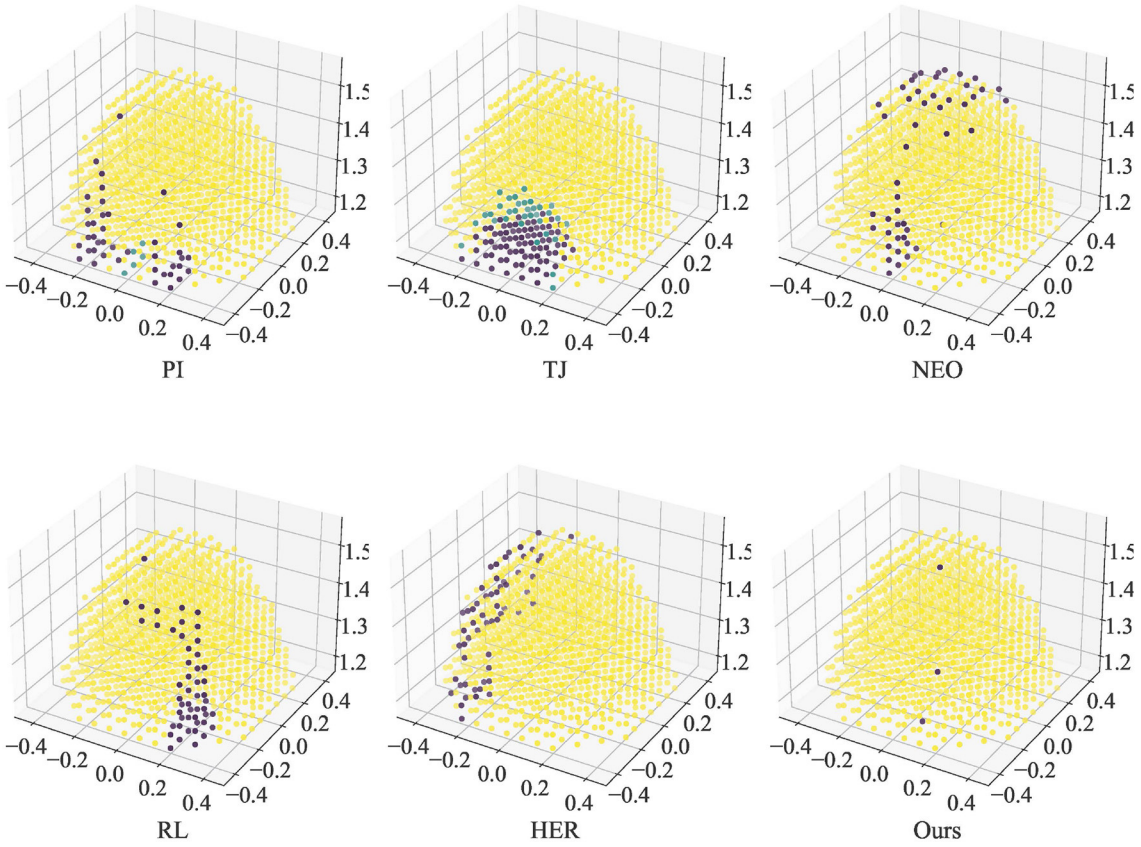


FIGURE 3
Ability of the end-effector to reach a single target position under control of different approaches. Yellow dots indicate successful arrival at the target. Green dots indicate that the number of steps is used up and the target is not reached (except NEO). Purple dots indicate collisions (except NEO).

counts the average reward per step. In this simulation, the reward is calculated according to Equation (11), reflecting the tracking performance of the end-effector on the target point under the control of different methods.

In Figure 3, yellow dots indicate the positions where the end-effector can reach the target successfully. Green dots indicate positions that cannot be reached before the steps run out. Purple dots indicate collisions when reaching these positions. The NEO method did not collide, so points indicating Run out are marked in purple for clarity in observation. The figure shows that moving the end-effector from the initial position to the back of the robot arm is challenging for the controller. The RL method without an improved replay buffer does not have an advantage over the traditional numerical method. The optimization-based collision avoidance approach NEO has difficulty in finding a suitable solution for avoiding irregularly moving robotic arm links, especially when a well-defined path is given.

According to the data in Table 3, it can be seen that our method has the highest arrival rate (99.72%), but the Avg reward is slightly lower. This could be attributed to a decrease in the end-effector movement speed under Agent control (resulting in a higher average number of steps compared to PI) and slight deviations from the planned trajectory, leading to a reduction in rewards. NEO is able to avoid collisions completely in this simulation but has more Run out cases. In the simulation, it was observed that NEO causes the robot arm to get stuck in certain poses and cannot continue tracking the target. Also, the NEO method prefers to find the trajectory freely, so it cannot track the target well under strict constraints on the trajectory, which is reflected by having a lower Avg reward. Attributed to encouraging target tracking, HER has lower Avg steps and higher Avg reward but does not do better in avoiding collisions and reaching the final target.

Figures 4A, B show how our method avoids self-collisions during tracing the target. The blue curve in the figure shows the trajectory of the end-effector. It can be seen that the end-effector maintains a smooth motion path under both methods. In Figure 4A, the PI method is employed to control the motion of the robotic arm. This resulted in a collision between the end-effector and the robot arm linkage at the red circle. In Figure 4B, our method controls the robot arm to rotate its own mechanism to move away from the motion path of the end-effector. Because of this behavior, our method can avoid self-collisions.

## 5.3 Three target positions arrival simulation

In each episode of this simulation, three final target positions are randomly generated. The controller needs to guide the end-effector to reach the three positions consecutively. Unlike the single target experiment, the initial of each segment of the moving

TABLE 3 Details of the end-effector reaching single position under the control of different methods.

| Methods | Success | Run out | Collision | Avg steps | Avg reward |
|---|---|---|---|---|---|
| PI | 1,025 | 5 | 41 | 97.68441 | -0.04706 |
| TJ | 961 | 33 | 77 | 514.761 | -0.18328 |
| NEO | 1,023 | 48 | 0 | 222.2605 | -0.10953 |
| RL | 1,021 | 0 | 50 | 219.8609 | -0.05474 |
| HER | 1,014 | 0 | 57 | 162.2745 | -0.0493 |
| OURS | 1,068 | 0 | 3 | 234.6667 | -0.06235 |

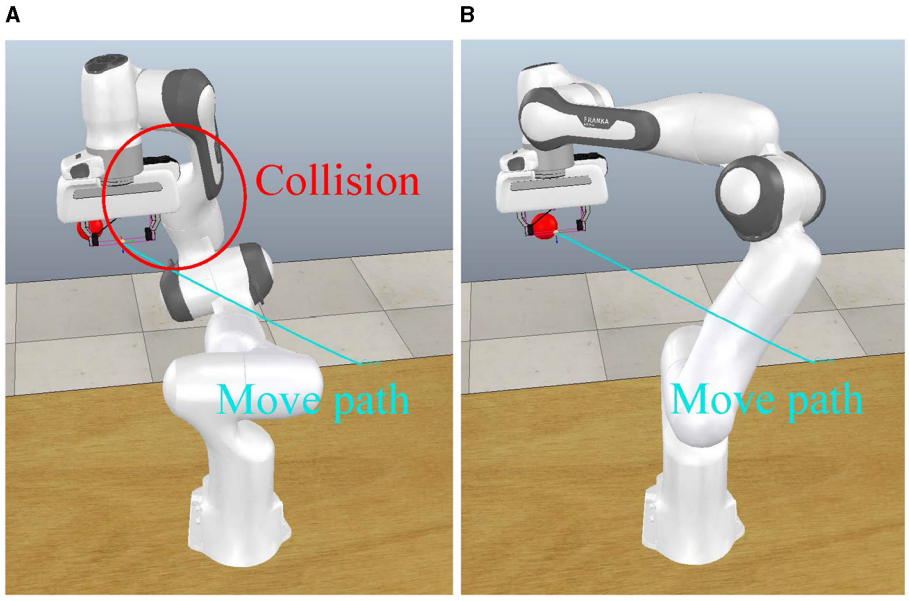


FIGURE 4
Path under different control methods. The cyan line depicts the motion trajectory of the end-effector, while the red sphere indicates the target position of the end-effector. Red circles highlight locations where collisions occur under the control of the pseudoinverse method. **(A)** PI posture. **(B)** Ours posture.

robot arm cannot be predicted. The experiment is executed for 1,000 episodes. The upper limit of steps per episode is 3,000. The performance of the six methods is compared. The target positions are pre-generated to ensure that each method faces the same challenge.

The results of the simulation are shown in Table 4. Similar to Section 5.2, the controller guides the end-effector to reach the three final target positions in sequence and is recorded as Success. Failure to reach any target position or collision in the middle of the episode is recorded as run out or collision. The table also shows the average steps per episode and the average reward per step.

The data in the Table 4 shows that our method has a higher successful arrival percentage than the method without the improved replay buffer, which indicates that the improved replay buffer enhances the performance of the RL algorithm. Our method also has an advantage over the traditional method, which indicates that the combined reinforcement learning and pseudo-inverse methods give the robot arm a better ability to avoid self-collision. The lower average reward for all methods compared to the single target position experiments is due to the longer average number of steps, thus generating more negative rewards.

## 5.4 Ablation experiments

The purpose of this section is to evaluate whether the improvements for different challenges improve performance. To this end, we repeat the experiments of the Section 5.2 section and keep all other parameter settings identical.

To verify the effectiveness of the reward adjustment in Equation (12), we select different $\gamma$ for training and tested the training results in simulation. The results of the test are shown in Table 5.

As can be seen from the table, the adjusted reward has a positive effect on avoiding self-collision of the robot arm, but the effect does not increase linearly with $\gamma$. The success rate of tracking showed two peaks when $\gamma$ is close to 0.2 and 0.7. In addition, the algorithm shows better stability when $\gamma$ is set to 0.2 in repeated training.

TABLE 4 Details of the end-effector reaching three different positions under the control of different methods.

| Methods | Success | Run out | Collision | Avg step | Avg reward |
|---|---|---|---|---|---|
| PI | 778 | 0 | 222 | 260.327 | -0.08718 |
| TJ | 796 | 112 | 92 | 1294.823 | -0.14426 |
| NEO | 742 | 160 | 98 | 528.74 | -0.10586 |
| RL | 751 | 23 | 226 | 463.561 | -0.08597 |
| HER | 723 | 5 | 272 | 393.234 | -0.07653 |
| Ours | 931 | 0 | 69 | 504.456 | -0.05887 |

TABLE 5 Details of simulated data with different success and failed buffer ratios.

| $\gamma$ | Success | Run out | Collision | Avg step | Avg reward |
|---|---|---|---|---|---|
| 0.99 | 1,061 | 0 | 10 | 171.2754 | -0.0412 |
| 0.95 | 1,041 | 0 | 30 | 164.7703 | -0.05292 |
| 0.9 | 1,058 | 0 | 13 | 170.605 | -0.04269 |
| 0.7 | 1,061 | 0 | 10 | 172.6872 | -0.05023 |
| 0.5 | 1,050 | 1 | 20 | 168.5556 | -0.04323 |
| 0.2 | 1,061 | 0 | 10 | 172.6872 | -0.05024 |
| 0.1 | 1,042 | 0 | 29 | 166.549 | -0.04163 |
| 0.05 | 1,053 | 0 | 18 | 169.4585 | -0.04364 |
| 0.01 | 1,041 | 20 | 10 | 168.8571 | -0.06217 |

To verify the effectiveness of the dynamic balancing mechanism, a set of simulations with different ratios of success and failed replay buffer are performed. The results of the simulation are shown in Table 6.

This simulation compares the tracking of end-effectors with four different ratios. In the No balance simulation, there is no limit on the percentage of successful and failed replay, and the percentage of failed replay is about 32.37% (135,480 of 418,525). The remaining simulations limit the percentage of failed replay to about 50%, about 75%, and 100%.

As can be seen in Table 6, Balancing successful and failed replay by 50%-50% can effectively reduce the probability of self-collision of the robot arm, although this increases the average number of steps and slightly reduces the average reward. Slightly more failed replay (75%) also reduces the likelihood of self-collision, but not as effectively as keeping it at 50%. The probability of collision increases when using failed replays entirely, due to the lack of successful samples resulting in the agent's inability to properly evaluate the environment.

Two conclusions can be drawn from the simulation results as follows. A) The dynamic balance replay proposed in this paper is effective in avoiding self-collision of the robotic arm. B) Appropriate discarding of some replay may have a positive impact on the behavioral strategy of the agent.

To examine the impact of distinct weights for $R_l$ and $R_r$ on the agent as stipulated in Equation (11), the reward function is configured as Equation (14)

$$R_s = \alpha R_l + (2 - \alpha) R_r. \tag{14}$$

The performance of the proposed apporach is evaluated under varying $\alpha$ values, and the results are presented in Table 7.

From Table 7, it can be observed that the proposed method performs favorably when $\alpha$ is set to 1. Therefore, in this paper, $\alpha$ is chosen to be 1, as indicated in Equation (11).

## 5.5 Motion smoothness demonstration

To verify that the method in this paper not only smooths the end-effector motion but also keeps the velocity of the robotic arm joints smooth. All joint velocities for 200 consecutive steps were collected to verify the smoothness of the robot arm joint motion.

Let the Action of step $t$ be $\dot{q}^t$. The acceleration $\dot{a}^t$ is calculated by approximating the velocity of two adjacent steps as Equation (15)

$$\dot{a}^t = \dot{q}^{t+1} - \dot{q}^t. \tag{15}$$

To represent the acceleration as a single value, a first order norm for $\dot{a}^t$ has been taken. After that, divide it by the number of joints to get the average acceleration $\bar{a}^t$ of the joints at step $t$. The formula is expressed as Equation (16)

$$\bar{a}^t = |\dot{a}^t|/n. \tag{16}$$

To visualize the results, the acceleration curves of Our method (Combined deterministic algorithm TD3 and pseudo-inverse) and SAC (Combined stochastic algorithm and pseudo-inverse) have

TABLE 6 Details of simulated data with different success and failed buffer ratios.

| Percent | Success | Run out | Collision | Avg step | Avg reward |
|---|---|---|---|---|---|
| No balance | 1,042 | 0 | 29 | 166.3119 | -0.04251 |
| 50% | 1,061 | 0 | 10 | 172.3688 | -0.05038 |
| 75% | 1,056 | 0 | 15 | 170.0355 | -0.04462 |
| 100% | 819 | 0 | 252 | 118.535 | -0.04596 |

TABLE 7 Details of simulated data with different success and failed buffer ratios.

| Ratio $\alpha$ | Success | Run out | Collision | Avg step | Avg reward |
|---|---|---|---|---|---|
| 0.25 | 1,029 | 1 | 41 | 173.6502 | -0.05646 |
| 0.5 | 1,038 | 0 | 33 | 174.3946 | -0.05828 |
| 1 | 1,056 | 0 | 15 | 173.4608 | -0.05111 |
| 1.5 | 1,032 | 0 | 39 | 172.9067 | -0.05961 |
| 1.75 | 1,027 | 0 | 44 | 172.9543 | -0.05935 |

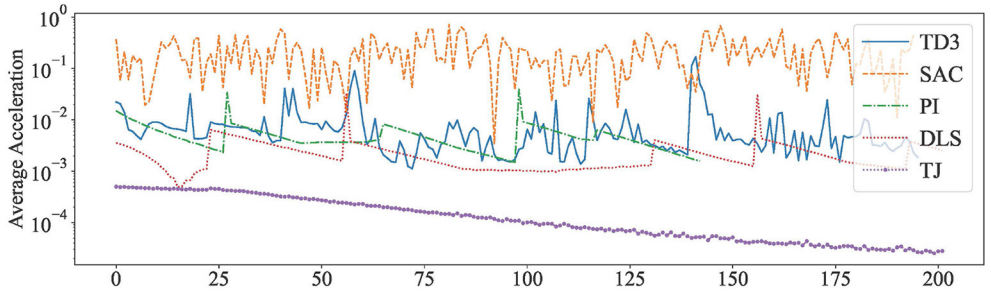


FIGURE 5
Average acceleration of robot arm joints under different methods of control.

TABLE 8 The average acceleration of each joint of the robot arm under different methods.

| Methods | Average | Maximum | Variances |
|---|---|---|---|
| Ours | 8.8e-3 | 0.17 | 2.8e-4 |
| SAC | 0.22 | 0.71 | 2.3e-2 |
| PI | 5.6e-3 | 0.05 | 3.7e-5 |
| DLS | 2.4e-3 | 0.03 | 9.9e-6 |
| TJ | 1.7e-4 | 5e-4 | 2.3e-8 |

been plotted. As a comparison, the acceleration curves of PI, DLS, and TJ are also plotted. The results are shown in Figure 5, and detailed data are shown in Table 8.

As can be seen in Figure 5, the curves of both our method and the SAC method produce significant fluctuations. This indicates that the control of the reinforcement learning method caused the jitter of the robot arm joints. However, the acceleration range of our method is closer to that of PI, which is two orders of magnitude lower than that of the SAC method. This is because our controller employed a deterministic reinforcement learning algorithm and calculate the final result by numerical methods. The PI, DLS, and TJ methods have smoother acceleration profiles because they are calculated in a purely mathematical way, with large fluctuations only in the case of target changes.

In Table 8, the mean, maximum, and variance of the acceleration for each of the five methods in 200 steps are calculated. These data are used to reflect the variation of acceleration for the above methods.

Our method is closer to the numerical methods PI and DLS in terms of average acceleration and maximum acceleration as shown in Table 8. The average acceleration of our method is 3 orders of magnitude lower, while the variance of acceleration is 2 orders of magnitude lower than the SAC method. Therefore, when the robot arm is controlled by our method, not only the trajectory of the end-effector is smooth, but also the motion of the other joints of the robot arm is steady.

The TJ method has the best acceleration performance. However, as it is known from previous experiments, the TJ method requires more steps to reach the target, so its lower acceleration is due to its slow movement speed.

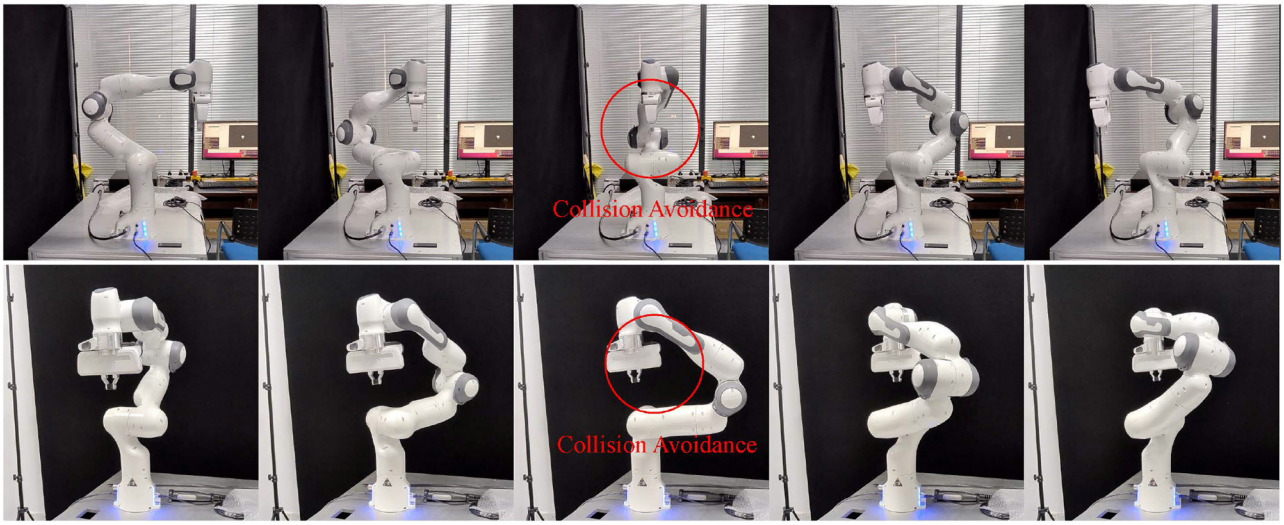


FIGURE 6
The upper row is the right view of the motion process, the lower row is the front view of the motion process.

### 5.6 Experiment

In this experiment, our method is deployed on a real Franka Emika Panda robot. The goal of the experiment is to move the end-effector along a straight line to the rear of the robotic arm to examine the ability of the algorithm to avoid collisions. The initial position of the end-effector is [0.5, 0.0, 0.36], and the target arrival position is [-0.4, 0.13, 0.46], measured in meters. The motion trajectory is a straight line connecting the initial position to the target position. To ensure that the end-effector moves along the designated path, 383 waypoints were inserted along the motion trajectory. Our method provides inverse kinematics (IK) solutions at a frequency of 10Hz, while the Franka Panda robotic arm receives control signals at a frequency of 1,000 Hz. To accommodate this, we performed B-spline linear interpolation along the trajectory, resulting in a total of 38,500 points in the final path, including the starting and ending points.

Figure 6 shows the motion process of the robotic arm from two views. Under the guidance of our controller, the robotic arm adjusts the linkage position and bypasses the end-effector to avoid collisions.

## 6 Conclusion

Introducing reinforcement learning into the field of robot control remains challenging. This encompasses searching for suitable solutions within complex solution spaces and ensuring smooth robot motions. This paper proposes a reinforcement learning-enhanced pseudo-inverse method for robotic arm control, aiming to maintain the smoothness of robot motions with self-collisions avoided. Although our current work is focuses solely on the inverse kinematics of the end effector position, there is potential for broader applications in the context of redundant robotic arms, such as in human-robot collaboration or medical scenarios.

## Data availability statement

The raw data supporting the conclusions of this article will be made available by the authors, without undue reservation.

## Author contributions

TH: Writing – original draft, Writing – review & editing. WL: Writing – review & editing. KH: Writing – review & editing.

## Funding

The author(s) declare that financial support was received for the research, authorship, and/or publication of this article. This work was supported in part by the National Natural Science Foundation of China under Grant 62206317, in part by the Guangdong Basic and Applied Basic Research Foundation under Grant 2022A1515012186, and in part by the Guangzhou Basic and Applied Basic Research Foundation under Grant 202201011523.

## Conflict of interest

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

## Publisher's note

All claims expressed in this article are solely those of the authors and do not necessarily represent those of

their affiliated organizations, or those of the publisher, the editors and the reviewers. Any product that may be evaluated in this article, or claim that may be made by its manufacturer, is not guaranteed or endorsed by the publisher.

## Supplementary material

The Supplementary Material for this article can be found online at: https://www.frontiersin.org/articles/10.3389/fnbot.2024.1375309/full#supplementary-material

## References


Agarwal, A., Nasa, C., and Bandyopadhyay, S. (2016). Dynamic singularity avoidance for parallel manipulators using a task-priority based control scheme. *Mechan. Mach. Theory* 96, 107–126. doi: 10.1016/j.mechmachtheory.2015.07.013

Al-Hafez, F., and Steil, J. J. (2021). "Redundancy resolution as action bias in policy search for robotic manipulation," in *Conference on Robot Learning* (Atlanta: PMLR), 981–990.

Andrychowicz, M., Wolski, F., Ray, A., Schneider, J., Fong, R., Welinder, P., et al. (2017). "Hindsight experience replay," in *Advance Neural Information Processing System*, 30.

Bing, Z., Brucker, M., Morin, F. O., Li, R., Su, X., Huang, K., et al. (2022a). Complex robotic manipulation via graph-based hindsight goal generation. *IEEE Trans. Neural Netw. Learn. Syst.* 33, 7863–7876. doi: 10.1109/TNNLS.2021.3088947

Bing, Z., Cheng, L., Huang, K., and Knoll, A. (2022b). Simulation to real: learning energy-efficient slithering gaits for a snake-like robot. *IEEE Robot. Autom. Magaz.* 29, 92–103. doi: 10.1109/MRA.2022.3204237

Bing, Z., Knak, L., Cheng, L., Morin, F. O., Huang, K., and Knoll, A. (2023a). "Meta-reinforcement learning in nonstationary and nonparametric environments," in *IEEE Transactions on Neural Networks and Learning Systems*, 1–15.

Bing, Z., Lerch, D., Huang, K., and Knoll, A. (2023b). Meta-reinforcement learning in non-stationary and dynamic environments. *IEEE Trans. Pattern Anal. Mach. Intell.* 45, 3476–3491. doi: 10.1109/TPAMI.2022.3185549

Bing, Z., Zhou, H., Li, R., Su, X., Morin, F. O., Huang, K., et al. (2023c). Solving robotic manipulation with sparse reward reinforcement learning via graph-based diversity and proximity. *IEEE Trans. Industr. Electron.* 70, 2759–2769. doi: 10.1109/TIE.2022.3172754

Boyd, S., and Vandenberghe, L. (2004). *Convex Optimization*. Cambridge: Cambridge University Press.

Cao, H., Chen, G., Li, Z., Feng, Q., Lin, J., and Knoll, A. (2023a). Efficient grasp detection network with gaussian-based grasp representation for robotic manipulation. *IEEE/ASME Trans. Mechatr.* 28, 1384–1394. doi: 10.1109/TMECH.2022.3224314

Cao, H., Chen, G., Li, Z., Hu, Y., and Knoll, A. (2022). Neurograsp: Multimodal neural network with euler region regression for neuromorphic vision-based grasp pose estimation. *IEEE Trans. Instrum. Meas.* 71, 1–11. doi: 10.1109/TIM.2022.3179469

Cao, H., Qu, Z., Chen, G., Li, X., Thiele, L., and Knoll, A. (2023b). Ghostvit: Expediting vision transformers via cheap operations. *IEEE Trans. Artif. Intellig.* 2023, 1–9. doi: 10.1109/TAI.2023.3326795

Cheng, F. T., Lu, Y. T., and Sun, Y. Y. (1998). Window-shaped obstacle avoidance for a redundant manipulator. *IEEE Trans.* Syst. *Man.* Cybern. *B Cybern.* 28, 806. doi: 10.1109/3477.735390

Colomé, A., and Torras, C. (2015). Closed-loop inverse kinematics for redundant robots: Comparative assessment and two enhancements. *IEEE/ASME Trans. Mechatron.* 20, 944–955. doi: 10.1109/TMECH.2014.2326304

Denavit, J., and Hartenberg, R. S. (1955). A kinematic notation for lower-pair mechanisms based on matrices. *J. Appl, Mech*, 22, 215–221. doi: 10.1115/1.4011045

Dereli, S., and Köker, R. (2018). Iw-pso approach to the inverse kinematics problem solution of a 7-dof serial robot manipulator. *Sigma J. Eng. Natural Sci.* 36, 77–85.

Dereli, S., and Köker, R. (2020). A meta-heuristic proposal for inverse kinematics solution of 7-dof serial robotic manipulator: quantum behaved particle swarm algorithm. *Artif. Intellig. Rev.* 53, 949–964. doi: 10.1007/s10462-019-09683-x

Duguleana, M., Barbuceanu, F. G., Teirelbar, A., and Mogan, G. (2012). Obstacle avoidance of redundant manipulators using neural networks based reinforcement learning. *Robot. Comp. Integrat. Manufact.* 2, 28. doi: 10.1016/j.rcim.2011.07.004

Fujimoto, S., Hoof, H., and Meger, D. (2018). "Addressing function approximation error in actor-critic methods," in *Proceedings of the 35th International Conference on Machine Learning*, eds J. G. Dy, and A. Krause (Stockholm: PMLR), 1582–1591. Available online at: http://proceedings.mlr.press/v80/fujimoto18a.html

Guo, Z. Y., and Hsia, T. C. (1990). Joint trajectory generation for redundant robots in an environment with obstacles. *IEEE Trans. Biomed. Eng.* 35, 153–60. doi: 10.1109/ROBOT.1990.125964

Haarnoja, T., Zhou, A., Abbeel, P., and Levine, S. (2018). "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," in *Proceedings of the 35th International Conference on Machine Learning* (Stockholm: PMLR), 1861–1870.

Haviland, J., and Corke, P. (2021). Neo: a novel expeditious optimisation algorithm for reactive motion control of manipulators. *IEEE Robot. Automat. Lett.* 6, 1043–1050. doi: 10.1109/LRA.2021.3056060

He, K., Zhang, X., Ren, S., and Sun, J. (2015). "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," in *Proceedings of the IEEE International Conference on Computer Vision* (Santiago: IEEE), 1026–1034.

James, S., Freese, M., and Davison, A. (2019). Pyrep: Bringing v-rep to deep robot learning. *arXiv*. Available online at: http://arxiv.org/abs/1906.11176

Köker, R., and Çakar, T. (2016). A neuro-genetic-simulated annealing approach to the inverse kinematics solution of robots: a simulation based study. *Eng. Comput.* 32, 553–565. doi: 10.1007/s00366-015-0432-z

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. Indust. Informat.* 8, 18. doi: 10.1109/TII.2021.3125447

Martin, P., and Millán, J. R. (1997). "Combining reinforcement learning and differential inverse kinematics for collision-free motion of multilink manipulators," in *Biological and Artificial Computation: From Neuroscience to Technology, International Work-Conference on Artificial and Natural Neural Networks*, eds J. Mira, R. Moreno-Díaz, and J. Cabestany (Canary Islands: Springer), 1324–1333. doi: 10.1007/BFb0032593

Momani, S., Abo-Hammour, Z. S., and Alsmadi, O. M. (2016). Solution of inverse kinematics problem using genetic algorithms. *Appl. Mathem. Informat. Sci.* 10, 225. doi: 10.18576/amis/100122

Nakamura, Y., and Hanafusa, H. (1986). Inverse kinematic solutions with singularity robustness for robot manipulator control. *J. Dyn. Syst. Meas. Control* 108, 163–171. doi: 10.1115/1.3143764

Paul, R., and Shimano, B. (1978). "Kinematic control equations for simple manipulators," in *IEEE Conference on Decision and Control including the 17th Symposium on Adaptive Processes* (San Diego, CA: IEEE), 1398–1406.

Rokbani, N., and Alimi, A. M. (2013). Inverse kinematics using particle swarm optimization, a statistical analysis. *Procedia Eng.* 64, 1602–1611. doi: 10.1016/j.proeng.2013.09.242

Schappler, M., and Ortmaier, T. (2021). "Singularity avoidance of task-redundant robots in pointing tasks: on nullspace projection and cardan angles as orientation coordinates," in *Proceedings of the 18th International Conference on Informatics in Control, Automation and Robotics*, eds O. Gusikhin, H. Nijmeijer, and K. Madani (SCITEPRESS), 338–349. doi: 10.5220/0010621103380349

Sutton, R. S., and Barto, A. G. (2018). *Reinforcement Learning: An Introduction*. MIT press. Available online at: https://www.worldcat.org/oclc/37293240

Tsai, L. W., and Morgan, A. P. (1985). Solving the kinematics of the most general six- and five-degree-of-freedom manipulators by continuation methods. *J. Mech. Design* 107, 189–200. doi: 10.1115/1.3258708

Wolovich, W., and Elliott, H. (1984). "A computational technique for inverse kinematics," in The *23rd IEEE Conference on Decision and Control* (Las Vegas, NV: IEEE), 1359–1363. doi: 10.1109/CDC.1984.272258

Zaplana, I., and Basanez, L. (2018). A novel closed-form solution for the inverse kinematics of redundant manipulators through workspace analysis. *Mech. Mach. Theory* 121, 829–843. doi: 10.1016/j.mechmachtheory.2017.12.005