IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received January 17, 2021, accepted January 24, 2021, date of publication February 3, 2021, date of current version February 17, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3056903

Motion Planning for Dual-Arm Robot Based on Soft Actor-Critic

CHING-CHANG WONG[1], SHAO-YU CHIEN[1], HSUAN-MING FENG[2], AND HISASUKI AOYAMA[3]

[1]Department of Electrical and Computer Engineering, Tamkang University, New Taipei City 25137, Taiwan
[2]Department of Computer Science and Information Engineering, National Quemoy University, Kinmen County 892, Taiwan
[3]Department of Mechanical and Intelligent Systems Engineering, University of Electro-Communications, Tokyo 182-8585, Japan

Corresponding author: Ching-Chang Wong (wong@ee.tku.edu.tw)

This work was supported in part by the Ministry of Science and Technology (MOST), Taiwan, under Grant MOST 108-2221-E-032-045, Grant MOST 109-2221-E-032-038, and Grant MOST 109-2918-I-032-002.

ABSTRACT In this paper, a motion planning method based on the Soft Actor-Critic (SAC) is designed for a dual-arm robot with two 7-Degree-of-Freedom (7-DOF) arms so that the robot can effectively avoid self-collision and at the same time can avoid the joint limits and singularities of the arm. The left-arm and right-arm of the dual-arm robot each have a neural network to control its position and orientation. Dual-agent training, distributed training structure, and progressive training environment are used to train neural networks. During the training process, the motion of one arm is regarded as the environment of the other arm, and the two agents are trained at the same time. In the input part of the neural network of the proposed method, all parameters come from the angle of each axis and kinematic calculation, no additional sensors are needed, so the method is easier to transplant to different dual-arm robots. With some appropriate neural network inputs and reward functions design, the robot can perform the expected self-collision avoidance and effectively avoid the joint limits and singularities of the arm. Finally, some experiments of the simulation tests in the Gazebo simulator and actual tests in a laboratory-made dual-arm robot are presented to illustrate the proposed SAC-based motion planning method is feasible and practicable in the avoidance of self-collision, joint limits, and singularities.

INDEX TERMS Dual-arm robot, soft actor-critic (SAC), deep reinforcement learning (DRL), motion planning, self-collision avoidance.

I. INTRODUCTION

Small-volume large-variety manufacturing process has become a trend in production. Compared with single-arm robots, dual-arm robots have better adaptability and flexibility. However, a dual-arm robot is like placing two robots close to each other, so the collision of robots needs to be considered. Zhou *et al.* used the concept of a flag, which will be activated when one arm enters a specific area to prevent the other arm from entering the same area [1]. Lam *et al.* used invisible sensitive skin inside the arm, but this method is mainly used to prevent the arm from colliding with the surrounding people [2]. Afaghani and Aiyama proposed a collision-map method for collision detection, which makes one arm to be an obstacle in the path of the other one [3]. There are also methods of using the redundant angle characteristics of the 7-DOF robotic arm to avoid self-collision. The objective function of the optimal trajectory is used to determine the angle of the redundant axis to deal with the problem of self-collision and singularity [4]. Su *et al.* also discussed the redundancy and operability of redundant robots, such as the redundancy optimization control of anthropomorphic robots [5], and the application of redundant robots in medical treatment [6]. Another similar method is to constrain arm movements by formulating potential collisions and joint limits into linear inequalities [7]. Currently, a more mature collision-free method has been developed to plan the trajectory in an off-line manner before the arm starts to move [8]–[12]. The method that is more widely studied today is to generate the repulsion vector by the distance between the links, which is inversely proportional to the distance between the links, thereby changing the trajectory of the arm movement to achieve the effect of self-collision avoidance [13], [14], some studies have added vision, and derived this method to avoid obstacles in the environment at the same time [15], [16].

The associate editor coordinating the review of this manuscript and approving it for publication was Ze Ji.

On the other hand, Machine Learning (ML) has been a very popular topic in recent years, and has been widely used in various fields. In 2012, the Reinforcement Learning (RL) method were used for the robot motion control [17], [18]. In 2013, Mnih *et al.* proposed Deep Q-Network (DQN) [19], which combined RL with the Deep Neural Network (DNN) and achieved breakthrough results. James and Johns used the DQN method to control the rotation angles of the joints of the robotic arm, so that the arm can perform gripping tasks [20]. In 2015, DeepMind proposed the Deep Deterministic Policy Gradient (DDPG) algorithm [21], which improved the problem of poor performance of previous RL methods when the range of output was continuous space, so there are more studies using DDPG for robot control, and all have achieved good results [22]–[24]. Meng *et al.* [25] and Ai *et al.* [26] used a learning-based method in robot control. Su *et al.* proposed an improved Recurrent Neural Network (RNN) scheme to perform the trajectory control of redundant robot [27]. In 2018, Haarnoja *et al.* proposed a Soft Actor-Critic (SAC) algorithm [28]. They used an additional value network to approximate the Q network, which has better performance in high-dimensional models, and use Hindsight Experience Replay (HER) to improve sample efficiency, and the entropy term of SAC makes it have better exploration performance. Due to the rise of ML, the application of robots has made rapid progress. Robots can gradually complete more complex and difficult tasks through ML methods. Su *et al.* proposed a model-free method based on Deep Convolutional Neural Network (DCNN) regression algorithm to achieve tool dynamics identification for bilateral teleoperation [29]. In 2019, Varin *et al.* compared the learning effects of two reinforcement learning methods (Proximal Policy Optimization and Soft Actor-Critic) in four action spaces (torque, joint PD, inverse dynamics, and impedance control). The experimental results showed that different action spaces will have different benefits for learning, and model-based controller modeling can effectively reduce the complexity of the problem [30]. In 2020, Akimov also used SAC to realize the movement control of the legged robot, and split the learning process into two stages. The experimental results showed that the agent after learning in the second stage can also handle the tasks in the first stage [31]. However, no matter what methods are used for motion planning, safety is one of the most important concerns. One is the safety of the people around, and the other is the safety of the robot itself. This paper mainly discusses the latter, that is, how to avoid damage caused by the robot's self-collision. Kim *et al.* proposed a motion planning algorithm that combines TD3 and HER, and applied the algorithm to 2-DOF and 3-DOF manipulators. The experimental results showed that the algorithm can generate smoother and shorter path, but it did not consider collision avoidance, which led to the failure of some tasks [32]. Prianto *et al.* used SAC to provide a path planning method for multi-arm, which can find the shortest path for any starting point and target point in a static environment [33]. Ha *et al.* used SAC to provide a method for learning a closed-loop decentralized multi-arm motion planner. Their experiments demonstrate that the resulting motion planning policy performs well not only in challenging multi-arm motion planning tasks but directly generalizes to tasks with a higher number of arms [34]. In this paper, a SAC-based motion planning method is proposed to control a dual-arm robot with two 7-DOF arms so that while learning to move the arm to the target point, it can also learn to dynamically avoid self-collision, joint limits, and singularities.

The rest of this paper is organized as follows: In Section II, the proposed SAC-based motion planning method and its network structure, reward functions, and training methods are clearly described. In Section III, some simulation test experiments in the Gazebo simulator with a large number of random tasks are presented to illustrate the necessity and effectiveness of these methods. Moreover, a laboratory-made dual-arm robot is used to do some practical experiments to verify that it can indeed run on the actual dual-arm robot. Finally, the conclusions are summarized in Section IV.

II. SAC-BASED MOTION PLANNING METHOD

A motion planning method based on the Soft Actor-Critic (SAC) is designed for a dual-arm robot with two 7-DOF arms. The design goal is to let a dual-arm robot move its arms to a target point while avoiding self-collision, joint limits, and singularities. SAC is one of the Deep Reinforcement Learning (DRL) algorithms. Its training is to update network parameters by maximizing entropy and rewards. Some training methods that help to improve the learning effect of agents are proposed in this paper. Moreover, the selected neural network structure and reward functions are also very important for the network training. Thus, four main parts are presented in this section: (a) deep reinforcement learning, (b) neural network design, (c) reward function, and (d) training method. They are described as follows:

A. DEEP REINFORCEMENT LEARNING

The main concept of Reinforcement Learning (RL) is to learn through the interaction between the agent and the environment. After the agent chooses an action, it will get a corresponding state and a reward from the environment. Through the learning in this continuous interactive process, an optimal policy can be obtained. The traditional reinforcement learning methods usually select the best action based on a lookup table, while the agent of DRL uses a DNN to select actions. This method greatly improves the ability to solve high-dimensional or more complex problems, and is also more applicable in a continuous action space to make an action policy.

In this paper, the left-arm and right-arm of the dual-arm robot are controlled by their respective agents. Therefore, the agent in the DRL is one of the arms and the environment is the entire dual-arm robot. Taking the right-arm as an example, the agent will receive the states and rewards from the environment, and the output control commands to move the arm to the target point, while avoiding collisions, joint limits, and singularities during the movement. The obstacles

of the right-arm to avoid collisions are the body and left-arm of the dual-arm robot.

The SAC algorithm is an off-policy actor-critic algorithm based on the maximum entropy RL framework. There are three neural networks: 1) value network (V): It is used to represent a state value function, 2) soft Q-network (Q): It is used to represent a soft Q-function, and 3) policy network (Π): It is used to represent a policy function. The policy network training of SAC is to update the network parameters by maximizing entropy and reward. The parameters of these three networks can be optimized by the training process that minimizes errors. The pseudocode of the SAC and the symbol table used in the proposed SAC-based motion planning method are respectively shown in Algorithm 1 and Appendix [28]. The parameter training of these three networks is described as follows:

Algorithm 1 Soft Actor-Critic

Input ψ, ϑ_1, ϕ; Initial parameters
$\bar{\psi} \leftarrow \psi$, $\vartheta_2 \leftarrow \vartheta_1$; Set target network weights
$D \leftarrow \sigma$; Initialize an empty replay buffer
for each iteration do
  $a_t \sim \Pi_\phi(a_t|s_t)$; Sample action a_t from
  $s_{t+1} \sim p(s_{t+1}|s_t, a_t)$; Execute a_t and sample next state s_{t+1}
  $D \leftarrow D \cup \{(s_t, a_t, r(s_t, a_t), s_{t+1})\}$; Store the transition in the replay buffer D
  if it's time to update then
   for each gradient step do
   $B = \{(s_t, a_t, r(s_t, a_t), s_{t+1})\}$; Sample a batch B from buffer D
   $\psi \leftarrow \psi - \lambda_V \widehat{\nabla}_\psi J_V(\psi)$; Update value network weights
   $\vartheta_i \leftarrow \vartheta_i - \lambda_\Pi \widehat{\nabla}_{\vartheta_i} J_Q(\vartheta_i)$; for $i \in \{1, 2\}$; Update the Q-function parameters
   $\phi \leftarrow \phi - \lambda_V \widehat{\nabla}_\phi J_\Pi(\phi)$; Update policy weights
   $\bar{\psi} \leftarrow \tau\psi + (1-\tau)\bar{\psi}$; Update target network weights
   end for
  end if
end for
Output ψ, $\bar{\psi}$, ϑ, ϕ; Optimized parameters

In the parameter training of the value network V_ψ, the parameters ψ are trained to minimize the squared residual error. It can be expressed by

$$J_V(\psi) = E_{s_t \sim D}\Big[1/2\,\big(V_\psi(s_t) - E_{a_t \sim \Pi_\phi}\big[Q_\vartheta(s_t, a_t) - \log \textstyle\prod_\phi (a_t \mid s_t)\big]\big)^2\Big] \quad (1)$$

where D is the replay buffer, which contains the distribution of the state and action of the previous sample. The equations for optimizing the parameters ψ can be expressed by

$$\hat{\nabla}_\psi \mathbf{J}_V(\psi) = \nabla_\psi V_\psi(s_t)\Big[V_\psi(s_t) - \Big(Q_\vartheta(s_t, a_t) - \log \textstyle\prod_\phi (a_t \mid s_t)\Big)\Big] \quad (2)$$

Based on the gradient of (1), its purpose is to optimize the parameters of the network to minimize the squared residual error between the output of the value network and the output of the Q-network minus the entropy of the policy function Π_ϕ.

In the parameter training of the soft Q-network Q_ϑ, the parameters ϑ are trained to minimize the soft Bellman residual. It can be expressed by

$$J_Q(\vartheta) = E_{(s_t, a_t) \sim D}\Big[1/2\,\Big(Q_\vartheta(s_t, a_t) - \widehat{Q}(s_t, a_t)\Big)^2\Big] \quad (3)$$

where

$$\widehat{Q}(s_t, a_t) = r(s_t, a_t) + \gamma E_{st+1 \sim p}\big[V_{\overline{\Psi}}(s_{t+1})\big] \quad (4)$$

and the equations for optimizing the parameters ϑ can be expressed by

$$\widehat{\nabla}_\vartheta J_Q(\vartheta) = \nabla_\vartheta Q_\vartheta(s_t, a_t)\big[Q_\vartheta(s_t, a_t) - \big(r(s_t, a_t) + \gamma V_{\bar{\psi}}(s_{t+1})\big)\big] \quad (5)$$

where $V_{\bar{\psi}}$ is an additional value function called the target value function. Based on the stochastic gradients of the loss function described by (3), its purpose is to update the parameters of the network to minimize the error between the output of Q network prediction and the sum of reward and the output of value function.

In the parameter training of the policy network Π_ϕ, the parameters ϕ are trained to minimize the equation which is expressed by

$$J_\Pi(\phi) = E_{st \sim D}\big[\log \Pi_\phi(a_t \mid st) - Q_\vartheta(st, at)\big] \quad (6)$$

and the equations for optimizing the parameters ϕ can be expressed by

$$\widehat{\nabla}_\phi J_\Pi(\phi) = \nabla_\phi \log \Pi_\phi(a_t \mid s_t) + \big[\nabla_{at} \log \Pi_\phi(a_t \mid s_t) - \nabla_{at} Q(s_t, a_t)\big]\nabla_\phi a_t \quad (7)$$

where

$$a_t = f_\phi(\varepsilon_t; s_t) \quad (8)$$

where ε_t is an input noise vector sampled from spherical Gaussian. Based on the approximate gradient of (6), its purpose is to update the parameters of the network to minimize the error between $\log \Pi_\phi(a_t|s_t)$ and the output of Q-network. The algorithm collects experience and current policy from the environment and uses batch random gradients sampled from the buffer to update the function to approximate the target value. It is feasible to use off-policy data from the buffer, because both expectations and policy can be trained entirely on off-policy data and evaluated for state and action.

B. NEURAL NETWORK DESIGN

In the neural network design section, there are three main parts: 1) neural network structure, 2) outputs of policy network, and 3) inputs of policy network. They are described as follows:

1) NEUAL NETWORK STRUCTURE

In the neural network structure of the SAC, there are three main neural networks: value network, soft Q network, and policy network. In this paper, the used network structure and the related hyperparameters are shown in Table 1. All layers are connected in the form of full connection layer. The outputs of the proposed policy network are the variation of the desired position, orientation, and redundant angle of the pose of the robot arm, including 8 parameters, while the output of the value network and the soft Q network is a single parameter. Therefore, the number of neurons in the output layer of the three networks is 8, 1, and 1, respectively. The inputs of the three networks are the robot state and target information, including 55 parameters, and the input of the soft Q network additionally includes the output of the policy network. Therefore, the number of neurons in the input layer of the policy network, soft Q network, and value network are 55, 63, and 55, respectively. In the part of hidden layers, the number of layers used in the policy network, soft Q network and value network are 5, 3, and 3, respectively. The activation function of the hidden layers are the Leaky-ReLU function [35]. The difference from the ReLU function is that Leaky-ReLU can make the neural network less prone to gradient vanishing during the training process [36]. In the part of selecting the activation function of the output layer, if the ReLU function or the Leaky-ReLU function is used, the output will only be a positive value. Since the control command of the arm requires a positive or negative value, the *Tanh* function is selected as the activation function of the output layer of the policy network. And the range of *Tanh* function is $[-1, 1]$, which can prevent the output value of the policy network from being too large and causing the arm to move too fast.

2) OUTPUTS OF POLICY NETWORK

The outputs of the policy network are the variation of the desired position, orientation, and redundant angle of the pose of the robot arm. All output values are in the form of the amount of change per unit time to control the arm. In the orientation part, the quaternion representation is used, thus, there are a total of 8 parameters in these three outputs of the policy network. The definitions of outputs, parameters, and symbols are shown in Table 2. Due to the *Tanh* function, these 8 outputs range from -1 to 1. Thus, the output values are respectively multiplied by a constant term to limit the maximum amount of change per unit time in the position, orientation, and redundant angle. In order to limit the moving speed of the arm to less than 0.6 m/s, the angular velocity of the orientation and the redundant angle are less than 1.25π rad/s, and because the cycle time of the arm is 0.008 s, so these three outputs are multiplied by the constant terms 1/300, 1/100, and 1/100, respectively.

3) INTPUTS OF POLICY NETWORK

The inputs of the policy network are the information that the network can obtain in the environment. Therefore, it is important to properly select some useful information to let the network have a good learning efficiency and result. The main purpose of this paper is to discuss how to effectively move the arm from the current position to the target position and avoid self-collision during the moving process. Therefore, it is necessary to let the network know the current information and target information of the arm, and the relative relationship between the two arms. In addition, because the outputs of the network are the control commands in the workspace, it is necessary to let the network know the joint information, the working reach, and the expected angle of each axis. Thus, the selected input of the policy network mainly includes six items and there are a total of 55 parameters in these six inputs. The definitions of inputs, parameters, and symbols are shown in Table 3. The six inputs are described as follows:

The current information S_{curr} is mainly used to provide information about the current position, orientation, and redundant angle of the arm to the neural network. It is defined by

$$S_{curr} = \left(pos_{curr}, quat_{curr}, \varphi_{curr} \right) \tag{9}$$

where pos_{curr} is the current position of the arm, $quat_{curr}$ is quaternion of the current orientation of the arm, and φ_{curr} is the current redundant angle of the arm. In the orientation part, the interpolated quaternion representation is used to ensure that the rotation of orientation will not encounter the gimbal lock problem [37]. In the redundant axis part, the redundant axis can be effectively used to avoid joint limits and singularities. Therefore, the angle of the redundant axis will be determined by the policy network in this paper. In addition,

TABLE 1. Network structure and hyperparameters of three neural networks of SAC.

Parameter	Policy Network	Soft Q-Network	Value Network
Number of input neurons	55	55+8	55
Number of output neurons	8	1	1
Number of hidden neurons per layer	512	512	512
Number of hidden layers	5	3	3
Activation function of hidden layer	Leaky-ReLU	Leaky-ReLU	Leaky-ReLU
Activation function of output layer	*Tanh*		
Optimizer	Adam		
Learning rate	3×10^{-4}		
Discount (γ)	0.99		
Replay buffer size	10^6		
Target smoothing coefficient (τ)	0.01		

TABLE 2. Outputs used in the policy network.

Output	Parameter and Symbol	Constant
Variation of position	$\Delta P = (\Delta x_p, \Delta y_p, \Delta z_p)$; (m)	1/300
Variation of orientation	$\Delta O = (\Delta w_o, \Delta x_o, \Delta y_o, \Delta z_o)$	1/100
Variation of redundant angle	$\Delta\varphi$ (rad/π)	1/100

TABLE 3. Inputs used in the policy network.

Input (# of parameters)	Parameter and Symbol
Current information of the arm S_{curr} (8 parameters)	position $pos_{curr} = (x_{pc}, y_{pc}, z_{pc})$
	orientation $quat_{curr} = (w_{qc}, x_{qc}, y_{qc}, z_{qc})$
	redundant angle φ_{curr}
Target information of the arm T_{vec} (11 parameters)	position vector $pos_{vec} = (x_{pv}, y_{pv}, z_{pv})$
	orientation vector $quat_{vec} = (w_{qv}, x_{qv}, y_{qv}, z_{qv})$
	reverse orientation vector $quat_{rev} = (w_{qr}, x_{qr}, y_{qr}, z_{qr})$
Collision information between two links C_{indx} (15 parameters)	link collision index $C_{indx} = (c_1, c_2, \cdots, c_{15})$
Joint Information of the arm J_{info} (13 parameters)	joint limit $\theta_{lmt} = (\theta_{lmt1}, \theta_{lmt2}, \cdots, \theta_{lmt7})$
	elbow joint position $P_e = (x_e, y_e, z_e)$
	wrist joint position $P_w = (x_w, y_w, z_w)$
Working reach of the arm R_{lmt} (1 parameter)	working reach index r_{indx}
Expected angle of each axis of the arm at the target θ_{tar} (7 parameters)	target angle of 7 axes $\theta_{tar} = (\theta_{tar1}, \theta_{tar2}, \cdots, \theta_{tar7})$

if the standard deviation between the input parameters is too large, it will cause training difficulties, so the ranges of all the parameter values of S_{curr} are normalized between -1 and 1 in this paper.

The target information T_{vec} is mainly used to provide information about the position and orientation of the arm at the target to the neural network. Compared with the absolute target position and orientation, the relative relationship has a higher correlation with the network output. It can point out the reference direction of the control command to a certain extent. It is defined by

$$T_{vec} = (pos_{rec}, quat_{vec}, quat_{rev}) \tag{10}$$

where pos_{vec}, $quat_{vec}$, and $quat_{rev}$ are respectively the position vector, the orientation vector, and the reverse orientation vector. Among them, the orientation vector is the shortest trajectory in spherical linear interpolation, and the reverse orientation vector is the longer trajectory opposite the shortest trajectory in spherical linear interpolation. The position vector pos_{vec} is defined by

$$pos_{vec} = pos_{curr} - pos_{tar} \tag{11}$$

where pos_{tar} is the target position of the arm. The orientation vector $quat_{vec}$ is defined by

$$quat_{vec} = q_1 - quat_{curr}; \quad q_1 = Slerp(quat_{tar}, quat_{curr}) \tag{12}$$

where q_1 is the first quaternion calculated by the spherical linear interpolation method [38] and $quat_{tar}$ is the quaternion of the target orientation of the arm. The reverse orientation vector $quat_{rev}$ is defined by

$$quat_{rev} = q_{rev} - quat_{curr};$$

$$q_{rev} = Slerp_{rev}(quat_{tar}, quat_{curr}) \tag{13}$$

where q_{rev} is the first quaternion calculated by the reverse spherical linear interpolation method. Since the redundant angle does not affect the position and orientation of the arm, the target information of the arm in this paper does not include the redundant angle.

The collision information C_{indx} is mainly used to provide information about the degree of collision between the links of the two arms to the neural network. The degree of approaching collision is converted from the distance between the links. One arm of a dual-arm robot contains five links, which are the body, shoulder, upper arm, forearm, and end links. There are twenty-five combinations of the distance between each link of the dual-arm robot. However, when considering the collision situation between each link, it can be found that because the left-arm and the right-arm have the same body link, and the position of the shoulder link is fixed. Only the positions of the upper arm link, forearm link, and end link will change during the movement of a single arm, thus it does not need to consider the collision between the body link and shoulder link and other links. Therefore, twenty-five combinations can be reduced to fifteen. Thus, the collision information between two links C_{indx} is defined by

$$C_{indx} = (c_1, c_2, c_3, \cdots, c_{15}) \tag{14}$$

where c_i is the i-th link-collision index and defined by

$$c_i = d_{lmt\,i} - d_{curr\,i} + 1; \quad \text{for } i \in \{1, 2, \cdots, 15\} \tag{15}$$

where d_{lmt} is the minimum limit distance between two links and d_{curr} is the current distance between two links. The defined minimum limit distance between two links of each group is shown in Table 4. The difference in this value is due to different links with different diameters. If the distance between two links is less than this limit, it will be regarded as a collision event. The closer the value of the link-collision index is to 1, the higher the chance of collision between two links.

The joint information J_{info} is mainly used to let the neural network learn to avoid exceeding the joint limits during movement, and it also helps the neural network to avoid reaching the singularities of the arm during the motion planning process. It can be expressed by

$$J_{info} = (P_e, P_w, \theta_{lmt}) \tag{16}$$

where P_e, P_w, and θ_{lmt} are respectively the elbow joint position, the wrist joint position, and the index of joint limit. The purpose of the elbow and wrist joints as the inputs is to enable the neural network to understand how different orientation and redundant angle changes during movement will affect the position changes of the links. The i-th index of joint limit is defined by

$$\theta_{lmt\,i} = \left(2 \times \frac{\theta_{curr\,i} - \theta_{min\,i}}{|\theta_{max\,i} - \theta_{min\,i}|} - \theta_{mu\,i}\right); \quad \text{for } i \in \{1, 2, \cdots, 7\} \tag{17}$$

TABLE 4. Minimum distance (m) between two links of each group.

	Body Link	Shoulder Link	Upper Arm Link	Forearm Link	End Link
Upper Arm Link	d_{lmt1} = 0.16	d_{lmt2} = 0.16	d_{lmt3} = 0.14	d_{lmt4} = 0.14	d_{lmt5} = 0.12
Forearm Link	d_{lmt6} = 0.16	d_{lmt7} = 0.14	d_{lmt8} = 0.14	d_{lmt9} = 0.14	d_{lmt10} = 0.12
End Link	d_{lmt11} = 0.14	d_{lmt12} = 0.12	d_{lmt13} = 0.12	d_{lmt14} = 0.12	d_{lmt15} = 0.10

where $\theta_{min\ i}$ and $\theta_{max\ i}$ are respectively the minimum limit angle and maximum limit angle of each axis.

The working reach r_{indx} is mainly used to let the neural network learn to avoid exceeding the working reach of the arm during movement. It is defined by

$$r_{indx} = \left(\frac{\|P_w - P_s\|}{r_{lmt}} \right) \tag{18}$$

where P_w and P_s are respectively the current positions of the wrist joint and shoulder joint, and r_{lmt} is the working reach limit.

The target angle θ_{tar} of each axis is mainly used to provide the neural network to compare with the current angle of each axis. It is defined by

$$\theta_{tar} = (\theta_{tar1}, \theta_{tar2}, \cdots, \theta_{tar7}) \tag{19}$$

where $\theta_{tar\,i}$ is the target angle of i-th axis. These 7 input values are the angles of each axis obtained by the inverse kinematics solution under the condition that the redundant angle is zero degree at the target point.

C. REWARD FUNCTIONS

In the reward functions of the proposed method, there are two main parts: 1) dense reward: It guides the neural network to learn how to control the movement of the arm to the target point, and 2) sparse reward: It guides the neural network to learn how to avoid self-collision, joint limits, and singularities. They are described as follows:

1) DENSE REWARD

The dense reward is to give a reward value for each output (action) of the neural network (agent). The purpose of the dense reward defined in this paper is mainly used to guide the learning of the neural network to move the arm to the position and orientation of the target. The defined dense reward R_d is expressed by

$$R_d = R_{pos} + R_{ang} + R_{quat} - 1 \tag{20}$$

where R_{pos}, R_{ang}, and R_{quat} are a position reward, an angle reward, and an orientation reward, respectively. The dense reward is the sum of the three rewards of position reward, angle reward, and orientation reward plus a constant term "-1". The purpose of this constant term is to enable the neural network to learn effectively. The position reward R_{pos} is defined by

$$R_{pos} = -\left\| pos_{tar} - pos_{curr} \right\| \tag{21}$$

where pos_{curr} and pos_{tar} are respectively the current position and target position of the arm. Equation (21) directly takes the negative value from the Euclidean distance of these two positions, so if the current position is closer to the target position, the value of the position reward is larger. However, if the reward is calculated simply by the distance in space, it means that a sphere is drawn with the target position as the center, and the reward value at any point on this sphere is the same. Therefore, the angle reward R_{ang} is also added to the dense reward to consider the direction of the movement vector pos_{vec} of the arm. The angle reward R_{ang} is defined by

$$R_{ang} = -\frac{R_{pos} \times \cos\theta_v + R_{pos}}{2} \tag{22}$$

where

$$\cos\theta_v = \frac{pos_{vec} \cdot (pos_{tar} - pos_{curr})}{\|pos_{vec}\| \, \|pos_{tar} - pos_{curr}\|} \tag{23}$$

where θ_v is the angle between the arm's movement vector pos_{vec} and the vector from the starting point to the target point. When the arm is closer to the target position, the value of R_{pos} will be closer to zero, and the cosine value of the angle θ_v does not have this characteristic. This will cause that the impact of distance reward will gradually decrease when the arm is closer to the target position, while the impact of angle reward will be expanded. Therefore, through the calculation of (22), the angle reward R_{ang} can be changed from the original cosine range of $[-1, 1]$ to the range of $[0, R_{pos}]$, so that it can be balanced the position reward R_{pos} and the angle reward R_{ang} both affect the online learning in terms of distance and angle. Finally, the orientation reward R_{quat} is defined by

$$\begin{aligned} R_{quat} &= -\left\| quat_{curr} - quat_{tar} \right\| + \left\| quat_{curr} - quat_{optar} \right\| - 2; \\ quat_{optar} &= -quat_{tar} \end{aligned} \tag{24}$$

where $quat_{curr}$ and $quat_{tar}$ are respectively the current quaternion and target quaternion of the arm. $quat_{optar}$ is the opposite vector of $quat_{tar}$. However, two quaternions opposite to each other represent the same orientation, which means that $quat_{tar}$ and $quat_{optar}$ are both target quaternions of the arm, so it is necessary to consider the distance between the current quaternion and these two target quaternions at the same time. Since the Euclidean distance of two target quaternions is 2, the range of the sum of the distance between any quaternion and these two target quaternions is $[2, 2\sqrt{2}]$, then add the constant term "-2" to make its value range become $[0, 2\sqrt{2}-2]$. Finally, it takes a negative value to indicate that when the current quaternion is closer to the target quaternion, the value of the orientation reward R_{quat} is the larger.

2) SPARSE REWARD

The sparse reward is to give a reward value given by the environment only under certain conditions. The purpose of the sparse reward defined in this paper is to guide the learning of the neural network to avoid the problems of self-collision, joint limits, and singularities, so the sparse rewards in this paper include four items: (a) self-collision occurs; (b) joint limit is exceeded, (c) working reach is exceeded, and (d) near singularity. The reward value is set to -8 when the above situation occurs. Among them, the definition of self-collision is the same as (15). When c_i is greater than 1, it means that a collision occurs. The definition of exceeding the joint limit is the same as (17). When $\theta_{\text{lmt i}}$ is greater than 1 or less than -1, it means that the joint limit is exceeded. The definition of exceeding working reach is the same as (18). When r_{indx} is greater than 1, it means that the working reach is exceeded. And the definition of near the singularity is defined as the case where the angular velocity of any axis exceeds π rad/s.

D. TRAINING METHODS

The proposed method for the dual-arm robot has two agents, which control the left-arm and the right-arm, respectively. Since the training process of the two neural networks is the same, the network training flowchart of the SAC using the right-arm as an example is shown in Fig. 1. There are three training methods in the training process: 1) dual-agent training, 2) distributed training architecture, and 3) progressive training environment. They are described as follows:

1) DUAL-AGENT TRAINING

The main purpose of the dual-agent training is to train two neural networks simultaneously in one environment [39]. The flowchart is shown in Fig. 2. At the beginning of each round, the environment will randomly generate a set of starting points and target points. The agent will output control commands to the controller, and the controller will control the rotation of each axis motor to move the arm according to the joint angles obtained by kinematics, and then the current state of the arm will be used to make a new environment state.

Because the left-arm and the right-arm are learning at the same time, and the randomly generated starting point will cause the arm to have a large displacement in an instant. This phenomenon does not happen in actual situations. Therefore, when one of the arms reaches the target point, it must wait for the other arm to reach the target point, and then the new starting points and target points of the two arms will be regenerated for a new training round. This method can ensure that the status information of the two arms is continuous during each round of training, so that the networks of the two arms can correctly learn at the same time.

2) DISTRIBUTED TRAINING ARCHITECTURE

The main purpose of the distributed training architecture is to quickly accumulate more empirical data and improve the efficiency of network training [40]. The distributed training

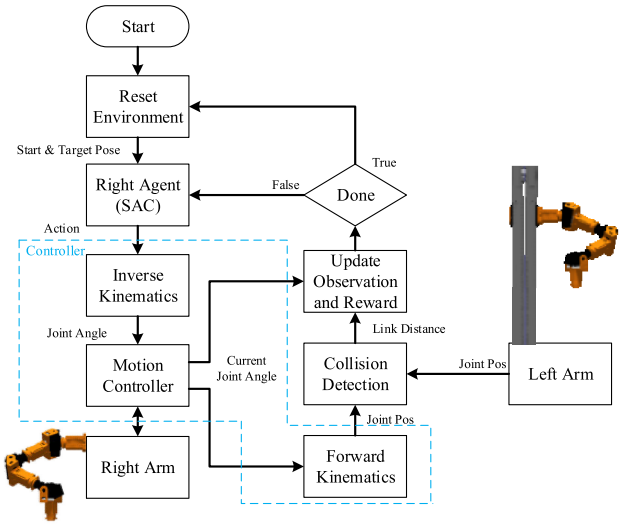

FIGURE 1. Network training flowchart of the SAC to avoid self-collision, joint limits, and singularities for the right-arm.

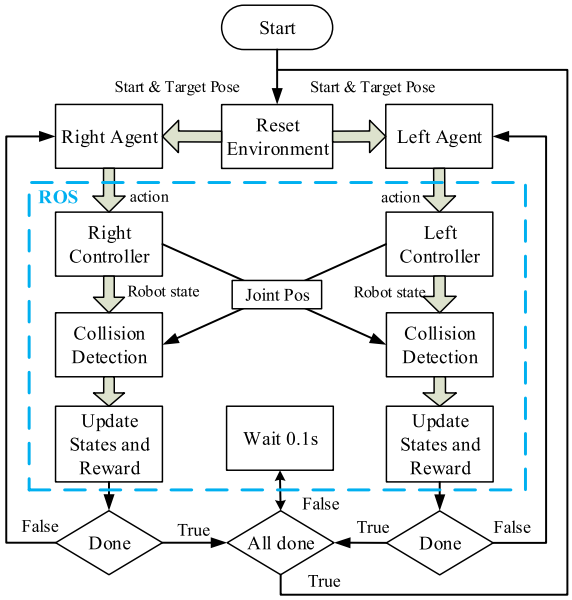

FIGURE 2. Flowchart of the dual-agent training for the motion planning.

architecture used in this paper is shown in Fig. 3. Similarly, it is implemented in a multiple thread manner. An agent interacts with four environments at the same time during the network training, and each environment uses a different random seed to randomly generate starting points and target points. The empirical data of the four environments are respectively stored in four buffers as training samples, and then samples are sequentially extracted from the four buffers to train the neural network. Since each movement of the arm is continuous and related, the empirical data generated by each group of the dual-arm robot are respectively stored in different registers. This method can ensure that the neural network extract data from a buffer, it will not get empirical

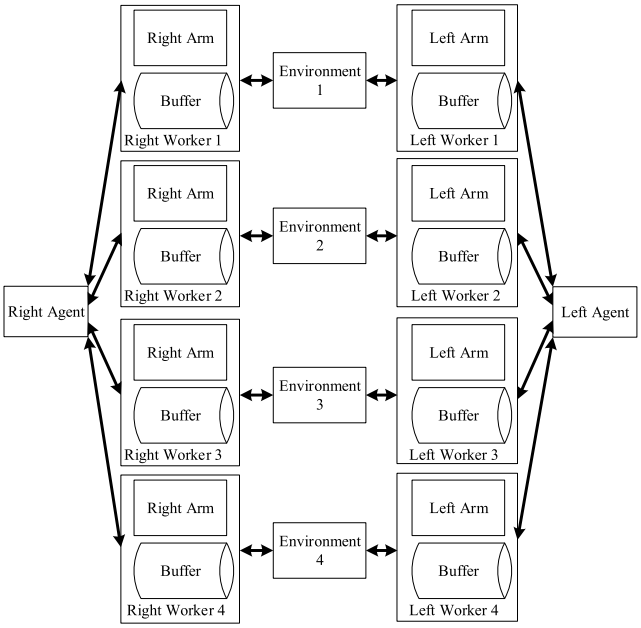

FIGURE 3. Distributed training architecture of the dual-agent training for the dual-arm robot.

data from different dual-arm robots. In the four environments, the arms between different environments will not affect each other, which means that each arm will only interact with arm agent in the same environment.

3) PROGRESSIVE TRAINING ENVIRONMENT

The main purpose of the progressive training environment is to effectively train the network. It takes an average of several hundreds of times to move the arm from the starting point to the target point. It is quite difficult at the beginning of the training, so a way that gradually increases the difficulty of completing the task is used to train the network. At first, the training environment is set up to make it easy for the network to complete the task. When the number of completed tasks gradually increases, slowly increase the difficulty of the environment to effectively train the network. There are two main ways: (i) conditions for completing the task and (ii) ranges of the randomly generated starting point and target point. They are described as follows:

In the part of conditions for completing the task, the Euclidean distance between the current position (orientation) and the target position (orientation) of the arm is used to make a judgment whether the neural network completed a task. In order to make the neural network easier to obtain positive rewards, a larger error from the target point is given at the beginning of the training. It makes the neural network easier to learn how to complete the task, and then gradually reduce this error to increase difficulty. The initial value of the error from the target point is 0.08 meters. Each time the task is successfully completed, the distance is multiplied by 0.993 until it is reduced to 0.01 meters. And the initial value of the Euclidean distance error between the current quaternion and the target quaternion is 0.4 units. Each time the task is successfully completed, the distance is multiplied by 0.993 until it is reduced to 0.2 units.

In the randomly generated range of the starting point and target point, due to the limited working reach of the arm, the randomly generated direction may cause the angle of each axis to exceed its joint limit. Therefore, seven random values are used to generate the angle θ_i of the seven axes of the arm, and then the corresponding information needed for the network training is calculated from the kinematics. The angle θ_i of each axis can be expressed by

$$\theta_i = \mathrm{ran}_i \times S \times g_i + \mathrm{mu}_i; \quad \text{for i} \in \{1, 2, \cdots, 7\} \tag{25}$$

where ran_i is a random value from -0.5 to 0.5, S is a random angle range control item, g_i and mu_i are respectively expressed by

$$g_i = |\theta_{\max\ i} - \theta_{\min\ i}| \tag{26}$$

and

$$\mathrm{mu}_i = \frac{\theta_{\max\ i} + \theta_{\min\ i}}{2} \tag{27}$$

This method randomly generates an angle within the joint limit of each axis. The larger the random range S, the more difficult the task to be completed. The value of S is set to 0.5 at the beginning of the training, and each time the task is completed, the random angle range control item S is increased by 0.004 until the value of S increases to 0.95. For example, if the limit of the first axis is 180, the maximum random range of the first axis is 171 during training. This is because the output of the policy network has a certain degree of randomness during training, so the starting or target point too close to the joint limit will affect the effect of learning. And a small S value at the beginning of training can make the arm in a safer state that is less likely to encounter collision and joint limit problems.

III. EXPERIMENTAL RESULTS

In the experiment, there are two main parts: (a) simulation test experiment and (b) actual test experiment. They are described as follows:

A. SIMULATION TEST EXPERIMENT

The main purpose of this experiment is to verify the effectiveness of the proposed SAC-based motion planning method for the dual-arm robot and the necessity of various network inputs and reward functions used in the training modes. In this verification process, the Gazebo simulator is used and a large number of random tasks are used to test the proposed method, and these test results will be compared with the linear trajectory planning method in workspace, the trajectory planning method in joint space, and Decentralized Multi-arm Motion Planner (DMAMP) proposed by Ha *et al.* [34]. Among them, only the experimental results of DMAMP method are used for comparison, the other three methods use the same random seed to generate 1,000 random tasks in the experiment. The

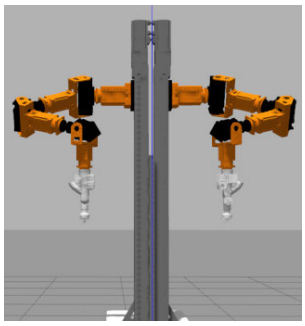

FIGURE 4. Description of the dual-arm robot constructed in Gazebo simulator.

dual-arm robot constructed in the Gazebo simulator is shown in Fig. 4. It mainly compares the occurrence probability of the following three items: self-collision, out of joint limits, and near the singularities. The test experiments are mainly divided into four parts: 1) completely random task, 2) random positions and fixed orientation, 3) collision-prone zone test, and 4) different network inputs and reward functions. They are described as follows:

1) COMPLETELY RANDOM TASK

In this experiment of completely random task, the purpose is to test all possible tasks within the working reach of the dual-arm robot. The starting point and target point of the completely random task are generated through (25) to generate random angles of each axis as in training, and the value of S is equal to 1, which means that the area of the completely random task is equal to the working reach of the arm. In the completely random task, the overlap ratio of the working reach of the two arms is about 0.3, so it is compared with the data obtained by the DMAMP method in easy tasks [34]. As shown in Table 5, the occurrence probabilities of three problems in the four methods are compared. Since the trajectory planning method in the joint space uses the starting angle and the target angle of each axis to plan the speed of the motor, and the output of the DMAMP method is the motion planning of each axis, the problem of joint limit and singularity does not occur in these two methods. From the experimental data described in Table 5, it can be found that in the self-collision item, the proposed method has a significantly lower occurrence probability than the other three methods. In the two items of joint limit and singularity, the performance of the proposed method is also superior to the linear trajectory planning method in workspace. In this experiment, the definition of near the singularity is defined as the case where the angular velocity of any axis exceeds π rad/s. From the experimental data in Table 5, it can be found that due to different orientation changes, it is frequent for the arm to produce a task that crosses the shoulder coordinate quadrant or the wrist coordinate quadrant in a completely random task, so there are still some cases of joint limits and singularities.

TABLE 5. Comparison of three methods in the experiment of completely random task.

Method/Item	Self-Collision	Joint Limit	Singularity
Trajectory Planning Method in Joint Space	13%		
Trajectory Planning Method in Workspace	25.1%	34.7%	27.4%
DMAMP Method in Easy Tasks [34]	4.1%		
Proposed Motion Planning Method	1.6%	1.3%	7.4%

TABLE 6. Comparison of three methods in the experiment of random position and fixed orientation.

Method/Item	Self-Collision	Joint Limit	Singularity
Trajectory Planning Method in Joint Space	11.2%		
Trajectory Planning Method in Workspace	25.4%	11.5%	23.6%
Proposed Motion Planning Method	0.3%	0.4%	5.4%

2) RANDOM POSITIONS AND FIXED ORIENTATION

In this experiment of random position and fixed orientation, the purpose is to test some simple scenario to verify the effectiveness of the proposed method. Compared with the completely random task, the random position and fixed orientation experiments has no orientation changes. The fixed orientation means that the Euler angles of the starting point and the target point of all test tasks are set to zero degrees, that is, roll = pitch = yaw = 0. The random position is the position of the wrist joint obtained from the angle randomly generated by the front four-axis motor of the arm, and based on this position, combine with a fixed orientation to obtain the random starting and target point. In this experiment, the occurrence probabilities of three problems in the three methods are compared, as shown in Table 6. From the experimental data described in Table 5 and Table 6, it can be found that since the test scenario is simpler than the completely random task, the occurrence probability of the three items by the proposed method has also decreased significantly, and the occurrence probability of the self-collision is only 0.3%. However, the other two methods did not greatly improve in these simpler tasks.

3) COLLISION-PRONE ZONE TEST

In this experiment of collision-prone zone, the purpose is to test random tasks in the zone where the task area of the two arms are completely overlapped to verify the effectiveness of the proposed method in an environment with a high risk of self-collision. The collision-prone zone is defined as an area centered on the body of the dual-arm robot and extending 20 cm to the left and right sides. That is, with the origin of the Y-axis as the center, the range of 20 cm each extends toward the positive axis and the negative axis. Similarly, this

TABLE 7. Comparison of three methods in the experiment of collision-prone zone test.

Method/Item	Self-Collision	Joint Limit	Singularity
Trajectory Planning Method in Joint Space	47.8%		
Trajectory Planning Method in Workspace	56.8%	23.5%	32.9%
DMAMP Method in Hard Tasks [34]	12.4%		
Proposed Motion Planning Method	2.9%	1.3%	5.0%

experiment is tested by 1,000 sets of random tasks, including random positions and random orientations. To ensure that all the random tasks are within the collision-prone zone, if the randomly generated starting point or target point is not within the defined zone, a new random task is regenerated until both are within the defined zone. Since the task areas of the two arms are completely overlapped, it is compared with the data obtained by the DMAMP method in hard tasks [34]. As shown in Table 7, the occurrence probabilities of three problems in the four methods are compared. Since this experiment limits all the starting point and the target point are in the zone where the task areas of two arms are completely overlapped, the occurrence probability of collision between the two arms will be higher and the occurrence probability of collision with the robot body also increases. From the experimental data described in Table 5 and Table 7, it can be found that in the self-collision item, all the occurrence probabilities of the four methods are increased. But in this experiment, the proposed method is still significantly better than the other three methods.

4) DIFFERENT NETWORK INPUTS AND REWARD FUNCTIONS

In this experiment of different network inputs and reward functions, the purpose is to verify the necessity of designing network inputs and the effectiveness of reward function. There are five main items for this comparison and verification: (i) network inputs without considering J_{info} (joint information), (ii) network inputs without considering R_{lmt} (working reach index), (iii) network inputs without considering θ_{tar} (each axis angle of the target), (iv) network inputs without considering $quat_{rev}$ (quaternion of reverse spherical linear interpolation), and (v) reward function without considering the constant term. The completely random task of the random position and random orientation is used to test in this experiment. Re-train the above five different network training mode, and also train 12,000 rounds. The comparison of the occurrence probability in the three items of self-collision, joint limit, and singularity is shown in Table 8. From the comparison of the experimental data, it can be seen that the reward function without considering the constant term is the most obvious effect on the self-collision, joint limit, and singularity. In the comparison of the four items that without considering the network input, the working reach index R_{lmt}, the joint information J_{info}, and the quaternion of reverse spherical linear interpolation $quat_{rev}$ have the largest impact on the self-collision, joint limit, and singularity, respectively. These results can verify that the proper selection of network inputs and reward functions have a great influence on the results of network training, and the proposed method does have a good effect on avoiding the problem of self-collision, joint limit, and singularity.

TABLE 8. Comparison of training modes of the proposed method with different network inputs and reward functions.

Network inputs or reward functions used in training mode	Self-Collision	Joint Limit	Singularity
Network inputs without considering J_{info}	10.1%	6.3%	38.7%
Network inputs without considering R_{lmt}	13.2%	11.7%	22.6 %
Network inputs without considering θ_{tar}	2.4%	5.1%	21.8%
Network inputs without considering $quat_{rev}$	3.2%	13.5%	18.7%
Reward functions without considering the constant term	18.4%	17.2%	43.7%
All network inputs and reward functions are considered	1.6%	1.3%	7.4%

B. ACTUAL TEST EXPERIMENT

The main purpose of this experiment is to verify the effectiveness of the proposed motion planning method applied to the actual dual-arm robot. An experimental scenario is designed and shown in Fig. 5. There are three areas named A, B, and C. In the C area, some white round objects are stacked in three positions named a, b, and c. In this experiment, one of the three positions a, b, and c in the C area is respectively selected to be the target position of the right-arm and the left-arm in a random way, and then let the right-arm and the left-arm respectively suck-and-place the object at the selected target position to the A area and the B area. The purpose is to verify that the proposed motion planning method can effectively avoid collisions when the right-arm and left-arm of the robot are performing the task of sucking and placing objects at the same time. In the experimental design, if the placed distances between the three positions a, b and c are too close to each other, the motion planning method will make the dual-arm robot only have one arm to suck object at a time, that is, two arms take turns to suck objects. Therefore, the placed distance between the three positions in this experiment satisfies the following two conditions: (i) The distance is large enough so that the left-arm and the right-arm are usually very close when sucking objects at the same time. For example, when the left-arm sucks the object at the "b" position, the right-arm can suck the object at the "a" position at the same time. (ii) The distance is small enough so that the left-arm and the right-arm cannot simultaneously suck objects far away from themselves. For example, when the left-arm sucks the object at the "a" position, the right-arm cannot suck the object at the

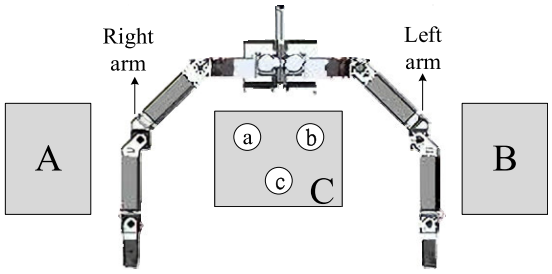

FIGURE 5. Experimental scenario of the actual dual-arm robot testing.

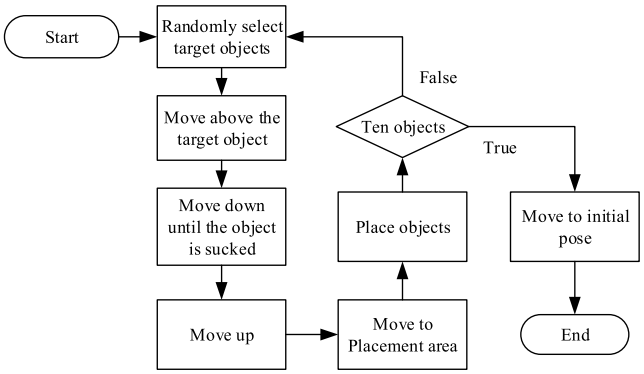

FIGURE 6. Flow chart of the actual dual-arm robot testing.

"b" position, otherwise the two arms will collide. This design can test that the proposed method can indeed make the robot have the ability to avoid self-collision, and the two arms can still suck objects when the distance between the two arms is relatively small.

The test process of this experiment is shown in Fig. 6. The left-arm and the right-arm will first randomly select a target position respectively, and the proposed motion planning neural network is used to control each arm to move above the selected target position. Then, the linear trajectory planning method in workspace is used to move the arm down and suck the object at the target position with the suction cup. After confirming that the object is sucked, raise the arm, and use the motion planning neural network to control the arm to the placement area A or B to place the object. If 12 objects have been sucked and placed, the test is completed and the two arms move back to their initial positions. For detail, one experimental video of the actual test of the proposed motion planning method is shown in https://www.youtube.com/watch?v=nFY-CMmZpXA. The experimental results are photographed in Fig. 7, where the blue dot and the red dot respectively represent the current target points of the left-arm and the right-arm, including the position of the target object or the location of the placement area. The movement states of the left-arm and right-arm in each picture of Fig. 7 are described as follows:

(a) The initial positions of the right-arm and the left-arm.
(b) The new target objects randomly selected by the right-arm and left-arm are the same object at the "a" position.
(c) The left-arm and right-arm simultaneously move to above the "a" position.
(d) Since the right-arm is closer to the "a" position, the left-arm performs the self-collision avoidance function to let right-arm suck the object at the "a" position first.
(e) After the right-arm sucks the object, the target position of the right-arm is changed to the "A" area.
(f) The right-arm places the sucked object to the "A" area and the left-arm moves to above the "a" position.
(g) The new target objects randomly selected by the right-arm and the left-arm are objects at the "b" position and the "c" position, respectively.
(h) The left-arm and the right-arm simultaneously move to above the target position selected by each other.
(i) Because the left-arm and the right-arm must be staggered to suck the target object, and the right-arm is closer to the "b" position, the left-arm performs a self-collision avoidance function to let the right-arm suck the object at the "b" position first.
(j) The right-arm places the sucked object to the "A" area, and the left-arm moves to above the "c" position.
(k) After the left-arm sucks the object, the target position of the left-arm is changed to the "B" area.
(l) The new target objects randomly selected by the right-arm and the left-arm are objects at the "b" position and the "a" position, and the two arms move to above the selected position, respectively.
(m) The left-arm performs the self-collision avoidance function to avoid collision with the right-arm.
(n) The right-arm completes the suck-and-place task.
(o) The left-arm completes the suck-and-place task.

Since the selected target objects are in a random manner, the target objects selected by the left-arm and the right-arm may be the same. For example, as shown in Fig. 7 (b), the target objects selected by the two arms are the same. Another situation is that the selected target object needs to interlace two arms to simultaneously suck the object. For example, as shown in Fig. 7 (l), the left-arm wants to suck the object at the "a" position and the right-arm wants to suck the object at the "b" position. When the dual-arm robot performs the task of sucking and placing objects with both arms, if the proposed method is not used, there is a very high probability that the left-arm and the right-arm will collide. Therefore, the test results of this experiment can verify that the proposed method has good execution results in the task of sucking and placing objects, and verify that it can indeed run on the actual dual-arm robot.

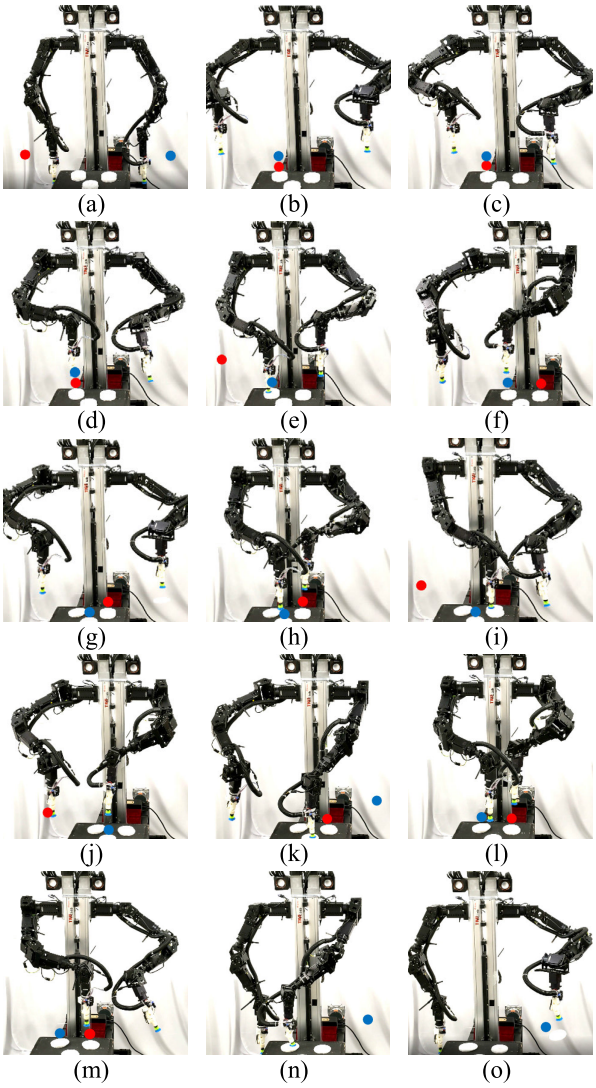

FIGURE 7. Experimental snapshots of the proposed motion planning method for the dual-arm robot.

On the other hand, it can be found from the experimental results that if the two arms will collide in the moving process, one arm will automatically reserve space for the other arm to avoid collision. The self-collision avoidance result of the motion planning method for the dual-arm robot is shown in Fig. 8. The movement states of the left-arm and right-arm in each picture are described as follows:

(a) The new target objects randomly selected by the right-arm and left-arm are the same object at the "c" position.
(b) The left-arm and right-arm simultaneously move towards to above the "c" position.
(c) The two arms tend to collide with each other. Since the left-arm is closer to the "c" position, the right-arm stops moving forward at this time.
(d) The left-arm moves above the "c" position and the right-arm keeps waiting.

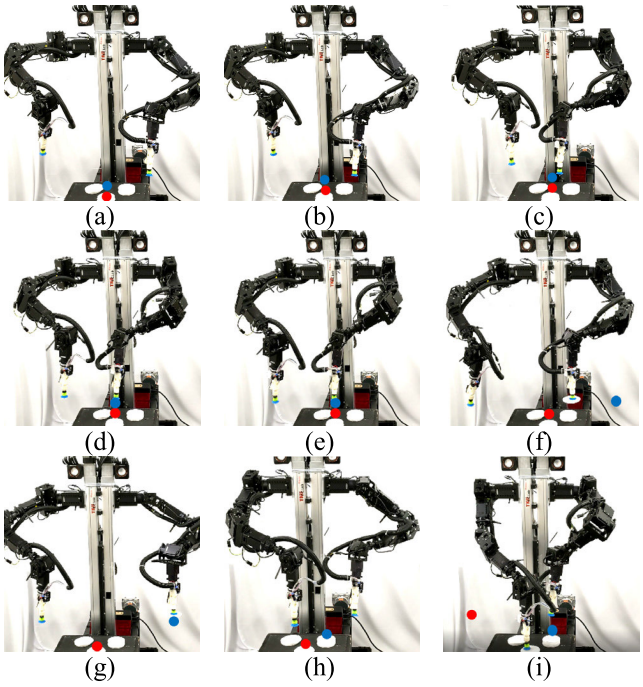

FIGURE 8. Snapshots of avoidance results of the proposed motion planning method for the dual-arm robot.

(e) Although the target point of the right-arm is above the "c" position, the right-arm moves slowly backward to avoid the collision because the left-arm is sucking the object at the "c" position.
(f) The left-arm moves toward the placement area "B".
(g) After the left-arm leaves, the right-arm moves to above the "c" position.
(h) The right-arm moves to above the "c" position, while the left-arm moves to above the "b" position (the two arms do not interfere with each other).
(i) The two arms perform sucking tasks at the same time.

If the left-arm and the right-arm will not interfere with each other in the process, the two arms can also perform the suck-and-place task at the same time without waiting. The process is shown in Fig. 8 (g) to Fig. 8 (i). In the entire task of sucking and placing objects, it is not necessary to set waiting points through a strategy. Instead, the proposed method is used to enable the dual-arm robot to autonomously decide when it needs to wait, move back, or move directly to the target point.

IV. CONCLUSION

A SAC-based motion planning method is proposed to make the dual-arm robot avoid self-collision, joint limits, and singularities. There are five main contributions in this research. First of all, in the motion planning part, a real-time method is proposed so that the implemented control network of one arm of the dual-arm robot can control the arm without considering the trajectory of the other arm. The proposed method does not need to plan the trajectories of

TABLE 9. Symbol table of the proposed SAC-based method.

Symbol	Definition
a_t	action at *t*-th control step (the output of policy network)
D	a buffer of previously sampled states, actions, and rewards.
E	expected function
ε_t	noise vector from spherical Gaussian
$f_\phi(\varepsilon_t; s_t)$	reparameterization from policy
γ	discount factor
$J_V(\psi)$	loss function of the value network
$J_Q(\vartheta_i)$	loss function of the soft q network
$J_\Pi(\phi)$	loss function of the policy network
$\log\Pi_\phi(a_t \mid s_t)$	entropy of the policy output
λ_V	learning rate of the value network
λ_Q	learning rate of the soft q network
λ_Π	learning rate of the policy network
$\hat{\nabla}_\psi$	partial differential of ψ
$\hat{\nabla}_{\vartheta_i}$	partial differential of ϑ
$\hat{\nabla}_\phi$	partial differential of ϕ
$\hat{\nabla}_\psi J_V(\psi)$	gradient of J_V
$\hat{\nabla}_{\vartheta_i} J_Q(\vartheta_i)$	gradient of J_Q
$\hat{\nabla}_\phi J_\Pi(\phi)$	gradient of J_Π
p	state transition function
$p(s_{t+1} \mid s_t, a_t)$	probability distribution of s_{t+1} under condition s_t and a_t
Π_ϕ	policy network
$\Pi_\phi(a_t \mid s_t)$	probability distribution of a_t under condition s_t
ϕ	parameters of the policy network
ψ	parameters of the value network
$\bar{\psi}$	exponentially moving average of the value network weights (the parameters of target value network)
Q_ϑ	Q network
r	reward
$r(s_t, a_t)$	reward function
s_t	state at *t*-th control step (the state of environment)
$s_{t+1} \sim p(s_{t+1} \mid s_t, a_t)$	sample next state s_{t+1} from the probability distribution of s_{t+1} under condition s_t and a_t
ϑ	parameters of the soft Q network
τ	target smoothing coefficient
V_ψ	value network
$V_{\bar{\psi}}$	target network
(s_t, a_t, r_t, s_{t+1})	(state, action, reward, next state)

the two arms in advance before performing the action so that the left-arm and the right-arm can be easily operated independently. Therefore, the proposed method allows the dual-arm robot to have higher flexibility. Secondly, in the part to transplant to other systems, the proposed method can simultaneously consider the collision problem between all links, without calculating the corresponding virtual repulsive force or the gradient of the minimum distance function between all links. Therefore, the proposed method can be more easily transplanted to different robot systems. Thirdly, in the neural network training part, three training methods of the dual-agent training, distributed training architecture, and progressive training environment are used to let the neural network training effectively. Fourthly, in the simulation training part, a 3D simulation environment is established in the Gazebo simulator to train the neural networks of the left-arm and the right-arm of the dual-arm robot. The proposed method can directly map the control commands of the arm according to various information calculated by the joint angles of the arm. Therefore, without adding additional sensors, the proposed method can let the trained network achieve the purpose of avoiding self-collision, joint limits, and singularities. Finally, in the part to verify the practicability and feasibility of the proposed method, a complete test on a real laboratory-made dual-arm robot is conducted. It can be seen from the experimental results that the proposed method can effectively let the dual-arm robot avoid the self-collision, joint limits, and singularities.

APPENDIX

See Table 9.

REFERENCES

[1] J. Zhou, K. Nagase, S. Kimura, and Y. Aiyama, "Collision avoidance of two manipulators using RT-middleware," in *Proc. IEEE/SICE Int. Symp. Syst. Integr.*, Kyoto, Japan, Dec. 2011, pp. 1031–1036.

[2] T. L. Lam, H. W. Yip, H. Qian, and Y. Xu, "Collision avoidance of industrial robot arms using an invisible sensitive skin," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, Vilamoura, Portugal, Oct. 2012, pp. 4542–4543.

[3] A. Y. Afaghani and Y. Aiyama, "On-line collision avoidance between two robot manipulators using collision map and simple escaping method," in *Proc. IEEE/SICE Int. Symp. Syst. Integr.*, Kobe, Japan, Dec. 2013, pp. 105–110.

[4] Y. Chua, K. P. Tee, and R. Yan, "Robust optimal inverse kinematics with self-collision avoidance for a humanoid robot," in *Proc. IEEE RO-MAN*, Gyeongju, South Korea, Aug. 2013, pp. 496–502.

[5] H. Su, W. Qi, Y. Hu, H. R. Karimi, G. Ferrigno, and E. De Momi, "An incremental learning framework for human-like redundancy optimization of anthropomorphic manipulators," *IEEE Trans. Ind. Informat.*, early access, Nov. 9, 2020, doi: 10.1109/TII.2020.3036693.

[6] H. Su, C. Yang, G. Ferrigno, and E. De Momi, "Improved human–robot collaborative control of redundant robot for teleoperated minimally invasive surgery," *IEEE Robot. Autom. Lett.*, vol. 4, no. 2, pp. 1447–1453, Apr. 2019.

[7] P. Bosscher and D. Hedman, "Real-time collision avoidance algorithm for robotic manipulators," *Ind. Robot, Int. J.*, vol. 38, no. 2, pp. 186–197, Mar. 2011.

[8] Y. Choi, D. Kim, S. Hwang, H. Kim, N. Kim, and C. Han, "Dual-arm robot motion planning for collision avoidance using B-spline curve," *Int. J. Precis. Eng. Manuf.*, vol. 18, no. 6, pp. 835–843, Jun. 2017.

[9] B. Lee and C. G. Lee, "Collision-free motion planning of two robots," *IEEE Trans. Syst., Man, Cybern.*, vol. SMC-17, no. 1, pp. 21–32, Jan. 1987.

[10] C. Chang, M. J. Chung, and B. H. Lee, "Collision avoidance of two general robot manipulators by minimum delay time," *IEEE Trans. Syst., Man, Cybern.*, vol. 24, no. 3, pp. 517–522, Mar. 1994.

[11] Z. Bien and J. Lee, "A minimum-time trajectory planning method for two robots," *IEEE Trans. Robot. Autom.*, vol. 8, no. 3, pp. 414–418, Jun. 1992.

[12] J. Lee, "A dynamic programming approach to near minimum-time trajectory planning for two robots," *IEEE Trans. Robot. Autom.*, vol. 11, no. 1, pp. 160–164, Feb. 1995.

[13] R. C. Luo, B.-H. Shih, and T.-W. Lin, "Real time human motion imitation of anthropomorphic dual arm robot based on Cartesian impedance control," in *Proc. IEEE Int. Symp. Robot. Sensors Environ.*, Washington, DC, USA, Oct. 2013, pp. 25–30.
[14] R. C. Luo, M.-C. Ko, Y.-T. Chung, and R. Chatila, "Repulsive reaction vector generator for whole-arm collision avoidance of 7-DoF redundant robot manipulator," in *Proc. IEEE/ASME Int. Conf. Adv. Intell. Mechatronics*, Besaçon, France, Jul. 2014, pp. 1036–1041.
[15] F. Flacco, T. Kröger, A. De Luca, and O. Khatib, "A depth space approach to human-robot collision avoidance," in *Proc. IEEE Int. Conf. Robot. Autom.*, Saint Paul, MN, USA, May 2012, pp. 338–345.
[16] A. De Luca and F. Flacco, "Integrated control for pHRI: Collision avoidance, detection, reaction and collaboration," in *Proc. 4th IEEE RAS EMBS Int. Conf. Biomed. Robot. Biomechatronics (BioRob)*, Rome, Italy, Jun. 2012, pp. 288–295.
[17] E. A. Pohlmeyer, B. Mahmoudi, S. Geng, N. Prins, and J. C. Sanchez, "Brain-machine interface control of a robot arm using actor-critic rainforcement learning," in *Proc. Annu. Int. Conf. IEEE Eng. Med. Biol. Soc.*, San Diego, CA, USA, Aug. 2012, pp. 4108–4111.
[18] S. Adam, L. Busoniu, and R. Babuska, "Experience replay for real-time reinforcement learning control," *IEEE Trans. Syst., Man, Cybern. C, Appl. Rev.*, vol. 42, no. 2, pp. 201–212, Mar. 2012.
[19] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing Atari with deep reinforcement learning," 2013, *arXiv:1312.5602*. [Online]. Available: http://arxiv.org/abs/1312.5602
[20] S. James and E. Johns, "3D simulation for robot arm control with deep *Q*-learning," 2016, *arXiv:1609.03759*. [Online]. Available: http://arxiv.org/abs/1609.03759
[21] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," 2015, *arXiv:1509.02971*. [Online]. Available: http://arxiv.org/abs/1509.02971
[22] S. Gu, E. Holly, T. Lillicrap, and S. Levine, "Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, Singapore, May 2017, pp. 3389–3396.
[23] M. Vecerik, T. Hester, J. Scholz, F. Wang, O. Pietquin, B. Piot, N. Heess, T. Rothörl, T. Lampe, and M. Riedmiller, "Leveraging demonstrations for deep reinforcement learning on robotics problems with sparse rewards," 2017, *arXiv:1707.08817*. [Online]. Available: http://arxiv.org/abs/1707.08817
[24] L. Tai, G. Paolo, and M. Liu, "Virtual-to-real deep reinforcement learning: Continuous control of mobile robots for mapless navigation," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst. (IROS)*, Vancouver, BC, Canada, Sep. 2017, pp. 31–36.
[25] W. Meng, S. Q. Xie, Q. Liu, C. Z. Lu, and Q. Ai, "Robust iterative feedback tuning control of a compliant rehabilitation robot for repetitive ankle training," *IEEE/ASME Trans. Mechatronics*, vol. 22, no. 1, pp. 173–184, Feb. 2017.
[26] Q. Ai, D. Ke, J. Zuo, W. Meng, Q. Liu, Z. Zhang, and S. Q. Xie, "High-order model-free adaptive iterative learning control of pneumatic artificial muscle with enhanced convergence," *IEEE Trans. Ind. Electron.*, vol. 67, no. 11, pp. 9548–9559, Nov. 2020.
[27] H. Su, Y. Hu, H. R. Karimi, A. Knoll, G. Ferrigno, and E. De Momi, "Improved recurrent neural network-based manipulator control with remote center of motion constraints: Experimental results," *Neural Netw.*, vol. 131, pp. 291–299, Nov. 2020.
[28] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," 2018, *arXiv:1801.01290*. [Online]. Available: http://arxiv.org/abs/1801.01290
[29] H. Su, W. Qi, C. Yang, J. Sandoval, G. Ferrigno, and E. D. Momi, "Deep neural network approach in robot tool dynamics identification for bilateral teleoperation," *IEEE Robot. Autom. Lett.*, vol. 5, no. 2, pp. 2943–2949, Apr. 2020.
[30] P. Varin, L. Grossman, and S. Kuindersma, "A comparison of action spaces for learning manipulation tasks," 2019, *arXiv:1908.08659*. [Online]. Available: http://arxiv.org/abs/1908.08659
[31] D. Akimov, "Distributed soft actor-critic with multivariate reward representation and knowledge distillation," 2019, *arXiv:1911.13056*. [Online]. Available: http://arxiv.org/abs/1911.13056
[32] M. S. Kim, D. K. Han, J. H. Park, and J. S. Kim, "Motion planning of robot manipulators for a smoother path using a twin delayed deep deterministic policy gradient with hindsight experience replay," *Appl. Sci.*, vol. 10, no. 2, pp. 575–589, Jan. 2020.
[33] E. Prianto, M. S. Kim, J. H. Park, J. H. Baen, and J. S. Kim, "Path planning for multi-arm manipulators using deep reinforcement learning: Soft actor–critic with hindsight experience replay," *Sensors*, vol. 20, no. 20, pp. 5911–5932, Oct. 2020.
[34] H. Ha, J. Xu, and S. Song, "Learning a decentralized multi-arm motion planner," 2020, *arXiv:2011.02608*. [Online]. Available: http://arxiv.org/abs/2011.02608
[35] A. L. Maas, A. Y. Hannun, and A. Y. Ng, "Rectifier nonlinearities improve neural network acoustic models," in *Proc. 30th Int. Conf. Mach. Learn.*, Atlanta, GA, USA, 2013, pp. 3–8.
[36] B. Xu, N. Wang, T. Chen, and M. Li, "Empirical evaluation of rectified activations in convolutional network," 2015, *arXiv:1505.00853*. [Online]. Available: http://arxiv.org/abs/1505.00853
[37] A. L. Schwab and J. P. Meijaard, "How to draw Euler angles and utilize Euler parameters," in *Proc. Int. Design Eng. Tech. Conf., Comput. Inf. Eng. Conf.*, Philadelphia, PA, USA, 2006, pp. 259–265.
[38] K. Shoemake, "Animating rotation with quaternion curves," in *Proc. 12th Annu. Conf. Comput. Graph. Interact. Techn.*, 1985, pp. 245–254.
[39] L. Busoniu, R. Babuska, and B. De Schutter, "A comprehensive survey of multiagent reinforcement learning," *IEEE Trans. Syst., Man, Cybern. C, Appl. Rev.*, vol. 38, no. 2, pp. 156–172, Mar. 2008.
[40] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. P. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," in *Proc. Int. Conf. Mach. Learn.*, New York, NY, USA, 2016, pp. 1928–1937.

CHING-CHANG WONG received the B.S. degree from the Department of Electronic Engineering, Tamkang University, Taiwan, in 1984, and the M.S. and Ph.D. degrees from the Department of Electrical Engineering, Tatung Institute of Technology, Taiwan, in 1986 and 1989, respectively. He joined the Department of Electrical and Computer Engineering, Tamkang University (TKU), in 1989, where he served as the Department Chairman, from 2006 to 2010. He established the Robotics Engineering Institute, in 2007. He is currently a Distinguished Professor. He established the Intelligent Automation and Robotics Center, in 2011. He has published and coauthored over 300 technical articles and 20 patents. His current research interests include intelligent control, humanoid robot, mobile robot manipulator, and deep reinforcement learning for robotic applications. He was elevated as a Fellow of The Institution of Engineering and Technology (IET), the Chinese Automatic Control Society (CACS), and the Robotics Society of Taiwan (RST), in 2009, 2015, and 2019, respectively. He was a recipient of the Outstanding Automatic Control Award from the CACS, Taiwan, in 2009, and the Outstanding Robotics Engineering Award from the RST, in 2018. He received the Outstanding and Premium Research Awards (2011∼2020) from the Ministry of Science and Technology (MOST), Taiwan. From 2009 to 2010, he served as the Chair of the IEEE Robotics and Automation Taipei Chapter.

SHAO-YU CHIEN was born in Keelung, Taiwan, in 1995. He received the B.S. and M.S. degrees from the Department of Electrical and Computer Engineering, Tamkang University, Taiwan, in 2017 and 2019, respectively, where he is currently pursuing the Ph.D. degree. His major research interests include robot manipulator, robotic applications, and machine learning.

HSUAN-MING FENG received the B.S. degree in automatic control engineering from Feng-Chia University, Taichung, Taiwan, in 1992, and the M.S. and Ph.D. degrees in computer science and information engineering from Tamkang University, New Taipei City, Taiwan, in 1994 and 2000, respectively. He is currently a Full Professor with the Department of Computer Science and Information Engineering, National Quemoy University. His current research interests include fuzzy systems, machine learning, neural networks, wireless networks, optimal learning algorithms, image processing, and robot systems.

HISASUKI AOYAMA was born in Japan, in 1958. He received the bachelor's degree in mechanical engineering for production, the master's degree in precision machinery, and the Ph.D. degree in precision engineering from the Tokyo Institute of Technology, Tokyo, Japan, in 1981, 1983, and 1988, respectively. From 1983 to 1988, he worked with the Research Laboratory of Precision Machinery and Electronics, Tokyo Institute of Technology. He became an Associate Professor with the Department of Precision Engineering, Shizuoka University. From 1989 to 1990, he was a Visiting Researcher with the College of Manufacturing, Cranfield Institute of Technology, U.K. He was with the Department of ECE, North Carolina State University, USA, in 1996. He was an Associate Professor with the Department of Mechanical Engineering and Intelligent Systems, University of Electro-Communications, in 1997. He became a Professor, in 2002, and the Director of the Micro Robotics and Mechatronics Group and the Global Alliance Laboratory Project. He was with the King Mongkut's Institute of Technology Ladkrabang, Thailand, in 2017. His research interests include micro/precision engineering, micro metrology, and its industrial and biomedical applications.