2022 IEEE 18th International Conference on Automation
Science and Engineering (CASE)
August 20-24, 2022. Mexico City, Mexico

2022 IEEE 18th International Conference on Automation Science and Engineering (CASE) | 978-1-6654-9042-9/22/$31.00 ©2022 IEEE | DOI: 10.1109/CASE49997.2022.9926603

A Flexible Collision-Free Trajectory Planning for Multiple Robot Arms by Combining Q-Learning and RRT*

Tomoya Kawabe[1] and Tatsushi Nishi[2]

Abstract— **In this paper, we propose an approach for real-time collision-free trajectory planning of multiple robot manipulators in a common workspace. In recent years, robot arms are often introduced to factories in place of human beings, and it has become important how efficiently multiple robot arms can be operated in a small space. The problem of trajectory planning for multiple robot arms is often solved by graph search algorithms, however, it is difficult for the conventional approach to provide flexible trajectory planning to cope with unexpected situations such as robot arm failure. Therefore, we propose a combined method for Q-learning and RRT for the trajectory planning problem. The effectiveness of the proposed method is further verified using numerical experiments. The planned trajectories is able to guarantee a certain degree of optimality when the motion trajectory is generated by combining reinforcement learning than by using only the graph search algorithm. The results indicate that the time required to generate the motion trajectory is reduced by the proposed method.**

I. Introduction

In recent years, robot arms have been introduced to replace humans in factories to perform complex assembly tasks. The introduction of robots requires a large space. In many cases, multiple robot arms have to be placed in a common workplace. Therefore, effective approach for collision-free trajectory planning of multiple robot arms is required. Many motion planning methods for a single robot arm using optimization approaches have been proposed in the past [1]. Shiller $et\ al.$ used graph search methods to optimize the motions of a six-degree-of-freedom manipulator operated in a three-dimensional space with obstacles [2]. However, the method requires the shape of the obstacle to be given directly, making it difficult to apply in complex environments such as real space. In order to solve the problem, there have been so many methods using probabilistic methods such as random sampling-based methods like PRM (Probabilistic Roadmap Method) and RRT (Rapidly-Exploring Random Tree) to the motion planning problem of robot arms. RRT does not guarantee the optimality of the trajectory, however, RRT* does guarantee convergence to the optimal trajectory [3][4]. This is because RRT* requires a process to recalculate the trajectory during the search.

These methods are applied to the motion planning problem of single robot arms. The motion planning problems for multiple robot arms are more complex. Traditionally, two types of approaches have been proposed: distributed approaches and centralized approaches [5]. Distributed multi-robot motion planning method [6][7] divides the problem into several subproblems and solves them separately, thereby reducing the size of the search space. However, optimality is not guaranteed in those approaches. Centralized approaches [8] tend to be slower than distributed approaches, however, they often can guarantee optimality. Those methods have received much attention in recent years. A hybrid centralized/distributed approach has also been proposed [9]. Random sampling-based approaches such as PRM and RRT can be extended to the multi-robot motion planning by regarding multiple robots as a single robot group. For one of the conventional methods, a method for collision-free trajectory planning of multiple robot arms in a common workspace using PRM has been proposed [10]. In the case of these approaches, unexpected events such as robot arm failures or delays may cause collisions that have not appeared in the simulation phase.

Therefore, recently, a reinforcement learning method has been proposed to solve the trajectory planning of multiple robot arms [11][12][13]. Reinforcement learning is often used to solve problems in which multiple robot arms execute cooperative actions. A method using deep reinforcement learning has also been addressed [14]. One of the advantages of using reinforcement learning is that the trajectory can be flexibly re-planned under unexpected situations. However, reinforcement learning is a trial-and-error method of searching for suboptimal solutions.

In this paper, we propose a method that combines a random sampling approach with reinforcement learning. We choose RRT* as the random sampling method and Q-learning as the reinforcement learning method. RRT* is used to search the trajectories of robot arms for each arm, and Q-learning is used to avoid collisions between robot arms. By combining the RRT* with Q-learning, each robot arm has the flexibility to accommodate the changes that can cope with unexpected situations, which is an advantage of reinforcement learning, while guaranteeing the optimality, which is an advantage of graph search. In addition, by incorporating elements of the RRT* into the state space and reward settings used in Q-learning, state space is simplified and the time required for trajectory planning can be reduced. Through several computational experiments, Through simulation experiments, our method reduces the time required for motion planning to less than 20% of that required by only RRT*, while avoiding collisions between robot arms.

[1]T. Kawabe is with Graduate School of Natural Science and Technology, Okayama University, 3-1-1 Tsushima-Naka, Kita-ku, Okayama 700-8530, JAPAN (phone: +81-86-251-8058, e-mail: pvle84mm@s.okayama-u.ac.jp).

[2]T. Nishi is with Graduate School of Natural Science and Technology, Okayama University, 3-1-1 Tsushima-Naka, Kita-ku, Okayama 700-8530, JAPAN (phone: +81-86-251-8058, e-mail: nishi.tatsushi@okayama-u.ac.jp).

II. Motion Planning

A. Problem Statement

In this research, multiple robot arms are placed in a limited space, and a trajectory planning method combining RRT* and Q-learning is used to reach the target posture while avoiding collisions. First, each robot arm independently explores its trajectory to the target posture by RRT*. Then, each robot arm is moved, and if a collision is likely to happen, Q-learning is applied and the trajectory is updated. The robot arm that we consider in the simulation is the VS-068 [15] of Denso Wave Corporation as shown in Fig. 1. The VS-068 is a robot used for a wide variety of tasks such as assembly, transportation, picking, inspection, and processing, and it is characterized by its high-speed operation. In addition, its slim design makes it suitable for use in this research for collision-free trajectory planning of robot arms.

B. RRT* Algorithm

Motion planning for the each 6-DOF robots were executed by using RRT*, an extension of one of the best-known tools, Rapidly Exploring Random Trees (RRT). RRT* is a method based on random sampling based on RRT, while modified to minimize the objective function. One of the disadvantages of RRT is that the obtained solution may deviate significantly from the optimal solution, however, RRT* solves this problem and is guaranteed to converge to the optimal solution [4]. The algorithm for RRT* is shown in Algorithms 1. Fig. 2 shows the process of extending the tree $\mathcal{T}$ in RRT*.

First, the tree structure $\mathcal{T}$ is built as in the RRT. V is a node and E is an edge. Then, randomly sample points z_{rand} in the space and extend the tree $\mathcal{T}$ in that direction from the nearest node $z_{nearest}$. RRT* create a set Z_{near} of neighboring nodes around the extended node z_{new} and reconstructs the node with the lowest cost among them as the parent node. The recalculation targets nodes within the radius expressed by the following equation (1).

$$r = \min\left(R\left(\frac{\log N}{N}\right)^{\frac{1}{d}}, \eta\right) \tag{1}$$

Fig. 1: VS-068 [15]

Algorithm 1 $\mathcal{T} = (V, E) \leftarrow$ RRT* $(z_{\text{init}}, z_{\text{target}})$

$\mathcal{T} \leftarrow \text{InsertNode}(\emptyset, z_{\text{init}}, \mathcal{T})$;
while $z_{\text{new}} \neq z_{\text{target}}$ **do**
  $z_{\text{rand}} \leftarrow \text{Sample}()$;
  $z_{\text{nearest}} \leftarrow \text{Nearest}(\mathcal{T}, z_{\text{rand}})$;
  $z_{\text{new}} \leftarrow \text{Steer}(z_{\text{nearest}}, z_{\text{rand}})$;
  if $\text{ObstacleFree}(z_{\text{new}}, z_{\text{near}})$ **then**
    $Z_{\text{near}} \leftarrow \text{Near}(\mathcal{T}, z_{\text{new}}, \min(R\left(\frac{\log N}{N}\right)^{\frac{1}{d}}, \eta))$;
    $z_{\min} \leftarrow z_{\text{near}}$;
    $c_{\min} \leftarrow \text{Cost}(Z_{\text{near}} + c(\text{Line}(z_{\text{near},z_{\text{new}}})))$;
    for each $z_{\text{near}} \in Z_{\text{near}}$ **do**
      if $\text{ObstacleFree}(z_{\text{near}}, z_{\text{new}})$ **and** $\text{Cost}(Z_{\text{near}} + c(\text{Line}(z_{\text{near},z_{\text{new}}}))) < c_{\min}$ **then**
        $z_{\min} \leftarrow z_{\text{near}}$;
        $c_{\min} \leftarrow \text{Cost}(Z_{\text{near}} + c(\text{Line}(z_{\text{near},z_{\text{new}}})))$;
      end if
    end for
    $\mathcal{T} \leftarrow \text{InsertNode}(z_{\min}, z_{\text{new}}, \mathcal{T})$;
    for each $z_{\text{near}} \in Z_{\text{near}}$ **do**
      if $\text{ObstacleFree}(z_{\text{near}}, z_{\text{new}})$ **and** $\text{Cost}(Z_{\text{new}} + c(\text{Line}(z_{\text{near},z_{\text{new}}}))) < \text{Cost}(z_{\text{near}})$ **then**
        $\mathcal{T} \leftarrow \text{ReConnect}(z_{\min}, z_{\text{near}}, \mathcal{T})$;
      end if
    end for
  end if
end while
return $\mathcal{T}$

In equation1, N is the number of nodes, R is the weight, d is the number of dimensions of the state space, and η is the upper bound of the radius. In this study, R=0.5, d=7 (6 axes + end-effector), and η =0.3. The length of the edge to be extended at each step is 0.1[m], and the target posture z_{target} is selected as the sampling point with a probability of 20%.

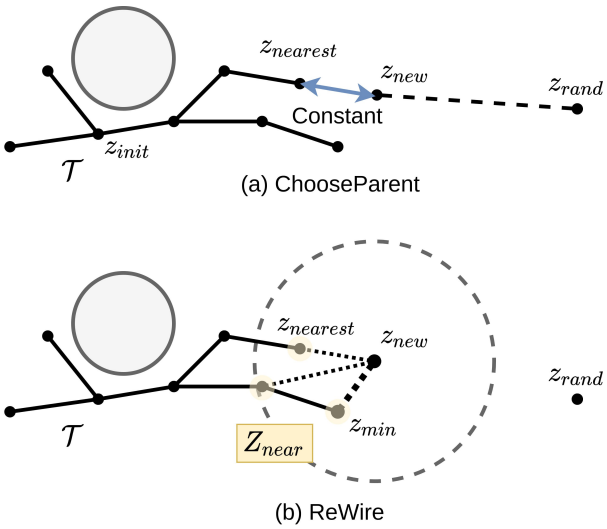

Fig. 2: Additions to the tree structure $\mathcal{T}$

C. Q-Learning

1) Learning Processes: In this study, Q-learning, one of the most well-known reinforcement learning methods, is used to realize collision avoidance between robot arms. Q-learning could select the optimal action in a particular state, i.e., how the joints should be operated when the robot arms are close to each other. Updating the Q function is performed by the following equation (2). s_t and a_t represent the state and action at step t, respectively, α is the learning rate, which takes a number between 0 and 1, and γ is the discount factor. The experiment was conducted with $\alpha = 0.5$ and $\gamma = 0.99$. The part A could be regarded as the error of the value function with the target value. This is called the TD (Temporal Difference) error, and learning convergence is judged when the TD error becomes sufficiently small.

$$Q_{new}(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[r_t + \gamma \max_{\alpha'} Q(s_{t+1}, \alpha') - Q(s_t, a_t) \right] \tag{2}$$

2) Definition of State Spaces: In this study, the state s_t of the robot arm used in Q-learning is defined as shown in Fig. 3. Instead of defining a single state space for the whole system, we define each robot arm to have its own state space. This method can be easily expanded to accommodate an increase in the number of robot arms. 13 spheres are placed around the end-effector, and the robot arm observes the inside of the spheres. The radius of the sphere is set to 0.05 [m]. The distance of each sphere from the end-effector is shown in TABLE I.

3) Action Settings: The ϵ-greedy method was used as a method for selecting actions based on policy. The ϵ-greedy method selects a random action that is not related to Q for a given probability ϵ. When ϵ is large, the emphasis is on search; when ϵ is small, the emphasis is on knowledge use.

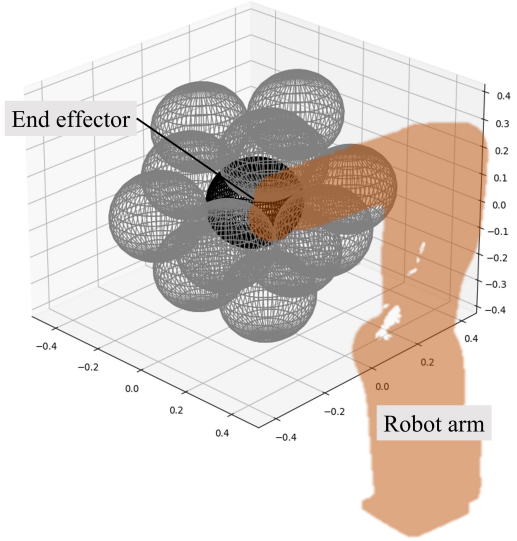

Fig. 3: Definition of state space

The value of ϵ was given by the following equation and was set to decrease as the learning progressed.

$$\epsilon = 0.5 \times \frac{1}{\text{Number of episode} + 1} \tag{3}$$

6 types of actions a_t used in the Q learning were prepared in the rectangular coordinate system. Move the end-effector position parallel to the x-, y-, or z-axis by a fixed amount in the positive or negative direction. The number of states s_t is 13, since that is the number of spheres, and the number of actions a_t is 6, so the size of the Q table is $2^{13} \times 13$.

4) Reward Settings: The reward r_t used for Q-learning is defined as follows, using the path planned by RRT*. By operating along the trajectory explored by RRT*, it is guaranteed to reach the target posture. Therefore, a high reward is given when the robot arm moves as explored by RRT*. On the other hand, the reward becomes smaller as the distance to another robot arm becomes closer. In particular, if the robot is close to another robot arm and chooses an action in the direction of the other robot arm, it will be given the lowest reward because of the high risk of collision. The equation (5) is an additional reward given regardless of the condition, and is higher the more it operates in the target posture direction. Z_t represents each joint angle of the robot arm at the current step, and Z_{target} that of the target posture.

$$r_t = \begin{cases} +10 & \text{If move as explored by RRT}^* \\ +1 & \text{If there is not other robot around} \\ -10 & \text{If collision with other robots} \end{cases} \tag{4}$$

$$r_t = r_t - 0.1 + (20 - \|Z_{target} - Z_t\|)/10 \tag{5}$$

D. Proposed Method

The proposed method is applied according to the flowchart shown in Fig. 4. First, each robot arm explores the trajectory to the target posture by RRT*, ignoring another robot arm. Then, move all the robot arms along the explored trajectory one action step at a time. The robot arm observes its neighbors and updates its actions by Q-learning when another robot arm is approaching. If the robot arm does not move closer, the searched trajectory will eventually converge to

TABLE I: Distance of the spheres from the end-effector

sphere no.	x [m]	y [m]	z [m]
1	$1/10$	0	0
2	$-1/10$	0	0
3	$1/20$	$\sqrt{3}/20$	0
4	$-1/20$	$\sqrt{3}/20$	0
5	$1/20$	$-\sqrt{3}/20$	0
6	$-1/20$	$-\sqrt{3}/20$	0
7	0	$\sqrt{3}/30$	$\sqrt{3}/20$
8	$1/20$	$-\sqrt{3}/60$	$\sqrt{3}/20$
9	$-1/20$	$-\sqrt{3}/60$	$\sqrt{3}/20$
10	0	$\sqrt{3}/30$	$-\sqrt{3}/20$
11	$1/20$	$-\sqrt{3}/60$	$-\sqrt{3}/20$
12	$-1/20$	$-\sqrt{3}/60$	$-\sqrt{3}/20$
13	0	0	0

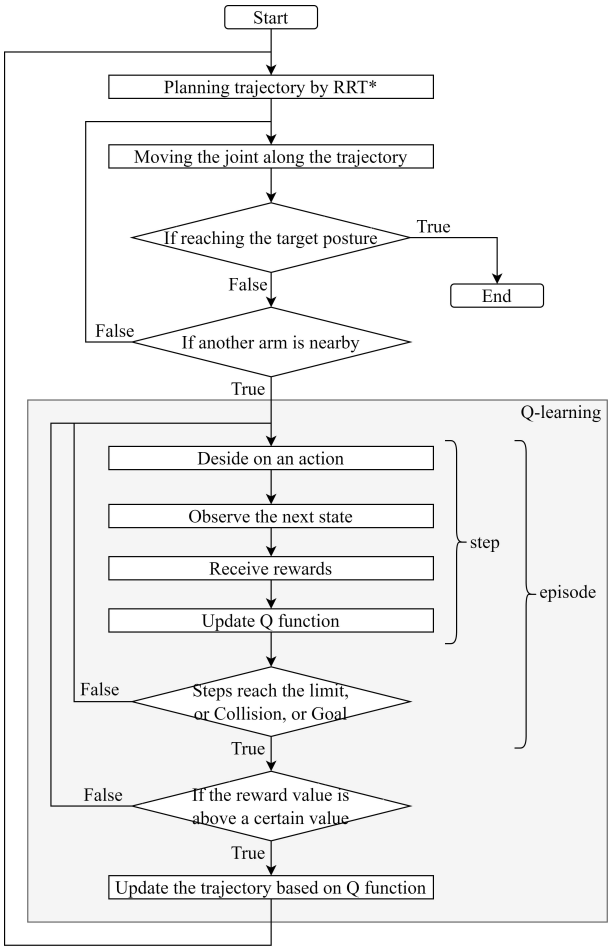

Fig. 4: The algorithm for applying the proposed method to trajectory planning

the shortest trajectory due to the characteristic of RRT*. In Q-learning, the robot arm first chooses an action and gets a reward according to the state changed in response to that action. The Q function is updated every step of the process based on the rewards obtained. This is repeated as one step until the robot arm collisions, reaches the goal, or the number of steps reaches the limit. If any of the conditions are met, the episode is terminated and the total rewards received are calculated. If the reward value exceeds a certain value, the training is terminated and the trajectory is updated based on the Q function. After updating the trajectory to avoid collision, the trajectory to the target posture is explored again with RRT*.

III. Numerical Experiments

A. Problem Settings

In this study, the environment shown in Fig. 5 was created with two 6-DOF robot arms in close locations. The installation position, initial posture, and target posture of each robot arm given as initial conditions are shown in the TABLE II. According to the initial conditions, each robot arm plans its trajectory to the target posture while avoiding collisions with each other.

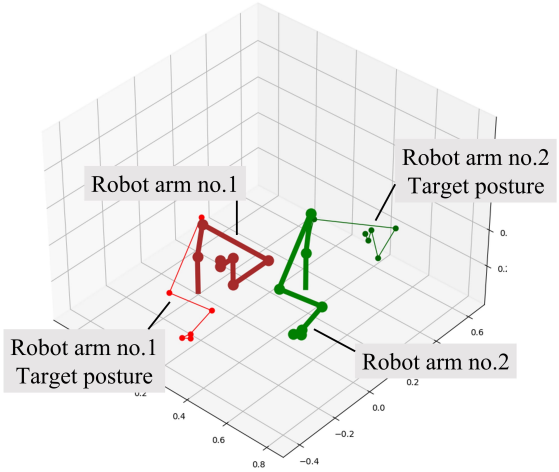

Fig. 5: Robot arm placement

TABLE II: Initial conditionst

Robot arm no.	1	2
Installation position (x, y, z)	$(0.3, 0.3, 0)$	$(0, 0, 0)$
Initial Posture θ_n[rad]	4	0
Target Posture θ_n[rad]	0	4

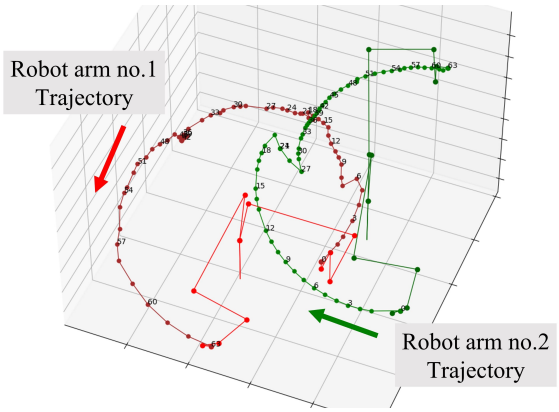

Fig. 6: Example of trajectory planning

B. Experimental Results

One of the results of the trajectory planning by the proposed method is shown in Fig. 6. When the robot arms are far away from each other, the robot arm moves efficiently to the target posture along the trajectory explored by RRT*. When two robot arms approach each other, robot arm no. 2 changes its trajectory to avoid collision with robot arm no. 1. After a sufficient distance is again secured between the two robots, the robot arm no. 2 replanning its trajectory and moving to the target posture.

The figure shows an example of trajectory planning when the radius of the sphere used for state observation is increased from 0.05[m] to 0.1[m]. A trajectory was planned in which robot arm no. 2 largely avoids the neighborhood from the base of robot arm no. 1. By adjusting the size of the state-space sphere, the margin between the robot arms could be adjusted as desired.

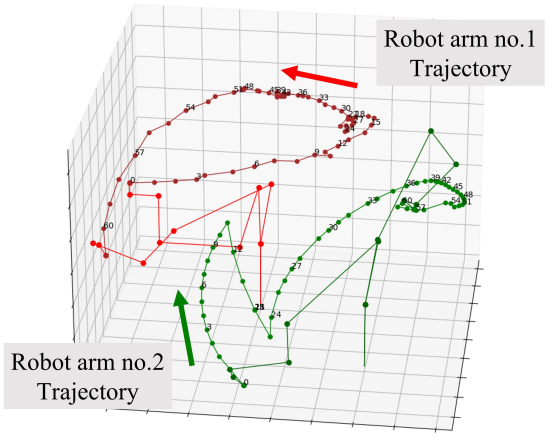

Fig. 7: Example of trajectory planning (large state space)

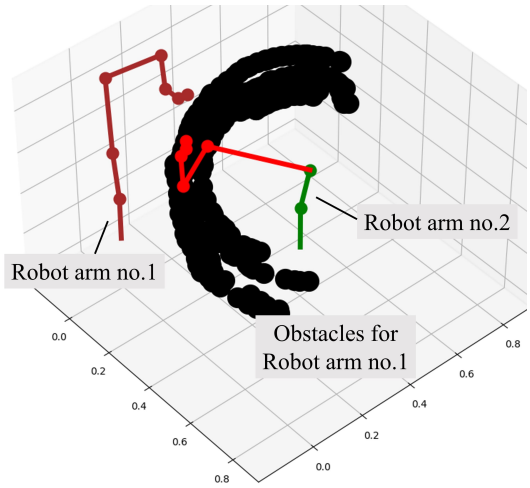

Fig. 8: Trajectory planning for only RRT*

TABLE III: Initial conditionst (Case 1)

Robot arm no.	1	2
Installation position (x, y, z)	$(0.4, 0.4, 0)$	$(0, 0, 0)$
Initial Posture θ_n[rad]	4	1.5
Target Posture θ_n[rad]	1.5	0

TABLE IV: Initial conditionst (Case 2)

Robot arm no.	1	2
Installation position (x, y, z)	$(0, 0.6, 0)$	$(0, 0, 0)$
Initial Posture θ_n[rad]	0	π
Target Posture θ_n[rad]	$-\pi$	0

TABLE V: Initial conditionst (Case 3)

Robot arm no.	1	2
Installation position (x, y, z)	$(0.5, 0.5, 0)$	$(0, 0, 0)$
Initial Posture θ_n[rad]	6	0
Target Posture θ_n[rad]	4	2

($n = 1, 2, \cdots, 6$, All joint angles are the same.)

TABLE VI: Results of computational experiments

	Proposed method (RRT*+QL)		Only RRT*	
Case	Obj.1	Time[s]	Obj.1	Time[s]
1	64.2	1.77	63.4	12.23
2	122.0	5.25	85.8	32.30
3	62.2	2.02	53.0	10.54

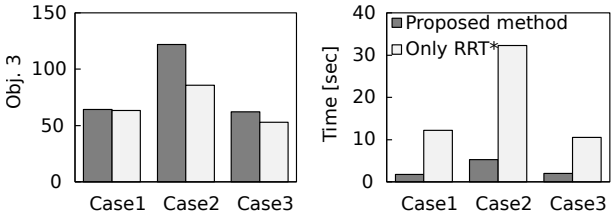

Fig. 9: Comparison of results

C. Comparison Experiment with Another Method

We investigate the effectiveness of the proposed method. The trajectories planned by the proposed method and that planned by other methods are compared. The length of the planned trajectory and the computation time required for trajectory planning are measured. The comparison is conducted with a method that plans trajectories using only RRT*. In the case of RRT* only, robot arm no.1 plans the trajectory by itself using RRT* at first. Then, the planned trajectory is used as an obstacle, and the robot arm no.2 plans its trajectory to avoid it. Fig. 8 shows how one robot arm explores the trajectory first and then uses the trajectory as obstacle for the other robot arm. To identify collisions, the robot arm that collides with an obstacle changes its color. The layout shown in Fig. 5 is adopted, and the initial conditions, such as the position of the robot arm, initial posture, and target posture, are changed for the experiment. TABLEs III, IV and V show the initial conditions used in the experiment. The experimental results obtained for each initial condition are shown in TABLE VI and Fig. 9. Obj.1 is the number of action steps until all the robot arms reach the target posture.

From TABLE VI, it can be seen that the proposed method can generate the trajectory plan with the same length of trajectory in Case 1 and Case 3 as the case where the trajectory is planned only by RRT*. However, in Case 2, the length of the trajectory planned by the proposed method is considerably longer. Fig. 10 shows an example of trajectory planning under the initial conditions of Case 2. From Fig. 10, we can see that the end-effector of robot arm no. 2 moves in the opposite direction immediately after the start. This seems to be an action to avoid the approaching robot arm no.1, however, it appears to be a useless movement. Currently, rewards are determined manually, however, using inverse reinforcement learning to automatically set rewards would eliminate this type of action.

TABLE VI shows that the computation time for all initial conditions is reduced by 10% to 20% compared with the case of trajectory planning using only RRT*. In the case of trajectory planning using only RRT*, the computation time becomes longer as the number of obstacles increases. This is the reason that the computation time could be significantly reduced.

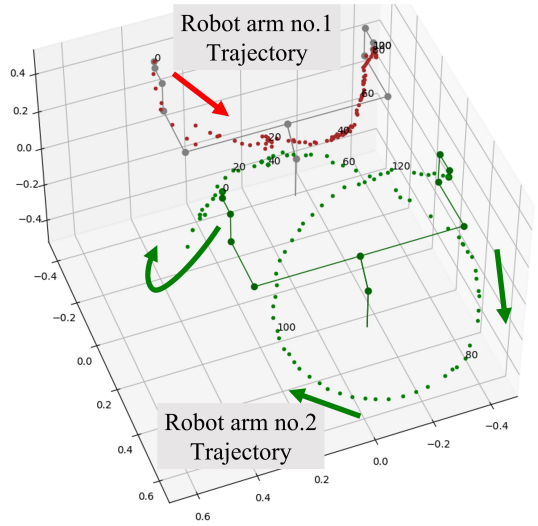

Fig. 10: Example of trajectory planning (Case 2)

IV. Conclusion

In this paper, we proposed a method combining RRT* and Q-learning, for a collision-free trajectory planning problem with multiple robot arms. Each robot arm first searched for a trajectory to the target posture using RRT*, and then used Q-learning to avoid collisions between robot arms. Simulation experiments showed that real-time collision-free trajectory planning could be performed. The time required for trajectory planning was reduced to less than 20% of that for trajectory planning using RRT* alone. However, depending on the initial conditions, the trajectory to the target posture sometimes became longer. In the future, it is important to revise the definitions of state space and reward and to use reinforcement learning methods or deep reinforcement learning instead of Q-learning to plan shorter trajectories.

Acknowledgment

This research is subsidized by New Energy and Industrial Technology Development Organization (NEDO) under a project JPNP20016. This paper is one of the achievements of joint research with and is jointly owned copyrighted material of ROBOT Industrial Basic Technology Collaborative Innovation Partnership.

References

[1] K. Harada, "Optimization in robot motion planning," *Journal of the Robotics Society of Japan*, vol. 32, no. 6, pp. 508–511, 2014.

[2] Z. Shiller and S. Dubowsky, "Global time optimal motions of robotic manipulators in the presence of obstacles," *In Proc. of IEEE Int. Conf. on Robotics and Automation*, pp. 370–375, 1988.

[3] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *Int. J. of Robotics Research*, vol. 30, no. 7, pp. 846–894, 2011.

[4] S. Karaman, M.R. Walter, A. Perez, E. Frazzoli and S. Teller, "Anytime motion planning using the RRT*," *In Proc. of IEEE Int. Conf. on Robotics and Automation*, pp.1478–1483, 2011

[5] R. Shome, K. Solovey, A. Dobson, D. Halperin and K.E. Bekris, "dRRT*: Scalable and informed asymptotically-optimal multi-robot motion planning," *Auton Robot*, vol. 44, pp. 443–467, 2020.

[6] M. Erdmann and T. Lozano-Perez, "On multiple moving objects," *In Proc. of IEEE Int. Conf. on Robotics and Automation*, pp. 1419–1424, 1986.

[7] J. P. van den Berg and M. H. Overmars, "Prioritized motion planning for multiple robots," *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, pp. 430–435, 2005.

[8] P. A. O'Donnell and T. Lozano-Perez, "Deadlock-free and collision-free coordination of two robot manipulators," *In Proc. Int. Conf. on Robotics and Automation*, vol. 1 pp. 484–489, 1989.

[9] J. Alonso-Mora, R. Knepper, R. Siegwart and D. Rus, "Local motion planning for collaborative multi-robot manipulation of deformable objects," *IEEE Int. Conf. on Robotics and Automation (ICRA)*, pp. 5495–5502, 2015.

[10] G. Sánchez and J.-C. Latombe, "On delaying collision checking in PRM planning: Application to multi-robot coordination," *Int. J. Robot. Res.*, vol. 21, no. 1, pp. 5–26, 2002.

[11] F. Kitano, H. Uchikata, K. Matsushita, M. Shiga and M. Sasaki, "Simultaneous optimization of mechanism and control in cooperative work of multiple robot arms," *Journal of the Japan Society of Applied Electromagnetics and Machines*, vol. 29, no. 1, pp. 161–167, 2021.

[12] K. Yamada, K. Ohkura and K. Ueda, "Cooperative behavior acquisition of autonomous arm robots through reinforcement learning," *Journal of the Society of Instrument and Control Engineers*, vol. 39, no. 3, pp. 266–275, 2003.

[13] A. Zeng, S. Song, S. Welker, J. Lee, A. Rodriguez and T. Funkhouser, "Learning synergies between pushing and grasping with self-supervised deep reinforcement learning," *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, pp. 4238–4245, 2018.

[14] E. Prianto, M. Kim, J.-H. Park, J.-H. Bae, J.-S. Kim, "Path planning for multi-arm manipulators using deep reinforcement learning: Soft actor–critic with hindsight experience replay," *Sensors*, vol. 20, no. 20:5911, 2020.

[15] VS-068/087 — Denso Wave Incorporate, https://www.denso-wave.com/en/robot/product/five-six/vs068-087.html (Accessed on 2022/02/27)