A Path Planning Approach Based on Q-learning for Robot Arm

Mengyu Ji[1,2], Long Zhang[1], Shuquan Wang[1]
[1]Key Laboratory of Space Utilization,
Technology and Engineering Center for Space Utilization
[2]University of Chinese Academy of Sciences
Beijing, China
e-mail: jimengyu17@mails.ucas.ac.cn

***Abstract*—A path planning method based on Q-learning is proposed for robot arm. As reinforcement learning, Q-learning is widely used in the field of mobile robot navigation due to its simple and well-developed theory. However, most of researchers avoid using it for solving the robot arm path planning problem because it has to take each joints motion into account. In this study, approximate regions instead of accurate measurements are used to define new state space and joint actions. Moreover, the reward function takes the distance from current position of robots' end-effector to the goal position into account. The experimental simulation shows that the Q-learning approach is efficient and has ability to plan a collision free path for robot arm when the order of magnitude of state space is below 10^4. Furthermore, the experimental results indicate that the number of obstacles affects the calculation time for the same iterations. The more obstacles, the less calculation time the algorithm needs. The weight coefficient in the reward function affects the convergence speed and the quality of the solutions.**

Keywords—robot arm, Q-learning, path planning, high-dimensional space

I. INTRODUCTION

Robots aim to liberate humans from doing heavy and trivial work. For example, robots working on the space station are assigned to carry out dangerous missions such as load operation and extravehicular status monitoring [1]. In human's daily life, robots have ability to help humans do various housework, such as caring for patients' physical health, and responding to humans' emotional appeals [2]. Faced with different kinds of complex working environments, robots sometimes have difficulty in planning a feasible path successfully to reach the goal. In these cases, traditional methods for path planning are unable to meet the requirements. Recently, with the development of artificial intelligence, the intelligent level of robot is getting higher and higher. Therefore, it is necessary to study how to let the robot intelligently plan an optimal path. Robot path planning problems can be divided into two types:

- Mobile robot: robot itself is regarded as a particle without size which works on a plane.
- Robot arm: is multi-joint structure. The motion of each joint needs to be planned.

For the robot arm, feasible solutions must be searched in the high-dimensional configuration space (C space). At present, there is no algorithm that gives consideration to both real-time path planning and optimal performance synchronously. For the past decades, the most widely used algorithms in robot arm path planning include Artificial Potential Field Method (APF) [3], Probabilistic Roadmaps (PRM) [4], Rapidly-exploring Random Tree (RRT) [5] and the improved methods based on them [6]-[8]. APF does well in real-time planning and can be used in dynamic environment. However, in some particular cases, it is easy to get into the local minimum and cannot reach the goal. PRM and RRT are applicable to the problem of high-dimensional multi-constraint path planning without the local minimum problem. However, this approach is time consuming and requires re-planning when the initial position changes. In some cases, feasible solution cannot be found.

Recently, artificial intelligence approach is emerging. Q-learning (QL) [9], as a reinforcement learning (RL) method, has begun to attract more and more scholars in terms of automatic navigation due to its high computational efficiency. First of all, an agent gets information about the environment by interacting with it, then the reward or the punishment for each action it takes will be store in a Q-table. By updating the table, the experiences begin to accumulate. In short, QL determines how an agent is able to take a series of actions to maximize cumulative returns. This algorithm is closer to biological nature. So far, QL is widely used in mobile robot navigation.

Dennis and Pablo [10] defined the region where the robot locates in and orientation of the robot as states of QL. The conducted tests show that almost in all cases the path generated by QL ensures low orientation error when the robot reaches the target point. However, the resulting path is not smooth enough due to discrete states and actions.

Reference [11] proposed a new QL method used for mobile navigation in the dynamic environment. Relative position and angle between robots' current pose and the goal pose are defined as new states so that this approach is able to be used in the unknown dynamic environment with a small size of Q-table. The conducted tests indicate the path generated by this method is quite smooth, and the success rate that the robot arrives at target pose in a collision free path can reach 98%.

978-1-7281-0855-1/19/$31.00 ©2019 IEEE

Yuan [12] proposed a modified QL for mobile robot navigation in dynamic environments. By imitating human reasoning, the state space and the reward functions are defined according to human perception and evaluation. And the success rate in avoiding obstacles can reach 90.5% after training. However, only a small number of scholars use QL to plan the robot arms motion.

In this study, a path planning approach based on QL for robot arm is proposed. Path planning can be considered as a constrained optimization problem. In this paper, section 2 presents the methodology for QL including the description for states, actions, reward functions and training procedure. Section 3 presents the results of simulation in MATLAB. Section 4 provides a brief conclusion for the proposed approach.

II. METHODLOGY

A. State Space and Actions

The first step to apply QL approach is defining states. The robot arm is a multi-rigid series structure with a number of degrees of freedom. Therefore, the rotation of each joint needs to be taken into account. The C space of robot is discretized. Thus, it can be defined as state space. Each joint moves within the range of [180, 180] degrees, which can be divided into 12 subregions according to joint angle θ:

$$S_i = \begin{cases} L_1, \theta \in [-180, -180 + res] \\ L_2, \theta \in [-180 + res, -180 + 2 \cdot res] \\ \vdots \\ L_{12}, \theta \in [-180 + 11 \cdot res, 180] \end{cases} \quad (1)$$

where *res* denotes angular resolution for each region. Since each joint has to work together to avoid obstacles, joint state space containing all joints information needs to be defined:

$$S = \{S_1, S_2, ..., S_N\} \quad (2)$$

where S is the Cartesian product between each tuple S_i (i=1, 2, … N). It represents the current joint state, and N is the number of robots DOF. All the joints are assumed to be revolute pairs. Each joint has three modes of motion: clockwise rotation (denoted by 1), counter-clockwise rotation (denoted by -1), and standing in place (denoted by 0). The angle of each rotation is equal to angular resolution for each region. Then motion of each joint can be represented by three numbers, which forms a tuple:

$$A_n = \begin{cases} 1, \theta = \theta + res \\ -1, \theta = \theta - res, n = 1, 2..., N \\ 0, \theta = \theta \end{cases} \quad (3)$$

The definition of joint actions A uses the same method as that of joint states. A is defined as follows:

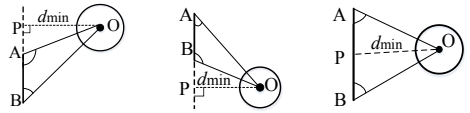
$$A = \{A_1, A_2, ..., A_N\} \quad (4)$$

B. The Reward Functions

Reward functions are used to give evaluation for the action at a given state. Imaging a bird flying in the sky, they tend to avoid all the obstacles and fly towards their goals in a shortest path. Hence penalty value has to be given by reward function when collision happens. In addition, the reward function should take into account the distance from the current position to the target position in the C space.

1) Collision detection: It is very dangerous when the robot arm collides with obstacles. In case that happens, the algorithm gives a negative infinite feedback value to avoid obstacles. In the cartesian space of end-effector, the vector $\overrightarrow{AO}$, $\overrightarrow{BO}$, $\overrightarrow{PO}$ and $\overrightarrow{AB}$ are defined to calculate the shortest distance between the obstacle and each robot's link. As shown in Fig. 1, d_{min} denotes the shortest distance. Line *AB* represents a robot's link, and the circle *O* represents an obstacle. *r* denotes the radius of the obstacles. When $d_{min} \leqslant$ r, collision happens. Let $w = \frac{\overrightarrow{AO} \cdot \overrightarrow{AB}}{|\overrightarrow{AB}|^2}$, and three cases for calculating d_{min} are discussed:

- In Fig. 1(a), w$\leqslant$0 is deduced from $\overrightarrow{AB} \cdot \overrightarrow{AO} \leq 0$. When w$\leqslant$0, $d_{\min} = |\overrightarrow{AO}|$.
- In Fig. 1(b), w$\geqslant$1 is deduced from $\overrightarrow{AB} \cdot \overrightarrow{AO} > 0$ and $|\overrightarrow{AP}| \geq |\overrightarrow{AB}|$. When w$\geqslant$1, $d_{\min} = |\overrightarrow{BO}|$.
- In Fig. 1(c), 0< w <1 is deduced from $\overrightarrow{AB} \cdot \overrightarrow{AO} > 0$ and $|\overrightarrow{AP}| < |\overrightarrow{AB}|$. When 0< w <1, $d_{\min} = |\overrightarrow{PO}|$.



(a) case one (b) case two (c) case three

Figure 1. Three geometrical relations between a link and an obstacle

2) Calculation of the Euclidean distance: Using inverse kinematics, the goal state in the C space can be figured out. The difference between the current joint angle and the goal needs to be considered so that each joint is able to rotate at the minimum angle when algorithm passes through the same number of iterations. In section III, it is presented in detail. Let X denote the vector of the current joint angle, and X' denote the vector of the next angle. X_g is the goal joint angle. Then Euclidean distance from the current joint angle to the goal angle is calculated by following formula:

$$D = \sqrt{(X' - X_g)^2} \quad (5)$$

3) The reward functions: Two terminal states and one transient state are defined as:

- The winning states (WS) if the robot reaches the goal.
- The failure states (FS) if the robot collides with any obstacle.
- The safe states (SS) if the robot is located in safe regions without obstacles.

In order to evaluate the actions of robot arm better, the reward function $R(s, a)$ is defined as follows:

- $R(s, a)$ = -inf, if robot moves from an SS to FS.
- $R(s, a)$ = 10, if robot moves from an SS to WS.
- $R(s, a) = -\mu \cdot D$, if robot moves from an SS to another SS, where μ is the penalty coefficient.

C. The Value Functions

The Q values of state-action pairs is stored in Q-table. All Q values stored in the table are initially set to zero. The rows in the table represent the robot's position in the C space, and the columns represent the action corresponding to the given state. During training, each entry in the table is populated by updating the Q values. After sufficient training, the Q values converges. The updating equation is shown below:

$$Q(s,a) \leftarrow (1-\alpha)Q(s,a) + \alpha\left[R(s,a) + \gamma \max_{a} Q(s',a)\right] \quad (6)$$

where s is the current joint state, and a is the selecting action. s' is the next state that robot transfers to from the current state after performed by the action a. α is the learning factor. γ is the discount factor. After training, the robot is able to find a collision free path in a real environment using learned policy.

In this study, ε-greedy is used for selecting action. At the beginning of learning, agent explore the environment with greater probability. With the accumulation of experience, the probability that agent uses previous experiences increases. Let ε denote the probability of exploration. ε in this process should decrease with the increasing of learning times. According reference [13], ε is designed as follows:

$$\varepsilon(k) = 1 - 0.99\frac{k}{\mathrm{M}} \quad (7)$$

where k is the current planning times in the training process, M is total planning times defined in advance.

III. SIMULATION

A. The Results

Puma560 robot is used to test the performance of the algorithm. The motion of the first three joints affects the position and posture of the end-effector, while the latter three has no influence on the position and only affects the posture of the end-effector. Therefore, the position and posture of the Puma560 are decoupled and can be planned separately, greatly reducing the complexity of the problem. The parameter values for simulation are set in TABLE I. The coordinates of each joint in the base coordinate system are calculated by the DH scheme [14]. In this study, only the first three joint motions are considered. Thus, the total number of states is 12^3=1728. The more subregions of motion range for each joint are divided, the better the generated path is in the environment with obstacles. However, if the number of subregions for each joint's motion is 24, the calculation time is about 7 hours. Therefore, the divided region should take into account both the computing time and algorithm performance.

TABLE I. PARAMETER VALUES

Parameters	α	γ	u	M	res	$r(m)$
Value	0.6	0.9	0.5	20000	30°	0.1

Initial position is [0.90,-0.20,-0.12] and terminal position is [0.40,0.60,0.60]. Let s_g denotes the goal state. It represents the region where the goal joint locates in. s_g can be calculated by using inverse kinematics. If s=s_g, a collision free path is found at this time and another path plan process begins.

With learned policy, robot arm is able to find a collision free path to reach the goal. First, the algorithm is able to retrieve the specific row in the Q-table according to the initial state, and the action with highest Q values is chosen. Second, sequence of all the states formed a path for the joint motion. Finally, in order to make the joint motion stable, cubic spline interpolation is used to interpolate the generated path. The flowchart of the algorithm for path planning is shown in Fig. 2. Simulation experiments were carried out on the MATLAB. The results are presented in Fig. 3-5.

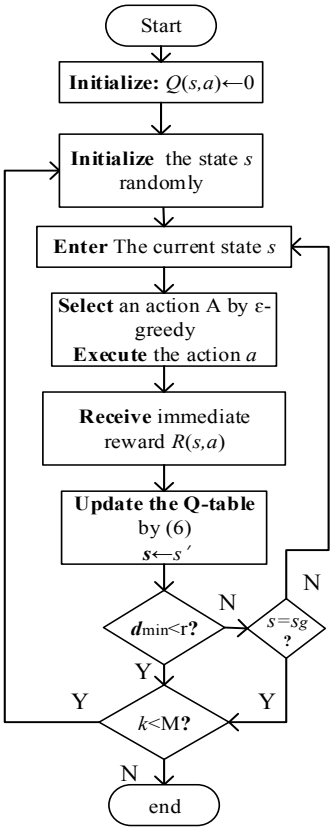


Figure 2. The flowchart of the QL for robot arm

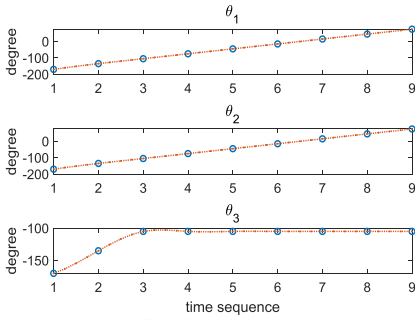
The first scenario is shown in Fig. 3. There are no obstacles in the operation space. Each joint pass through a minimum angle. The trajectory of the end-effector is quite smooth. In order to achieve adequate training, the learning times are set at 20,000. The optimal solution is obtained.

The second scenario is shown in Fig. 4. There is one obstacle. Compared with the first condition, it can be seen that the third joint of the robot arm changes its original angle at the third step in order to avoid the obstacle. After avoiding the obstacle, the end-effector returned to the trajectory of the first case because it is the shortest path in the C space. The optimal solution is obtained.

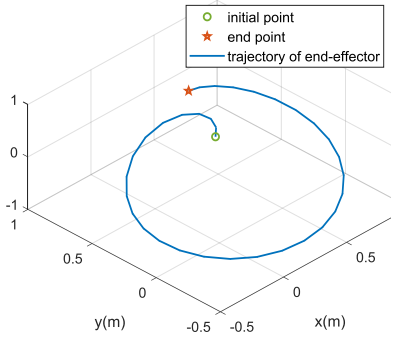
The third scenario is shown in Fig. 5. There are two obstacles. As can be seen from the Fig. 5(a), in order to avoid these two obstacles, the robot arm takes one more step

than that in the first case. It avoided two obstacles at the third and eighth step respectively. And the optimal solution is obtained.

The calculation time for different number of obstacles is shown in TABLE II.

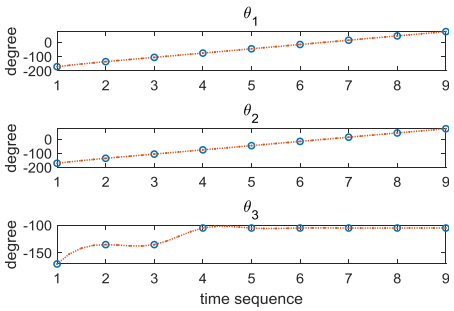


(a) The first three joints' motion

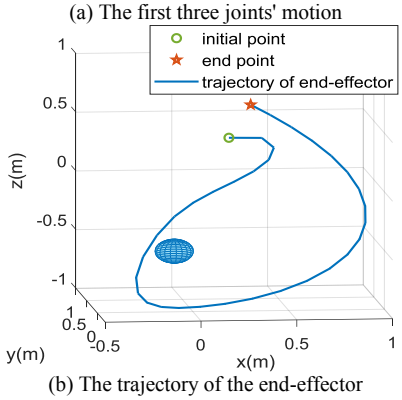


(b) The trajectory of the end-effector

Figure 3. The first three joints' motion and the trajectory of the end-effector without obstacle

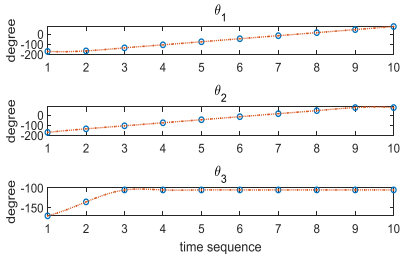


(a) The first three joints' motion

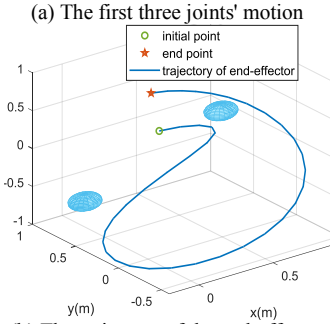


(b) The trajectory of the end-effector

Figure 4. The first three joints' motion and the trajectory of the end-effector with one obstacle



(a) The first three joints' motion



(b) The trajectory of the end-effector

Figure 5. The first three joints' motion and the trajectory of the end-effector with two obstacles

B. *The Influence of the Number of Obstacles*

TABLE II. shows the calculation time for 20000 iterations, which indicates that the more obstacles there are in operation space, the less calculation time the algorithm needs. The reasons are as follows:

- The states with obstacles are terminal states. If the algorithm achieves a terminal state, it will stop this iteration. Therefore, the time for each iteration is reduced.
- The Q values of the FS and WS is always -inf and 10 respectively. Since the algorithm is forced to stop when it reaches the terminal states. Thus, the Q value of terminal states is always the immediate reward and it converges quickly. The Q-table does not need to update these two values, which accelerates the convergence speed of the Q-table.

TABLE II. THE CALCULATION TIME FOR DIFFERENT NUMBER OF OBSTACLES

Number	0	1	2	3	4
Time(s)	521	425	307	260	200

C. *The Influence of the Penalty Coefficient μ*

TABLE III. shows the quality of solutions for the algorithm with 10000 iterations. There is no obstacle in the environment. It can be observed that when μ=0.3,0.4 or 0.5, the algorithm performs better. The reasons are as follows:

- When μ is small, the algorithm gives less punishment to the robot arm, so that the robot arm may not improve its strategy in some iterations. Therefore, the convergence speed of the algorithm becomes slow and the optimal solution cannot be obtained within 10000 iterations in most cases. Only

by increasing the number of iterations can the optimal solution be obtained.

- When μ is large, the algorithm tends to punish the robot arm a lot. Even if the robot arm reaches the target point, it still receives more cumulative punishment than the reward. Thus, the robot arm would rather stay where it is, which give a zero reward, rather than a penalty. Thus, the convergence speed of the algorithm becomes slow and the optimal solution cannot be obtained within 10000 iterations in most cases.

TABLE III. THE INFLUENCE OF THE PENALTY COEFFICIENT.

μ	0.01	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1
Optimal Solution	N	N	N	Y	Y	Y	N	N	N	N	N

IV. CONCLUSION

The QL method is adapted to plan a collision free path in a high-dimensional configuration space. Paths generated by this approach are more easily and efficiently when the order of magnitude of state space is below 10^4. From the simulation results, it can be seen that the more obstacles there are in operation space, the less calculation time the algorithm needs. And weight coefficient μ in the reward function affects the convergence speed and the quality of the solutions.

ACKNOWLEDGMENT

This research was supported by the National Natural Science Foundation under grant No. 11672294.

REFERENCES

[1] W. Xu, B. Liang, C. Li, Y. Liu, and Y. Xu, "Autonomous target capturing of free-floating space robot: Theory and experiments," Robotica, vol. 27, no. 3, pp. 425–445, 2009.

[2] T. Wang and Y. Tao, "Research status and industrialization development strategy of chinese industrial robot," Journal of Mechanical Engineering, vol. 50, no. 9, p. 1, 2014.

[3] O. Khatib, "Real-time obstacle avoidance for manipulators and mobile robots," International Journal of Robotics Research, vol. 5, no. 1, pp. 90–98, 1986.

[4] L. Kavraki, P. Svestka, and M. H. Overmars, Probabilistic roadmaps for path planning in high-dimensional configuration spaces. Unknown Publisher, 1994, vol. 1994.

[5] S. M. LaValle, "Rapidly-exploring random trees: A new tool for path planning," 1998.

[6] F. Chen, P. Di, J. Huang, H. Sasaki, and T. Fukuda, "Evolutionary artificial potential field method based manipulator path planning for safe robotic assembly," in 2009 International Symposium on Micro-NanoMechatronics and Human Science. IEEE, 2009, pp. 92‑97.

[7] S. Karaman, M. R. Walter, A. Perez, E. Frazzoli, and S. Teller, "Anytime motion planning using the rrt," in 2011 IEEE International Conference on Robotics and Automation. IEEE, 2011, pp. 1478–1483.

[8] S. S. Ge and Y. J. Cui, "New potential functions for mobile robot path planning," IEEE Transactions on robotics and automation, vol. 16, no. 5, pp. 615–620, 2000.

[9] C. J. Watkins and P. Dayan, "Q-learning," Machine learning, vol. 8, no. 3-4, pp. 279–292, 1992.

[10] D. B. Aranibar and P. J. Alsina, "Reinforcement learning-based path planning for autonomous robots," in EnRI-XXIV Congresso da Sociedade Brasileira de Computac¸ao, 2004, p. 10.

[11] M. A. K. Jaradat, M. Al-Rousan, and L. Quadan, "Reinforcement based mobile robot navigation in dynamic environment," Robotics and Computer-Integrated Manufacturing, vol. 27, no. 1, pp. 135–149, 2011.

[12] R. Yuan, F. Zhang, Y. Wang, Y. Fu, and S. Wang, "A q-learning approach based on human reasoning for navigation in a dynamic environment," Robotica, vol. 37, no. 3, p. 445C468.

[13] L. I. Yan-Hui, H. Zhao, and L. I. Shan-Shan, "New q-learning algorithm for trajectory plan of manipulator," Journal of Jilin University, vol. 31, no. 1, pp. 90–94, 2013.

[14] P. Corke, Robotics, Vision and Control: Fundamental Algorithms in MATLAB. Springer, 2011, vol. 73.