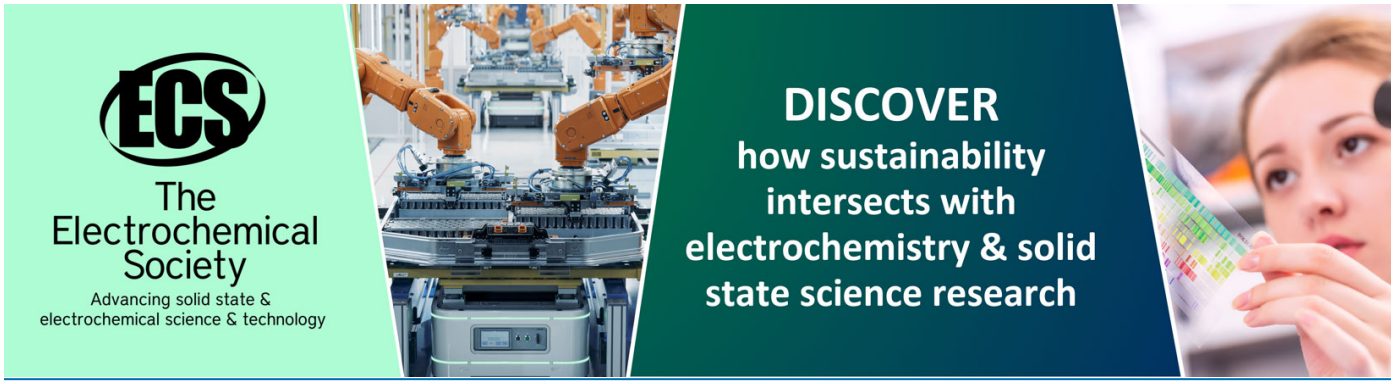
A simple implement of Q-learning in robot path planning



A simple implement of Q-learning in robot path planning

Haoran Gao[1, †], Yujie Liu[2, †], Shiqi Su[3, †], Wetao Yao[4,*, †]

[1]RDFZ Chaoyang Branch School, Beijing 100028, China

[2]GuangYi International Academy, Changsha 410000, China

[3]Mathematics and Applied Mathematics, Shanghai University, Shanghai 200436, China

[4]School of Overseas Education, Changzhou University, Changzhou 213164, China

*Corresponding author. Email: zsb@cczu.edu.cn

[†]These authors contributed equally.

Abstract—This paper firstly gives an introduction of robot path planning problem, which includes the brief definition of path planning, some representative methods and previous applications of Q-learning. Secondly, the paper compares some typical methods, like Breadth First Search and Depth-First-Search, A* and deep learning, with corresponding pseudo codes in detail. Their advantages and disadvantages are also listed in this part. Thirdly, we carry out a simple simulation experiment by applying Q-learning method. The experiment is clearly presented in several parts which includes environment establishment, realizing the Q-learning, simulation experiment and interpretation of the Q-table. In the end, a short conclusion summarizes the achievement of our results.

1. Introduction

Mobile robots have been widely used in several aspects of our life, namely, military, agriculture, and logistics [1]. Under such circumstance, it's necessary to do researches about the moving of robots. Robot navigation is a technique that shows the robot's ability to establish its own position and orientation within the frame of reference. The robot navigation mainly consists of three steps: (1) self-localisation, (2) path planning, (3) map-building and interpretation. Among them, the 'path planning' path has been discussed most.

Robot path planning is to navigate a mobile object from the start position to the goal position. There are a set of moving or fixed obstacles in the whole two-dimensional plane [2]. Refining the robot path planning not only can improve the efficiency of robot's work but also avoid some dangerous situations happen in real life in future.

Principle of artificial potential field method is a simple mobile robot path planning algorithm, it will target point location as potential energy low, obstacles in the map as a potential energy high, calculating the potential field figure known map, and ideally, the robot is like a rolling ball and roll automatically to avoid the various obstacles to the target point

In the PRM (1996) method, configuration-free space (the prearrangement of motion that can be realized) is withdrawn, reduced to, or plotted on a one-dimensional linear system. The scan of the response is limited to the system, and the moving arrangement becomes a graph search problem. This particular approach is also known as the skeleton approach, the retreat approach, or the highway approach.

A Virtual Impedance technique for multi-purpose robot improves the motion stability of the robot. This technology improves the robot's performance under dynamic conditions of static and portable obstacles. Furthermore, it is inferred from the direct framework of the above algorithm that it is more efficient than previous strategies. Program analysis is carried out on a physically moving multi-functional robot platform under internal conditions. The impedance control strategy is introduced into collision avoidance calculation, and the separated multi-functional robot and obstacle are switched to virtual obstacle resolving robot. Production efficiency and smooth variation are also the key problems of multi-functional robot headway control. Considering the high performance of independent collision avoidance in a small local space, the teleoperation headway controller realized by self-collision avoidance is a suitable path for practical application.

Another famous and basic method is Q-learning, which our article will also focus on. Q-learning is a value-based reinforcement learning algorithm depends on stochastic transitions and rewards. The key element of this algorithm is state, action and reward. Usually, the target function is Q(s,a), where 's' is the current state and 'a' is the action. The value of Q can be interpreted by the reward to carry out action 'a' under the state 's'. The main ideal of Q-learning algorithm is to make a table which include state and action to store the Q value. After that, the learning subject will choose the action that can get maximal benefit according to the table.

Moreover, another important concept in reinforcement learning is policy. There are two policies, one is behavior policy and the other is target policy. The behavior policy is used to make decision in practice, while the target policy uses the data from behavior policy. Under the condition of on-policy, the target policy approaches to its value during the implement of behavior policy. Although it seems simple, the value of target policy maybe regional optimal, not global optimal. Because the data is not comprehensive enough to find out the global optimal value. However, in the process of off-policy, the target policy will not make any decision until the behavior policy collect all data for it. Thus, the value of the target policy in off-policy is the global optimal.

A series of significant researches in different areas have been carried out by using Q-learning. For instance, Wyatt proposed Q-value sampling to solve bandit problem [3]. Also, the reinforcement Q-learning can be used to study model-free optimal tracking control for discrete-time system [4]. However, Q-learning has a better prospect in the aspect of robot. It has applications in the path planning of both solitary robot and multiple robots. After setting a reward mechanism, with the Q-learning, robot can find an optimal path which obtains the max rewards among all, staring from the current state. On the one hand, for solitary robots, Q-learning combined with neural network present a new hybrid path planning method for robot arm [5]. Besides, using Q-learning helps high-level path planning for an autonomous sailboat robot [6]. Moreover, the method based on Voronoi and Q-learning algorithm gives a path planning for car-like robot [7]. On the other hand, for multiple robots, one of the most typical cases maybe using a deep reinforcement learning Q-network(DQN) to realize the path planning for robots in unmanned warehouse [8]. Similarly, fuzzy Q-learning help deal with the underactuated dynamics of the unmanned aerial vehicles(UAVs) [9].

2. Some common path-planning algorithm principles

The path planning problem of robot is to find an optimal path to avoid obstacles in the workspace from the initial state of the robot to the target state according to some or some optimization criteria (such as minimum work cost, shortest walking path, shortest walking time). In this essay will set up a simple maze environment, red position, black position represents failure, yellow position represents success, let the red block slowly through continuous exploration and learning to walk to the yellow position as shown in Fig.1.The goal is to find an algorithm to get the best path to the end with higher efficiency.

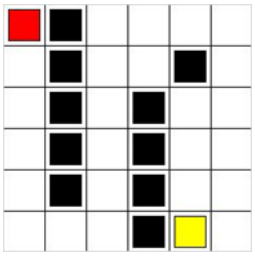


Fig.1 path planning diagram

2.1. Breadth First Search and Depth-First-Search.

The most basic principles for path-planning are Breadth First Search (bfs) and Depth-First-Search(dfs). bfs takes the starting point "A" as the center of the circle, first searches all points around A to form "A" point search area similar to a circle, and then expands the search radius to get a larger search area until the end "B" point is found in the search area. bfs guarantees minimal cost to get from the starting point to the end point that means getting the shortest path regardless of whether the process involves searching many grids, so this way unless can get the best path, but have low efficiency. [10] The table Algorithem.1 show Pseudo code for bfs[11].

Algorithem.1 pseudo code

```
1    q.push(head);
2    while(! q.empty()) {
3      temp=q.front();
4      q.pop();
5      If (tempI is the target state)
6          Output or record
7     If (temp is invalid)
8          continue;
9    If (temp)
10         Q.p ush (temp + ¦ Δ);
11   }
```

dfs takes starting point "A" and searches in the direction of end point "B". dfs is to search as far away from A as possible and as close to B as possible, unless don't know where is the shortest path and can't find it. dfs is that by constantly correcting the relationship between the direction of travel and the direction of the destination, this way need search less area but can't find the best path between start point to end point.[12] .The table Algorithem.2 show Pseudo code for dfs[13].

Algorithem.2 dfs pseudo code

```
1   Void DFS (state A) {
2    If (A is illegal)
3       return;
4    If (A is the target state)
5      Output or record path
6    If (A is not the target state)
7       DFS (A + Δ)
8   }
```

*2.2.A**

A new algorithm can be obtained by combining bfs and dfs that is A*. A* considering the cost from the starting point through the current route, and constantly calculating whether the current route direction is closer to the direction of the end. A* formula is

$$F(x) = G(x) + H(x).$$

F(x)is the comprehensive priority of node x and we will use the smallest one. G(x) is the cost of the distance between node x and the starting point, H(x) is the estimated cost of node x to the end point. And then each move one spot will afresh to confirm the direction until to the end point.

A* is the most efficient way to find the path when not considering going to different points have different costs, such as having amplitude between the spots.

2.3. deep learning

The deep learning way is Q-learning and sarsa. Q-Learning algorithm is a kind of off-policy reinforcement Learning algorithm. The algorithm uses the value of each step to perform the next action. The agent based on Q-Learning algorithm can judge the next step only by the current state without knowing the overall environment. Q-learning is a value-based algorithm among reinforcement Learning algorithms. Q refers to the expected benefits of taking an action in a certain state at a certain time. The environment will feedback corresponding returns according to the actions of the agent, so the main idea of the algorithm is to construct a Q value table between the states and actions, and then select the actions that can obtain the maximum benefits according to the Q value. But Q-learning will take the behavior with the highest expected reward every time, you may not be able to explore other possible behaviors in the training process, and even enter the "local optimal".

Sarsa is state-action-reward-state-action, it is also store action value functions in q-table mode. The difference is that Sarsa updates differently than Q-learning, it is off-policy and will update data after action.

3. Test Results

3.1. Environment Establishment

For prior declaration, we use Python as our experiment tool. To simulate the experiment, we should firstly set up the environment. The definition of path planning is to find a feasible route from the starting point to the end point avoiding obstacles. According to such requirement, we create a maze which is 6 units in length and 6 units in width. We present it in the form of grid in Figure 2. The Figure 1 also gives an overview of the initial environment setting. The small red square is our experiment robot with the initial position on the upper left corner of the maze. The black blocks represent traps. In contrast, the white squares have no properties. The robot can move freely through out the whole maze. When robot move onto a square, it can move to adjacent squares. The yellow square stands for the treasure with an extra reward. The blue square on the bottom right corner of the maze is the terminal. When the robot get to the terminal successfully, then a feasible path is achieved. However, such feasible path can only be found after continuously learning and exploring.

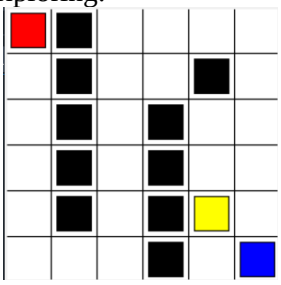


Fig.2 experiment environment

Secondly, we are about to define the movements in the maze. Our explanation will still base on Figure 1. By choosing different moving directions (up/down/left/right), the current position changes following by different outcomes. It is worth noting that because the scale of the maze is 6*6, the first row cannot manage to carry out the ‘up’ action. Similarly, the last row cannot take ‘down’ action, the first column cannot take ‘left’ action and the last column can not take ‘right’ action. These restrictions should be added to the movement of robot to ensure the robot always adopting valid actions.

Thirdly, it is also important to build a incentive system. The matrix below shows the reward matrix which is according to the Figure 1. For instance, if moving to the black block, the score of this route will get a -10 credit punishment. Contrarily, if moving to the yellow block, the score of this route will have a 3 credit reward. The white squares have neither rewards nor punishments, so 0 is given to be these positions' credits. When get to the terminal, the robot will get 10 credits and end this trial. All the credits should return to the score of current moving strategy.

$$\text{Reward Matrix}=\begin{bmatrix} 0 & -10 & 0 & 0 & 0 & 0 \\ 0 & -10 & 0 & 0 & -10 & 0 \\ 0 & -10 & 0 & -10 & 0 & 0 \\ 0 & -10 & 0 & -10 & 0 & 0 \\ 0 & -10 & 0 & -10 & 3 & 0 \\ 0 & -10 & 0 & -10 & 0 & 10 \end{bmatrix}$$

Fig.3 reward matrix

3.2.Realizing the Q-learning

Q-learning is a typical off-policy method in path planning. The off-policy means the learning policy differs from the enforcement policy, which is reflected in the Q-learning policy. The key equation in Q-learning is:

$$Q(s,a) <- Q(s,a) + \alpha[r+\gamma max_{a'}Q(s',a')-Q(s,a)]. \quad (1)$$

In the stage of learning, the agent has a greedy attribute, which means it will always take the action that maximizes the reward. However, in the stage of enforcement, the parameter ε-greedy is introduced. In our experiment, an ε-greedy is set to be 0.8, so there is a 80 percent possibility for the agent to follow the greedy attribute, choosing the action which results in the largest reward according to the Q-table. While under the left 20 percent possibility, the agent will take a totally random action regardless to the Q-table. Another parameter α is the study rate that decides the amount of error to be learned. Moreover, the γ represents the loss of the reward which is introduced to make up for the difference between the real reward and the estimated reward. But our both the real and estimated rewards in our experiment are the same, then the γ is 1 in this experiment. In order to be clear, the Pseudo Code of Q-learning is shown in Figure 3. Q(s,a) represents a reward value by taking action 'a' under the state of 's'.

Algorithem.3 Q-learning pseudo code

```
Initialize Q(s,a) arbitrarily
Repeat (for each episode):
    Initialize s
    Repeat (for each step of episode):
        Choose a from s using policy derived from Q(e.g., ε − greedy)
        Take action a, observe r, s′
        Q(s,a) ← Q(s,a) +α[r+γmax_a′Q(s′,a′)-Q(s,a)]
        s ← s′
    until s is terminal
```

In conclusion, to realize the Q-learning, we should first draw the Q-table, which record the value of each state and action. Next, an ε-greedy is set to influence the agent choice in the enforcement stage. Finally, the agent is able to find a feasible path according to the Q-table. Thus, the Q-learning is achieved.

3.3.Simulation Experiment

Our simulation follows three steps:

1. get the next process according to the current state,
2. carry out the process and obtain the new state,
3. record the value calculated by the algorithm.

Our algorithm gets its first reward at the 25th trail, and at about the 50th trail, the shortest route is almost obtained. Fig 3 shows the situation when achieving the success.

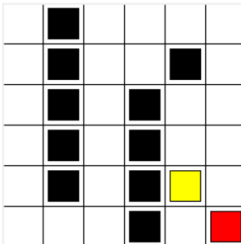


Fig.4 achieve the success

3.4.Interpretation of the Q-table

Here is part of the Q-table:

	0	1	2	3
[5.0, 5.0, 35.0, 35.0]	0.0	0.0	-0.01	0.0
[5.0, 45.0, 35.0, 75.0]	0.0	0.0	-0.01	0.0
[45.0, 5.0, 75.0, 35.0]	0.0	0.0	0.00	0.0
[5.0, 85.0, 35.0, 115.0]	0.0	0.0	-0.01	0.0
[45.0, 85.0, 75.0, 115.0]	0.0	0.0	0.00	0.0
[45.0, 45.0, 75.0, 75.0]	0.0	0.0	0.00	0.0
[5.0, 125.0, 35.0, 155.0]	0.0	0.0	-0.01	0.0
[45.0, 125.0, 75.0, 155.0]	0.0	0.0	0.00	0.0
[5.0, 165.0, 35.0, 195.0]	0.0	0.0	-0.01	0.0.

Fig.5 The first 9 rows of the Q-table

The four values in bracket stands for current state (s). Here [5.0, 5.0, 35.0, 35.0] represents the initial position (state) in the maze. The first two numbers show the left top coordinate and the last two show the right bottom coordinates. The four numbers in first row imply four actions: up/right/right/left. And 4 columns below them shows four rewards by taking different actions according to the current state. Taking the second row '[5.0, 5.0, 35.0, 35.0] 0.0 0.0 -0.01 0.0' as an example, if the agent turn to right, it will be caught by a trap. Thus, the value here is -0.01. Moreover, 0.0 means that the agent moves to a blank square, 0.003 means the agent gets the treasure and 0.01 means the agent reaches the terminal point.

With refinement of the table, the first success happened at approximately the 30th trail.

	0	1	2	3
[205.0, 205.0, 235.0, 235.0]	0.00	0.00	0.000000	0.010000

Fig.6 the first success

The last step from above table is a turning point. After that, the following trails will help achieve the optimal path.

4. Conclusion

In this paper, we introduced the methods of path-planning and how it can be solved by using Q-leaning, including the benefits of Q-leaning, as well as three principles of path-planning algorithm, Breadth First

Search and Depth-First-Search, A* and deep learning. Then, we used Python to achieve Q-learning based on a maze which is 6 units in length and 6 units in width. In short, to realize the Q-learning, we should first draw the Q-table, which record the value of each state and action. Next, an ε-greedy is set to influence the agent choice in the enforcement stage. Finally, the agent is able to find a feasible path according to the Q-table.

We got the result, it is concluded that the boundary conditions at ends of the specimens have great effect on static mechanical properties of the composite slabs presented here, and the composite slab with restraint at four corners has the largest bearing capacity. We have proved that robot can learn by themselves to find the best path, this further confirms the importance of Q-learning in path-planning research. We are looking forward to realize the goal of deeper implement of Q-learning and to learn more knowledge about path-planning in the future

References

[1] Jiang, J., Xin, J. (2019). Path planning of a mobile robot in a free-space environment using Q-learning. Progress in artificial intelligence, 8(1), 133-142.
[2] Fujimura, K., Samet, H. (1989). A hierarchical strategy for path planning among moving obstacles (mobile robot). IEEE transactions on robotics and Automation, 5(1), 61-69.
[3] Dearden, R., Friedman, N., Russell, S. (1998). Bayesian Q-learning. In Aaai/iaai (pp. 761-768).
[4] Liu, Y., Yu, R. (2018). Model-free optimal tracking control for discrete-time system with delays using reinforcement Q-learning. Electronics Letters, 54(12), 750-752.
[5] Abdi, A., Adhikari, D., Park, J. H. (2021). A Novel Hybrid Path Planning Method Based on Q-Learning and Neural Network for Robot Arm. Applied Sciences, 11(15), 6770.
[6] Silva Junior, A. G. D., Santos, D. H. D., Negreiros, A. P. F. D., Silva, J. M. V. B. D. S., & Gonçalves, L. M. G. (2020). High-level path planning for an autonomous sailboat robot using Q-Learning. Sensors, 20(6), 1550.
[7] Alhassow, M. M., Ata, O., Atilla, D.C. (2021). Car-Like Robot Path Planning Based on Voronoi and Q-Learning Algorithms. International Conference on Engineering and Emerging Technologies (ICEET), 2021, pp. 1-4.
[8] Yang, Y., Juntao, L., Lingling, P. (2020). Multi-robot path planning based on a deep reinforcement learning DQN algorithm. CAAI Transactions on Intelligence Technology, 5(3), 177-183.
[9] Shi, H., Li, H., Hwang, K. S., Pan, W., Xu, G. (2018). Decoupled Visual Servoing With Fuzzy Q-Learning. IEEE Transactions on Industrial Informatics, vol. 14, no. 1, pp. 241-252.
[10] Zhang, Y., Han , L. (2012). Implementation of path algorithm with breadth first search algorithm.
[11] DFS and BFS idea template pseudocode. https://blog.csdn.net/dlpf_c/article/details/76695130.
[12] Ru, Z., Huajun, W. (2021). Spatial search algorithm based on breadth first search. School of computer science and technology. Chengdu University of Technology.
[13] A* algorithm and pseudocode. https://blog.csdn.net/weixin_43821874/article/details/94737711.