Implementing Q-learning on Simple Vehicle: Learning Obstacle Avoidance in Different Environments

Peiheng Lu
University of Edinburgh
801, Block 18, Mengyuan Community, Haiguan road, Hefei, Anhui, China
*s1948149@ed.ac.uk

Abstract. Q-learning has been applied to many research projects in the past few decades. Its simple logic of mapping a corresponding action to each possible state has made it one of the most popular methods in reinforcement learning. In this implementation of Q-learning, a simple controlling scheme was used. The percept of the robot is taken as the state, and the actions give instruction to the actuating device of the robot. The goal is for this robot to pick up the obstacle avoidance skill after series of training. A sanity check for such an idea was provided in this essay. The basic disc robot from Robotics Playground was used, and the basic form of Q-learning was utilized. Our vehicle’s obstacle avoidance ability was assessed in environments with different levels of complexity using the Collision-Time graphs produced. The results showed very limited improvements in terms of reducing the number of collisions in all of the environments, and some fluctuations in the number of collisions were shown in the diagram. Combining the fluctuations with observations during the experiment process, we came to a conclusion that using Q-learning to navigate an arbitrary environment, the state space, and action space could be too large for any efficient training to be done.

1. Introduction
Pathfinding is a fundamental component of many important applications in the fields of GPS, video games, robotics, logistics, and crowd simulation and can be implemented in static, dynamic, and real-time environments [1]. And obstacle avoidance is an important component for path finding, as avoiding collisions with the environment and not having unrealistic plans to go through an obstacle is an import utility of a navigation program for a mobile robot. Considering its importance, we shouldn’t be surprised to see the wide variety of approaches people came up with to tackle it, and one of the most popular solutions would be using reinforcement learning, a general framework for decision making under uncertainty [2].

Reinforcement learning proposes a way of programming agents by reward and punishment without needing to specify how the task is to be achieved [3]. It punishes the agent for actions that are against the expected behavior and rewards the agent if the action is helpful for getting the designated task achieved. It does come with a formidable computational expense and has not really caught on until until the last two to three decades. Two kinds of approaches for reinforcement learning were proposed, which are model-based and model-free reinforcement learning. Model-free means learning a controller without learning a model, while model-based methods learn a model, then use it to derive a controller. Q-learning and its variations, being one of the simplest methods in model-free approaches of reinforcement learning, has appeared in a large amount of research.
Q-learning constructs a two-dimensional Q-table for choosing an action under a certain situation. The input from sensors is used to choose a corresponding row from the Q-table. Each row of the Q-table corresponds to the same set of actions, and each action is assigned with a value, default to be 0. The action with the maximum value will be chosen as the action to take under the current situation. The value that corresponds to a state action pair will be decreased if it leads up to failure and increases if it contributes to success [4].

One of the classical applications of Q-learning requires a fully observable environment that can be discretized. It uses the discretized environment as input and comes up with a path for the mobile robot to follow using a control scheme like inverse kinematic control. It converts planning in a continuous environment into grid path planning, which has a great range of applications in areas like robotics and video games [5]. There have been many efficient solutions to grid path planning, but all of them have the same limitation – every time the agent encounters a new environment, it has to start the learning process all over again. An application of Q-learning that can be applied to a wide variety of vehicles and only needs to train the robot once for it to tackle any possible environment is desired.

This project checked whether a simple Q-learning implementation is efficient enough to handle three environments of increasing complexity. The implementation is straightforward - the data from the two distance sensors on the agent is used as ‘state’ required by the Q-learning algorithm, and the algorithm provides a suitable ‘action’, which would be mapped to the torque of the agent’s two motors. Tests were implemented on each of the three environments, and three histograms showing the number of collisions were produced to reflect the result of the training.

2. Methods
The disc robot we used has two types of sensors mounted, two distance sensors and one contact switch. The input from two distance sensors is mapped as the state for Q-table, and the contact switch is used to approximate collisions with obstacles in the environment.

2.1. Packages and parameters
The platform chosen for conducting this experiment was MATLAB 2020b [6]. Robotic playground and their dependencies were used for the modeling of the disc robot and obstacle environment [7]. The default disc robot from the package was used, and its detailed parameters were as follows: The distance between wheels of the disc robot was 0.5 meters, with wheels of radius 0.1m. The two mounted distance sensors had a position offset of [0.2 0.15] with orientation offset 45 and a position offset of [0.2 -0.15]. Both sensors had a maximum range of 20 meters and a minimum range of 0 meters, with a resolution of 0.01 meter and a sample time of 0.1 seconds. The input value will be processed to be accurate to 0.2m and with a range of 0 to 1.8 meters inclusive. The contact switch had a position offset of [0 0], with an orientation offset of 0 degrees. The distance for it to be triggered was set at 0.02 meters. It was much less than the radius of the disc robot itself, but this setting gave the best estimation of collision. The arena was of size 5 x 8, and the starting position of the disc robot was [-1.4 0]. In the simple setting, a block of 2 x 5 was placed in the position [0 0] of the arena. The medium setting had two blocks of size 2 x 2, placed at [0 2] and [0 -2], and the hard setting has three blocks, two of size 3 x 1 at [0 2.5] and [0 -2.5], one of size 1.5 x 2 at [0 0]. (figure 1)

![Figure 1 Layouts of the environments](image-url)
2.2. **Implementation of Q-learning**

Broadly speaking, the Q-learning algorithm requires two types of input and produces one kind of output. A reward signal is needed for punishing or rewarding the robot. The current (state, action) as well as the (state’, action’) after the action was implemented was needed for the update of Q-table entry corresponding to (state, action). When given a state, Q-learning will search the list of actions for the highest value and output the corresponding action.

The update function we used is as follows: [8]

\[
Q_{\text{new}}(s,a) \leftarrow Q(s,a) + \alpha \times (r_i + \gamma \times \max_a Q(s',a) - Q(s,a)) \tag{1}
\]

In our implementation, the reward signal was provided by the contact switch. When an obstacle comes into close proximity with the contact switch, the switch will send out the signal of 1, telling the system that a collision has happened and the agent needs to be punished. The vehicle will reverse for some time (such time can vary) to get the agent away from the obstacle. Collision detection is deactivated during such a period so that it won’t be stuck in an infinite loop of reversing into an obstacle.

The states were provided by the two distance sensors. They are accurate to 0.2 meters and ranged from 0 to 1.8 meters inclusive. Thus 10 discrete data will be possible for each distance sensor, and two sensors will provide us with 100 possible states, denoted by [leftS rightS], where leftS are the data from the left sensor and rightS is the data from the right sensor.

Five actions were provided for each state and will be denoted by [leftM rightM], where leftM is the torque for left motor and rightM is the torque for right motor. The five possible actions were: [100 100], goes straight; [100 -10], turn right; [-10 100], turn left; [80 -30] sharp right; [-30 80] sharp left. This gave us a 100 x 5 Q-table.

We updated the Q-table and chose an action every 0.1 seconds, which was the same as the sampling frequency of all our sensors. This frequency can be increased for higher precision. However, this increases the requirement for computing power and training time.

The update function was a direct implementation of function 1 above, however, with some minor changes. The \(Q(s,a)\) in our update function was not really the current state and action in our implementation, but the state and action chose 0.1 seconds ago. \(Q_{\text{new}}(s,a)\) was the current state and action. And the reward was the reward signal from the last state. This is because we cannot accurately predict the next state our agent will be in; we can only update it after the next state has been reached.

The entire process was implemented in a Simulink simulation for easy monitoring, and all the variables, including the Q-table, were stored in the Simulink Data Store Memory block for convenient read and write.

3. **Results**

The diagrams show the changes in the number of collisions as the experiment time progresses. The y axis ‘count’ shows the number of collisions that occurred in a 100 second period and is ranged from 0 to 30. The x axis ‘time’ is ranged from 0 to 10000 with a bin width of 100 seconds. A roughly decreasing trend can be observed from all three graphs during the first 1000 (2000 for the easy environment) seconds. It shows that the algorithm did help with our agents’ ability of obstacle avoidance. However, we can see that after the initial decreasing period is over, the number of collisions rises sharply to nearly reach the level before any learning was done, then the value just fluctuates with no clear trend.

Not surprisingly, when comparing the three figures side by side, the histogram for the ‘easy’ environment in figure 2 has the lowest overall number of collisions. However, the training seems to have the least effect on this environment as it still could reach the same number of collisions as when the training started. At around 9300 seconds, the collision number was even higher than at the beginning.

The ‘medium’ environment in figure 3 started out with a much higher number of collisions than ‘easy’, which was slightly decreased by the learning process, and afterward, the trend of change in collision number was very similar to the ‘easy’ environment. The training helped bring down the highest number of collisions by about 5, which is still far from the goal of eliminating all the collisions, and it
did bounce back to a much higher value than the start at around 9600 seconds, which indicated some severe instability.

The ‘hard’ environment (figure 4) had the highest overall number of collisions. The training had some effects as the collision number never reached the level at the start of the training. Still, the effects were very limited and similar to ‘medium’. The collision number was brought down by about 5. Still, its result was the most promising as the collision number never reached the level at the start.
4. Discussion
By observing the Q-table generated and the simulation produced, we can see that there are two main causes for the fluctuating period after the initial decreasing stage: First, there are some areas in the graph that are harder to traverse than others, such as the corners of the arena, and thus the number of collisions will increase. To increase the agility and allow the vehicle to make more appropriate choices, we need to increase the action space. But 10 more actions add 1000 more entries in the Q-table, and at the beginning of the project, we have a state space of 16, we trained our model for 14 hours, and the performance still did not improve.

However, the second cause has a much more important effect. After training for 1000 to 2000 seconds, the vehicle could have a clear idea about what to do in most of the scenarios, in other words, when to turn right and when to turn left, also how sharp a turn should it make. Then the limiting sensory power started to cause problems. Some scenarios that require different actions are reflected in the same state. For example, when heading head-first towards a wall with a corner nearby requires different actions when the corner is to the left of the vehicle than when it is to the right, but the state will be [a a] for both scenarios, with a being a positive decimal value. This can cause confusion in the Q-table and overwrites some previous choices. But having more sensors requires more state. One more sensor in our current Q-table will bring 10 times more states, while decreasing the number of states of each sensor can cause the vehicle to lose some of its ability to reason under different scenarios, still causing damage to the obstacle avoidance ability.

The fact that a possible increase in vehicle performance may bring exponential growth in state and action space is abysmal, and the prospect of wider application is dim.

Nonetheless, there was a relevant project that manages to achieve some level of success. In an environment designed to help the robot make consistent decisions about turning left, turning right, or going straight, the limited sensory power can be made up of, and this method can be useful [9]. The robot still cannot deal with many situations. For example, when there is a wall right in front of it, it still won’t know whether turning left is better than turning right.

Thus, the applicability of basic Q-learning in a general environment is still questionable, but in some more ideal experiment environments, it can work satisfactorily.

5. Conclusions
We have presented the design of our vehicle, both at the software and hardware level. We also showed its performance in various environments in terms of the number of collisions. The relatively insignificant learning outcome and constant fluctuation in the vehicle’s performance have shown some severe limitations in the design of the vehicle. We realized that for Q-learning to work in an arbitrary environment, the abstracted state and action space will have to be enlarged greatly and thus more likely to cause efficiency issues. This has revealed a bleak fact that the simple design using Q-learning alone may not be enough for learning obstacle avoidance in an efficient manner. For researchers and students working on similar navigation or obstacle avoidance task, using only Q-learning is likely to be insufficient. Combining Q-learning with more advanced techniques might be necessary in real environment with continuous state and action space [10].

References
[1] Zeyad Abd Algfoor, Mohd Shahrizal Sunar, and Hoshang Kolivand. “A Comprehensive Study on Pathfinding Techniques for Robotics and Video Games.” In: Int. J. Comput. Games Technol. 2015 (2015), 736138:1–736138:11. url: http://dblp.uni-trier.de/db/journals/ijcgt/ijcgt2015.
[2] Nicolas Carrara et al. “Scaling up budgeted reinforcement learning”. In: CoRR abs/1903.01004 (2019). arXiv: 1903.01004. url: http://arxiv.org/abs/1903.01004.
[3] L. P. Kaelbling, M. L. Littman, and A. W. Moore. “Reinforcement learning: A survey”. In: Journal of Artificial Intelligence Research 4 (1996), pp. 237–285.
[4] Christopher John Cornish Hellaby Watkins. “Learning from Delayed Rewards”. PhD thesis. Cambridge, UK: King’s College, May 1989. url: http://www.cs.rhul.ac.uk/~chrisw/new_thesis.pdf.

[5] Aleksandr I. Panov, Konstantin S. Yakovlev, and Roman Suvorov. “Grid Path Planning with Deep Reinforcement Learning: Preliminary Results”. In: Procedia Computer Science 123 (2018). 8th Annual International Conference on Biologically Inspired Cognitive Architectures, BICA 2017 (Eighth Annual Meeting of the BICA Society), held August 1-6, 2017 in Moscow, Russia, pp. 347–353. issn: 1877-0509. doi: https://doi.org/10.1016/j.procs.2018.01.054. url:https://www.sciencedirect.com/science/article/pii/S1877050918300553.

[6] MATLAB 9.9.0.1592791 (R2020b) Update 5. The MathWorks, Inc. Natick, Massachusetts, 2020.

[7] MathWorks Student Competitions Team (2021). Robotics Playground. url: https://github.com/mathworks-robotics/robotics-playground/releases/tag/20.1.4.

[8] R. S. Sutton and A. G. Barto. Reinforcement Learning: An Introduction. Cambridge, MA: MIT Press, 1998.

[9] T. Ribeiro et al. “Q-Learning for Autonomous Mobile Robot Obstacle Avoidance”. In: 2019 IEEE International Conference on Autonomous Robot Systems and Competitions (ICARSC). 2019, pp. 1–7. doi: 10.1109/ICARSC.2019.8733621.

[10] Chris Gaskett, David Wettergreen, and Alexander Zelinsky. “Q-Learning in Continuous State and Action Spaces”. In: Proceedings of the 12th Australian Joint Conference on Artificial Intelligence: Advanced Topics in Artificial Intelligence. AI ’99. Berlin, Heidelberg: Springer-Verlag, 1999, pp. 417–428. isbn: 3540668225.