Lab 5 Report: Robot Learning – Reinforcement Learning

Mohammad Rami Koujan

1 Introduction

Reinforcement learning offers to robotics a framework and a set of tools for the design of sophisticated and hard-to-engineer behaviors. A remarkable variety of problems in robotics may be naturally phrased as ones of reinforcement learning. Reinforcement learning (RL) enables a robot to autonomously discover an optimal behavior through trial-and-error interactions with its environment. Instead of explicitly detailing the solution to a problem, in reinforcement learning the designer of a control task provides feedback in terms of a scalar objective function that measures the one-step performance of the robot.

Q-learning is a method for solving reinforcement learning problems. It has the potential to reduce robot programming effort and increase the range of robot abilities. In fact, the aim of this lab assignment is to analyse and implement a reinforcement learning algorithm (Q-learning) to send the robot to a goal position.

2 Algorithm Analysis

This section presents in details the programmed Q-learning algorithm in Matlab function where the main objective is to find the goal in a finite 2D environment that is closed and contains some obstacles. The chosen map is of size 20x14 which means there are 280 distinct states that the robot will choose from depending on its movement where only up, down, left, and right actions are allowed.

The idea of the accomplished implementation is to consider the problem as an episodic one and then to repeat the entire learning process for several number of episodes where the following steps were followed:

1. Defining a loop to handle the iteration for n number of episodes
2. Inside the previous loop, first, the state (initial location) of the robot is initialized randomly to be in any free cell inside the tested map. Therefore, the randomly generated number has to be checked to make sure it is neither an obstacle nor the goal position.
3. Another loop is defined to repeat the procedure for m number of iterations.
4. Inside the previous loop, an action is chosen based on ϵ-greedy policy. This means that with probability ϵ the action is generated randomly from the set $\{up, down, left, right\}$ and with probability 1-ϵ the value of the Q function is inspected at the chosen location to obtain the action that leads to the highest value. Indeed, the size of the defined Q function is equal to the tested map size multiplied by 4, which is the total number of allowed robot movements.
5. Moving the robot according to the previously selected action.
6. Checking the new state of the robot and setting the reward "r" as follows:
   (a) If the new state is an obstacle position, the robot stays in its current position and "r" takes a value of -1.
   (b) If the new state is a free cell other than the goal, "r" takes a value of -1 as well.
   (c) Finally, if the new state is the goal position, the value of "r" is set to 1 since the robot has achieved the task.
7. Updating the Q function according to the following equation:
   Q(s,a)= Q(s,a)+α.(r+γ. $maxQ_{a\prime}(s\prime + a\prime) - Q(s,a)$)

   where α is the learning rate of the algorithm, γ is the discount factor, and ϵ is the random action probability as described in step 4.

8. Updating the current state to the new one which is generated in step 5.
9. Checking whether the goal is achieved so as to finish the episode.

3 Tests and Results

This section presents and discusses the results of testing the previously implemented algorithm on the selected map. In order to test the algorithm, first, the effectiveness of the algorithm is computed by executing a separate procedure after each 100 episodes of the main algorithm. This procedure works as follows:

1. Runs 100 episodes.
2. Inside each episodes, it generates the state randomly as in step 2 of the main algorithm.
3. For the same number of the previously defined iterations (m), greedy policy is followed to choose the action.
4. In the same way as in step 4 of the main algorithm, the value of r is set but this time it is updated in a cumulative way and after that the current state is updated according to the one generated in step 2.
5. Finally, at the end of the last episode, the cumulative reward is averaged over 100.

Figure 1 shows the result of running such a test where the following parameters are used as an input to the implemented algorithm: α=0.1, γ=0.9, ϵ=0.3, $n\_episodes$=20000, $n\_iterations$=50.

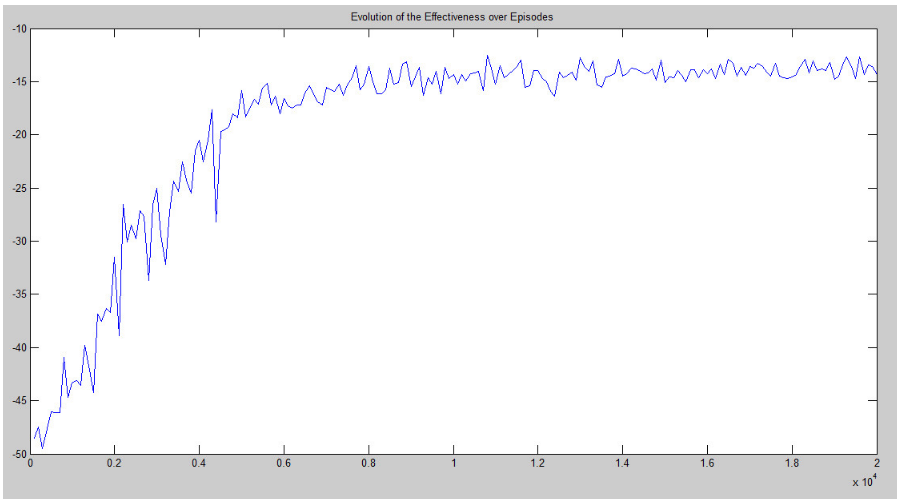

Figure 1: Effectiveness of the implemented Q-learning algorithm over episodes

As it is shown in preceding figure the effectiveness fluctuates and decreases during some episodes since the learning was enabled in those episodes.

Figure 2 shows the value of the state value function V where it is computed by choosing the action with the highest reward at each state in the Q function. The red cells in the shown representation indicate obstacles and the cell at location (18,3) which is surrounded by four dark red cells is the goal. Other colors indicate how the rewards are increasing (up to zero) toward the defined goal.

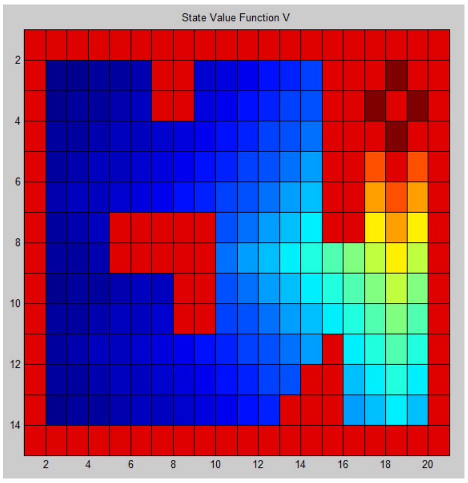

Figure 2: State value function of the implemented algorithm

The optimal policy is presented in the following figure. This representation shows clearly the action that is the best to choose at each state in order to navigate as less cells as possible before reaching the final goal.

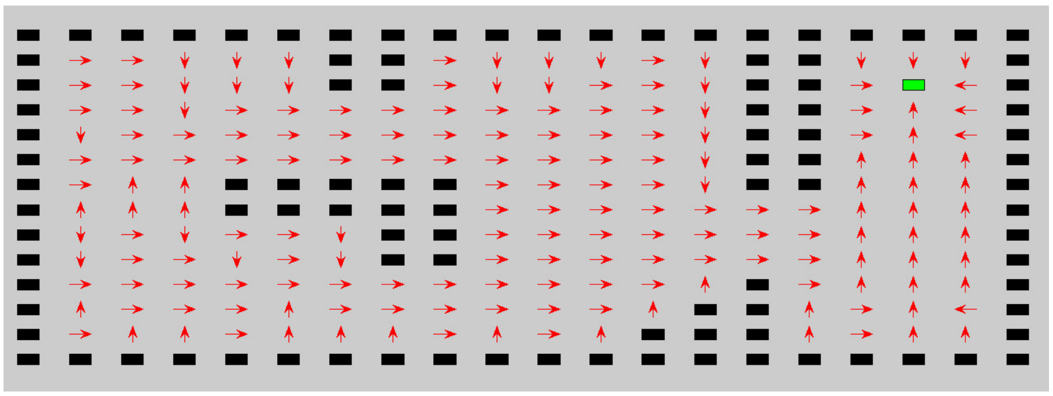

Figure 3: The optimal policy of the tested algorithm

The Q-learning algorithm with an ϵ-greedy policy has three main parameters: the learning rate (α) , the discount factor (γ) , and the ϵ-greedy parameter. In fact, α and γ are usually between 0.0 and 1.0, and ϵ is usually small. The learning rate parameter limits how quickly learning can occur. In the Q-learning algorithm, it governs how quickly the Q-values can change with each state/action change. If the learning rate is too small for the task at hand, learning will occur very slowly. If the rate is too high, then the algorithm might not converge, i.e., Q-values might not stabilize near values that represent an optimal policy. The discount factor controls the value placed on future rewards. If the value is low, immediate rewards are optimized, while higher values of the discount factor cause the learning algorithm to more strongly count future rewards. The value of ϵ is a probability of taking a non-greedy (exploratory) action in ϵ-greedy action selection methods. A non-zero value of ϵ insures that all state/action pairs will be explored as the number of trials goes to infinity. In a greedy method, $\epsilon = 0$, the algorithm might miss optimal solutions; however, in a near greedy method, a value of ϵ that is too high will cause the robot to waste time exploring suboptimal state/action pairs that have already been visited.

The following figure demonstrates the optimal policy achieved by the programmed Q-learning algorithm with the following parameters: α=0.8, γ=0.9, ϵ=0.3, $n\_episodes$=2000, $n\_iterations$=50.

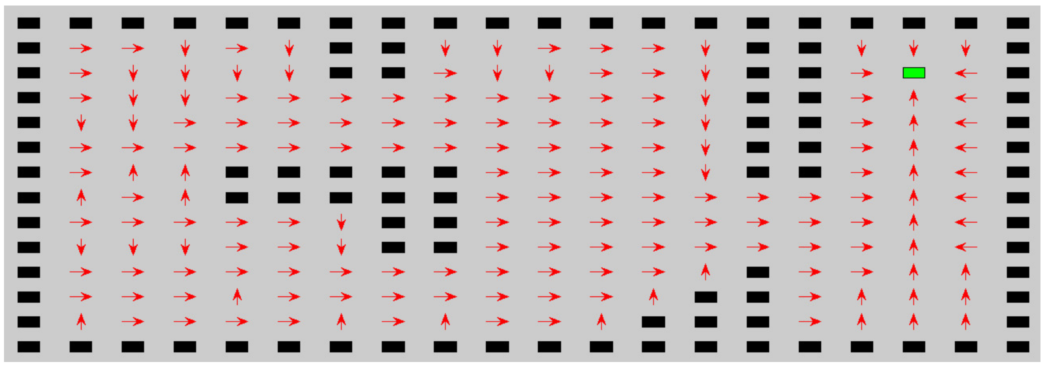

Figure 4: The optimal policy of the tested algorithm with α=0.8 and $n\_episodes$=2000

Where it is evident from the previous figure how the policy is also an optimal one with much less number of episodes but with a higher value of the learning rate (α). However, as it is discussed earlier, when the rate is too high, then the algorithm might not converge.

4 Conclusion

In this lab assignment, a reinforcement learning algorithm (Q-learning) is designed, analysed, and implemented in Matlab function. Furthermore, it is tested with a map of a size 20x14 and different parameters. The presented results reveal that choosing the optimal set of Q-learning parameters is always a trade-off between many issues governed by the application itself.