Computational Robotics

Lab 1 - Markov Decision Processes

Andrew Choi

October 2019

1 Introduction

This goal of this lab is to explore finite Markov Decision Processes to control a simple discretized robot. To do so, a gridworld is constructed with well-defined boundaries and rewards. In this gridworld, we develop and implement a model for robot behavior and use it to accomplish a prescribed task.

2 Mathematical Formulation

2.1 Markov Decision Processes

Before delving into the details of the experiment, the finite Markov Decision Process as well as the methods employed (value iteration and policy iteration) are described mathematically.

Markov decision processes (abbreviated MDP from hereon) are a classical formalization of sequential decision making. Driven by rewards, these decisions are made not just influenced by immediate rewards, but also by possible future rewards. The MDP can be described as a tuple

$$(S, A, P, R, H, \gamma),$$

where S is the state space, A is the action space, P is the transition probabilities between states given a certain action, R is the immediate reward received from transitioning to a certain state, H is the horizon, and γ is the discount factor. With the following parameters, the goal of an MDP is to find an optimal policy $\pi^*(s)$ that outputs the best action for every state in the state space. To find such a policy, we introduce the concept of value functions in the next section and explain its role in deriving the optimal policy.

2.2 Value Functions and Policies

This formulation allows us to construct a value function $V(s)$ $\forall s \in S$, which is the expected discounted returns for every state.

$$V^\pi = \mathbb{E}[\sum_t \gamma^t r(s_t)]$$

This value function is different depending on the policy $\pi(s)$ which deterministically chooses an action depending on the current state. Using the value function, we can compute optimal policies for finite state spaces using two methods, policy iteration and value iteration. It should be noted that for these two methods to converge to an optimal policy, the state space must be discrete and computationally finite and a perfect model of the world must be provided.

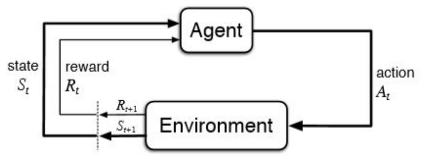


Figure 1: The agent-environment interaction in a Markov decision process.

2.3 Policy Iteration

For policy iteration, an initial policy is randomly intialized. Afterwards, a two step loop starts until an optimal policy is reached. This loop consists of

1. Policy Evaluation $\rightarrow$ Computing V^π for the current policy
2. Policy Improvement $\rightarrow$ Using knowledge gained from the policy's value function to improve the policy

Policy evaluation consists of its own iterative loop in which the value function for the policy is computed using the Bellman operator in which each loop step can be thought of as extending the time horizon starting from 0 towards H. With each step, we use the previous value function estimate to obtain a new value function which propagates the rewards throughout the states encountered.

$$V^\pi = \mathbb{E}_{s'}[r(s,a,s') + \gamma V(s')]$$

Once a value function for a policy is computed, policy improvement can then be undergone to create a new improved policy. This is done by analyzing the value function and changing actions for certain state if they lead to a higher expected discounted return.

$$\pi^{i+1} = argmax_a V^{\pi_i}(s)$$

This iterative loop ends when the policy no longer changes. This can be thought of as being due to convergence to the optimal value function.

$$\|V^{H+1} - V^H\| < threshold \tag{1}$$

$$V^* = V^{H+1} \tag{2}$$

$$\pi^* = \pi^{H+1} \tag{3}$$

2.4 Value Iteration

In contrast with policy iteration, value iteration never explicity deals with a policy. Instead the Bellman operator is applied to an initialized value function iteratively until the value function converges to the optimal value function. From this optimal value function, an optimal policy can be generated.

$$V_0^*(s) = 0 \; \forall s \in S \tag{4}$$

$$V_1^*(s) = max_a \mathbb{E}_{s'}[r(s,a,s') = max_a \sum_{s'} P_{sa}(s') r(s,a,s') \tag{5}$$

$$V_{H+1}^*(s) = max_a \sum_{s'} P_{sa}(s')[r(s,a,s') + \gamma V_H^*(s')] \tag{6}$$

$$stop\ when\ \|V^{H+1} - V^H\| < threshold \tag{7}$$

$$\pi^*(s) = argmax_a V^*(s) \tag{8}$$

3 Experimental Setup

3.1 Robot Description

For the experiment, a simple robot is is considered that will reside in the gridworld and attempt to reach a prescribed goal state. The robot will occupy one grid in the gridworld at any given time and can face any of the twelve headings identified by the hours on a clock $h \in \{0...11\}$. Each action is characterized by a movement value of 1, -1, or 0 which can be thought of as moving forward, backward, or staying still, respectively. This movement is then followed by a rotation of 1, -1, 0 which can be thought of as clockwise, counter-clockwise, and no rotation. Following this, it can be seen that the action of the robot can be described as a tuple

$$(a, b),$$

where a is the movement value and b is the rotation value. The following restrictions are applied to the robot's actions:

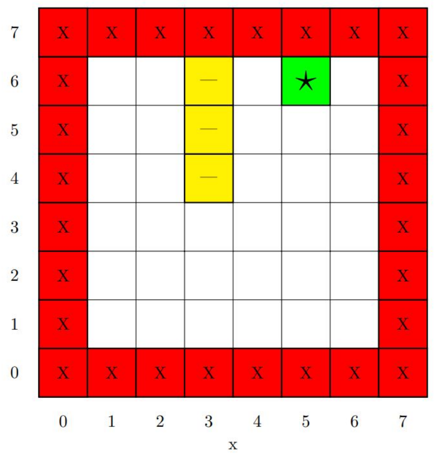


Figure 2: Gridworld with set Rewards

1. Attempting to move off the grid results in no linear movement, but the rotation portion will still happen.
2. Aside from restriction 1, the robot can only rotate if it also moves forwards or backwards.
3. If the robot chooses to move, a pre-rotation error may occur with probability p_e
   (a) With probability p_e each, robot will first rotate by +1 or -1 before it moves. No pre-rotate with probability $1 - 2 * p_e$.
   (b) Choosing to stay still will not incur an error rotation.

Then, it can be seen that the robot is defined by the following discrete action space.

$$(1, 0), (1, 1), (1, -1), (0, 0), (-1, 0), (-1, 1), (-1, 1) \ N_e = 7$$

The discrete state space is the set of all possible (x, y) positions and heading angles. Following this, the state space size can be computed as

$$N_s = X * Y * R = 8 * 8 * 12 = 768$$

3.2 Created Gridworld and Rewards

To allow for trajectory simulation, an 8x8 gridworld is constructed. Provided for us are the defined rewards for the states of the gridworld as shown in figure 2.

As the rewards for each state are independent of heading angle, they can be represented graphically on a 2D grid image. As shown above, the boundary states (red and marked X) have a reward of -100. The lane markers (yellow and marked -) have reward -10. Finally, the goal square (green and marked *) has reward +1. All other states have reward 0.

3.3 Benchmark

To be able to properly analyze the benefits provided by following an optimal policy, we first must set a benchmark. This benchmark will be the trajectory generated by a hand-engineered initial policy π_0. This policy crudely gets to the goal state by simply prioritizing closing the x distance and then the y distance without any regard for rewards *(code in utils.py)*. The plotted trajectory can be seen in figure 2.

3.4 Scenarios

With the following setup, we will explore four different scenarios:

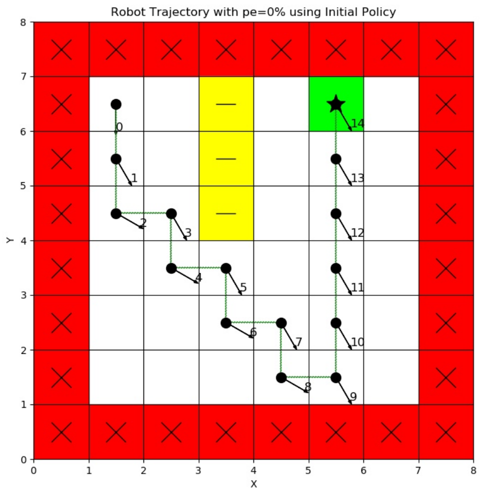
1. Reward at goal state located at (x=1, y=1) independent of heading, $p_e = 0\%$

2. Reward at goal state located at (x=1, y=1) independent of heading, $p_e = 25\%$
3. Reward at goal state +1 only when robot is pointing down $h = 6$, $p_e = 0\%$
4. Reward at goal state +1 only when robot is pointing down $h = 6$, $p_e = 25\%$

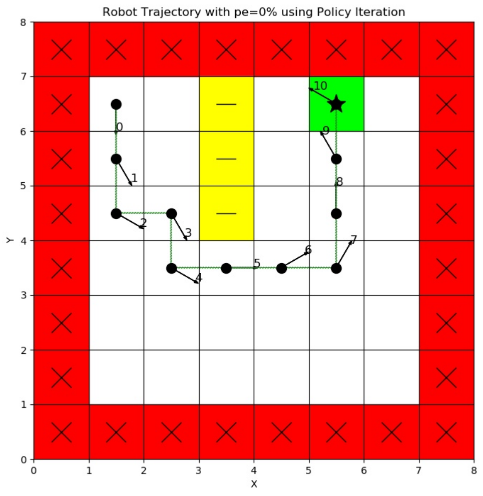
In addition to this, all four scenarios will start out at an initial state of $(x = 1, y = 6, h = 6)$. A discount factor of $\gamma = 0.9$ is also employed for all scenarios. Code has been generated that can graphically plot and generate trajectories according to the different parameters above. In the next section, we analyze the results of such trajectories and discuss the significance that arises from such results. We also discuss any notable computational differences between the two methods using Python's time library.

4 Experimental Results

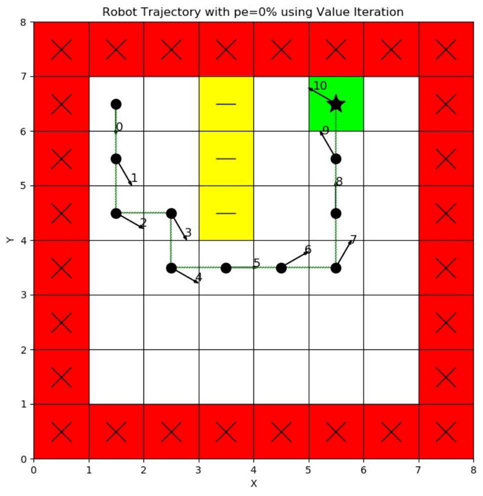
4.1 Scenario 1



(a) Initial Policy Trajectory



(b) Policy Iteration Trajectory



(c) Value Iteration Trajectory

Figure 3: Scenario 1 Trajectory Results

Generated trajectories can be seen above in figure 3 for when $p_e = 0\%$ and the goal state is independent of robot heading. With p_e equaling zero, this effectively eliminates any stochasticy in our environment and thus, the above trajectories will be identical if generated again since our actions deterministically get us to the state we desire. In subfigure a, it can be seen that the initial policy does fairly well, but takes an unnecessarily wide turn around the lane markers resulting in a total of +1 reward over 14 time steps to reach the goal. In contrast, both value iteration and policy iteration, result in a total of +1 reward while taking only 10 time steps to reach the goal by optimizing to take the tightest possible turn around the lane markers. In fact, the trajectories for both methods are actually identical. Although they are identical, it should be noted that each method may come up with different optimal policies as often there are multiple optimal policies for a single optimal value function as discussed earlier in section 2.2.2. To compare computational differences, the runtimes for both methods to convergence were measured over ten separate runs and averaged with the results being as follow.

$$PI_{time} = 6.201 \; seconds \;\; VI_{time} = 6.323 \; seconds$$

With a difference of only about a tenths of a second, there seems to be no noticeable difference in terms of computational speed between the two methods for scenario 1. As the state space is at a manageable size, we are most likely not to see noticeable differences until the state space rises in magnitude.

4.2 Scenario 2

For scenario 2, we not introduce stochasticity to the environment by setting the p_e to 25%. Due to this stochasticy, different runs will result in different trajectories. Therefore, we include two plots each for the benchmark initial policy, policy iteration, and value iteration as shown in figure 4 to have a more comprehensive representation of possible trajectories. The results of the six shown trajectories can be summarized as follow.

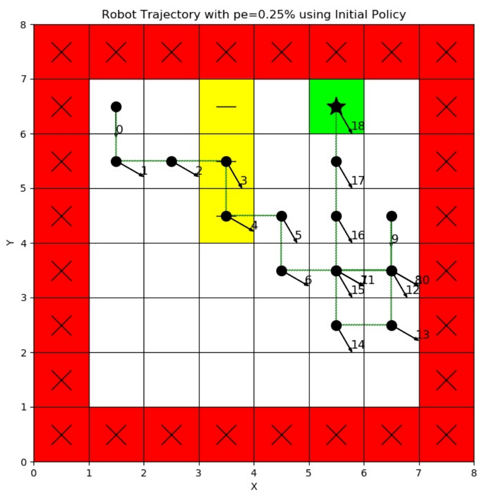
Trajectory	Total Reward	Number of Time Steps to Goal
IP Traj 1	-19	18
IP Traj 2	-9	6
PI Traj 1	+1	32
PI Traj 2	+1	14
VI Traj 1	+1	18
VI Traj 2	+1	18

Table 1: Scenario 2 Trajectory Results

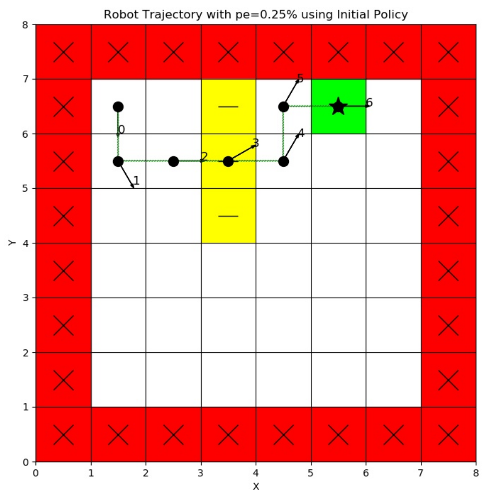
At first glance, one thing that all the trajectories have in common is the fact that they often have detours and must turn around or even end up in cyclical paths due to random chance rotations. Unique to the initial policy, is that it often runs into lane markers accruing negative reward during its episode. In contrast, both value iteration and policy iteration successfully avoid negative reward states even if it ends up increasing the time it takes to the goal, since we have not assigned a penalty for increasing number of time steps for this environment. Differences in trajectory between policy iteration and value iteration are arbitrary and are due to complete chance. The important takeaway is that regardless of which method is used, both will produce policies that maximizes the reward. Compared to scenario 1, we now start to see a noticeable difference in computational time as shown below.

$$PI_{time} = 10.492 \; seconds \;\; VI_{time} = 6.664 \; seconds$$

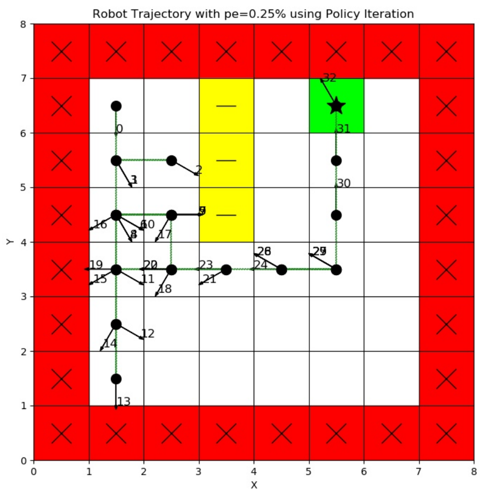
laksdfkljsdflkjsdjkf



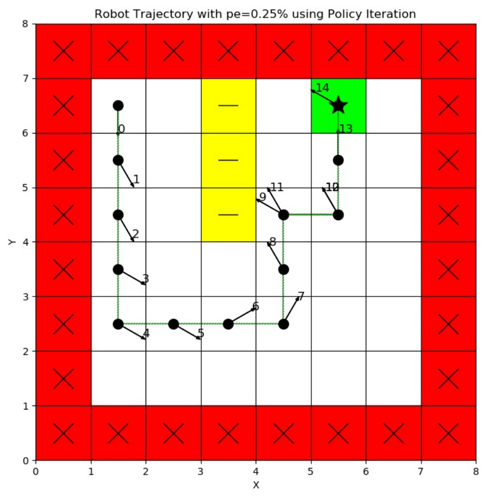
(a) Initial Policy Trajectory 1



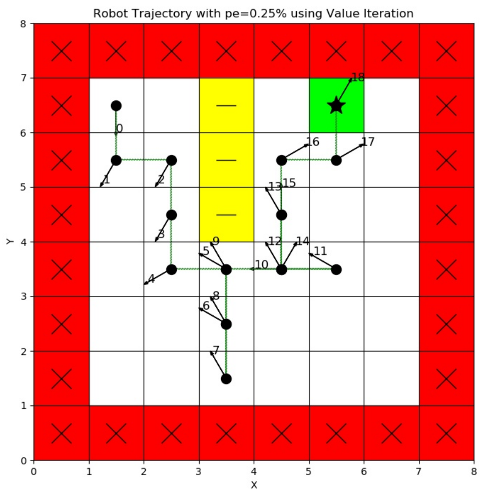
(b) Initial Policy Trajectory 2



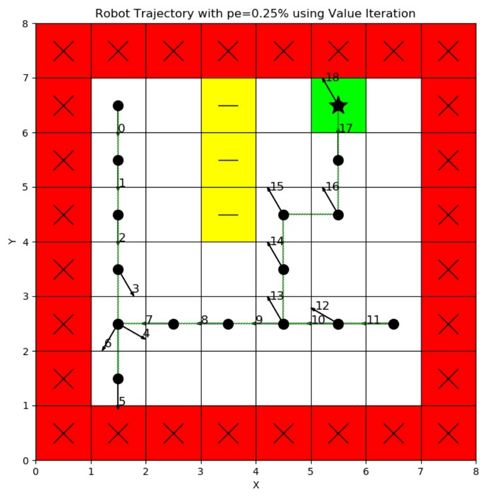
(c) Policy Iteration Trajectory 1



(d) Policy Iteration Trajectory 2



(e) Value Iteration Trajectory 1



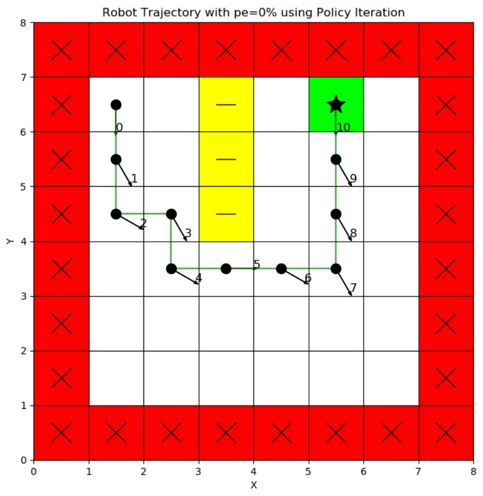
(f) Value Iteration Trajectory 2

Figure 4: Scenario 2 Trajectory Results

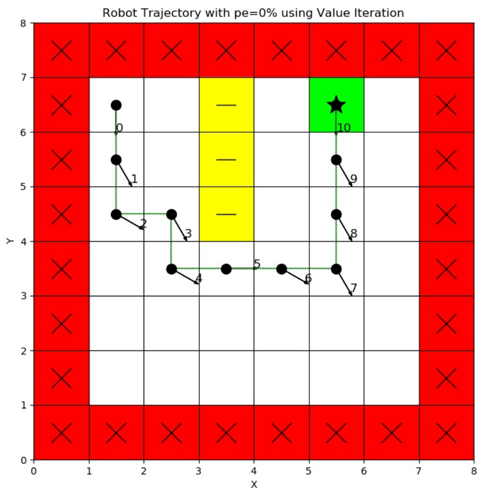
4.3 Scenario 3

For scenario 3, we return to a deterministic setting by setting p_e back to 0. For this scenario, rather than comparing to the initial policy's trajectory as the benchmark, we compare the trajectories produced by value and policy iteration to those produced in scenario 1. The trajectories are shown in figure 5. Comparing the two scenarios, the trajectories are exactly identical aside from the robot's heading. It can be seen that in scenario 1, since the goal state was independent of the robot's heading angle, the robot arbitrarily moved "forwards" while facing up. In contrast, due to the goal state requiring the robot to be in the $h = 6$ heading position, the robot takes advantage of its ability to move backwards to align itself in correct orientation without sacrificing time efficiency. Much like before, we store the computational times as the average of 10 runs.

$$PI_{time} = 10.048\ seconds\ \ VI_{time} = 7.016\ seconds$$



(a) Policy Iteration Trajectory



(b) Value Iteration Trajectory

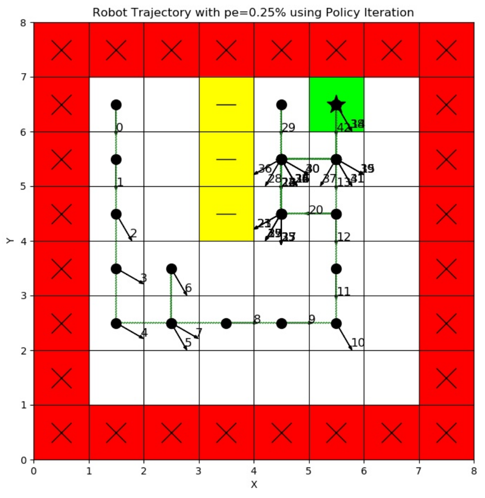
Figure 5: Scenario 3 Trajectory Results

4.4 Scenario 4

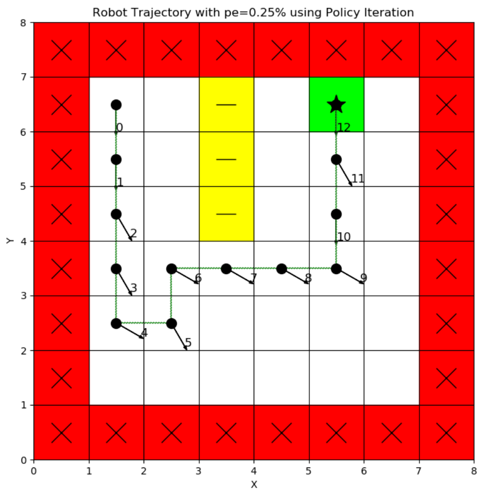
Finally, we analyze the results of scenario 4. Much like scenario 2, we show two trajectories each for value and policy iteration to get a better representation of trajectory possibilities. Furthermore, we will compare the results of both methods to those in scenario 2. Quite easily the most difficult scenario for the robot to solve, the results of scenario 4 show

Trajectory	Total Reward	Number of Time Steps to Goal
PI Traj 1	+1	42
PI Traj 2	+1	12
VI Traj 1	+1	26
VI Traj 2	+1	14

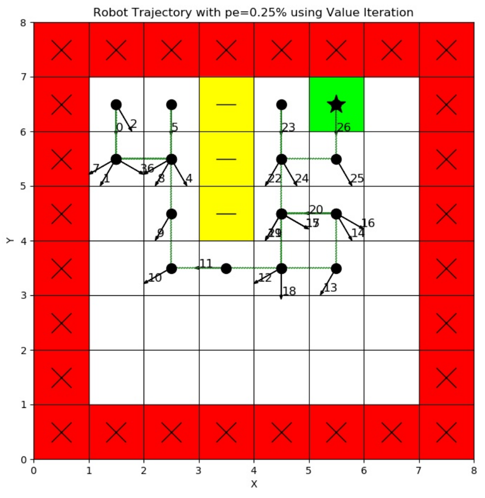
Table 2: Scenario 4 Trajectory Results



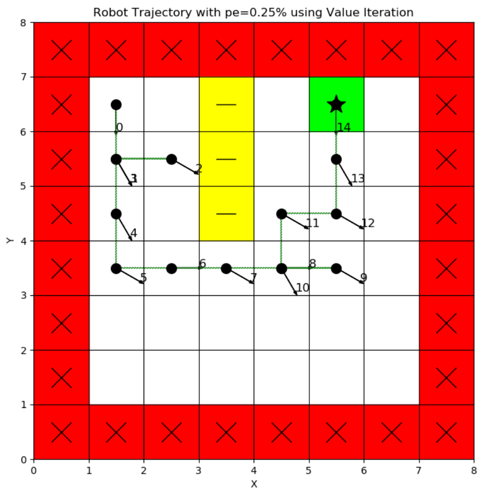
(a) Policy Iteration Trajectory 1



(b) Policy Iteration Trajectory 2



(c) Value Iteration Trajectory 1



(d) Value Iteration Trajectory 2

Figure 6: Scenario 4 Trajectory Results

5 Conclusion

6 Sources

Sutton, R. S. and A. G. Barto. 2017. Reinforcement Learning: An Introduction (2nd Edition, in progress). MIT Press.