Lab 3 Report

Use of of Markov Decision Process (MDP) to Control Simple Discretized Robot

Aaron Wilhelm, Andrew Wilhelm, Byeong Cheon Choi, Calvin Wu
Team Jayhawk
2/19/2019

Abstract:

In this paper, we demonstrate the utility of a Markov Decision Process (MDP) as a controller to guide our discretized robot in a 2D grid world. Our approach uses both policy and value iteration to determine an optimal policy, which allows us to compare the speed of each algorithm using the parameters below. We also demonstrate the effects of a probabilistic error on the trajectory of a robot using an optimal policy generated by the MDP. The results from this paper can be extended to more complex systems if some simplifying assumptions or approximations are made.

Introduction:

The Markov Decision Process (MDP) is a process to determine the optimal polich of a stochastic dynamic system in discrete time. The MDP relies on the Markov Assumption, which states that a system's entire state history can be captured by the system's current state. As such, the MDP is independent of all earlier states and actions and generates an action dependent upon only the current state. In this lab, we develop and implement a MDP to control a simulated robot in a 2D grid world to reach a target state with optimal reward.

By requiring a discrete state space, a discrete action space, and a discrete time domain, MDPs can only be directly applied to a few real-world systems that readily meet these demands. However, more complex systems can be simplified or approximated so that MDPs can then be applied. In addition, other control algorithms can be combined with MDPs. Thus, MDPs are useful tools to solve control problems for real-world applications.

Background:

By definition, a MDP can be described by S, A, P, R, γ and H, where S is a finite set of states, A is a finite set of actions, P is a state transition probability matrix, R is a real-valued reward function, γ is a discount factor, and H is the event horizon.

$$S = State$$

$$A = Action$$

$$P = Probabilities = \left\{ p\left(S,\ A,\ S' \right) \right\} = p_{SA}\left(S' \right)$$

$$R = Reward\ Function = \left\{ r\left(S,\ A,\ S' \right) \right\} = \{ r(s) \}$$

$$\gamma = Discount\ Factor,\ where\ 0 < \gamma < 1$$

$$H = Event\ Horizon$$

$$Total\ Discounted\ Reward = \sum_{i}^{H} \gamma^{i-1} r_i$$

$$\pi = Policy,\ \pi(S) = a$$

The goal of the MDP is to find an optimal policy π* that provides an optimal action for any state. A policy is a mapping from states to actions. The optimal policy π* maximizes expected rewards. It does so by considering the chances of going from one state to another, given an action and the probability of error.

$$Bellman\ Equation\ =\ V^*(S)\ =\ \max_a E\left[r + \gamma V^*\left(S'\right)\right]$$

We take two approaches in determining the optimal policy with the MDP: value iteration and policy iteration. In value iteration, the optimal policy is obtained by iteratively updating the value function using the Bellman Equation. The next value function is a function of the reward for each state and the value function in the next state. Iteration stops once the update to the value function is sufficiently small. With the optimal value function, the optimal policy can be found by selecting the action for each state that will maximize the value of the next state.

Value Iteration:

$$V^0(S) = 0,\ \ for\ all\ S$$

$$Assume,\ \left\|V^H - V^*\right\| = \max_a \sum_S \left|V^H(S) - V^*(S)\right| < \varepsilon$$

$$Loop\ until\ \left|\left|V^H - V^{H+1}\right|\right| < \frac{\varepsilon(1-\gamma)}{\gamma}:$$

$$V^{H+1}(S) = \max_a \sum_{S'} p\left(S,\ A,\ S'\right) \bullet \left(r\left(S,\ A,\ S'\right) + \gamma \bullet V^H\left(S'\right)\right.$$

End Loop

The second approach, policy iteration, determines the optimal policy by once again recursively updating the value function through the Bellman Equation. However, after updating the value function, the optimal policy for the new value function is generated. Iteration continues until the previous and current optimal policies are identical. When this occurs, the policy obtained is the optimal policy for the system.

Policy Iteration:

$$\pi^0(S) = a,\ \ for\ all\ S$$

Choose an arbitrary policy $\rightarrow \pi'$

Loop until $\pi = \pi'$:

Compute the value function of policy π *by using the Bellman Equation*

$$V^{\pi}(S) = \max_{a} \sum_{S'} p\left(S, \pi(S), S'\right) \bullet \left(r\left(S, \pi(S), S'\right) + \gamma \bullet V^{\pi}\left(S'\right)\right)$$

Improve the policy at each state

$$\pi'(S) = arg \sum_{S'} p\left(S, A, S'\right) \bullet \left(r\left(S, A, S'\right) + \gamma \cdot V^{\pi}\left(S'\right)\right)$$

End Loop

Methods:

Our implementation to test the MDP uses a 2D grid world with length and width of 6 units. The simulated robot has a XY location and a rotation as its state. The XY location is a coordinate pair that indicates the cell of a 6 by 6 grid. The robot can rotate into one of twelve possible headings (akin to the numerical positions of a clock) $h \in \{0...11\}$ where h = 0, 3, 6, 9 is the compass north, east, south, and west respectively. Therefore, the size of the state space N_S is W * L * H = 6 * 6 * 12 = 432.

The simulated robot has the following possible actions: stay stationary (no change in state), move forwards or move backwards. If it chooses to move, then after moving, it can also choose to rotate left, right, or not at all. Therefore, the size of the action space N_A is 7. If the robot is attempting to move in the direction of an edge (i.e. attempting to move out of the possible defined states), then the robot's XY position will remain the same while its heading may change. For example, if the simulated robot is in state (5,5,6) or (5,5,9) and the robot is given the command to move forward and turn, the robot will turn without moving. There is also an error probability pe that the robot will turn without explicit instruction to do so. This error only occurs if the robot moves forward or backwards (i.e. does not remain still) and affects the rotation prior to the robot moving.

The rewards of the decision process within the simulated environment are independent of the action taken and are therefore only dependent upon the current state of the simulated robot. Generally, the decision process uses four different types of rewards. The border of the environment (i.e. the states that are located along the boundary) have a reward of -100. Lane markers, which are used to guide the direction of the robot, have a reward of -10. The goal has a reward of +1 and all other states have a reward of 0.

When using policy iteration to determine the optimal policy, we use an initial policy where the action that will move the robot closer to the goal is selected. For example, if the goal is directly in front of the simulated robot, the robot will move forward in that direction. When using value iteration, we assume an initial value function V(s) = 0 for all states.

All images and code for this lab can be found in the Lab 3 folder of our online repository[1] .

Results:

Below are figures showing the policies and trajectories of the robot for each of the scenarios outlined in the lab. The red, yellow, and green cells are the border states, lane markers, and goal, respectively. A dot means an action of no movement should be taken. An arrow indicates that the robot should move either forwards or backwards. A black, magenta, and teal arrow specify no rotation, clockwise rotation, and counter-clockwise rotation, respectively when the robot moves. For figures that show the policy at a given heading, the value of the optimal path is shown. For figures that show a sample trajectory of the robot, the robot's current heading is specified in the bottom corner of each cell it traverses.

Figure 1 shows the initial policy actions for the initial policy π_0. Figure 2 shows the trajectory using this initial policy with discount factor $\lambda = 0.9$, $p_e = 0$ and initial state x = 1, y = 4, and h = 6. The value of this trajectory is -0.656.

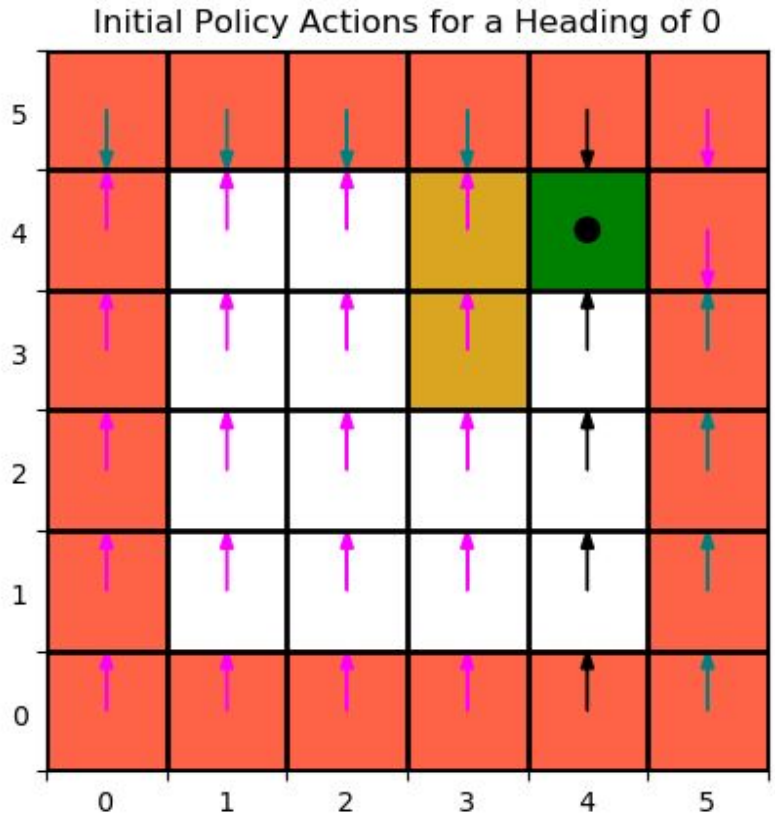

Figure 1: Initial policy π_0 for the robot at heading zero. Note that there are no values for each square as this policy was generated independent of a value function.

[1] https://git.uclalemur.com/TeamJayhawk2019/ECE183DA.git

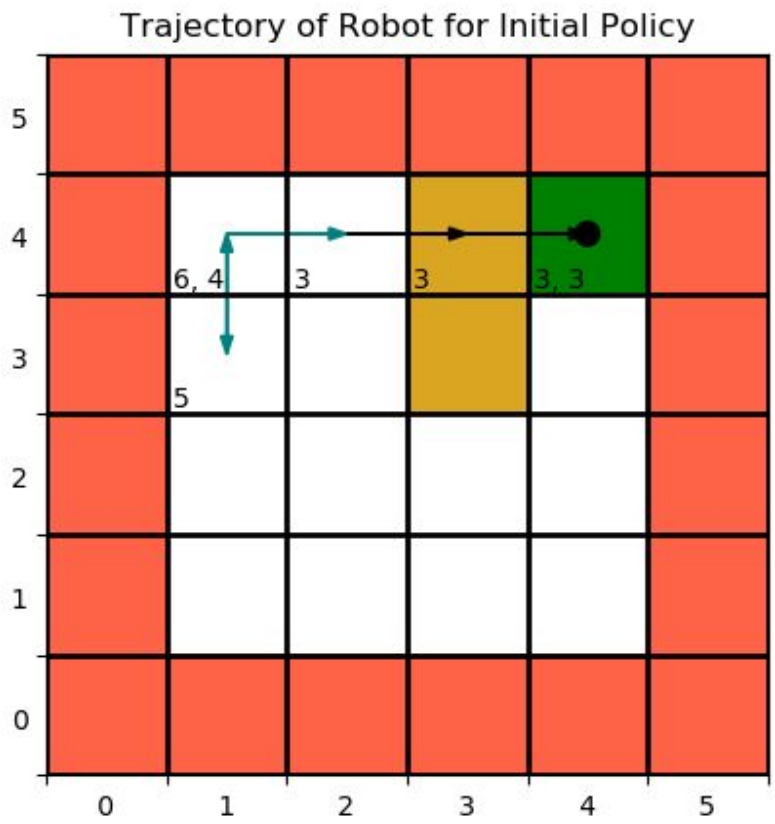

Figure 2: Trajectory using the initial policy π_0 and discount factor $\lambda = 0.9$ for a robot with initial state x = 1, y = 4, and h = 6. The value of this trajectory is -0.656. Note that since the robot started with a heading of 6 and it had to go forward and backward to turn to the proper heading before proceeding to the goal.

Figure 3 shows the optimal policy after policy iteration. This also uses the same initial policy with discount factor $\lambda = 0.9$, turn error $p_e = 0$, and initial state x = 1, y = 4, and h = 6 as the previous trajectory graph. Figure 4 shows the trajectory for this optimal policy which has a value of 4.783. It took 219 seconds for the optimal policy to be generated using policy iteration.

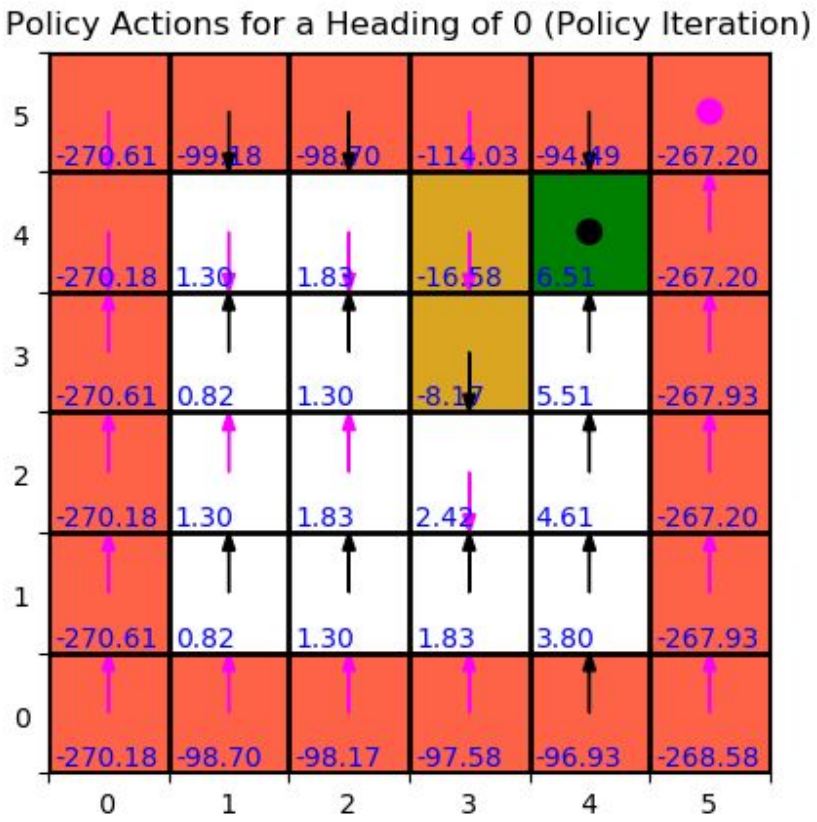

Figure 3: Optimal policy π* for the robot at heading zero. The numbers within each cell is the value function for that state.

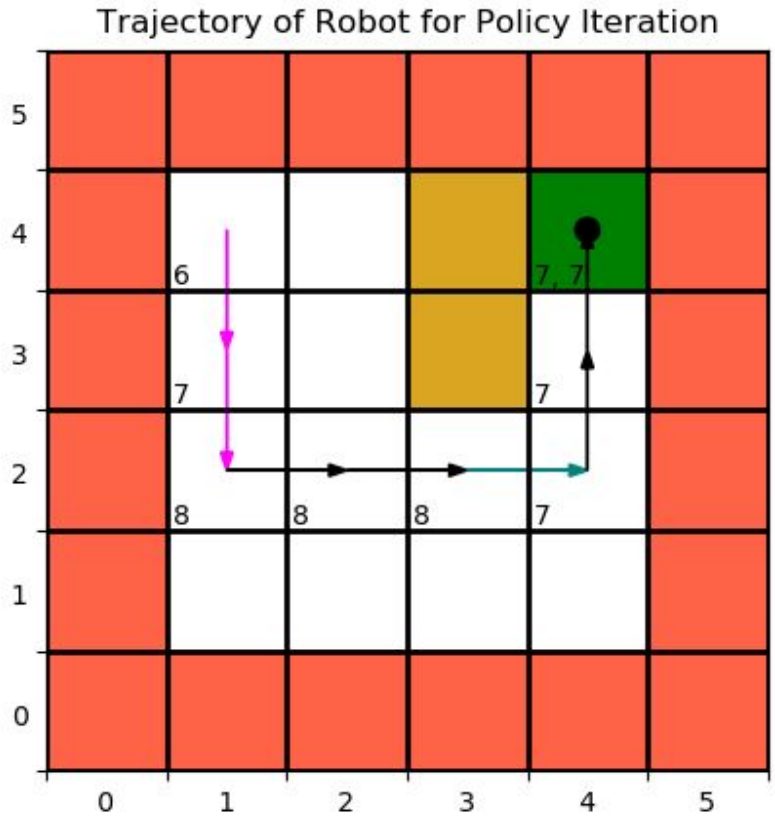

Figure 4: Trajectory using the optimal policy π* and discount factor λ = 0.9 for a robot with turn error probability p_e = 0 and initial state x = 1, y = 4, and h = 6. The value of the trajectory is 4.783.

Figure 5 shows the optimal policy after value iteration. This uses a discount factor λ = 0.9, turn error p_e = 0, and initial state x = 1, y = 4, and h = 6. Figure 6 shows the trajectory for this optimal policy which has a value of 4.783. It took 382 seconds for the optimal policy to be generated using value iteration.

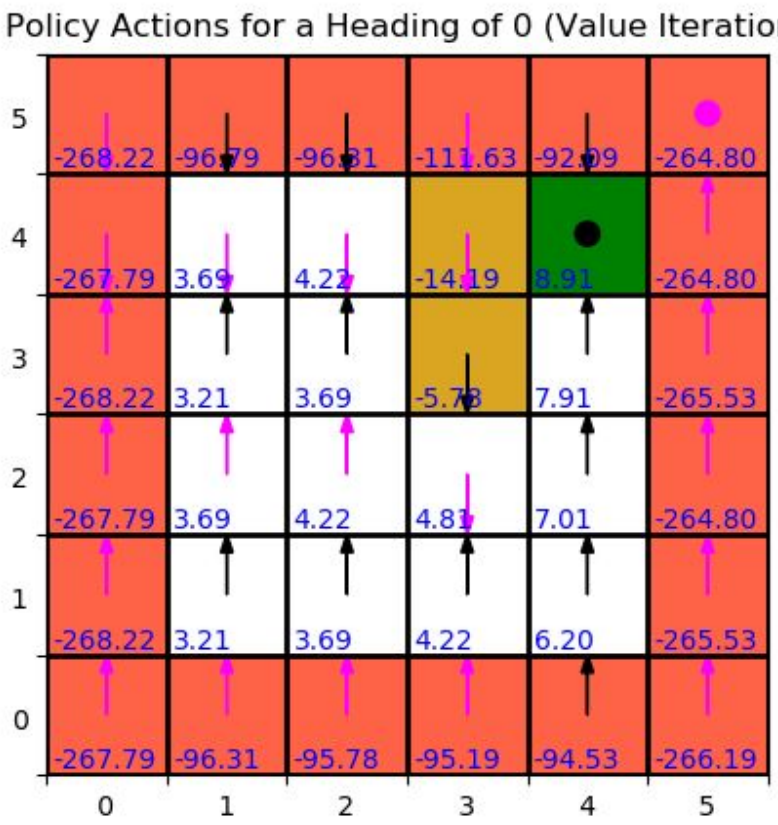

Figure 5: Optimal policy π* for the robot at heading zero. The numbers within each cell is the value function for that state.

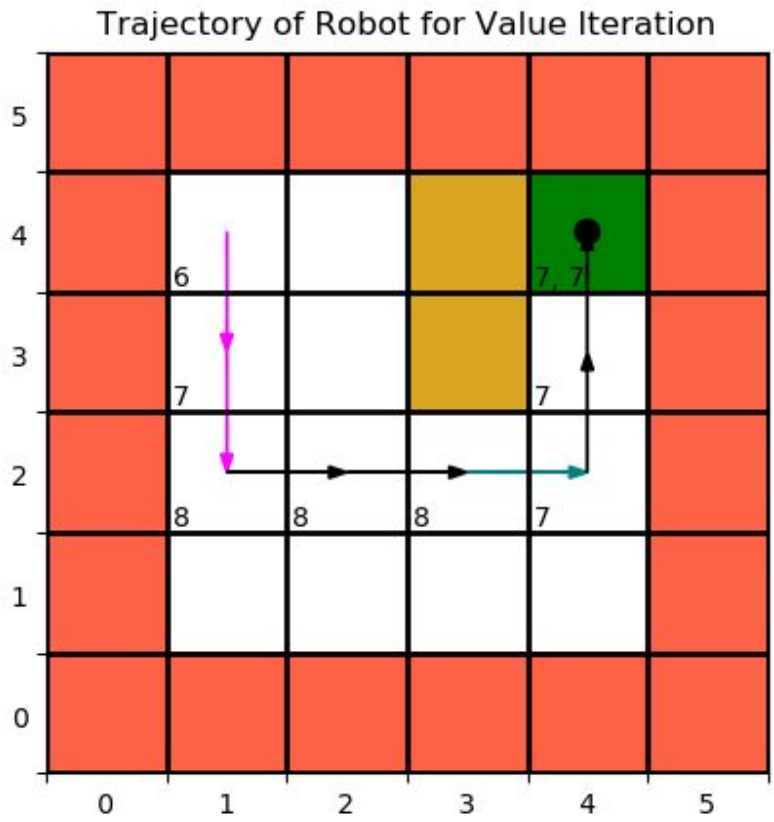

Figure 6: Trajectory using the optimal policy π* and discount factor λ = 0.9 for a robot with turn error probability p_e = 0 and initial state x = 1, y = 4, and h = 6. The value of the trajectory is 4.783.

Figure 7 shows the optimal policy after policy iteration for a robot with turn error p_e = 0.1. This also uses the same initial policy with discount factor λ = 0.9 and initial state x = 1, y = 4, and h = 6 as the previous trajectory graph. Figure 8 shows the trajectory for this optimal policy which has a value of 3.138. It took 425 seconds for the optimal policy to be generated using policy iteration.

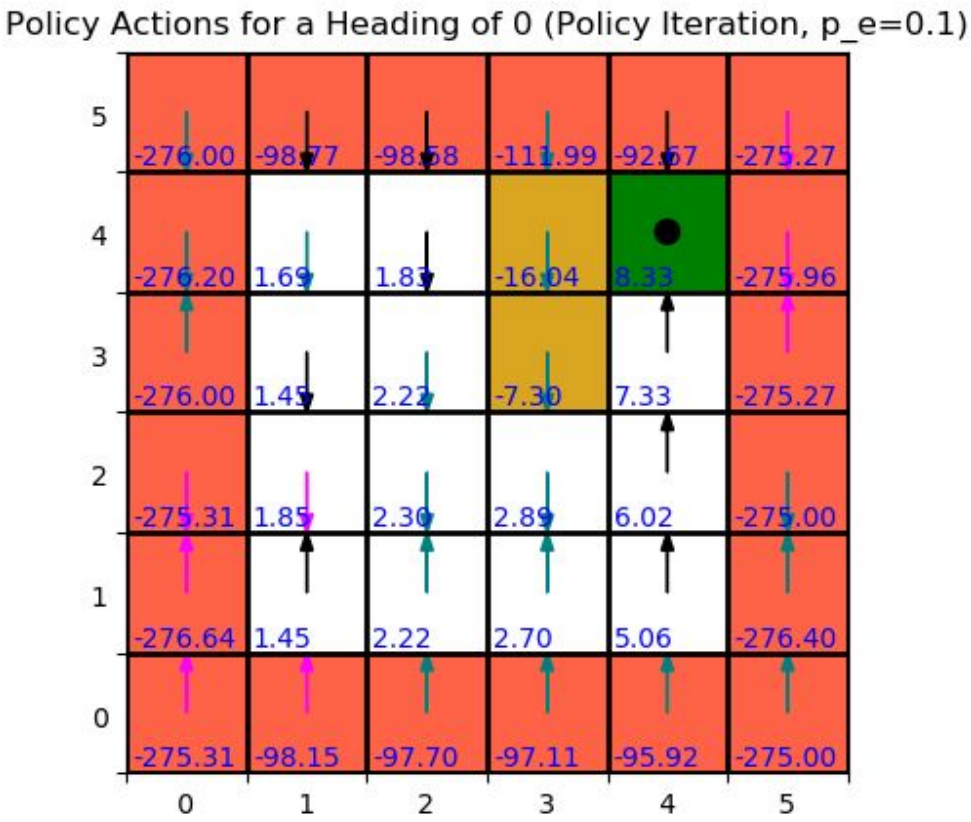

Figure 7: Optimal policy π* for the robot at heading zero. The numbers within each cell is the value function for that state.

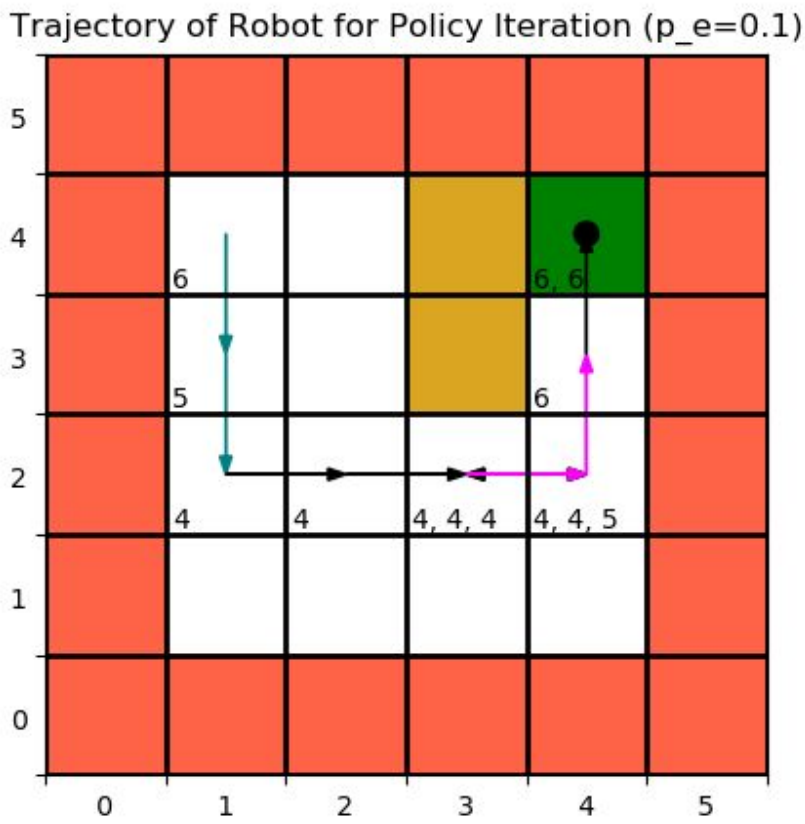

Figure 8: Trajectory using the optimal policy π* and discount factor λ = 0.9 for a robot with turn error probability p_e = 0 and initial state x = 1, y = 4, and h = 6. The value of the trajectory is 3.138. Note that due to a turning error, the robot had to readjust its heading during the trajectory.

Figure 9 shows the optimal policy after value iteration for a robot with turn error p_e = 0.1. This uses a discount factor λ = 0.9 and initial state x = 1, y = 4, and h = 6. Figure 10 shows the trajectory for this optimal policy which has a value of 4.783. It took 575 seconds for the optimal policy to be generated using value iteration.

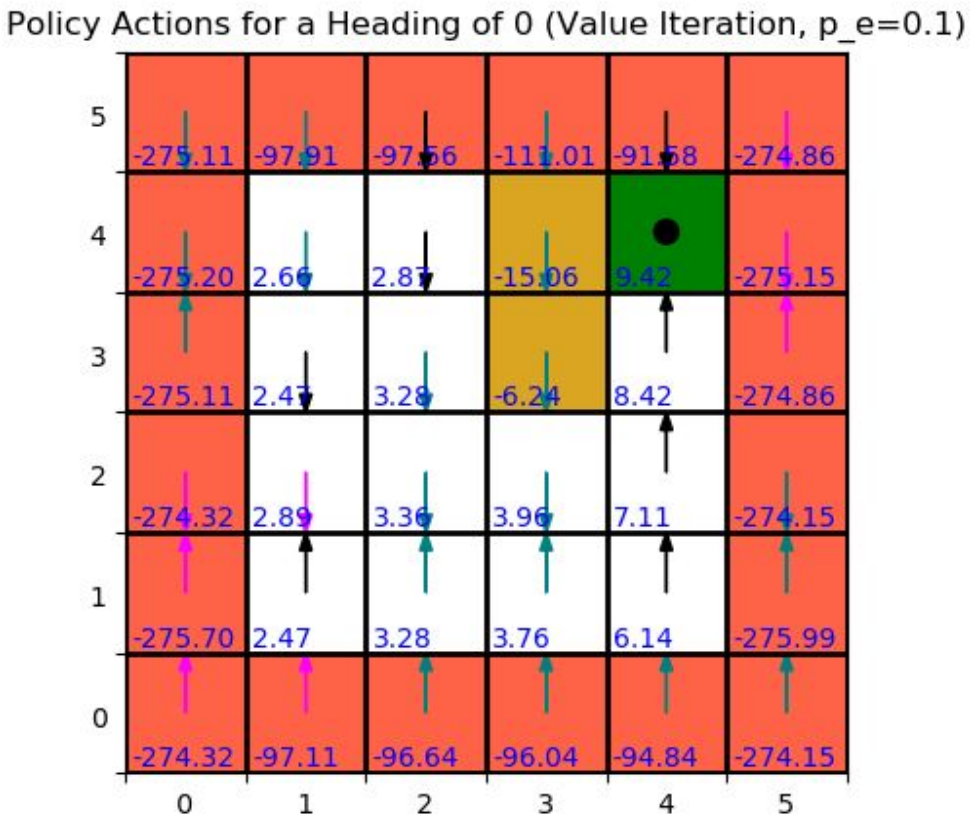

Figure 9: Optimal policy π* for the robot at heading zero. The numbers within each cell is the value function for that state.

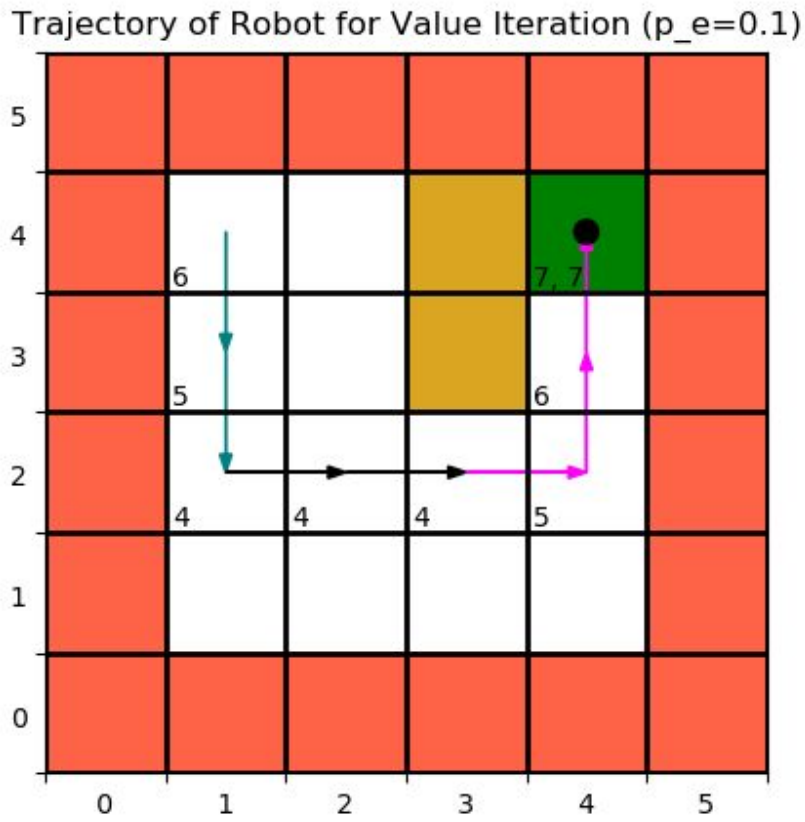

Figure 10: Trajectory using the optimal policy π* and discount factor λ = 0.9 for a robot with turn error probability p_e = 0 and initial state x = 1, y = 4, and h = 6. The value of the trajectory is 4.783.

Figure 11 and Figure 12 show the optimal policy after policy iteration for a robot with turn error p_e = 0 and p_e = 0.25 that only receives an award if it is at the goal and has a heading of 6 (i.e. goal state is (4,4,6)). This also uses the same initial policy with discount factor λ = 0.9 and initial state x = 1, y = 4, and h = 6 as the previous trajectory graph. It took 205 and 464 seconds, respectively, for the optimal policy to be generated using policy iteration. Figure 13 and Figure 14 show the trajectory for these optimal policies which have values of 4.783 and 0, respectively.

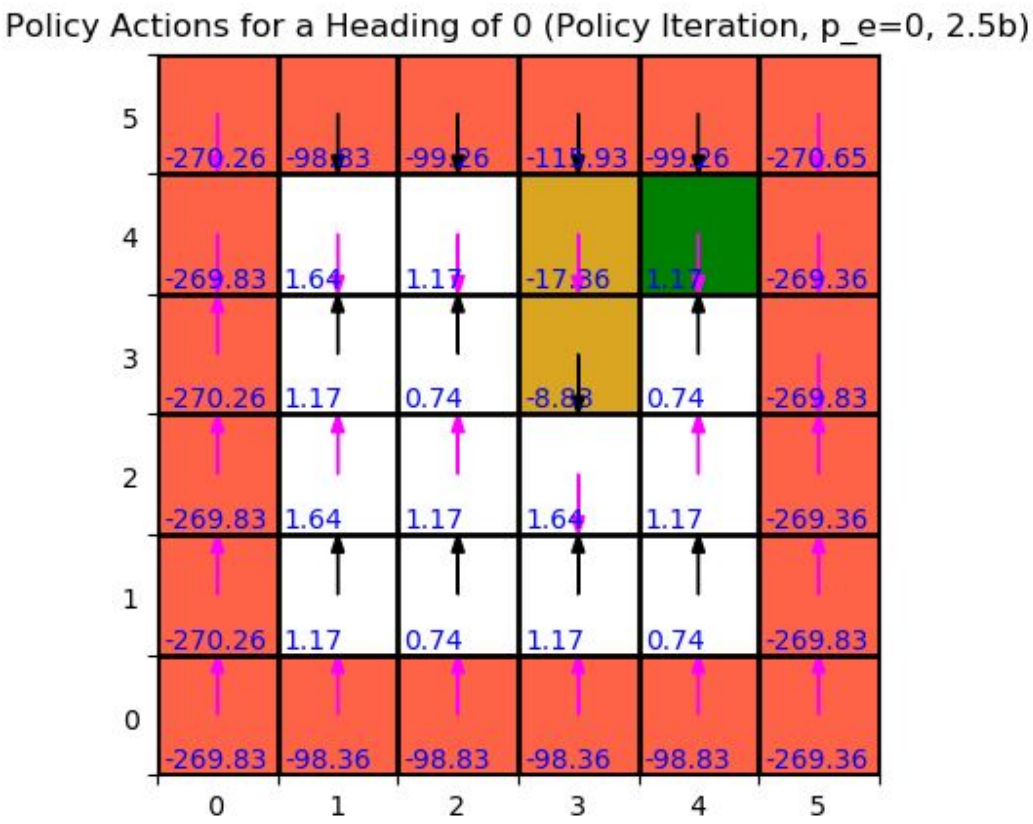

Figure 11: Optimal policy π* for the robot at heading zero with $p_e = 0$. The numbers within each cell is the value function for that state.

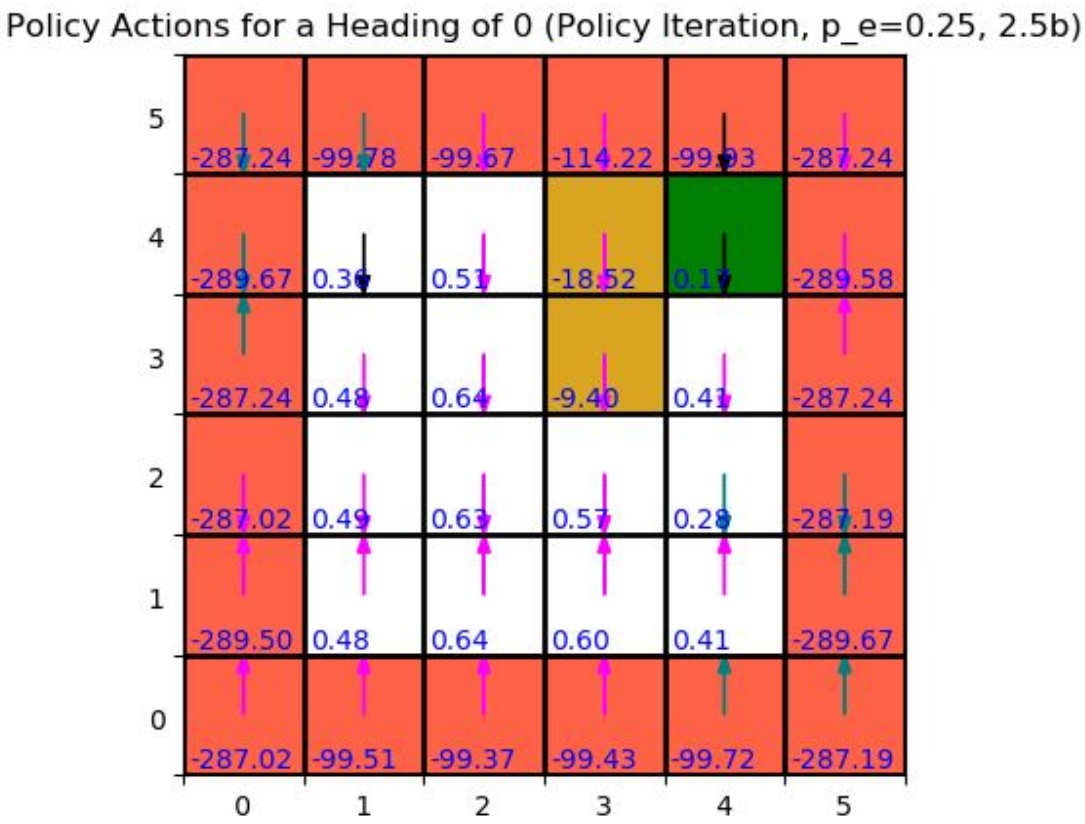

Figure 12: Optimal policy π* for the robot at heading zero with p_e = 0.25. The numbers within each cell is the value function for that state.

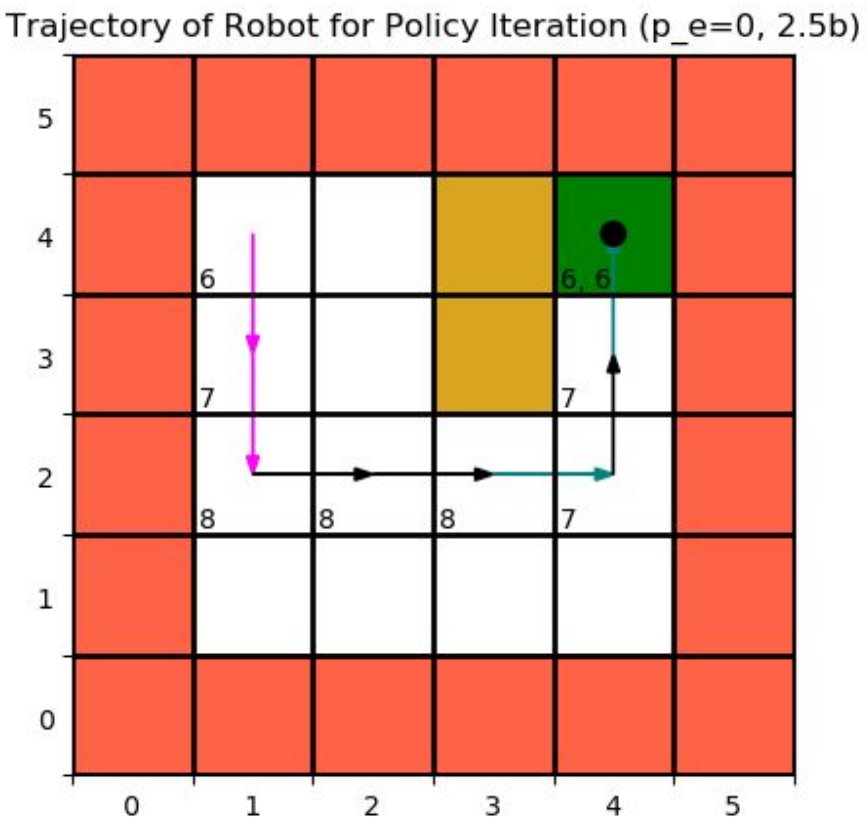

Figure 13: Trajectory using the optimal policy π* and discount factor λ = 0.9 for a robot with turn error probability p_e = 0 and initial state x = 1, y = 4, and h = 6. The value of the trajectory is 4.783.

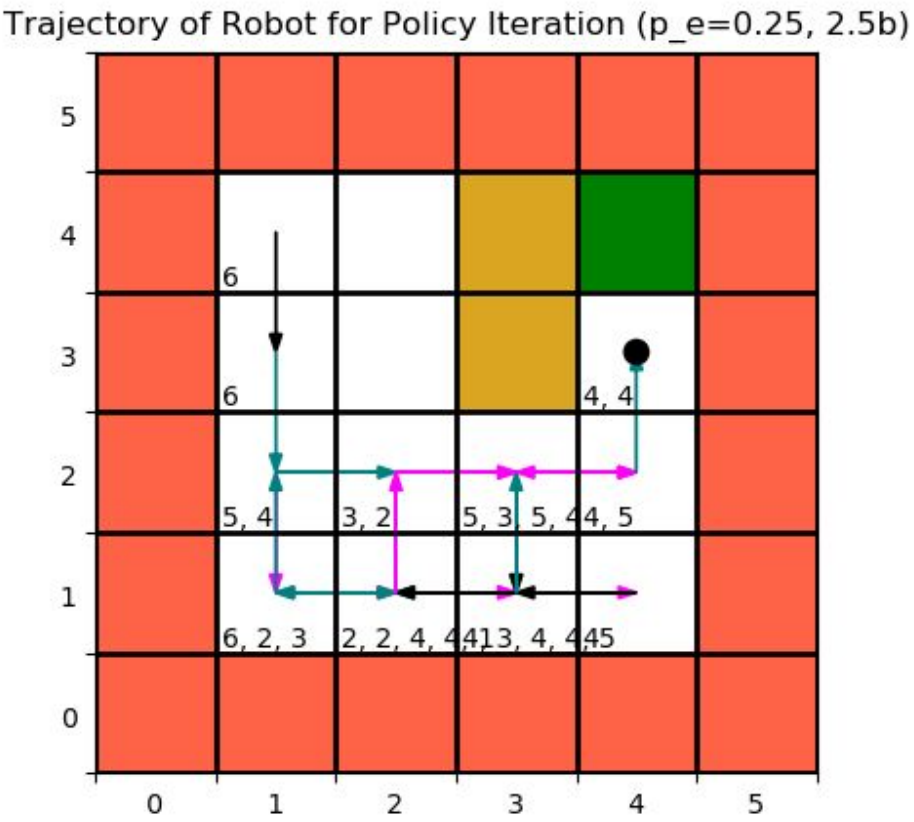

Figure 14: Trajectory using the optimal policy π* and discount factor λ = 0.9 for a robot with turn error probability p_e = 0.25 and initial state x = 1, y = 4, and h = 6. The value of the trajectory is 0.The reason that the robot decides to stop at (4,3,4) is that in order to move, it would have to move horizontally. This would cause a high penalty, so the robot decides to instead remain where it is at.

Figure 15 and Figure 16 show the optimal policy after value iteration for a robot with turn error p_e = 0 and p_e = 0.25 that only receives an award if it is at the goal and has a heading of 6 (i.e. goal state is (4,4,6)). This also uses the same initial policy with discount factor λ = 0.9 and initial state x = 1, y = 4, and h = 6 as the previous trajectory graphs. It took 496 and 652 seconds, respectively, for the optimal policy to be generated using value iteration. Figure 17 and Figure 18 show the trajectory for these optimal policies which have values of 4.783 and 0, respectively.

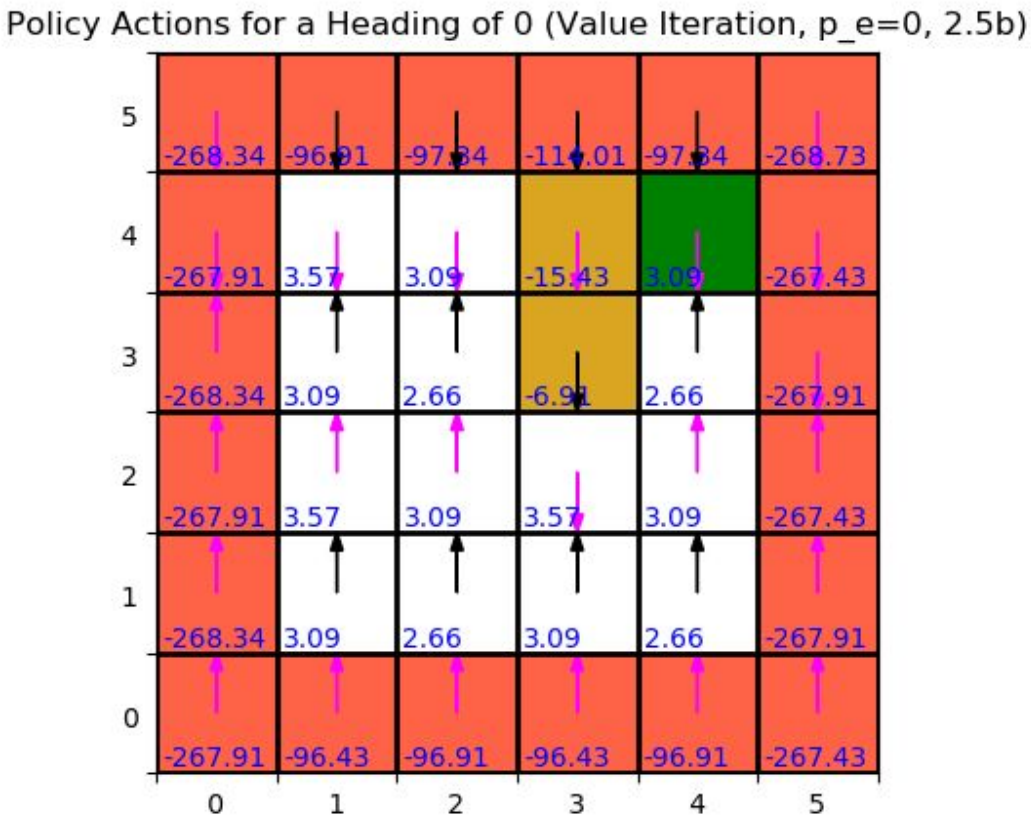

Figure 15: Optimal policy π* for the robot at heading zero with $p_e = 0$. The numbers within each cell is the value function for that state.

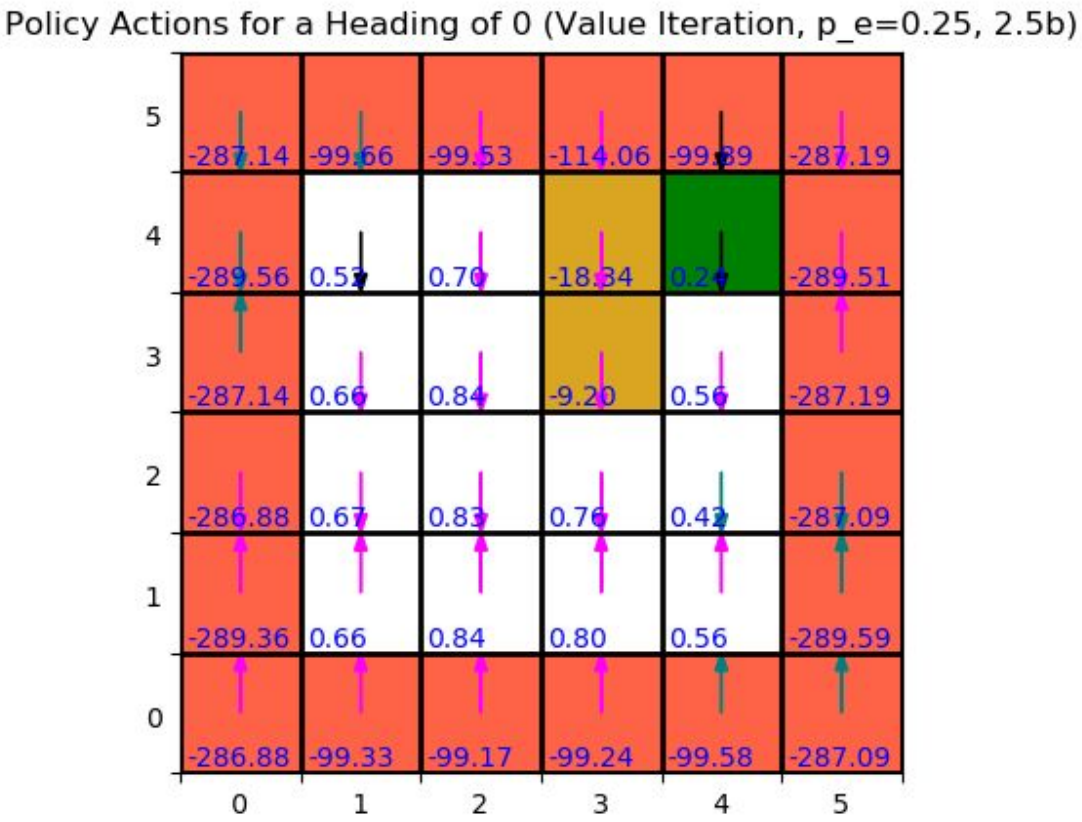

Figure 16: Optimal policy π* for the robot at heading zero with p_e = 0.25. The numbers within each cell is the value function for that state.

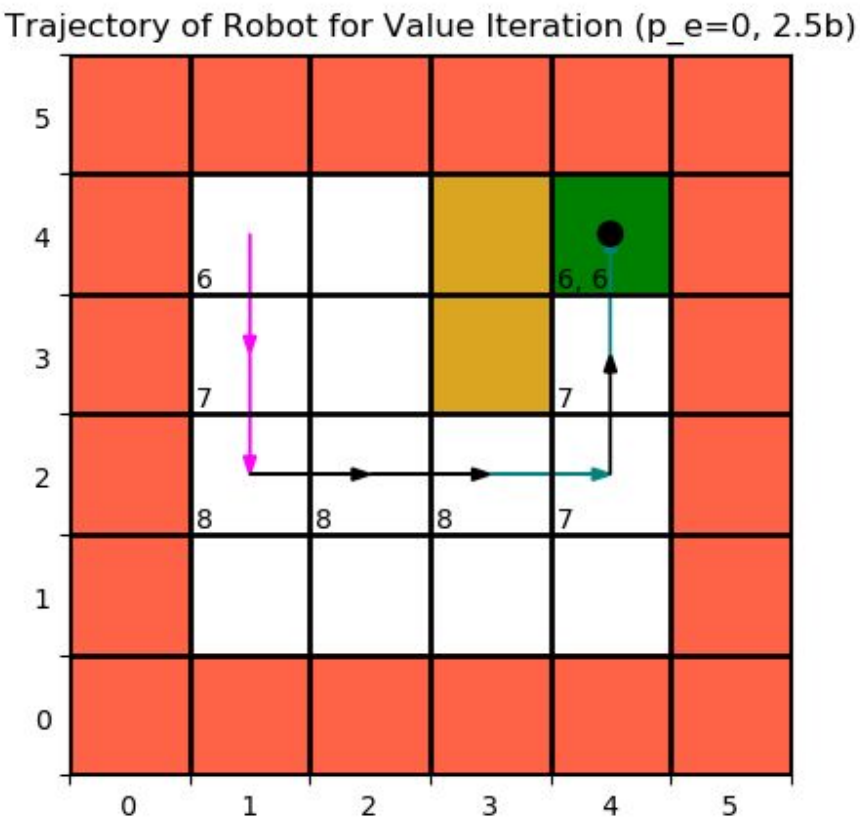

Figure 17: Trajectory using the optimal policy π* and discount factor λ = 0.9 for a robot with turn error probability p_e = 0 and initial state x = 1, y = 4, and h = 6. The value of the trajectory is 4.783.

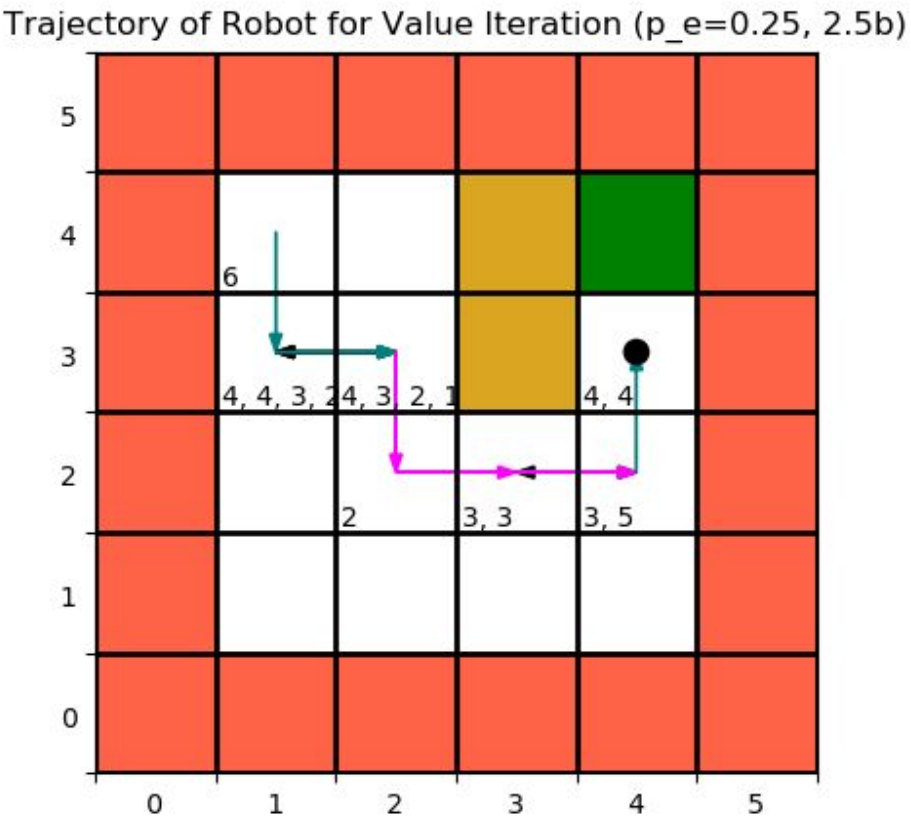

Figure 18: Trajectory using the optimal policy π* and discount factor λ = 0.9 for a robot with turn error probability p_e = 0.25 and initial state x = 1, y = 4, and h = 6. The value of the trajectory is 0. The reason that the robot decides to stop at (4,3,4) is that in order to move, it would have to move horizontally. This would cause a high penalty, so the robot decides to instead remain where it is at.

As would be expected, introducing a turn error complicated the path that the robot could take. In the case of the policy iteration with turn error of 0.25, the robot even became “stuck”, as it happened to rotate in such a way that it would not be able to reach its goal without receiving a major penalty from the boundary states.

Conclusion:

We have demonstrated a MDP with a discretized robot in a simulated environment. Our approach used both value iteration and policy iteration to determine the optimal policy. We then demonstrated that, under the parameters of our problem, policy iteration is quicker at determining an optimal policy. If we were to relax the error term for the value iteration, we could get the optimal value function (and hence optimal policy) quicker. However, we would also increase the chance of choosing a policy that is not the true optimum policy.

As expected, both policy and value iteration returned the same optimal policy. This makes sense as the MDP states that there is only one optimal policy for a given scenario.

For scenarios where we increased the probability of the turn error, we observed that for certain situations the robot could recover and continue to the goal. However, for others, the robot had to make a compromise and choose to stay at a state that was not its intended goal. This problem could be fixed by increasing the reward of the final goal and/or decreasing the penalty of the lane markers or borders.

For the scenarios where we specified the final heading of the robot for the reward, we saw little change in the trajectory of the robot. As the robot had enough space to rotate to the desired angle, its trajectory was minimally affected by this change in the reward parameters. The policy, however, was affected more as now each of the actions specified for it to move to the correct heading to get the reward.

Our work, although done entirely in discrete space, can be generalized to continuous systems if the continuous systems are discretized. This broadens the applicability of our work to other complex systems that can be found in the real world.

References:

1. https://www.cs.rice.edu/~vardi/dag01/givan1.pdf
2. https://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-825-techniques-in-artificial-intelligence-sma-5504-fall-2002/lecture-notes/Lecture20FinalPart1.pdf
3. http://www0.cs.ucl.ac.uk/staff/D.Silver/web/Teaching_files/MDP.pdf
4. https://ocw.mit.edu/courses/aeronautics-and-astronautics/16-410-principles-of-autonomy-and-decision-making-fall-2010/lecture-notes/MIT16_410F10_lec23.pdf
5. https://stackoverflow.com/questions/8194156/how-to-subtract-two-lists-in-python
6. https://stackoverflow.com/questions/5998245/get-current-time-in-milliseconds-in-python
7. https://stackoverflow.com/questions/43971138/python-plotting-colored-grid-based-on-values
8. https://stackoverflow.com/questions/43971138/python-plotting-colored-grid-based-on-values#
9. https://stackoverflow.com/questions/15971768/drawing-arrow-in-x-y-coordinate-in-python
10. https://matplotlib.org/api/pyplot_api.html#matplotlib.pyplot.arrow
11. https://docs.scipy.org/doc/numpy/reference/generated/numpy.flipud.html#numpy.flipud
12. https://docs.python.org/3/library/random.html
13. https://matplotlib.org/api/_as_gen/matplotlib.axes.Axes.arrow.html
14. https://stats.stackexchange.com/questions/76169/convergence-of-value-iteration