EE183DA
Team Buffalo
Lab 3: Markov Decision Processes (MDPs)

The University of California - Los Angeles

Iou-Sheng (Danny) Chang · UID: 804-743-003
William Argus · UID: 004-610-455
XianXing (Gray) Jiang · UID: 604-958-018
Ho (Bobby) Dong · UID: 604-954-176

February 19th 2019

Contents

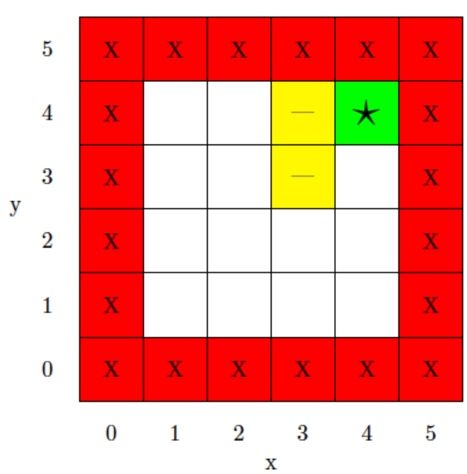
1 Introduction and Lab Overview

Markov decision process (MDP) is a model widely used for machine learning. MDP allows the robot to determine the optimal action to take for the given state it is currently in. In order to achieve this, the general MDP model has to be given four elements: state s, action a, transition function $T(s, a, s')$, and reward $R(s, a, s')$. The transition function and the reward are dependent on the current state, the action being taken, and the next state. The solution of the MDP model is to obtain the optimal policy that prescribes the action the robot should take at a given state, s. This is achieved by finding a policy that prescribes the actions that maximizes the cumulative reward in the following equation:

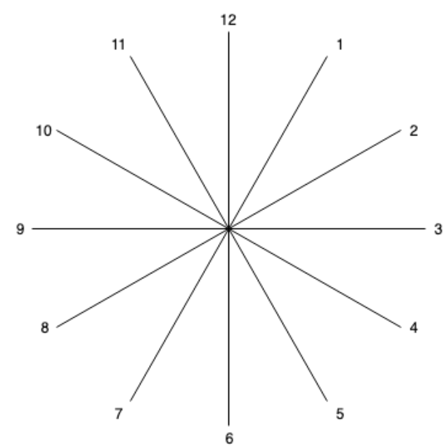
$$\sum_{t=0}^{\infty} \gamma^t R_{a_t}(s_t, s_{t+1}) \tag{1}$$

Where $a_t = pi(s_t)$, i.e. actions given by policy. Note that γ is the discount factor.

The state is the positional configuration plus the heading of the robot. In an MDP, all possible states are known, and the robot will always exist in exactly one of these states at any given time. For this lab, the robot runs in a grid world with finite cells, and its current state is represented by its x and y coordinates (each ranging from 1 to 6 for the 6 by 6 grid, shown in fig. 1a), and its heading (ranging from 1 to 12, laid out like the numbers on a clock, shown in fig. 1b).



(a) Grid World given in Lab 3 instruction



(b) 12 possible headings of the robot

The actions are a set of movements that the MPD model can dictate the robot to take in order to transfer it from the current state to the next state. For this lab, the robot is only allowed to take 7 actions: Forward, and turn left, no turn, or turn right (current heading -1, 0, and +1, respectively), backward, and turn left, no turn, or turn right (current heading -1, 0, and +1, respectively), and no forward or backward movement and no rotation.

The transition matrix describes the transition from state s to the state s' by taking action a. For an ideal system, the next state, s', is always the result of being at the current state and taking the given action. For a system that contains noise, the next state is determined by being at the state, s, and taking the given action with a probabilistic model. For this lab, the transition function follows a probabilistic model with error called "pre-rotation". "Pre-rotation" occurs before the robot performs the linear motion (forward/backward) part of a prescribed action. It will change the heading of the robot, resulting in the robot taking action while in a different state than the current state input. This "pre-rotation" occurs with the probability pe for both pre-rotating right and left; the probability of the robot not "pre-rotating" and taking the given action at the inputted state is $1 - 2pe$.

The reward describes the reward that the robot will get when it moves into state, s'. The reward in the MDP model serves as an indicator to the robot of which state is preferred and which states are to be avoided. For this lab, the reward is set to specific values, defined only by the state, indicated from the lab manual (shown in fig. 1a), with white being 0, red being -100, yellow being -10, and green being $+1$, the goal square.

2 MDP system

2.1 State space

The state space is $N_S \in \mathbb{R}^{6\times6\times\ 12}$. It consists of all possible states and is represented by a 6 by 6 by 12 three dimensional matrix. The first two dimensions give the x and y values, each with possibilities 1 through 6. The third dimension gives the heading of the state. The headings are numbered 1 through 12 and are arranged the same way as numbers on a clock. For example, moving forward with right rotation while in state x = 1, y = 1, and heading = 12, results in moving to state x = 1, y = 2, and heading = 1.

The state space N_S is implemented in MATLAB as *stateSpace* in the function *init.m* (listing 1).

2.2 Action space

The action space is $N_A \in \mathbb{R}^{7\times2}$. It consists of 7 actions. The first three are move forward, with left, no, right rotation respectively. The next three are move backward, with left, no, right rotation respectively. The last action is no movement and no rotation.

The state space N_S is implemented in MATLAB as *action* in the function *init.m* (listing 1).

2.3 Probability function

$p_{sa}(s')$ is the function that returns the probability, given p_e, s, a, s'. It first determines if s' is a possible next state given the current state (s), the action (a), and error probability (p_e). There will only be at most 3 possible next states for s'. This is due to the fact that the action dictate whether the robot will move forward or backward and its rotation, so the only thing that is not determined is the pre-rotation, which has 3 possibilities; left with probability p_e, right with probability p_e, or no rotation with probability $1-2p_e$. Accordingly, the function calculates the probability that each of these pre-rotations happen, meaning that the probability of all possible states is known given the current state, action, and p_e. The probability of the requested state s' can then be returned by the function, as required.

The probability function is implemented in MATLAB as *probability.m* (listing 2).

2.4 Next state function

The function that returns the next state s' given the current state s, action a, and error probability p_e uses the probability function defined in *probability.m* (listing 2). Since the probability of each of the possible next states is known, the MATLAB code generates a random number over a sufficient large range of values, (the size of which is given by the variable *pe_m*, calculated by 10 to the power of the number of decimal places in p_e ($10^{pe_{power}}$). This way it is assured that *pe_m* is a sufficiently large number. We then assign the first $pe \cdot 10^{pe_{power}}$ values to signify the state given in the event of a pre-rotation to right, the next $pe \cdot 10^{pe_{power}}$ values to signify the state given in the event of a pre-rotation to the left, and the remaining $10^{pe_{power}} - 2 \cdot pe \cdot 10^{pe_{power}}$ values to signify the state given in the event of no pre-rotation. Using this method, it can be guaranteed that the right, left, and no pre-rotations happen for each movement of the robot with the probability given by any p_e values.

The next state function is implemented in MATLAB as *next_state.m* (listing 3).

2.5 Matlab Implementation

Listing 1: MATLAB code for Initialization function

```
function [stateSpace,pe,lambda,s,action] = init()
% Initialization of value
value_int= zeros(6);

% Expend the value matrix to 12 headings' 3D matrix space
[rows,cols] = size(value_int);
for x = 1:rows
    for y = 1:cols
        for h = 1:12
            stateSpace(x,y,h) = value_int(x,y);
        end
    end
end

% Initialization of error percentage (pe)
pe = 0.1;

% Initialization of discount factor
lambda = 0.9;

% Initialization of initial state
s = [2;5;6];

% Initialization of actions set
% column 1: forward 1 / no movement 0 / backward -1
% column 2: leftturn -1 / no rotation 0 / rightturn 1
action = [1  -1;...
          1  0;...
          1  1;...
          -1 -1;...
          -1 0;...
          -1 1;...
          0 0];
end
```

Listing 2: MATLAB code for Probability function

```
function [prob,state_p] = probability(pe,state_c,action,state_n)
    % ===== Heading ===== %
    % circular loop to account for the flaw in mod 12
    heading = [12 ,1 ,2, 3, 4, 5, 6, 7, 8, 9, 10, 11 ,12, 1];

    % ===== Possible States ===== %
    % state_p is a R^{2}
    % cause there are three possible prerotation outcome
    state_p = [state_c  state_c  state_c];
    if (pe ~= 0 && action(1) ~=0)
        % prerotation -1 (left)
        state_p(3,1) = heading(state_p(3,1));
        % prerotation 0 (no prerotation)
        state_p(3,2) = heading(state_p(3,2)+1);
        % prerotation 1 (right)
        state_p(3,3) = heading(state_p(3,3)+2);
    end

    % ===== Update Possible Next State ===== %
    % State depedning on heading
    for i =1:3
        % (x,y)
        % If statements account for when robot is at the border
        % when @ the border, robot can still rotate but cannot
            perform movement
        switch state_p(3,i)
            case {11,12,1}
                state_p(2,i) = state_p(2,i) + action(1);
                if (state_p(2,i) == 7 || state_p(2,i) == 0)
                    state_p(2,i) = state_c(2);
                end
            case {2,3,4}
                state_p(1,i) = state_p(1,i) + action(1);
                if (state_p(1,i) == 7 || state_p(1,i) == 0)
                    state_p(1,i) = state_c(1);
                end
            case {5,6,7}
                state_p(2,i) = state_p(2,i) - action(1);
                if (state_p(2,i) == 0 || state_p(2,i) == 7)
                    state_p(2,i) = state_c(2);
                end
            case {8,9,10}
                state_p(1,i) = state_p(1,i) - action(1);
                if (state_p(1,i) == 0 || state_p(1,i) == 7)
                    state_p(1,i) = state_c(1);
                end
```

```
        end
        % Heading
        state_p(3,i) = heading((state_p(3,i)+1) + action(2));

    end
    % Probability
    % Attach the probability corresponding to each possible states
    % For prerotated (-1/1) states: pe
    % For non-prerotated (0) states: 1-2*pe
    state_p(4,:) = [pe, 1-2*pe, pe];

    % ===== Finding probability for the selected state ===== %
    prob = 0;
    for i = 1:3
        % Check if the possible next state == next state
        % If so save the probability attached to that state
        if state_p(1:3,i) == state_n
            prob = state_p(4,i);
        break;
    end
end
```

Listing 3: Matlab code for Next State function

```
function [state_n,prerotation] = next_state(pe,state_c,action)
    % ===== pe && prerotation ===== %
    if (pe == 0)
        prerotation = 0;
    else
        % initialization of pe array
        pe_power = 0;
        pe_m = [];
        % calculate array size needed to account for pe's # of digits after decimal point
        while (floor(pe*10^pe_power) ~= pe*10^pe_power)
           pe_power = pe_power + 1;
        end
        % create an array to store movements based on pe
        for i = 1:pe*power(10,pe_power)
            pe_m = [pe_m 1 -1];
        end
        pe_m = [pe_m zeros(1,power(10,pe_power) - 2*pe*power(10,pe_power))];
        % prerotation based on pe (defined in init.m)
        prerotation = pe_m(randi(power(10,pe_power)));
    end

    % ===== Initialize next state ===== %
```

```
    state_n = state_c;

    % ===== Update Next State ===== %
    % circular loop to account for the flaw in mod 12
    heading = [12 ,1 ,2, 3, 4, 5, 6, 7, 8, 9, 10, 11 ,12, 1];
    % Heading (prerotation)
    % account for pe = 0 (no error percentage), will not incur an error rotation
    % account for choosing to stay still, will not incur an error rotation
    if (pe ~= 0 && action(1) ~=0)
        state_n(3) = heading((state_n(3)+1) + prerotation);
    end
    % (x,y)
    % If statements account for when robot is at the border
    % when @ the border, robot can still rotate but cannot perform movement
    switch state_n(3)
        case {11,12,1}
            state_n(2) = state_c(2) + action(1);
            if (state_n(2) == 7 || state_n(2) == 0)
                state_n(2) = state_c(2);
            end
        case {2,3,4}
            state_n(1) = state_c(1) + action(1);
            if (state_n(1) == 7 || state_n(1) == 0)
                state_n(1) = state_c(1);
            end
        case {5,6,7}
            state_n(2) = state_c(2) - action(1);
            if (state_n(2) == 0 || state_n(2) == 7)
                state_n(2) = state_c(2);
            end
        case {8,9,10}
            state_n(1) = state_c(1) - action(1);
            if (state_n(1) == 0 || state_n(1) == 7)
                state_n(1) = state_c(1);
            end
    end
    % Heading
    state_n(3) = heading((state_n(3)+1) + action(2));
end
```

3 Planning Problem

3.1 Reward function

For the reward function $R(s)$, a matrix was created that contained the value of the reward at that state in the appropriate state. For example, when cell (5,5) was referenced, it gave the reward of the goal state (5,5) of +1. When a state s was inputted, the function simply returned the value of the cell of matrix given by the x and y values in s. This simple implementation was possible because reward in a state is independent of heading, so the heading in state s could simply be ignored, with only the x and y values taken into account. In order to check if the reward function returns correct reward for any given state, we used boolean statement to test for all cases in Matlab *main.m*.

The next state function is implemented in Matlab as *reward.m* (listing 4).

3.2 Matlab Implementation

Listing 4: Matlab code for Reward function

```
function reward_disp = reward(state_c)
    % ===== Grid World Reward ===== %
    % REFER to the grid world shown in the lab instruction
    gw_reward = [-100 -100 -100 -100 -100 -100;...
                 -100 0    0    -10  1    -100;...
                 -100 0    0    -10  0     -100;...
                 -100 0    0    0    0     -100;...
                 -100 0    0    0    0     -100;...
                 -100 -100 -100 -100 -100 -100];
    [rows,cols] = size(gw_reward);

    % ===== Map the reward value ===== %
    reward_disp = gw_reward((rows+1) - state_c(2),state_c(1));
end
```

4 Policy Iteration

4.1 Overview

Policy iteration changes the policy of the robot's actions after each movement and then recompute the value of the state. After repeated iterations, the policy will converge to a single policy, and this policy is guaranteed to be the optimal policy. Usually, the optimal policy is converged upon before the optimal value function is converged upon.

The optimal policy is found using the Bellman equation eq. (3) to perform policy iteration, which improves π until it converges on the optimal policy function. The equations for optimal value and optimal policy are eq. (4) and eq. (5), respectively.

$$V^{\pi}(s) = E\Big[\sum_{i=1}^{T} \gamma^{i-1} r_i\Big] \ \forall \ s \in \mathbb{S} \tag{2}$$

$$V^{*}(S) = \max_{a} \ \Big[R(s,a) + \gamma \sum_{s' \in \mathbb{S}} p(s'|s,a) V^{*}(s')\Big] \tag{3}$$

$$V^{*}(s) = \max_{\pi} \ V^{\pi}(s) \ \forall \ s \in \mathbb{S} \tag{4}$$

$$\pi^{*} = arg\max_{\pi} \ V^{\pi}(s) \ \forall \ s \in \mathbb{S} \tag{5}$$

The optimal Q-function (eq. (6)) will give the total expected reward gained by the robot given that the robot starts in state s, takes action a, and then continues to take the best action for the rest of its movement.

$$V^{*}(s) = \max_{a} \ Q^{*}(s,a) \ \forall s \in \mathbb{S} \tag{6}$$

4.2 Initial policy matrix

To create a matrix that stored the action $a = \pi_0(s)$, the action given by the initial policy, a function was written to assign the appropriate action to each state. This function took into account position (x,y) and heading and assigned the action prescribed the initial policy of moving forward or backward, whichever took the robot closest to the goal square, or, if the goal square was to the right of left of the robot, moving forward and turning toward the goal square.

The action $a = \pi_0(s)$ is implemented in MATLAB as *init_policy.m* (listing 5).

4.3 Robot trajectory plotting function

The function that plotted the trajectory given policy matrix π, initial state s_0 and probability of error p_e. We first implemented the Grid World shown in Lab 3 instructions (fig. 1a) into MATLAB figure, then live update robot trajectory onto that existing figure, with the heading indicated. The action and prerotation for next state are also shown in the Grid World for the ease to verify and debug the code, a sample figure when the robot reaches the goal square is shown in (fig. 2).

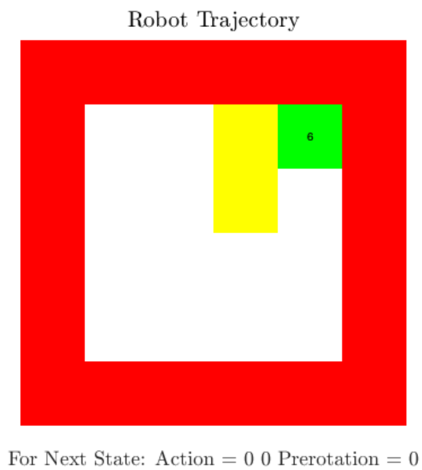


Figure 2: Grid World implemented on MATLAB and robot trajectory plot

The robot trajectory plotting function is implemented in MATLAB as *plot_route.m* (listing 6).

4.4 Trajectory plot using policy π_0

The plot of the trajectory using policy π_0 from initial state $x = 2, y = 5, h = 6$, assuming $p_e = 0$ is plotted using (listing 5) and (listing 6).

The trajectory plotted is shown in video in the Appendix and Demonstration section.

4.5 Policy evaluation computation

This function returns a matrix of value $v = V^{\pi}(s)$ when indexed by state s.

The function is implemented in MATLAB as *get_value.m* (listing 7).

4.6 Value of the trajectory in section 4.3

The value matrix is shown in *val* in MATLAB *policy_init.m* (listing 8).

4.7 One-step lookahead optimal policy π

This function returns a matrix π given a one-step lookahead on value

The function is implemented in Matlab as *get_policy.m* (listing 9).

4.8 Policy Iteration

Combining the *get_value.m* and *get_policy.m* functions, the policy iteration *policy.m* code returns optimal policy π^* and optimal value V.

Pseudo Code for Policy Iteration

Start with an arbitrary policy π_0
Rerun
  π *corresponds to* π_0
  Compute values using π by solving the following equation:
    $V^{\pi}(S) = E[r|s,\pi(s)] + \gamma \sum_{s' \in S} P(s'|s,\pi(s))V^{\pi}(s')$
  At each state, improve policy
    $\pi'(s) = argmax_a(E[r|s,a] + \gamma \sum_{s' \in S} P(s'|s,a)V^{\pi}(s'))$
Continue until $\pi = \pi'$

Figure 3: Pseudo code for Policy Iteration

The policy iteration function is implemented in Matlab as *policy.m* (listing 10).

4.9 Optimal policy trajectory plot

The plot of the trajectory under optimal policy π^* and $p_e = 0$.

The trajectory plotted is shown in video in the Appendix and Demonstration section.

4.10 Computation time

We use the Matlab built in $tic - toc$ function to determine the compute time of Policy Iteration, as shown in *main.m* (listing 11), the compute time is then printed to the commend window as shown in fig. 4, it is determined to be 4.6858 $[s]$.

```
========== 2.3(a) ==========
REFER to 'init_policy.m' for detail
========== 2.3(b) ==========
REFER to 'plot_route.m' for detail
========== 2.3(c) ==========
REFER to 'policy_init.m' for detail
REFER to figure plotted or video linked in the lab report reference
========== 2.3(d) ==========
REFER to 'get_value.m' for detail
========== 2.3(e) ==========
REFER to 'policy_init.m' for detail
========== 2.3(f) ==========
REFER to 'get_policy.m' for detail
========== 2.3(g) ==========
REFER to 'policy.m' for detail
========== 2.3(h) ==========
REFER to figure plotted or video linked in the lab report reference
========== 2.3(i) ==========
Policy Iteration Compute Time = 4.6858 [s]>>
```

Figure 4: Compute time of Policy Iteration

4.11 Matlab Implementation

Listing 5: MATLAB code for $a = \pi_0(s)$

```
function [policy] = init_policy(rows)
    % ===== Initialization of initial policy \pi_0 ===== %
    % REFER to lab 3 instructions for the initialization process
        % If the goal is in front of you, move forward
        % If it is behind you, move backward
        % Then turn the amount that aligns your next direction of travel closer towards the goal (if necessary).
        % If the goal is directly to your left or right, move forward then turn appropriately.
    for x = 1:6
        for y = 1:6
            % Iterate through all headings
            for h = 1:12
                % Case 1: (x,y) = (<5,<5)
                if (x < 5 && (rows+1) - y < 5)
                    switch h
                        case {11,12,1}
                            policy(y,x,h,1) = 1;
                            policy(y,x,h,2) = 1;
                        case {2,3,4}
                            policy(y,x,h,1) = 1;
                            policy(y,x,h,2) = -1;
                        case {5,6,7}
                            policy(y,x,h,1) = -1;
                            policy(y,x,h,2) = -1;
                        case {8,9,10}
                            policy(y,x,h,1) = -1;
                            policy(y,x,h,2) = 1;
                    end
                end
```

```
            % Case 2: (x,y) = (<5,5)
            if (x < 5 && (rows+1) - y == 5)
                switch h
                    case {11,12,1}
                        policy(y,x,h,1) = 1;
                        policy(y,x,h,2) = 1;
                    case {2,3,4}
                        policy(y,x,h,1) = 1;
                        policy(y,x,h,2) = 0;
                    case {5,6,7}
                        policy(y,x,h,1) = 1;
                        policy(y,x,h,2) = -1;
                    case {8,9,10}
                        policy(y,x,h,1) = -1;
                        policy(y,x,h,2) = 0;
                end
            end
            % Case 3: (x,y) = (<5,6)
            if (x < 5 && (rows+1) - y == 6)
                switch h
                    case {11,12,1}
                        policy(y,x,h,1) = -1;
                        policy(y,x,h,2) = 1;
                    case {2,3,4}
                        policy(y,x,h,1) = 1;
                        policy(y,x,h,2) = 1;
                    case {5,6,7}
                        policy(y,x,h,1) = 1;
                        policy(y,x,h,2) = -1;
                    case {8,9,10}
                        policy(y,x,h,1) = -1;
                        policy(y,x,h,2) = -1;
                end
            end
            % Case 4: (x,y) = (5,<5)
            if (x == 5 && (rows+1) - y < 5)
                switch h
                    case {11,12,1}
                        policy(y,x,h,1) = 1;
                        policy(y,x,h,2) = 0;
                    case {2,3,4}
                        policy(y,x,h,1) = 1;
                        policy(y,x,h,2) = -1;
                    case {5,6,7}
                        policy(y,x,h,1) = -1;
                        policy(y,x,h,2) = 0;
                    case {8,9,10}
```

```
            policy(y,x,h,1) = 1;
            policy(y,x,h,2) = 1;
    end
end
% Case 5: (x,y) = (5,5) REACHED GOAL SQUARE
if (x == 5 && (rows+1) - y == 5)
   policy(y,x,h,1) = 0;
   policy(y,x,h,2) = 0;
end
% Case 6: (x,y) = (5,6)
if (x == 5 && (rows+1) - y == 6)
    switch h
        case {11,12,1}
            policy(y,x,h,1) = -1;
            policy(y,x,h,2) = 0;
        case {2,3,4}
            policy(y,x,h,1) = 1;
            policy(y,x,h,2) = 1;
        case {5,6,7}
            policy(y,x,h,1) = 1;
            policy(y,x,h,2) = 0;
        case {8,9,10}
            policy(y,x,h,1) = 1;
            policy(y,x,h,2) = -1;
    end
end
% Case 7: (x,y) = (6,<5)
if (x == 6 && (rows+1) - y < 5)
    switch h
        case {11,12,1}
            policy(y,x,h,1) = 1;
            policy(y,x,h,2) = -1;
        case {2,3,4}
            policy(y,x,h,1) = -1;
            policy(y,x,h,2) = -1;
        case {5,6,7}
            policy(y,x,h,1) = -1;
            policy(y,x,h,2) = 1;
        case {8,9,10}
            policy(y,x,h,1) = 1;
            policy(y,x,h,2) = 1;
    end
end
% Case 8: (x,y) = (6,5)
if (x == 6 && (rows+1) - y == 5)
    switch h
        case {11,12,1}
```

```
                            policy(y,x,h,1) = 1;
                            policy(y,x,h,2) = -1;
                        case {2,3,4}
                            policy(y,x,h,1) = -1;
                            policy(y,x,h,2) = 0;
                        case {5,6,7}
                            policy(y,x,h,1) = 1;
                            policy(y,x,h,2) = 1;
                        case {8,9,10}
                            policy(y,x,h,1) = 1;
                            policy(y,x,h,2) = 0;
                    end
                end
                % Case 9: (x,y) = (6,6)
                if (x == 6 && (rows+1) - y ==6)
                    switch h
                        case {11,12,1}
                            policy(y,x,h,1) = -1;
                            policy(y,x,h,2) = -1;
                        case {2,3,4}
                            policy(y,x,h,1) = -1;
                            policy(y,x,h,2) = 1;
                        case {5,6,7}
                            policy(y,x,h,1) = 1;
                            policy(y,x,h,2) = 1;
                        case {8,9,10}
                            policy(y,x,h,1) = 1;
                            policy(y,x,h,2) = -1;
                    end
                end
            end
        end
    end
end
```

Listing 6: MATLAB code for robot trajectory plotting

```
function plot_route(robot_state,prerotation,action)
% Using [RGB] to plot the corresponding colored grid world
rgb_1 = [1 1 1 1 1 1;...
         1 1 1 1 0 1;...
         1 1 1 1 1 1;...
         1 1 1 1 1 1;...
         1 1 1 1 1 1;...
         1 1 1 1 1 1];
rgb_2 = [0 0 0 0 0 0;...
         0 1 1 1 1 0;...
         0 1 1 1 1 0;...
```

```
           0  1  1  1  1  0;...
           0  1  1  1  1  0;...
           0  0  0  0  0  0];
rgb_3 = [0  0  0  0  0  0;...
           0  1  1  0  0  0;...
           0  1  1  0  1  0;...
           0  1  1  1  1  0;...
           0  1  1  1  1  0;...
           0  0  0  0  0  0];
rgb = cat(3,rgb_1,rgb_2,rgb_3);
[rows,cols,deps] = size(rgb);
GW = figure(1);
gridWorld = imshow(rgb,'InitialMagnification','fit');
set(gca,'ticklabelinterpreter','latex');
title('Robot Trajectory','fontsize',20,'interpreter','latex');

% Plot the robot state on grid world
% Number shown on the grid is the heading
for gw_x = 1:rows
    for gw_y = 1:cols
        if (robot_state(gw_x,gw_y) ~= 0)
            text(gw_y,gw_x,num2str(robot_state(gw_x,gw_y)),...
                'HorizontalAlignment','center','VerticalAlignment','middle');
            xlabelStr = ['For Next State: Action = ',num2str(action),' Prerotation = ',num2str(prerotation)];
            xlabel(xlabelStr,'fontsize',18,'interpreter','latex');
            drawnow;    refreshdata;
        end
    end
end

% 'tightfig.m' is an open sourse function
% Trim off the extra region in the figure
tightfig;
end
```

Listing 7: Matlab code for computing policy evaluation

```
function [val] = get_value(command,lambda,pe,rows)
    val = init();
    value_difference = 10e5;
    % Policy Evaluation
    while value_difference > 10e-5
        % Store previous value to temp
        temp = val;
        % Compute and update new value and store
        for x = 1:6
```

```
            for y = 1:6
                for h = 1:12
                    % Find all possible movements due to prerotation
                    % Note: here the next state and doesn't matter
                    [garbage,s] = probability(pe,[x;y;h],command((rows+1)-y,x,h,:),[1,1,1]');
                    % Update the value calculated for policy before update
                    val((rows+1)-y,x,h) = s(4,1) * (reward([s(1,1);s(2,1)]) + lambda*val((rows+1)-s(2,1),s(1,1),s(3,1))) +...
                                          s(4,2) * (reward([s(1,2);s(2,2)]) + lambda*val((rows+1)-s(2,2),s(1,2),s(3,2))) +...
                                          s(4,3) * (reward([s(1,3);s(2,3)]) + lambda*val((rows+1)-s(2,3),s(1,3),s(3,3)));
                end
            end
        end
        % Compare new value with previous value
        value_difference = sum(sum(sum(abs(temp-val))));
    end
end
```

Listing 8: MATLAB code for running the π_0 case

```
%% EE 183DA Lab 3
% Team Buffalo
    % Iou-Sheng (Danny) Chang   UID: 804-743-003
    % William           Argus   UID: 004-610-455
    % XianXing  (Gray)  Jiang   UID: 604-958-018
    % Ho        (Bobby) Dong    UID: 604-954-176
% Markov Decision Processes (MDPs)
clc; clear all; close all;

%% For initial policy \pi_0
robot = zeros(6);
[rows,cols] = size(robot);

% Initialization
[optimal_value_policy,pe,lambda,s,action] = init();
policy_p = init_policy(rows);
prer = 0;
pe = 0;

```

```matlab
val = get_value(policy_p,lambda,pe,rows);

% Robot trajectory plot using initial policy
% NOTE: comment out rest of the code in section 2.3 before running this part
 % Initialization

robot((rows+1)-s(2),s(1)) = s(3);
for j = 1:1000
    act = policy_p((rows+1)-s(2),s(1),s(3),:);
    plot_route(robot,prer,act);
    optimal_policy_policy(j,1) = act(1,1,1,1);
    optimal_policy_policy(j,2) = act(1,1,1,2);
    if (s(1) == 5 && s(2) ==5)
        break;
    end
    robot((rows+1)-s(2),s(1)) = 0;
    [s prer]= next_state(pe,s,act);
    robot((rows+1)-s(2),s(1)) = s(3);
end

% Command Window
% 2.3(c)
fprintf('========== 2.3(c) ==========\n');
fprintf('REFER to figure plotted or video linked in the lab report reference\n')

% 2.3(e)
fprintf('========== 2.3(e) ==========\n');
fprintf('REFER to ''val'' matrix\n');
```

Listing 9: Matlab code for optimal policy

```matlab
function [command] = get_policy(command,val,lambda,pe,rows,action)

for x = 1:6
    for y = 1:6
        for h = 1:12
            set = [];
            for j = 1:6
            % Find all possible movements due to prerotation
            % Note: here the next state and doesn't matter
            [garbage,s] = probability(pe,[x;y;h],action(j,:),[1,1,1]');
            % Store the value calculated for each actions
            % Stored in a temp array
            set(j) = s(4,1) * (reward([s(1,1);s(2,1)]) + lambda*val((rows+1)-s(2,1),s(1,1),s(3,1))) +...
```

```
                     s(4,2) * (reward([s(1,2);s(2,2)]) + lambda*val((rows+1)-s(2,2),s(1,2),s(3,2))) +...
                     s(4,3) * (reward([s(1,3);s(2,3)]) + lambda*val((rows+1)-s(2,3),s(1,3),s(3,3)));
            end
            % Special case
            % When we reach the goal (green) square
            % Robot can perform action (0,0) --> stay still (no movement, no rotation)
            if (x == 5 && y == 5)
                [garbage,s] = probability(pe,[x;y;h],action(7,:),[1,1,1]');
                set(7) = s(4,1) * (reward([s(1,1);s(2,1)]) + lambda*val((rows+1)-s(2,1),s(1,1),s(3,1))) +...
                      s(4,2) * (reward([s(1,2);s(2,2)]) + lambda*val((rows+1)-s(2,2),s(1,2),s(3,2))) +...
                      s(4,3) * (reward([s(1,3);s(2,3)]) + lambda*val((rows+1)-s(2,3),s(1,3),s(3,3)));
            end
            % Find the best action (maximum value) and store it
            [M,I] = max(set);
            % Find value of original action
            [garbage,orig] = probability(pe,[x;y;h],command((rows+1)-y,x,h,:),[1,1,1]');
            temp = orig(4,1) * (reward([orig(1,1);orig(2,1)]) + lambda*val((rows+1)-orig(2,1),orig(1,1),orig(3,1))) +...
                orig(4,2) * (reward([orig(1,2);orig(2,2)]) + lambda*val((rows+1)-orig(2,2),orig(1,2),orig(3,2))) +...
                orig(4,3) * (reward([orig(1,3);orig(2,3)]) + lambda*val((rows+1)-orig(2,3),orig(1,3),orig(3,3)));
            % Store new policy if find better
            if temp < M
                % Store movement (forward/backward) into command
                command((rows+1)-y,x,h,1) = action(I,1);
                % Store rotation (left/none/right) into command
                command((rows+1)-y,x,h,2) = action(I,2);
            end
        end
    end
end
```

Listing 10: MATLAB code for Policy Iteration

```
function [command,val] = policy(policy_p,lambda,pe,rows,action)
    % Initialization of initial policy \pi_0
```

```
    command = policy_p;
    exit = 0;

    while exit == 0
        command_copy = command;
        % Initialization
        val = get_value(command,lambda,pe,rows);
        command = get_policy(command_copy,val,lambda,pe,rows,action)
            ;
        % Policy Improvement


        % If policy converges, exit the while loop and set to true
            (1)
        if isequal(command_copy,command) == 0
            exit = 0;
        else
            exit = 1;
        end
    end
    val = get_value(command,lambda,pe,rows);
end
```

Listing 11: MATLAB code for running policy iteration and value iteration

```
%% EE 183DA Lab 3
% Team Buffalo
    % Iou-Sheng (Danny) Chang   UID: 804-743-003
    % William           Argus   UID: 004-610-455
    % XianXing  (Gray)  Jiang   UID: 604-958-018
    % Ho        (Bobby) Dong    UID: 604-954-176
% Markov Decision Processes (MDPs)
clc; clear all; close all;

%% Robot State Initialization
robot = zeros(6);
[rows,cols] = size(robot);

%% 2.1 MDP System
% 2.1(a)
% REFER to stateSpace R^{3} matrix
fprintf('========== 2.1(a) ==========\n');
fprintf('REFER to stateSpace R^{3} matrix in ''init.m''\n');
% 2.1(b)
% REFER to action R^{2} matrix
fprintf('========== 2.1(b) ==========\n');
fprintf('REFER to action R^{2} matrix in ''init.m''\n');
% 2.1(c)
```

```
% REFER to 'probability.m' function for detail
fprintf('========== 2.1(c) ==========\n');
fprintf('REFER to ''probability.m'' function for detail\n');

% 2.1(d)
% REFER to 'next_state.m' function for detail
fprintf('========== 2.1(d) ==========\n');
fprintf('REFER to ''next_state.m'' function for detail\n');

%% 2.2 Planning problem
% 2.2(a)
% REFER to 'reward.m' function for detail
% Use boolean to check given state if return correct reward value
fprintf('========== 2.2(a) ==========\n');
fprintf('REFER to ''reward.m'' function for detail\n');
% border states (reward = -100)
fprintf('border states reward value = -100 (yes(1) / no(0)): %d\n', ...
        (reward([1,1]) == -100) && (reward([1,2]) == -100) && (reward([1,3]) == -100) && (reward([1,4]) == -100) && (reward([1,5]) == -100) &&...
        (reward([6,2]) == -100) && (reward([6,3]) == -100) && (reward([6,4]) == -100) && (reward([6,5]) == -100) && (reward([6,6]) == -100) &&...
        (reward([2,1]) == -100) && (reward([3,1]) == -100) && (reward([4,1]) == -100) && (reward([5,1]) == -100) && (reward([6,1]) == -100) &&...
        (reward([1,6]) == -100) && (reward([2,6]) == -100) && (reward([3,6]) == -100) && (reward([4,6]) == -100) && (reward([5,6]) == -100));
% lane markers (reward = -10)
fprintf('lane markers reward value = -10 (yes(1) / no(0)): %d\n',...
        (reward([4,4]) == -10) && (reward([4,5]) == -10));
% goal square (reward = 1)
fprintf('goal square reward value = 1 (yes(1) / no(0)): %d\n',...
        (reward([5,5]) == 1));
% other states (reward = 0)
fprintf('other states reward value = 0 (yes(1) / no(0)): %d\n',...
        (reward([2,2]) == 0) && (reward([2,3]) == 0) && (reward([2,4]) == 0) && (reward([2,5]) == 0) &&...
        (reward([3,2]) == 0) && (reward([3,3]) == 0) && (reward([3,4]) == 0) && (reward([3,5]) == 0) &&...
        (reward([4,2]) == 0) && (reward([4,3]) == 0) &&...
        (reward([5,2]) == 0) && (reward([5,3]) == 0) && (reward([5,4]) == 0));

%% 2.3 Policy Iteration
```

```
% Use 'Run Section' to only run the policy iteration
clc; clear all; close all;

% Robot State Initialization
robot = zeros(6);
[rows,cols] = size(robot);

% Initialization
[optimal_value_policy,pe,lambda,s,action] = init();
policy_p = init_policy(rows);
prer = 0;

% Using MATLAB built in 'tic toc' command to obtain compute time
tic
[policy_p,optimal_value_policy] = policy(policy_p,lambda,pe,rows,
    action);
computeTime_policyIterationStr = toc;

% Display initial state
robot((rows+1)-s(2),s(1)) = s(3);

% Plot robot state
% Grid World has the same configuration as the figure shown in lab3
    instruction
% Value shown on the graph is the heading
% Next action and prerotation is shown in the x axis
% when we reach the goal block, break out of the loop
for j = 1:1000
    act = policy_p((rows+1)-s(2),s(1),s(3),:);
    plot_route(robot,prer,act);
    optimal_policy_policy(j,1) = act(1,1,1,1);
    optimal_policy_policy(j,2) = act(1,1,1,2);
    if (s(1) == 5 && s(2) ==5)
        break;
    end
    robot((rows+1)-s(2),s(1)) = 0;
    [s prer]= next_state(pe,s,act);
    robot((rows+1)-s(2),s(1)) = s(3);
end

% Command Window
% 2.3(a)
fprintf('========== 2.3(a) ==========\n');
fprintf('REFER to ''init_policy.m'' for detail\n');
% 2.3(b)
fprintf('========== 2.3(b) ==========\n');
fprintf('REFER to ''plot_route.m'' for detail\n');
```

```
% 2.3(c)
fprintf('========== 2.3(c) ==========\n');
fprintf('REFER to ''policy_init.m'' for detail\n');
fprintf('REFER to figure plotted or video linked in the lab report reference\n')
% 2.3(d)
fprintf('========== 2.3(d) ==========\n');
fprintf('REFER to ''get_value.m'' for detail\n');
% 2.3(e)
fprintf('========== 2.3(e) ==========\n');
fprintf('REFER to ''policy_init.m'' for detail\n');
% 2.3(f)
fprintf('========== 2.3(f) ==========\n');
fprintf('REFER to ''get_policy.m'' for detail\n');
% 2.3(g)
fprintf('========== 2.3(g) ==========\n');
fprintf('REFER to ''policy.m'' for detail\n');
% 2.3(h)
fprintf('========== 2.3(h) ==========\n');
fprintf('REFER to figure plotted or video linked in the lab report reference\n');
% 2.3(i)
fprintf('========== 2.3(i) ==========\n');
computeTime_policyIterationStr = ['Policy Iteration Compute Time = 4.6858 [s]'];
%computeTime_policyIterationStr = ['Policy Iteration Compute Time = ',num2str(computeTime_policyIterationStr),' [s]'];
fprintf(computeTime_policyIterationStr);

%% 2.4 Value iteration
% Use 'Run Section' to only run the value iteration
clc; clear all; close all;

% Robot State Initialization
robot = zeros(6);
[rows,cols] = size(robot);

% Initialization
[value_prev,pe,lambda,s,action] = init();
optimal_value_value = value_prev;
value_difference = 10e5;
prer = 0;

% Using MATLAB built in 'tic toc' command to obtain compute time
tic
while value_difference > 10e-5
    [value_prev,policy_v] = value(optimal_value_value,action,lambda,
```

```
        pe,rows);
    value_difference = sum(sum(sum(abs(value_prev-
        optimal_value_value))));
    optimal_value_value = value_prev;
end
computeTime_valueIteration = toc;

% Display initial state
robot((rows+1)-s(2),s(1)) = s(3);

% Plot robot state
% Grid World has the same configuration as the figure shown in lab3
    instruction
% Value shown on the graph is the heading
% Next action and prerotation is shown in the x axis
% when we reach the goal block, break out of the loop
for j = 1:1000
    act = policy_v((rows+1)-s(2),s(1),s(3),:);
    plot_route(robot,prer,act);
    optimal_policy_value(j,1) = act(1,1,1,1);
    optimal_policy_value(j,2) = act(1,1,1,2);
    if (s(1) == 5 && s(2) ==5)
        break;
    end
    robot((rows+1)-s(2),s(1)) = 0;
    [s prer]= next_state(pe,s,act);
    robot((rows+1)-s(2),s(1)) = s(3);
end

% Command Window
% 2.4(a)
fprintf('========== 2.4(a) ==========\n');
fprintf('REFER to ''optimal_policy_value'' matrix\n');
fprintf('REFER to ''optimal_value_value'' matrix\n');
% 2.4(b)
fprintf('========== 2.4(b) ==========\n');
fprintf('REFER to figure plotted or video linked in the lab report
    reference\n');
% 2.4(c)
fprintf('========== 2.4(c) ==========\n');
computeTime_valueIterationStr = ['Value Iteration Compute Time = ',
    num2str(computeTime_valueIteration),' [s]'];
fprintf(computeTime_valueIterationStr);
```

5 Value Iteration

5.1 Overview

Value iteration uses a value function to assign a value to each state. The higher the value of a given state, the better it is for the robot to be in that state. The value function eq. (7) gives the state a value equal to the total expected rewards the robot will gain from starting in that state and moving according to its prescribed policy.

$$V^{\pi}(s) = E\Big[\sum_{i=1}^{T} \gamma^{i-1} r_i\Big] \forall s \in \mathbb{S} \tag{7}$$

The optimal policy is found using the Bellman equation eq. (8) to perform value iteration, which improves $V(s)$ until it converges on the optimal value function eq. (9). The optimal policy eq. (10) is the policy that is used in the optimal value function.

$$V^{*}(S) = \max_{a} \Big[R(s,a) + \gamma \sum_{s' \in \mathbb{S}} p(s'|s,a) V^{*}(s')\Big] \tag{8}$$

$$V^{*}(s) = \max_{\pi} V^{\pi}(s) \ \forall s \in \mathbb{S} \tag{9}$$

$$\pi^{*} = arg\max_{\pi} V^{\pi}(s) \ \forall s \in \mathbb{S} \tag{10}$$

The optimal Q-function eq. (11) will give the total expected reward gained by the robot given that the robot starts in state s, takes action a, and then continues to take the best action for the rest of its movement.

$$V^{*}(s) = \max_{a} Q^{*}(s,a) \ \forall s \in \mathbb{S} \tag{11}$$

5.2 Value Iteration

The function returns optimal policy π^* and optimal value V, assuming $V(s) = 0 \ \forall \ s \in \mathbb{S}$.

Pseudo Code for Value Iteration

Assign arbitrary values to V(s)
Rerun
    For all states in the state space
        For all actions in the action space
            Q(s,a), where Q(s,a) = $E[r|s,a] + \gamma \sum_{s' \in S} P(s'|s,a)V(s')$
        V(s), where V(s) is $\max_a$Q(s,a)
Continue until V(s) converges

Figure 5: Pseudo code for Value Iteration

The value iteration function is implemented in MATLAB as *value.m* (listing 12).

5.3 Optimal policy trajectory plot

The plot of the trajectory using optimal policy π^* and $P_e = 0$.

The trajectory plotted is shown in video in the Appendix and Demonstration section.

5.4 Computation time

We use the MATLAB built in $tic - toc$ function to determine the compute time of Value Iteration, as shown in *main.m* (listing 11), the compute time is then printed to the commend window as shown in fig. 6, it is determined to be 8.6225 $[s]$.

```
========== 2.4(a) ==========
REFER to 'optimal_policy_value' matrix
REFER to 'optimal_value_value' matrix
========== 2.4(b) ==========
REFER to figure plotted or video linked in the lab report reference
========== 2.4(c) ==========
Value Iteration Compute Time = 8.6225 [s]>>
```

Figure 6: Compute time of Value Iteration

5.5 Matlab Implementation

Listing 12: MATLAB code for Value Iteration

```matlab
function [val,command] = value(V,action,lambda,pe,rows)
    % ===== Initialization ===== %
    val = V;
    command = [];

    % ===== ?ompute the value ===== %
    for x = 1:6
```

```
        for y = 1:6
            % Iterate through 12 possible headings
            for h = 1:12
                % Iterate through 7 possible actions
                value_allActions = [];
                for j = 1:6
                    % Find all possible movements due to prerotation
                    % Note: here the next state and doesn't matter
                    [garbage,s] = probability(pe,[x;y;h],action(j,:),[1,1,1]');
                    % Store the value calculated for each actions
                    % Stored in a temp array
                    value_allActions(j) = s(4,1) * (reward([s(1,1);s(2,1)]) + lambda*val((rows+1)-s(2,1),s(1,1),s(3,1))) +...
                                          s(4,2) * (reward([s(1,2);s(2,2)]) + lambda*val((rows+1)-s(2,2),s(1,2),s(3,2))) +...
                                          s(4,3) * (reward([s(1,3);s(2,3)]) + lambda*val((rows+1)-s(2,3),s(1,3),s(3,3)));
                end
                % Special case
                % When we reach the goal (green) square
                % Robot can perform action (0,0) --> stay still (no movement, no rotation)
                if (x == 5 && y == 5)
                    [garbage,s] = probability(pe,[x;y;h],action(7,:),[1,1,1]');
                    value_allActions(7) = s(4,1) * (reward([s(1,1);s(2,1)]) + lambda*val((rows+1)-s(2,1),s(1,1),s(3,1))) +...
                                          s(4,2) * (reward([s(1,2);s(2,2)]) + lambda*val((rows+1)-s(2,2),s(1,2),s(3,2))) +...
                                          s(4,3) * (reward([s(1,3);s(2,3)]) + lambda*val((rows+1)-s(2,3),s(1,3),s(3,3)));
                end
                % Find the best action (maximum value) and store it
                [M,I] = max(value_allActions);
                val((rows+1)-y,x,h)= M;
                % Store movement (forward/backward) into command
```

```
                command((rows+1)-y,x,h,1) = action(I,1);
                % Store rotation (left/none/right) into command
                command((rows+1)-y,x,h,2) = action(I,2);
            end
        end
    end
end
```

6 Additional Scenarios

For the additional scenarios, we run the Policy Iteration with $p_e = 0.1$ using the original code, only changing the p_e value in *init.m* (listing 1). However for the altered reward section (the reward of goal square +1 only applies when the robot is pointing straight down $h = 6$, it is 0 otherwise), we duplicated the original code and further changed the *reward.m* function to account for the scenarios.

The modified *reward.m* is combined with the *init.m* into the Matlab function *add_init.m* (listing 13).

6.1 Trajectory with error

The plot of the trajectory using Policy Iteration under optimal policy π^* and $p_e = 0.1$.

The trajectory plotted is shown in video in the Appendix and Demonstration section, we can also see from the video that the compute time is now 6.1113 $[s]$.

6.2 Trajectory with altered reward

The plot of the trajectory using Policy Iteration under optimal policy π^* and $p_e = 0$ & $p_e = 0.25$.

The trajectory plotted is shown in video in the Appendix and Demonstration section, we can also see from the video that the compute time for $p_e = 0$ is 3.8029 $[s]$ (fig. 7) and the compute time for $p_e = 0.25$ is 4.1386 $[s]$ (fig. 8).

```
========== 2.5(b) ==========
REFER to figure plotted or video linked in the lab report reference
Policy Iteration Compute Time = 3.8029 [s]>>
```

Figure 7: Compute time under altered reward $p_e = 0$

```
========== 2.5(b) ==========
REFER to figure plotted or video linked in the lab report reference
Policy Iteration Compute Time = 4.1386 [s]>>
```

Figure 8: Compute time under altered reward $p_e = 0.25$

6.3 Conclusions

Policy and Value iteration achieve the same goal using different algorithms. The compute time of policy iteration was much smaller than the compute time of value iteration. This is seen in sections 4.10 and 5.4, and is expected, as policy iteration's compute time is usually faster than value iteration compute time, as the policy iteration algorithm converges to the optimal policy faster than the value iteration algorithm converges to the optimal value policy.

The altered reward scenario (section 6.2) resulted in the robot only receiving the value given by the final state when it had a specific heading within the final state, also known as the goal square. The computation time of the policy iteration algorithm was less for this scenario than it was for the general scenario policy iteration computation. This is believed to be caused by the fact that because the final state was specific (there was an exact coordinate and heading required), compared to the general final state (only an exact coordinate required), there were less possible movements to achieve the final state that resulted in gaining reward.

6.4 Matlab Implementation

Listing 13: Matlab code for the altered reward

```
function [value,reward,pe,lambda,s_init,action] = init()
% Initialization of value
value_int= zeros(6);
reward_temp = [-100 -100 -100 -100 -100 -100;...
               -100  0    0    -10  0    -100;...
               -100  0    0    -10  0    -100;...
               -100  0    0    0    0    -100;...
               -100  0    0    0    0    -100;...
               -100 -100 -100 -100 -100 -100];

% Expend the value matrix to 12 headings' 3D matrix space
[rows,cols] = size(value_int);
for x = 1:rows
    for y = 1:cols
        for h = 1:12
            value(x,y,h) = value_int(x,y);
            reward(x,y,h) = reward_temp(x,y);
        end
    end
end

reward(2,5,6) = 1;

% Initialization of error percentage (pe)
pe = 0.25;

% Initialization of discount factor
lambda = 0.9;

% Initialization of initial state
s_init = [2;5;6];
```

```

% Initialization of actions set
% column 1: forward 1 / no movement 0 / backward -1
% column 2: leftturn -1 / no rotation 0 / rightturn 1
action = [1   -1;...
          1   0;...
          1   1;...
          -1  -1;...
          -1  0;...
          -1  1;...
          0  0];
end
```

7 Appendix and Demonstration

All class materials and documents can be found on GitHub.
All codes, videos, figures and related documents for Lab 3 can be found on GitHub.

For specific experiment videos and demonstrations please visit the links attached to the topics:

- Video: Matlab live robot trajectory using π_0, $p_e = 0$
- Video: Matlab live robot trajectory for Policy Iteration under π^*, $p_e = 0$
- Video: Matlab live robot trajectory for Value Iteration under π^*, $p_e = 0$
- Video: Matlab live robot trajectory for Policy Iteration under π^*, $p_e = 0.1$
- Video: Matlab live robot trajectory for Policy Iteration under altered reward & π^*, $p_e = 0$
- Video: Matlab live robot trajectory for Policy Iteration under altered reward & π^*, $p_e = 0.25$
- Matlab code - original
- Matlab code - altered reward
- Images and Figures
- Images for compute time

8 References

1. Mehta, Ankur. ”Design of Robotic Systems I.” EE183DA. University of California Los Angeles, Los Angeles, California. Lectures.

2. Mehta, Ankur. ”Design of Robotic Systems I.” EE183DA. University of California Los Angeles, Los Angeles, California. Lab 3 Instructions.

3. Feinberg, Eugene A., and Adam Shwartz. Handbook of Markov Decision Processes: Methods and Applications. Kluwer Academic Publishers, 2002.

4. Ross, Sheldon M. Introduction to Stochastic Dynamic Programming. Acad. Press, 1995.

5. Alzantot, Moustafa. “Deep Reinforcement Learning Demysitifed - Policy Iteration, Value Iteration and...” Medium.com, Medium, 9 July 2017.