## CS 545 - Introduction to Robotics

## Homework Assignment 3 (Due April 30 ${ }^{\text {th }}$ )

1) (40 points) In this question, you will implement, step-by-step, a task called the pole-balancing task, represented as a Markov Decision Process. It is described as follows (see accompanying figure):


Figure 1: The Pole Balancing Task
It is an episodic task where the agent is a cart trying to balance a pole placed on top of it. The cart can move either left or right, but will break if it hits either the left or right steps. Pole balancing is achieved by compensating the fall of the pole in one direction by moving in that direction (think of balancing a stick in your hand for an approximate idea of how to do this).

More formally, we have the variables:

| Variable | Unit | Meaning | Valid Range |
| :---: | :---: | :---: | :---: |
| $\dot{\theta}$ | Radians | Angle of pole from vertical | $\left[-\frac{\pi}{10}, \frac{\pi}{10}\right]$ |
| $\dot{\theta}$ | Radians $/ \mathrm{sec}$ | Angular velocity of pole | $[-\pi, \pi]$ |
| $x$ | Meters | Cart position | $[-2.4,2.4]$ |
| $\dot{x}$ | Meters/s | Cart velocity | $[-2,2]$ |

The reward is +1 on every time step, unless the pole has become unbalanced or the cart hits either wall, in which case the reward is -100 . That is, the agent is rewarded for keeping the pole balanced. At every step, we can compute the new values of the variables using a classical mechanics system. The Matlab function cartStep(s,a) provides you with an implementation of this system: it takes as an argument a four-dimensional vector $[\theta \dot{\theta} x \dot{x}]$ and an action $a \in\{-1,1\}$ and returns the reward and next state. The episode should terminate as soon as a reward of -100 is returned, which occurs when $\theta$ or $x$ falls outside
of the valid range defined in the provided table. Note that the valid ranges for $\dot{\theta}$ and $\dot{x}$ are only given as an indication; the variables themselves are not bounded in the system. The initial starting position is [00 00 ]. The discount factor is $\gamma=0.9$.
a) Benchmark Policies : Begin by implementing a fixed policy, namely one that takes action -1 when $\theta<0$ and action 1 when $\theta \geq 0$. This function should be called fixedPolicy.m. Compute the number of steps until 1the episode terminates. Also implement a fixed policy which takes action -1 when $\dot{\theta}<0$ and action 1 when $\dot{\theta}>0$. These two policies will serve as your benchmark for the other questions; report on the number of steps it takes for each policy to fail.
b) Tile Coding: The next logical step is to implement tile coding. Divide the four-dimensional state space into a number of tiles of a specific size. A good initial choice for tile size is $\left[\frac{\pi}{60} \frac{\pi}{8} 0.10 .2\right]$. Write a function getTile.m which takes a state vector as angument and returns a fourdimensional index. Also define a four dimensional Q-value table where each entry represents a tile. Recall that you need one such table for each possible action. You might need to use the CELL Matlab command to do this. Remember to handle cases where the given state vector falls outside of the valid range. Discuss any issues that arise in the implementation of this function.
c) Implement Q-learning using your tile coding functions from part a) and the cartStep function. Name your function qlearning.m. You should iteratively generate an action from your Q-learning policy, use cartStep to step through the environment, and then update your Q-value table. For this part of the assignment, you should use an $\epsilon$-greedy algorithm. Run your algorithm for a varying number of episodes (at least up to 10000). Report the performance of the algorithm in terms of the number of steps until the end of the episode. Compare with the benchmark policies. Discuss any extra assumptions which you made.
d) Now that we have a basic Q-learning algorithm, we want to study the effects of exploration. For this purpose, we will look at another exploration mechanism, the Boltzmann exploration. Now that we have a basic Q-learning algorithm, we want to study the effects of exploration. For this purpose, we will look at another exploration mechanism, the Boltzmann exploration:

$$
\pi(s, a)=\frac{e^{\frac{Q(s, a)}{T}}}{\sum_{a^{\prime}} e^{\frac{Q\left(s, a^{\prime}\right)}{T}}}
$$

There is a temperature parameter. When $T \rightarrow 0, \pi(s, a)$ becomes greedy with respect to $\mathrm{Q}(\mathrm{s}, \mathrm{a})$; when $\mathrm{T} \rightarrow \infty$, the policy becomes uniformly random over all actions. Modify your original Q-learning implementation to use Boltzmann exploration instead of q-greedy. Name this function Boltzmannqlearning.m. Run it for the same number of episodes as in Question 2 , and report the performance of the algorithm in function of the number of steps. Compare the two exploration methods with each other and with the benchmark policies. Discuss any additional assumptions you made.


Figure 2: Gridworld Task
2) ( 30 points) Included with the assignment is MATLAB code for a MDP representing two $3 \times 3$ rooms connected by a $2 \times 1$ hallway (see Fig. 3). An agent in this environment can move noisily in each of the four directions: up, down, left, right. Also, every action yields a reward of -1, except in state 20 and 21. Any action taken from state 20, the goal state, takes the agent into state 21 and gives it a reward of +100 . Any action taken in state 21 transitions back to state 21 , making it a terminal state; no reward is given in this state. The following functions are provided:
loadMDP.m - Loads the MDP parameters, including the transition function and the reward function. An example policy is also included.
drawpolicy.m - Draws a given policy. Can handle probabilistic policies. Assumes the agent takes the action with highest probability; for visualization purposes.
a) Write a function which converts the $Q$-value table into a policy. Let $Q(s, a)$ be the value of taking action a in state s and policy(s, a) be 1 for the optimal action and 0 otherwise.
b) Using the MDP format provide, implement value iteration to determine the optimal policy. Examine loadMDP.m to see the format for the MDP. Initialize the value function as random or uniform. How many iterations did it take for value iteration to converge? Plot the value function after one iteration, after converging, and after an iteration during the procedure. Provide a brief description of the results. Repeat using the other initialization procedure. Compare the results.
3) (30 points) For this question, you will be asked to implement a set of potential fields. You will find the matlab functions meshgrid and quiver useful. Assume the map is a MxN grid, with obstacles, goals, and the robot occupying a single gridcell. The goal is to determine the vector at each gridpoint based on the desired potential field.
a) Implement a potential field to avoid an obstacle at an ( $x, y$ ) position. Visualize this potential field using the functions describe above for a single obstacle at a position of your choice. How did you determine the vector at each point? Address the magnitude and orientation of each
vector. What happens (in this simplified case) when a robot is at the same x or y plane as the obstacle?
b) Implement a random potential field, which provides a random vector at each gridpoint. Add this random field to the random field implement in a). Visualize the resulting potential field. Discuss what happens when the robot is at the same x or y plane as the obstacle now.
c) Using the function from a), generate the potential field for two separate obstacles. Combine those two fields into a single potential field.
d) Implement a potential field drawing the robot towards a goal. Briefly discuss the implementation. Generate a final potential field for a robot in an environment with 3 obstacles and 1 goal. Include the random field.

