EE209AS (Fall 2018)

Computational Robotics

Prof. Ankur Mehta
mehtank@ucla.edu

Problem set 2
Due 2pm Tue. Oct. 16, 2018

Objectives

The goal of this lab is to explore Markov Decision Processes (MDPs) to control a simple discretized robot. You will develop and implement a model of the robot behavior and use it to accomplish a prescribed task.

Deliverables

This project will require you to write code. If you do not have one already, create an account on `http://github.com`, and create a project for this class. Make sure the **well commented** code for this lab is committed and pushed, and submit a link to the repository. For some possible resources on git, see below.

You may work individually or in pairs on this assignment. Each person needs to submit their own solutions, but the team can submit common code. Indicate on your solutions who you worked with, and for each person identify 1) the specific contributions made by each, and 2) an aggregate percentage of the total work done.

Upload your solutions to gradescope.

Preliminaries

0(a). What is the link to your (fully commented) github repo for this pset?

0(b). Who did you collaborate with?

0(c). What were the specific contributions of each team member?

0(d). What was the aggregate % contributions of each team member?

1 Setup

Consider a simple robot in a 2D grid world of length L and width W. That is, the robot can be located at any lattice point $(x, y) : 0 \leq x < L, 0 \leq y < W; x, y \in \mathbb{N}$. At each point, the robot can face any of the twelve headings identified by the hours on a clock $h \in \{0 \dots 11\}$, where $h = 0$ represents 12 o'clock is pointing up (i.e. positive y) and $h = 3$ is pointing to the right (i.e. positive x).

In each time step, the robot can chose from several actions. Each singular action will consist of a movement followed by a rotation.

- The robot can choose to take no motion at all, staying still and neither moving nor rotating.
- Otherwise the robot can choose to either move "forwards" or "backwards".
  - This may cause a pre-rotation error, see below.
  - This will cause the robot move one unit in the direction it is facing, rounded to the nearest cardinal direction.

  That is, if the robot is pointing towards either 2, 3, or 4 and opts to move "forwards", the robot will move one unit in the $+x$ direction. Similarly, if the robot is pointing towards either 11, 0, or 1 and opts to move "backwards", it will move one unit in the $-y$ direction.
- After the movement, the robot can choose to turn left, not turn, or turn right. A left (counter-clockwise) turn will decrease the heading by 1 (mod 12); right (clockwise) will increase the heading by 1 (mod 12). The robot can also keep the heading constant.
- Attempting to move off of the grid will result in no linear movement, but the rotation portion of the action will still happen.

Note that aside from the at edges of the grids, the robot can only rotate if it also moves forwards or backwards; it can move without rotating though.

The robot has an error probability p_e: if the robot choses to move, it will *first* rotate by +1 or -1 (mod 12) with probability p_e each, before it moves. It will not pre-rotate with probability $1 - 2p_e$. Choosing to stay still (take no motion) will not incur an error rotation.

Create code to simulate this system. You may need to create objects to represent states $s \in S$ and/or actions $a \in A$. Note that the state s will later be used as an index to a matrix/array when computing policies and values.

1(a). Create (in code) your state space $S = \{s\}$. What is the size of the state space N_S?

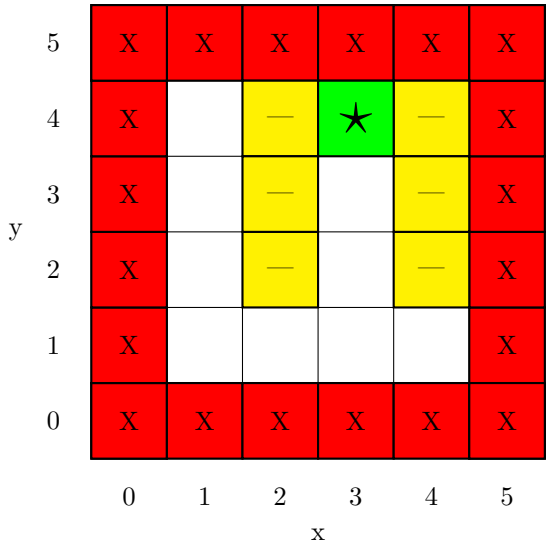
1(b). Create (in code) your action space $A = \{a\}$. What is the size of the action space N_A?

1(c). Write a function that returns the probability $p_{sa}(s')$ given inputs p_e, s, a, s'.

1(d). Write a function that uses the above to return a next state s' given error probability p_e, initial state s, and action a. Make sure the returned value s' follows the probability distribution specified by p_{sa}.

2 Problem

Consider the grid world shown below, with $L = W = 6$:



The rewards for each state are independent of heading angle (or action taken). The border states $\{x = 0, x = L, y = 0, y = W\}$ (red, marked X) have reward -100. The lane markers (yellow, marked —) have reward -1. The goal square (green, marked $\star$) has reward +1. Every other state has reward 0.

2(a). Write a function that returns the reward $R(s)$ given input s.

3 Policy iteration

Assume an initial policy π_0 of taking the action that gets you closest to the goal square. That is, if the goal is in front of you, move forward; if it is behind you, move backwards; 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.

3(a). Create and populate a matrix/array that stores the action $a = \pi_0(s)$ prescribed by the initial policy π_0 when indexed by state s.

3(b). Write a function to generate and plot a trajectory of a robot given policy matrix/array π, initial state s_0, and error probability p_e.

3(c). Generate and plot a trajectory of a robot using policy π_0 starting in state $x = 1, y = 4, h = 6$ (i.e. top left corner, pointing down). Assume $p_e = 0$.

3(d). Write a function to compute the policy evaluation of a policy π. That is, this function should return a matrix/array of values $v = V^{\pi}(s)$ when indexed by state s. The inputs will be a matrix/array storing π as above, along with discount factor λ.

3(e). What is the value of the trajectory in 3(c)? Use $\lambda = 0.9$.

3(f). Write a function that returns a matrix/array π giving the optimal policy given a one-step lookahead on value V.

3(g). Combine your functions above in a new function that computes policy iteration on the system, returning optimal policy π^* with optimal value V^*.

3(h). Run this function to recompute and plot the trajectory and value of the robot described in 3(c) under the optimal policy π^*.

3(i). How much compute time did it take to generate your results from 3(h)? You may want to use your programming language's built-in runtime analysis tool.

4 Value iteration

4(a). Using an initial condition $V(s) = 0 \ \forall s \in S$, write a function (and any necessary subfunctions) to compute value iteration, again returning optimal policy π^* with optimal value V^*.

4(b). Run this function to recompute and plot the trajectory and value of the robot described in 3(c) under the optimal policy π^*. Compare these results with those you got from policy iteration in 3(h).

4(c). How much compute time did it take to generate your results from 4(b)? Use the same timing method as in 3(i).

5 Additional scenarios

5(a). Recompute the robot trajectory and value given initial conditions from 3(c) but with $p_e = 25\%$.

5(b). Assume the reward of $+1$ only applies when the robot is pointing down, e.g. $h \in \{5, 6, 7\}$ in the goal square; the reward is 0 otherwise. Recompute trajectories and values given initial conditions from 3(c) with $p_e \in \{0, 25\%\}$.

5(c). Qualitatively describe some conclusions from these scenarios.

Git Resources

- Getting started with GitHub: `https://guides.github.com/activities/hello-world/`
- Detailed documentation on how to use git: `https://git-scm.com/book/en/v2`
- Try git in the browser: `https://try.github.io/levels/1/challenges/1`