Robot Learning

Lab Exercise 1

Wednesday 18th January 2023

Edward Johns

Welcome to the first lab exercise for the Robot Learning course! By the end of this exercise, you will have become familiar with the 2D simulation environment that will be used throughout the course, in both the subsequent lab exercises and the coursework. You do not need to submit anything and this exercise is not assessed, but it will help you to later understand how to do the courseworks. There are four parts to this exercise. As well as this document, you should also have downloaded three Python files: *robot-learning.py*, *robot.py*, and *environment.py*.

Part 1: Setting up your Python environment (~10 minutes)

First, you will need to set up your Python environment. You should use Python 3 and a virtual environment, and it is helpful to use an IDE, such as PyCharm. The code should run on a range of operating systems, and you are welcome to use either your own machine or one of the machines in the lab. You will need to install the following Python packages: *numpy*, *pyglet*, and *scikit-image*. Click here for guidance on using virtual environments and installing Python packages. Finally, run the script *robot-learning.py*. If you can see a red circle moving around the screen, then it looks like everything is running well so far! You may wish to take a quick look at the code in the three Python files to see if you can understand what is going on.

Part 2: Exploring the simulator (~40 minutes)

Next, you can spend some time exploring the simulator, by making a number of modifications.

1) The robot's initial state is the blue square, and the goal state is the green star. They exist in a square 2D environment, where states in this environment range from (0, 0) to (1, 1). Set the initial state to the bottom-left of the environment, at coordinates (0.1, 0.1), and set the goal state to the top-right of the environment at coordinates (0.9, 0.9). You can do this in *Environment.generate_init_and_goal_states()*. Then, run *robot-learning.py* again to observe this change in the initial state and goal state.

2) The environment consists of terrain, which can be of any height between 0 and 1. The lower the terrain, the faster the robot can move at that point. You can think of this like the robot travelling through hills and valleys, where hills slow down the robot. The terrain is shaded according to its height, where lower terrain is darker and higher terrain is lighter. The terrain is stored in the variable *Environment.terrain_map*, which is a 100-by-100 array storing a height for every element in the terrain map. Change the terrain so that the bottom of the map has a height of 0 and the top of the map has a height of 1, with a linearly varying height between. You can do this in

Environment.generate_terrain_map(). Then, run *robot-learning.py* again and you should now see the environment's shading change from the bottom to the top.

3) Currently, the robot takes random actions at every step. Change this so that the robot moves upwards at every step. You can do this in *Robot.step()*. Then, run *robot-learning.py* again to observe the change. You should see the robot move to the top of the environment, but as it reaches the top, it should slow down. This is because the terrain is higher at the top of the environment, and therefore the robot moves slower.

4) Next, you will change the environment's dynamics again. But this time, instead of changing the terrain, you will change the effect that the terrain has on the robot. Take a look at *Environment.step()*, where the *impedance* variable determines how far the robot can move. Originally, the impedance is just the same as the terrain height. Modify this code now, so that the robot slows down even more quickly as the robot moves up the environment. To do this, set the impedance to the terrain height raised to the power 0.2. Then, run *robot-learning.py* again to observe the change. You should see that the robot slows down even more quickly as it moves upwards.

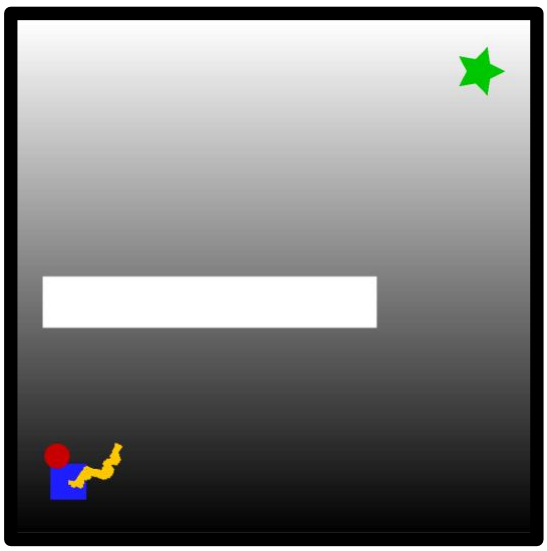
5) You will now add an obstacle to the environment, so that the robot cannot move to the goal in just a straight line. Return to *Environment.generate_terrain_map()*, and add a rectangular region filled with a height of 1, to represent the obstacle. The bottom-left of this region should be at (5, 40) in the terrain map, and the top-right should be at (70, 50). Then, run *robot-learning.py* again to observe the change. You should observe that the robot stops when it reaches the obstacle, and cannot go upwards any further.

6) Finally, change *Robot.step()* back so that the robot takes a random action in each step. Then, add code to *Robot.step()* so that after every 1000 steps, the robot resets to its initial state. In other words, the length of the episode is 1000 steps. Then, run *robot-learning.py* again to observe these episode resets.

Part 3: Visualising the behaviour (~20 minutes)

In this part, you will visualise some of the robot's behaviour, which may help later in the course with debugging, or understanding the effect of different design choices. In *Robot.step()*, add some code to check how close the robot is to the goal at the end of an episode. Then write some code such that, if this distance is the shortest so far from all the episodes, the robot's path (the list of robot states) for that episode is recorded. Then, visualise the best path in yellow, by drawing it onto the window in each call to *Robot.draw()*. To draw a line between each state, use *pyglet.shapes.Line* to create a line, and you should draw the lines in a batch following these instructions: (https://pyglet.readthedocs.io/en/latest/modules/graphics/index.html). To prevent this from slowing down the program too much, you may want to only draw a line between every 10 or even every 100 steps along this best path.

By now, you should have a window which looks something like this:



Part 4: Solving the task (??? minutes)

If you still have time to spare, then this part is an open-ended opportunity to design your own robot controller, and see if you can enable the robot to reach the goal. Of course, you could just manually calculate the optimal path in this simple environment, which would be considered an analytical method rather than a robot learning method. But to prepare for the exercise next week, it would be better to try the following. (1) Execute a set of random episodes. (2) Calculate the best few episodes from these. (3) Take the actions for these few episodes, and generate a new set of actions by perturbing these actions a little. Or, you could calculate the average action sequence and then perturb this a few times. (4) Repeat steps 2 and 3 in a loop. Using a strategy like this, you should observe that over time, the robot can get closer and closer to the goal. Good luck!

End of Exercise