Robot Learning

Lab Exercise 2

Wednesday 25th January 2023

Edward Johns

Welcome to the second lab exercise for the Robot Learning course! By the end of this exercise, you will have implemented some of the methods introduced in Lecture 2. By following this exercise, you should then be able to complete the tasks in Parts 1 and 2 of Coursework 1. Part 1 in this exercise corresponds to Part 1 in Coursework 1, and Part 2 in this exercise corresponds to Part 2 in Coursework 1.

As with Lab Exercise 1, there are the following three Python files: *robot-learning.py*, *robot.py*, and *environment.py*. However, there is also now a fourth Python file: *graphics.py*. Whilst the overall code structure is similar to that in Lab Exercise 1, there have been some changes to make the code easier to manage, now that you will be implementing more complex ideas.

Preliminaries (~20 minutes)

To begin, I suggest that you look through the code and try to understand the role of the some of the functions. First, take a look at functions *update()* and *on_draw()* in *robot-learning.py*. These define the overall flow of the main program, in terms of both the robot's behaviour (*update*) and the graphics drawn in the window *(on_draw)*. Then, take a look at the function *Robot.next_action()*. This function is called in *robot-learning.py* at every timestep, before the robot takes a physical action in the environment. If this is the first timestep, then the robot will first do some planning before executing the first action in the plan, whereas for subsequent timesteps, the robot will simply execute the actions in this plan. Then, take a look at the function *Robot.model()*. This is the robot's model of the environment's dynamics. Note that in this implementation, the robot is assumed to have full access to the true dynamics. If you run *robot-learning.py*, you should see an orange path which the robot is following; this orange path is the robot's plan.

Part 1: Random Shooting (~20 minutes)

In this part, you will implement the Random Shooting algorithm that was introduced in Lecture 2. Random Shooting is a planning method which aims to compute an optimal sequence of actions, by simply randomly sampling sequences and choosing the best one.

To begin, take a look at the *Robot.planning()* function. Currently, this only samples a single sequence of actions, so there is not really any intelligent planning yet. Therefore, you should now modify this function. First, you should rename this function to *Robot.planning_random_shooting()* and tell the robot to use this function in *Robot.next_action()*. Then, you should modify *Robot.planning_random_shooting()* by creating an outer loop, so that the robot samples 100 action sequences, simulates each of them using the model, and then chooses the best one to execute in the environment.

Once you have done this, you should be able to answer Part 1 of Coursework 1.

Part 2: Cross Entropy Method (~60 minutes)

In Part 1, only a very basic planning algorithm was implemented. Next, you will implement a more sophisticated algorithm which, although still relatively simple to implement, is significantly more powerful. This algorithm is the Cross Entropy Method.

To begin, make a copy of your current *Robot.planning_random_shooting()* function, and rename it to *Robot.planning_cross_entropy()*. Then, you should modify this function by introducing an outer loop which loops through 5 iterations of the Cross Entropy Method. In other words, by the end of the algorithm, the algorithm should have refit the action distribution 5 times.

You should use a multivariate Normal distribution, which NumPy offers, where each dimension corresponds to the angle at which the robot moves for a particular timestep. In other words, each timestep has its own mean and variance. However, your covariance matrix should be diagonal and you should not be modelling the covariance between different timesteps. For the first iteration, you should set the mean to be 0, and the covariance diagonals to be very large such that the distribution is effectively a uniform distribution.

Once you have the main code implemented, you can then play around with two important parameters: k, the % of action sequences which are used to refit the action distribution in each iteration, and N, the number of sampled action sequences in each iteration. Experiment with different values for k and N and see if you can enable the robot to reach the goal, whilst keeping the planning horizon (the length of the planned action sequence) at 100.

Once you have done this, you should be able to answer Part 2 of Coursework 1.

End of exercise