CPSC 8810
Motion Planning

Assigned: Feb 12, 2020
Due: 11:59pm, Feb 23, 2020

Programming Assignment 3 — Discrete Planning *(Grading: 0–10 points)*

Problem Description

In the third assignment, you have to use A* to plan the discrete motion of a simple car. For simplicity, we will assume that the state space is 3-dimensional and is given by an x position, a y position, and an orientation θ (so, the speed is always fixed).

Basic Requirements

Please modify the `astar.py` code available on the website to implement your planner. The code assumes that:

1. The car is moving on a 2D grid and its orientation can assume four distinct directions: {0:North, 1:West, 2:South, 3:East}.

2. There are 3 possible actions that the car can take:
   $U = \{$turn right then move forward, move forward, turn left then move forward$\}$

3. The heuristic for the A* search is the number of steps to the 2D goal cell.

As the state space is 3D-dimensional, implementing A* may be more involved than planning for just 2D grid positions as we covered in the class. However, the extension should be easy if you denote the three actions of the car as [-1, 0, 1]. Every time you expand a state and generate its successors, you can add each action value to the index orientation of the state. The new orientation can then be used to determine the neighboring position on the 2D grid. If, for example, the car is on the grid cell [x, y] and its current orientation is 2, i.e. oriented downwards, taking the right turn action having index -1 will lead to a new orientation of 1, i.e., going west. This means that the new state after taking this action will be [x, y-1, 1].

Please, test your A* implementation with different costs. Start with a unit cost for all three actions, then try [10, 1, 1] (high cost for turning right) and [1, 1, 10] (high cost for turning left). In all tests, you should return the optimal plan from the state [4,3,0] to the state [2,0,1]. While expanding states, you will have to maintain an open and closed list. Please use the data structures provided in the code.

Extra credits (3pts)

Consider implementing Dijkstra's algorithm for the above setup where the source is the [2,0,1] state. Since Dijkstra returns the minimal cost path between the source and all available states, you should be able to answer multiple queries from different target states to the source. As an example you can test your implementation to return the optimal path between [4,3,0] and [2,0,1] assuming a [1,1,10] cost vector.

There are different ways to implement Dijkstra. Given your A* implementation in the basic requirements, the simplest is to use a priority queue along with a closed list. You can either insert all nodes in the queue at the beginning and modify the values during expansion or insert new nodes and/or modify their values as you expand.

Submission

Submit the assignment using Canvas. You can work in pairs if you want to. If you do so, though, please let us know in advance and also attach a `Readme` file indicating the corresponding group members.

Help

If you get stuck, please do not hesitate to contact us for help, and stop by during the office hours. We also encourage you to post questions and initiate discussions on Canvas. Your colleagues are also there to help you.