Robotics: Capstone
Week 1: Path Planning

1 Introduction

This assignment is supposed to be relatively simple in order for you to get comfortable with python and be prepared for the later parts of the project. Your goal is to implement Dijkstra's algorithm on a graphs. You should notice that this is very similar to the assignment in the Computational Motion Planning Course. This will be useful for planning trajectories in later parts of the course.

Dijkstra's algorithm is a path-finding algorithm that is guaranteed to find the shortest path from a source node s to every other node in a graph. The A^* algorithm is similar to Dijkstra's, but uses a heuristic to make the search faster. If you choose to implement A^*, you may use the distance to the end goal as the heuristic. You will be implementing the algorithms in Python using numpy. The exact specification is given below and in the code.

For a review of these two algorithms, please see the Wikipedia article for Dijkstra's algorithm.

2 Occupancy Map

You will need to generate a graph from an occupancy map. You will be loading the occupancy map from the YAML file. Refer to the "Using the Simulator" document to see how exactly to load the YAML files. Once loaded, you will need to convert it from its original format, an `N` by `M` numpy array of 0s and 1s, to a graph format.

The locations of each cell in the occupancy map determine the metric location of that cell as expected. If the cell is at location `(i,j)` (zero indexed) then it should go to metric location `x = (j + 0.5)*x_spacing` and `y = (i + 0.5)*y_spacing`. In other words, the column number gives the x value and the row number gives the y value. Note that the intuition that the cells on top of the table have a higher y value is false, so you can flip this map accordingly.

3 Shortest Path Implementation

You need to the function in the file "ShortestPath.py". It will

```
def dijkstras(occupancy_map,x_spacing,y_spacing,start,goal):
    """
    Implements Dijkstra's shortest path algorithm
    Input:
    occupancy_map - an N by M numpy array of boolean values (represented
        as integers 0 and 1) that represents the locations of the obstacles
        in the world
    x_spacing - parameter representing spacing between adjacent columns
    y_spacing - parameter representing spacing between adjacent rows
    start - start node
    goal - goal node
    Output: a tuple (path, numopened)
            path: list of the indices of the nodes on the shortest path found
                starting with "start" and ending with "end"
    """
    # Your code here
    pass
```

We will proved a few sample graphs with solutions to test your implementation. It is encouraged that you generate your own graphs.

4 Grading

To test your implementation, we have provided two sample graphs with solutions in the file dijkstra_test_paths.txt that you can copy to the params.yaml file. There are also two sample graphs without solutions, and you will input the length of the shortest paths for these as your solution.