Shreya Gummadi

ENPM 661: Planning for Autonomous Robots

Project 2

Project Statement:

1. Generate a path from initial to goal position which avoids obstacles using:
    - Dijkstra algorithm.
    - Astar algorithm.
2. Show the nodes explores and the optimal path generated.
3. Take initial position, goal position, resolution, radius of the robot and clearance as user input.

Files:

There are two files for the project:

- Dijkstra_p2.py
- Astar_p2.py

Modules:

The following modules are needed to run the program:

- Numpy (to use the pi,sqrt,trig functions etc.)
- Matplotlib.pyplot (to plot)
- Matplotlib.path (to check existence of point in the polygon)
- Time (to get the run time of the particular algorithm)
- Ipython (to get the animation of the plots in an outside window)

Run:

To run the programs just hit run and provide the necessary user inputs.

User Inputs:

- The initial and goal positions can be gives as: x_co-ordinate,y_co-ordinate
  Ex: initial position: 5,3 goal position: 10,50
- Resolution, radius and clearance can be given as the digit.
  Ex: resolution: 1 clearance: 0 radius: 0
- The user can provide radius as ‘0’ for point robot. Whereas for a rigid robot the user can specify the radius of his choice.

Animation:

The animation makes the code slower as it has to plot everything multiple times. To view the animation for node exploration:

- Uncomment lines 125-133 in Dijkstra_p2.py
- Uncomment lines 133-140 in Astar_p2.py

Note: In case, from IPython import get_ipython, get_ipython().run_line_magic('matplotlib','qt') doesn’t plot the graph in a window instead of inline. Type %matplotlib qt in the ipython console.

Implementation:

Both the algorithm makes use of a class Node to store the position of the node, its corresponding cost and its parent id.

They then have 4 helper functions in common:

- **Movement()** : specifies the increment in x,y co-ordinates and the cost when we move the current node in the 8 connected space.
- **Exists()** :This function takes the node, resolution, clearance and radius as its parameters and determines if the node exists in the obstacle free space.
- **Generate_id()** : This function takes the node as its parameter and generates a unique id for it. The node is stored in the dictionary with this id as the key.
- **Plot()** : This function plots the map of the obstacle space using half planes and semi-algebraic model.

The Astar algorithm has an extra helper function called **Heuristic()** it takes two nodes as its parameters and gives the Euclidean distance between the two.

The two main functions are;

- **Dijkstra()** :It takes co-ordinates of the initial and goal position, resolution, clearance and the radius as its arguments. It returns the co-ordinates of the nodes in the optimal path. It first checks if the initial and goal node lie in the obstacle free space otherwise it tells the user that the following position is not possible. The algorithm works on expanding the node with the least cost and moving forward until we reach the goal node. If the current node has a lower cost but it is already been explored but its cost is higher then it is updated in the list.
- **Astar()** : This function takes the same arguments as Dijkstra and also returns the co-ordinates of the nodes in the optimal path. The only difference between the two is that Astar algorithm uses a heuristic cost to go. It takes the node with the minimum cost+heuristic_cost and expands that until we reach the goal. The heuristic cost in the program is the Euclidean distance between the current node and the goal node.