Motion Planning RBE 550

Project 2: Point Planning and Discrete Search

DUE: Thursday, September 21th at (8:30 am) before class starts.

All the written components should be typesetted with LaTeX. A template is provided on Overleaf. However, feel free to use your own format as long as it is in LaTeX. The written report should be submitted as a PDF file.

This assignment can be completed (optionally) in pairs. Present yours and your partner work only. In the written report include both your names in the title. You must *explain* all of your answers. Answers without explanation will be given no credit.

Submit two files **a)** your code as a zip file **b)** the written report as a pdf file. If you work in pairs, only **one** of you should submit the assignment, and write as a comment the name of your partner in Canvas.

Theoretical Questions (40 points)

1. **(5 points)** Write two key-differences between the Bug 1 and Bug 2 algorithms.
2. **(15 points)** A fundamental part of A* search is the use of a heuristic function to avoid exploring unnecessary edges. To guarantee that A* will find the optimal solution, the heuristic function must be admissible. A heuristic h is admissible if

$$h(n) \leq T(n)$$

   where T is the true cost from n to the goal, and n is a node in the graph. In other words an admissible heuristic never overestimates the true cost from the current state to the goal.

   A stronger property is consistency. A heuristic is consistent if for all consecutive states n, n'

$$h(n) \leq T(n, n') + h(n')$$

   where T is the true cost from node n to its adjacent node n' .

   Answer the following:

   (a) **(5 points)** Imagine that you are in the grid world and your agent can move up, down, left, right, and diagonally, and you are trying to reach the goal cell $f = (g_x, g_y)$. Define an admissible heuristic function h_a and a non-admissible heuristic h_b in terms of x, y, g_x, g_y. Explain why each heuristic is admissible/non-admissible.

   (b) **(10 points)** Let h_1 and h_2 be consistent heuristics. Define a new heuristic $h(n) = max(h_1(n), h_2(n))$. Prove that h is consistent.
3. **(20 points)** Suppose you are planning for a point robot in a 2D workspace with polygonal obstacles. The start and goal locations of the robot are given. The visibility graph is defined as follows:

- The start, goal, and all vertices of the polygonal obstacles compose the vertices of the graph.
- An edge exists between two vertices of the graph if the straight line segment connecting the vertices does not intersect any obstacle. The boundaries of the obstacles count as edges.

Answer the following:

(a) **(10 points)** Provide an upper-bound of the time it takes to construct the visibility graph in big-O notation. Give your answer in terms of n, the total number of vertices of the obstacles. Provide a short algorithm in pseudocode to explain your answer. Assume that computing the intersection of two line segments can be done in constant time.

(b) **(10 points)** Can you use the visibility graph to plan a path from the start to the goal? If so, explain how and provide or name an algorithm that could be used. Provide an upper-bound of the run-time of this algorithm in big-O notation in terms of n (the number of vertices in the visibility graph) and m (the number of edges of the visibility graph). If not, explain why.

Programming Component (60 points + 20 bonus)

In this assignment, you are going to implement BFS and DFS and A* algorithms with Python3. This template is provided to you as a starting point. After you finish coding, you would be able to create your own map to test different algorithms, and visualize the path found by them. Files included:

- **search.py** is the file where you will implement your algorithms. As we will use a grader code to help us grade your assignments more efficiently. Please do not change the existing functions names, or the docstring tests.
- **main.py** is the scrip that provides helper functions that load the map from csv files and visualize the map and path. You are not required to modify anything but you are encouraged to read and understand the code.
- **util.py** This file includes auxiliary structures that you could use to help you with your implementation. Using these structures is optional but potentially very helpfull. You are not required to modify anything but you are encouraged to read and understand the code.
- **maps/map.csv** is the map file you could modify to create your own map.
- **maps/test_map.csv** restores a test map for doc test purpose only. Do not modify this file.
- **maps/large_map.csv** This is a large map, that you should not modify.

Please finish reading all the instructions below before starting actual coding.

Installation Instructions (Optional)

This assignment was tested with

- `python 3.8.10`

- `matplotlib 3.4.2`
- `numpy 1.17.4`

Your default setup will probably work but if it does not, you can install these specific versions by creating a virtual environment with the following commands in the terminal:

```
# Creating the virtual environment
python3.8 -m venv venv search.py
# Activating the environment
source venv/bin/activate
# Installing the requirments
pip install -r requirements.txt
```

Now the virtual environment should be activated, and if you run any python commands it should be using the specified versions for the required packages. If you exit/change the terminal you should re-activate the environment.

Get Started

Before starting any coding, please run the code first:

```
python search.py
```

When running **search.py** as a main function, it will run a doc test for all the algorithms. It loads **test_map.csv** as the map for testing.

As you haven't written anything yet, you would see you fail all the doc tests. After implementing each algorithm, you should run this file again and make sure to pass the doc tests (you will see nothing if you pass the test).

But please note that, passing doc tests does not necessarily mean that your algorithm is done without any problems. It just shows that your algorithms are working in this simple **test_map.csv** with its given start and end position. (It may still fail, for example, if you change the goal to an obstacle.)

For visualization, please run:

```
 python main.py
```

There should be 3 maps shown representing the results of 3 algorithms. As said before, there would be no path shown in the graph as you haven't implemented anything yet. The **main.py** loads the map file **map.csv**, which you are free to modify to create your own map.

More details

- Please first read the algorithm description in search.py and make sure you understand the input/arguments and required output/return of the algorithms.

- Keep in mind that, the coordinate system used here is [row, col], which could be different from [x, y] in Cartesian coordinates.
- When you explore the nearby nodes in the map, please follow this order ”right, down, left, up“, which means “$[0,+1],[+1,0],[0,-1],[-1,0]$” in coordinates. There is nothing wrong using other exploring orders. It is just that the test result was gotten by algorithms using this order. A different order may result in different results, which could let you fail the test and the grader code.
- For A* you should use the Manhattan Distance as a heuristic.
- Also, for the output of the function, path should include the first / start position and the goal / end position. Step is the number of visited nodes. Until now, I hope you have a basic understanding of the template code and the requirements. After you go through the Rubrics below, you may start coding!
- In case there is no path your algorithm should return an empty list ([])

Grading

1. **(10 points)** A clear structure without duplicating your code. Please figure out the similarity of these three algorithms (hint: Lavalle, Chapter 2). As they are somewhat similar, if you implement them separately, you would end up writing a lot of duplicated codes or copying codes here and there. First, it's not a good practice to do so. It also does not reflect your deeper understanding of these algorithms. Please find out the similarity and construct your code in a way that you would not duplicate the codes. You are not required to write anything for this part.
2. **(10 points)** Your algorithms pass the basic doc test. After implementation, as mentioned before, you could run the doc test by: python search.py If you see nothing, it means you pass the simple doc test. You are not required to write anything for this part.
3. **(40 points)** Your algorithms pass all the tests for grading. We will use a grader code with other test cases for grading. Please think of different possible test cases yourself, modify test.csv to create your own map and check the result. It's a good practice to think of boundary cases, a map that does not have a valid path, for instance. Also use the main.py script to plot the results of **large_map.csv** and include them in your report.
4. **(20 bonus points)**. Create a better heuristic for A* that outperforms the Manhattan distance for a specific map. You can choose one of the existing maps or create your own by modifying **map.csv** . Your heuristic must still be admissible, for all maps but should outperform the Manhattan distance in your map choice. Include in your report the map with the solution found by your heuristic, the Manhattan Distance heuristic, and the number of steps required by each heuristic. Also explain why your heuristic is still admissible, and the reasoning behind your choice.