MAE593B– SPTP: Robot Motion Planning

Term: Fall 2024

Instructor: Guilherme Pereira

Homework Assignment II

I – Instructions

- **The assignment is individual**.
- Students are allowed to discuss their solutions with their classmates.
- If any external source is used to execute the assignment, the source must be cited. Examples of external sources are scientific papers, books, friends, classmates, parents, ChatGPT, Wikipedia and other internet pages – Anything, anywhere! You will not be penalized if you use external help but this must be correctly acknowledged.
- Using the work, or part of the work, of others as yours is plagiarism! Please, do not share your code with your classmates!
- **It is better do try to solve all the problems by yourself!**
- Doubts and discussions should be preferable made using Slack. Please, never send your code in Slack! Small parts of the code are allowed. Please, try to solve your classmates' doubts.
- Your assignment must be submitted using eCampus. Please, attach **a single zip file** with all your files**.** The file name must be **YourFirstName_YourLastName.zip. Please, use zip files only!** No tar, gz, rar and other formats, please.
- The assignment is due **Wednesday, September 25, 2024**.

II – Objectives

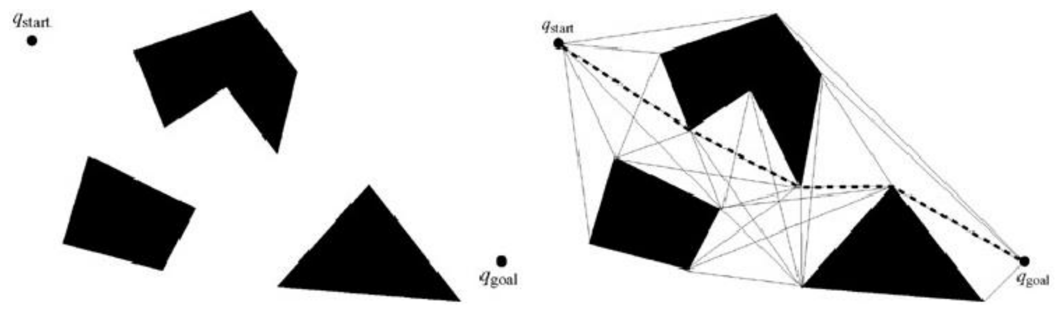
This assignment has the following objectives:

1. Practice graph implementation;
2. Implement graph search algorithms;
3. Create basic graph search functions for future homework;
4. Implement a simple motion planner.

III – Introduction

In this homework we will practice graph search algorithms. We will search for paths on a visibility graph represented with the help of the PGraph implementation in Peter's Corke Robotics toolbox. You can use the files on eCampus or the toolbox from this link: https://petercorke.com/toolboxes/robotics-toolbox/

A visibility graph is a graph of intervisible locations for a set of points and obstacles in the Euclidean plane. Each node in the graph represents a point location, and each edge represents a visible connection between them. That is, if the line segment connecting two locations does not pass through any obstacle, an edge is drawn between them in the graph. For motion planning, a visibility graph is constructed with vertices of the obstacles, the initial and the final positions. The next figure shows a typical visibility graph for a polygonal environment:



IV – What must be done?

a) Using the environment framework development in HW1, write a Matlab function that creates a Visibility Graph. Use the class PGraph to store your visibility graph. Please, refer to Appendix J of Corke's book for examples of the class. Hint: During the construction of your graph, you will need to write a function that determines if an edge belongs or not to the free configuration space. This can be done by discretizing the edge and testing each of the discrete points for collision. See an example on how to discretize the edge here:

   https://www.mathworks.com/matlabcentral/answers/121488-function-to-discretize-a-line-between-two-points

b) Create a BFS function (in class, we did not have a function – need to incapsulate the given code in a function) and find a path from a start position to the goal position using your function. Your function must implement the algorithm shown in class that includes the stopping criteria (stops when the goal is found) – this was not implemented in the given code. The user should use the mouse to enter start and goal positions.

c) Create a Dijkstra function and find the shortest path from a start position to the goal position, both entered using the mouse. Your function must implement the algorithm shown in class. Hint: To implement the priority Queue you can use an array or matrix (Remember that this is not the best choice!). Every time a node enters the PQ you must sort the matrix. Ex: Suppose your PQ is a n x 2 matrix where the first column is the node number and the second is the cost. You can use [B,I]=sort(PQ(:,2)) to sort by the second column and use indexes I to reorder the entire PQ as PQ=PQ(I,:). Hint: the PGraph representation has a field "data" that can be used to store information in each node. This can be used, for example, to store the back-pointer for the node and the cost, if you think this is necessary. Use G.setvdata(1, 2); to add data 2 to node 1. Instead of a number, the data can be an array or even an object. For example, G.setvdata(1, [12.3 16.4 30]); saves the array [12.3 16.4 30] to node 1. To recover the information of node 1, use G.vdata(1); Alternatively, you can create a separate array to store your data.

   [*Optional: extra 15% in the final grade*] Implement a Heap-based PQ instead.

d) Create an A* function and find the shortest path from a start position to the goal position. Your function must implement the algorithm shown in class. You can't use the A* implementation provided by PGraph as your own solution. Hint: Use the Euclidean distance as heuristics.

e) Compare and discuss the results (path length and time of computation) of the different methods for at least two different environments (a dense and a less dense environment). Include the A* algorithm provided by PGraph in your comparison. The comparison should show (in the same plot) the paths found by each method (BFS, Dijkstra, A* implemented by you, A* from PGraph) and a table that contains the path length and the computation time for the methods tested. A written discussion about the results is necessary. One table and one plot should be shown for each environment tested.

V – What must be submitted?

1. A single main Matlab script (along with other necessary files) that shows the answers for items

(a) to (d). Everything that was not specified in the assignment may be chosen by the students.

2. A pdf report (word documents are not allowed) with up to five pages containing explanations about the assignment (files, scripts, functions, etc), figures that exemplify the work, the response for item (e) in form of tables, plots and a written discussion, and references, if necessary.

   OBS: Remember that all the files must be in a single ZIP file.