University of Girona

Autonomous Robots

Lab 4

Topological Maps – Visibility Graph

Author:
Di MENG

Supervisor:
Dr. Marc Carreras

April 30, 2017

1 Objective

The objective of this lab is to grab the knowledge of Rotational Plane Sweep(RPS) algorithm for generating a visible graph for robot path planning and implement it in Matlab. With the start position, goal position and several obstacles represented by polygons, the visible graph should be generated which shows the available path from the start to goal. Also the path planning for topological Maps will be completed along with the A* algorithm which will be implemented in the next lab for choosing a shortest path.

2 Introduction

The topological map is a simplified map only containing the relationship of points. It can be represented by a graph where nodes are real locations and the edges join the positions in free space. And the edges are including distances. The visible graph can be used to build up a topological map. It is a graph of intervisible locations, typically 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 points. An edge can be drawn if the line segment connecting two locations does not pass through any obstacles. When the nodes are lying in one line, that means they are an ordered series which has the same sense as available path for robot planning.

In order to generate the visible graph, the RPS algorithm is used. The basic idea of RPS algorithm is that for every points, checking if it is visible from itself to all the rest points in the graph. A rotational half-line emanating from the current point is used to check if there's interrupt between two points. When the pair points are considered visible, the edges between are drawn in the visible graph representing the connection is in free space.

3 Implementation of RPS algorithm

The RPS algorithm is implemented in the form of a function named "RPS". The input of the function is the vertices representing the environment and the output is a set of edges which are the available path in the given environment.

function [edges]=RPS(vertices)

3.1 Environment configuration

The environment is including the start point, the goal points and the obstacles represented by polygons, so the polygons has to be defined to configure the environment. As the polygons are defined by vertices in programming, a set of vertices is defined with brute force. It is an nx3 dimensional matrix which every row represents a vertex and the first and the end row are always the start and goal points, the three columns are x, y coordinates and the ascending numbers of polygons(starting from 0).

3.2 Extracting the edges of polygons

With the given vertices, edges of polygons are extracted for the subsequent convenience. It is saved in an nx2 matrix which every row represents an edge and two columns represent the ending points of the edge respectively. Therefor, the vertices and the corresponding edges are linked to each other.

3.3 Creating a list of sorted angles

The rotational half-line has to stop only in the directions where there's a vertex to check if the vertex is visible. So the angles between every line segment which is built up by two vertices and horizontal lines are computed and sorted in an ascending order. Also the indices of angles are matched to the indices of vertices.

3.4 For every given vertex

3.4.1 Initializing the S list

The list S is an active list containing edges which have intersects with horizontal line at the beginning and will be updated as the half-line reaching directions of vertices.

Basically, a horizontal line is defined by two points, one is the current vertex in the loop and the other one is far away. All the edges are checked if there's an intersect with this horizontal line, if yes, the edge is saved in the S list. Also, if the vertex is on the polygon, the connecting edges of the vertex are not in the initial S list.

3.4.2 Check the visibility of the rest vertices with respect to the current one

Going through all the rest vertices in the angle-sorted order. There are two conditions when the vertex is considered to be visible with respect to the current vertex.

1. The current S list is empty.
2. The S list is not empty. The line segment from current vertex to the checking vertex has intersects with the edges in the S list. The intersects are the two vertices which are the ending points of the line segment.

If the vertex is considered visible, the vector containing the current vertex and the visible vertex which is actually the free-connecting edge is added to the visible graph list.

3.4.3 Update the S list

Every time the half-line stops in a vertex, the connecting edges of the vertex are taken into account. If the edge is in the S list, it is removed. Otherwise, it is added.

3.5 Post-processing of the visible edges list

After the above processes, a list of visible edges is built. The repeating edges in the list which are represented in different order of vertices have to be removed. Also, some edges which are inside the polygons and are not visible indeed should be removed as well.

For checking if the edges are inside the polygons, the method used in the code is that extracting the center point of the edge and checking if this center point is inside the polygon, if yes, it means this edge is inside the polygon and will be deleted.

At the end, the result from the post processing is saved in the "edges" list and exported.

4 Results from different environments

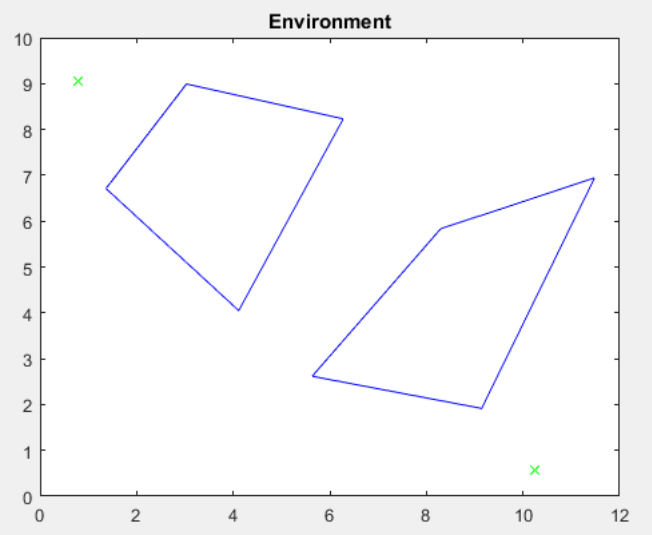
4.1 Small environment

Firstly the algorithm is tested in a relatively simple map which has ten vertices and two polygons. The input vertices and the environment are shown below(Figure 1). The green points in the (b) are the start and goal points, the blue polygons represent obstacles.

10x3 double

	1	2	3
1	0.7807	9.0497	0
2	3.0322	8.9912	1
3	1.3655	6.7105	1
4	4.1140	4.0497	1
5	6.2778	8.2310	1
6	8.2953	5.8333	2
7	5.6345	2.6170	2
8	9.1433	1.9152	2
9	11.4825	6.9444	2
10	10.2544	0.5702	3

(a) The input vertices



(b) The environment

Figure 1: The given environment

After applying the RPS function, the resultant edges and visible graph are showing as follows:

18x2 double

	1	2
1	1	3
2	1	2
3	2	3
4	2	5
5	3	4
6	4	6
7	4	5
8	4	7
9	5	7
10	5	6
11	5	9
12	6	9
13	6	7
14	7	10
15	7	8
16	8	9
17	8	10
18	9	10

Figure 2: The output edges

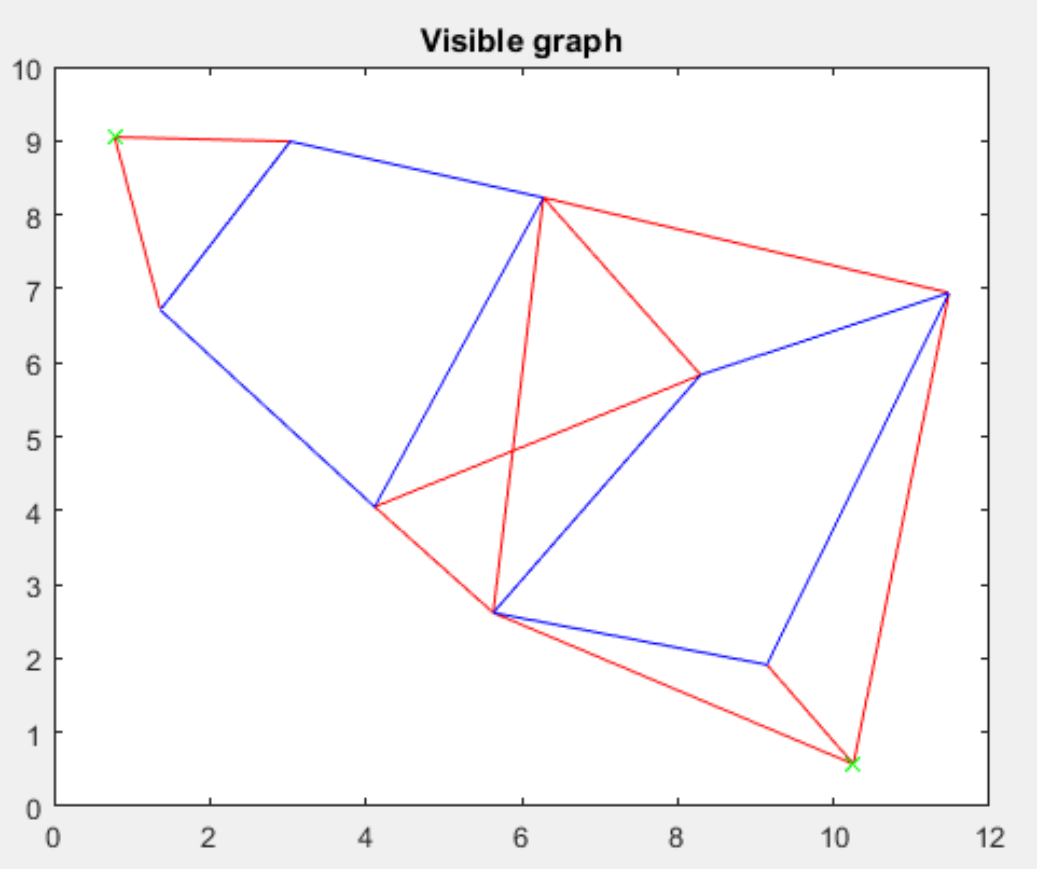


Figure 3: The visible graph

As we can see from the figure 3, the red lines are the edges which are the visible connections between vertices. The edges of polygons are also visible connections, but for better visual representation they are shown in blue. The visible graph shows many possible paths from start position to the goal position. The elapsed time is 0.528531 seconds(i5 core).

4.2 Convex polygon

For the robustness of the implementation, the function was also tested in the situation when the polygon is convex. The figure below shows proper result so that this algorithm works for multiple types of polygons.

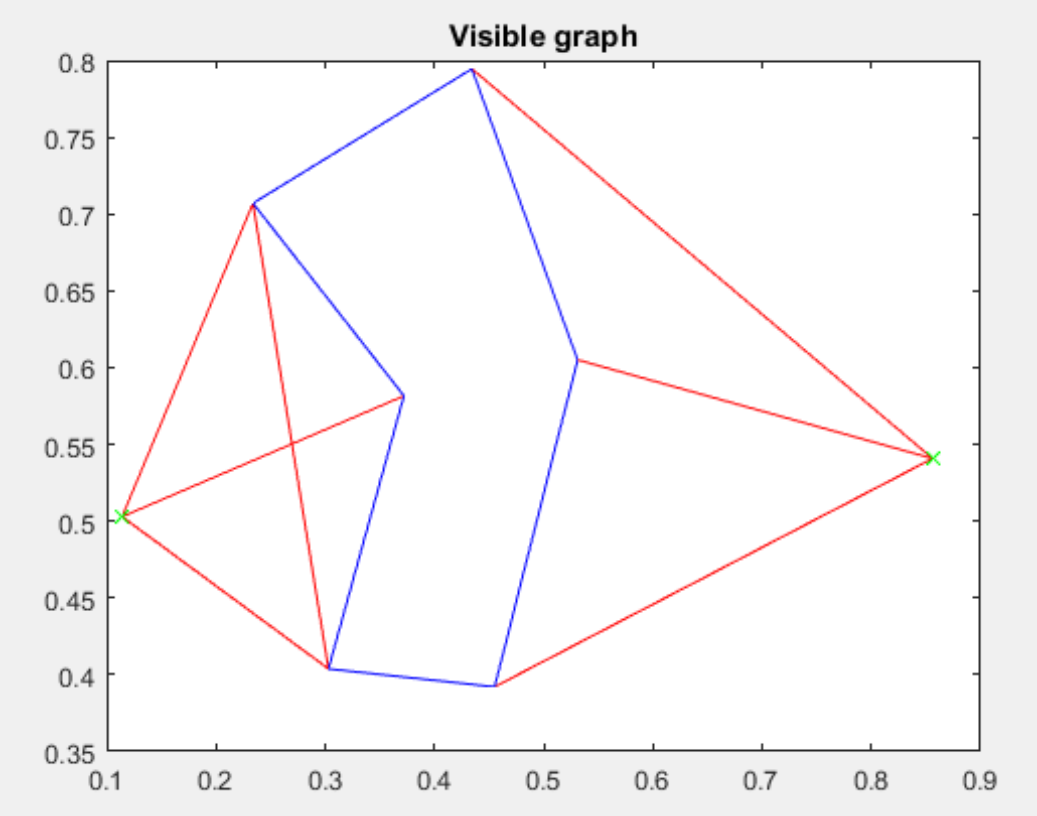
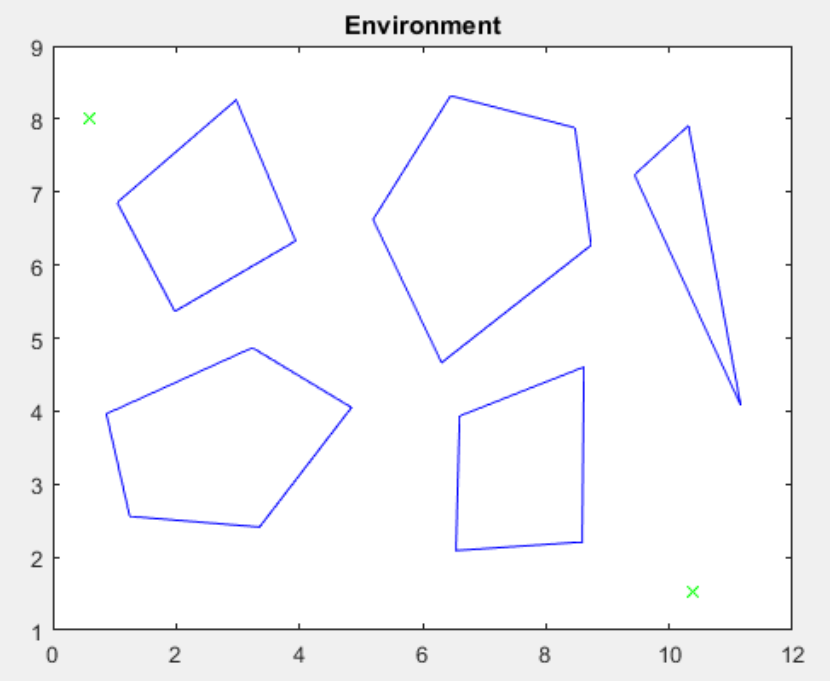


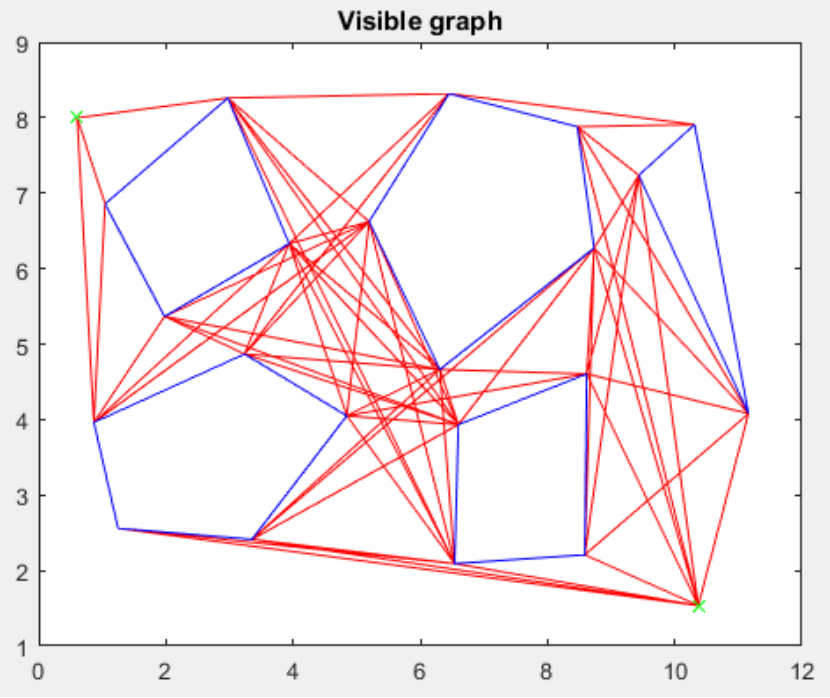
Figure 4: The visible graph of convex polygon

4.3 Big environment

The RPS function has also been tested in a relatively complicated map which has 23 vertices and 5 polygons. The output edges are 85 which are represented in red. The blue lines are edges of polygons also the visible edges. The elapsed time is 1.122202 seconds(i5 core).



(a) The environment



(b) The visible graph

Figure 5: Relatively complicated map

5 Conclusion

In this lab assignment, the visible graph was obtained by Rotational Plane Sweep algorithm. This algorithm was implemented in Matlab and tested in different environments. The results showed proper performance and time consumption.

The RPS algorithm is kind of different with the previous ones like wavefront and RRT algorithms. The given environment is a set of vertices of polygons which are actually the obstacles' bounding boxes. It can be used after the detection of the obstacles in the image from top view other than needing the full matrix of the map. This algorithm can give better performance for some specific applications.