# Autonomous Robots Lab Report 2
# Topological Maps - VISIBILITY GRAPHS

Raabid Hussain

April 3, 2016

## 1 Objective

The objective of the lab work was to get familiarized with topological maps used in mobile robotics. A basic implementation of visibility graphs was done using MATLAB. The algorithm requires vertices of all the obstacles in the environment along with the start and goal position of the robot in order to build a visibility graph that can later be used to find the optimal path. The algorithm is designed as such that it takes into account all the possible solutions.

## 2 Introduction

Robot mapping has always been an active research area in mobile robotics. Mapping involves obtaining features from the environment that are later used in path planning. After all the features have been extracted from the environment, it is customary in most path planning algorithms to build a topological map. This map generally represents all the possible paths that the robot can follow. One of these topological maps is known as Visibility Graphs.

Visibility graph algorithm requires all the vertices of the obstacles in the environment to be known before-hand. This means that all the obstacles have to be pre-processed to be approximated as polygon with prominent vertices. Visibiltiy graph can be seen as a simplified version of the topological maps with only relationships between key features in the map.

The rotational plane sweep algorithm starts with the starting position and decides which nodes or vertices in the topological map are visible from that location. It connects all those nodes with the current node before proceeding onto the next node and repeating the same process for all the nodes. Through this process a map of all the visible nodes from each node is built that can be used to perform path planning procedures as it contains all the possible solutions for navigation.

## 3 Implementation of Rotational Plane Sweep Algorithm

We were to implement the rotational plane sweep algorithm in MATLAB to extract a visibility graph of the map. A set of vertices or nodes in the environment were provided along with the obstacle number each node belonged to. The list of nodes also contained the starting position (as the first vertex) and the goal position (as the last vertex).

The first step in the RPS algorithm was to find out the edges of the obstacles. A list of all the edges was formed that contained the vertex number of both its end points. Only after this it became convenient to go through all the vertices and check the visibility for each node.

### 3.1 Sorting the Nodes

The first step in the RPS algorithm is to sort all the vertices in the map in terms of their angle with the current vertex. Angles for all the nodes is checked and saved in a list. The list is synchronized with the vertex numbers. Since the algorithm works only if the angles list is sorted in ascending order, we then sorted the list and saved only the vertex number as the angles are not required for further processing: we just need to know which vertex should next be processed.

The next thing is to initialize the ’S’ list which contains the edge number that the node line intersect. In order to initialize this list, a horizontal line is stretched from the current node to the right side and all the obstacle edges that intersect it are entered into the S list. The list is then sorted in terms of distance of the edges from the current vertex or node.

## 3.2 Checking Visibility

After initialization of the S list, we create another list that contains all the vertices that are visible to the vertex. This list is empty at the start. We then check the vertex with the least angle with the current vertex and check all the edges that intersect or obstruct the path. All the edges that intersect are added to the S list. For an infinite line, the equation of the line was found in homogeneous coordinates i.e 'ax+by+c=0'. The S list is arranged in terms of distance of the edge to the vertex. If the S list is empty and the edges that form an intersection with the line by the two vertices being considered are components of the same vertices, then the vertex is added to the vertex list. This list is called edges in the MATLAB code.

After this, the next vertex with the least angle is considered. Again all the edges are checked for intersection and added to the S list. If any of the edges is in the S list and it is found that the new line is leaving that edges, then it is removed from the S list. When the S list is not empty, after an edge is removed from the S list, then the first element of the S list is considered that if it was the one being removed then the vertex is also classified as visible and added to the 'edges' list.

The above process is repeated for all the vertices according to their angle with the current vertex and the S and edges lists are updated accordingly. After all the vertices have been considered the vertices in the edges list are kept while the S list is emptied for the next vertex. Also the edges list is transformed to a 2D array with the first element being the number of the current vertex and the second element its corresponding visible vertex. The process ends when all the vertices have been considered and all their corresponding visible vertices are found.

## 3.3 Updating the 'edges' list

As in the above process, if the vertices in the edges list belong to the same polygon/obstacle, they are not considered, but since they are part of the visibility graph, the edges of the obstacles are added to the edges list at the end of the algorithm. The next thing was to obtain a unique edges list. Since all the vertices are taken into account separately, it was observed that the edges list contained the same edges twice with opposite start and end positions. So, an additional check was done to remove any similar edges. The edge was also sorted in order for better visualization of the output. This edges list is output by the function. A diagram of the visibility graph is also done at the end of the function. Sample graphs are shown in the later sections. The obstacles are represented in blue, the visibility graphs in red and the start and goal positions in black and green stars.

## 4 Sample Results

This section contains few sample tests that were carried out using the algorithm.

### 4.1 Testing on Small Environments

Initially, the code was being tested on small environments in order to clearly apprehend what was going on. The following environment was used:-

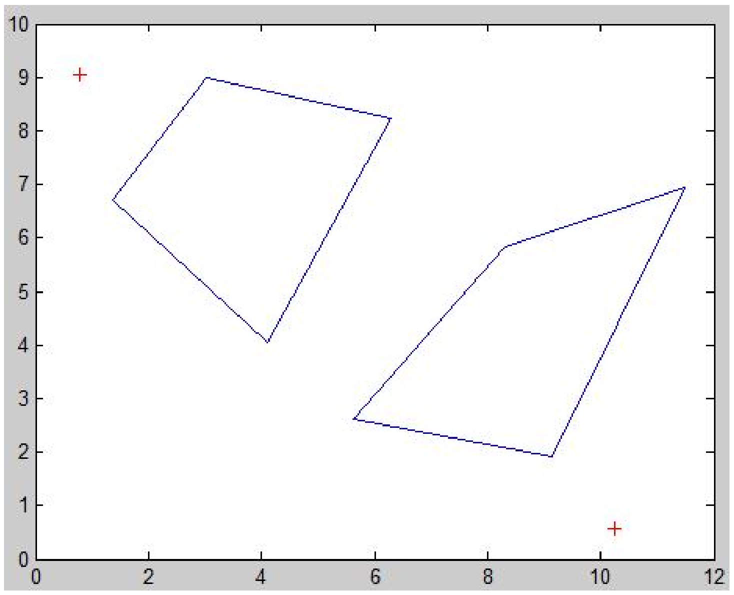

Figure 1: Samll environment for testing the algorithm

The vertices of the above environment were input to the algorithm function and the following output was obtained:-

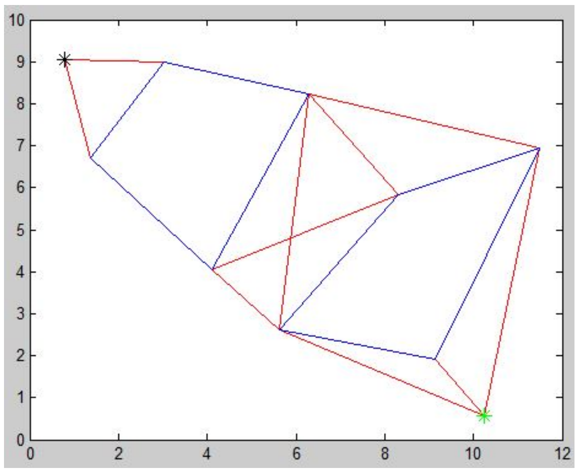

Figure 2: Visibility graph obtained from RPS algorithm for small environments

## 4.2 Testing on Large Environments

It was necessary to check whether the algorithm works on larger environments or not. So a sample test was run and the output is shown in figure 3.

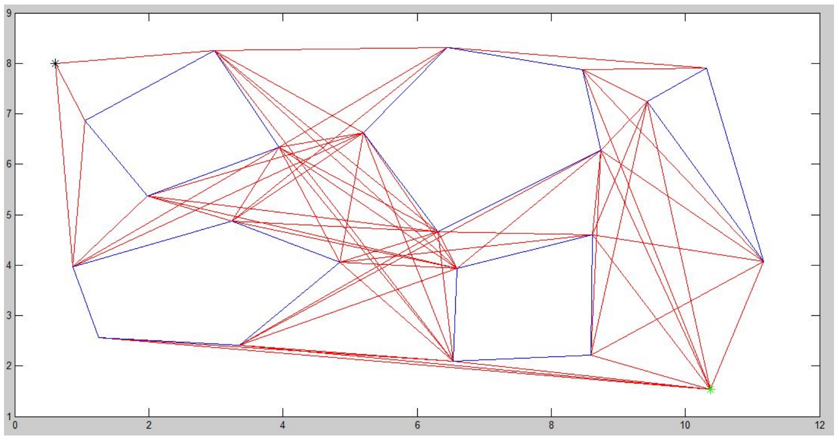

Figure 3: Visibility graph obtained from RPS algorithm for large environments

## 4.3 Dealing with Concave Polygons

The most grueling task of this lab was to enable the code to deal with concave obstacles. However, when the initial version of the code was run, the following output was obtained as shown in figure 4.

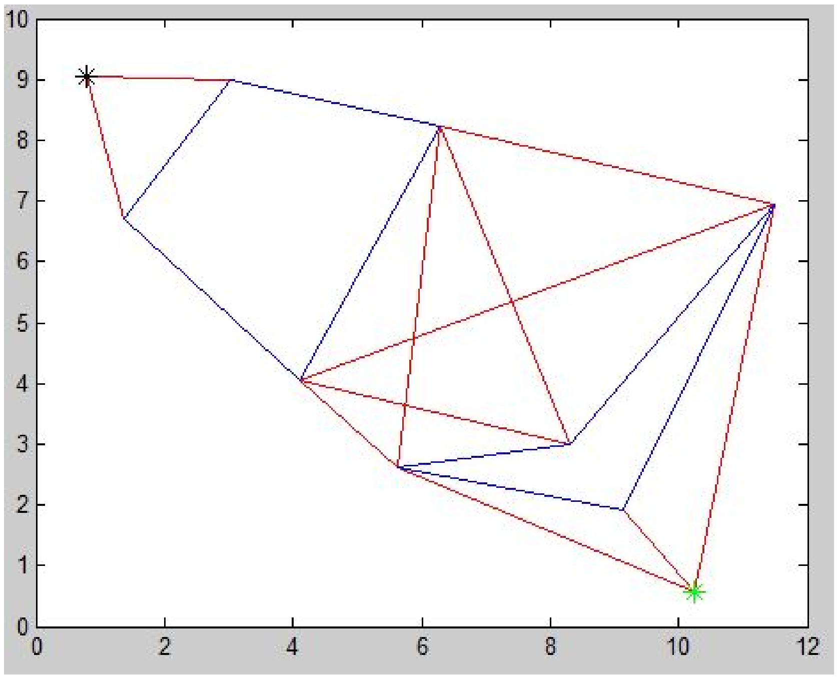

Figure 4: Initial visibility graph obtained from RPS algorithm for concave obstacles

However, this is not the right output. The vertices in the concave area are visible to each other but are shown to be hidden. So, the check at the end of the algorithm that deals with the vertices being of the same polygon was updated. A line was formed between the two vertices being considered and stored in terms of parametric form. Then sample points were taken at an interval of 0.1 distance on that line. All these points were checked whether they were inside the polygon or not. If they were outside, then they were not ignored. The result of the updated algorithm is shown in figure 5.

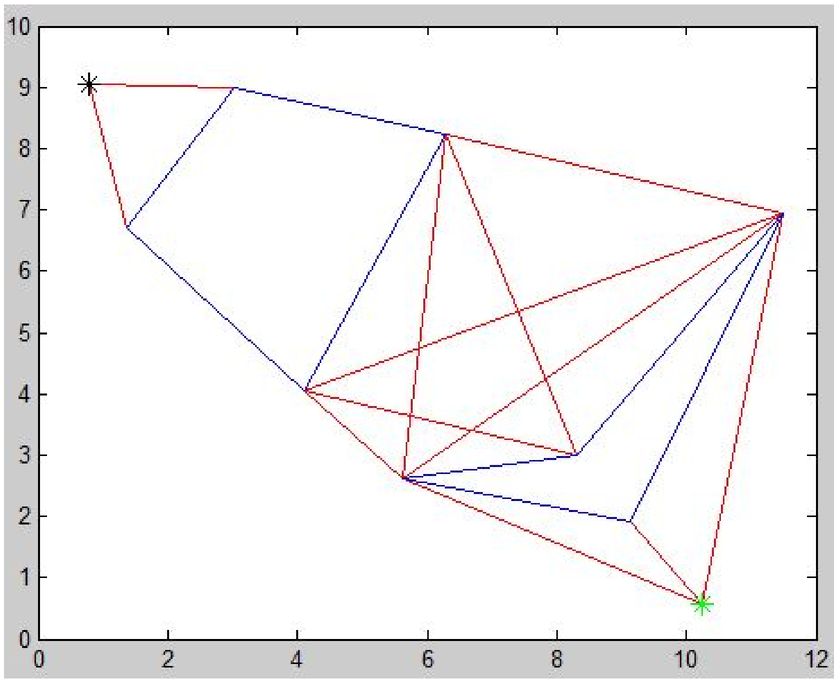

Figure 5: Visibility graph obtained from RPS algorithm for concave obstacles

## 5 Conclusion

In this lab assignment, visibility graph was obtained using the rotational plane sweep algorithm. The task was accomplished using MATLAB. Lecture notes and sample codes on internet were consulted to clear any confusion. The algorithm was tested on both small and large environments along with different forms of polygons. The algorithm produces correct results for all the test environments used.