Practical Exercise Visibility Graph

Goals:

In this practical exercise you should learn one of the motion planning algorithms, which relies on a roadmap as a subspace of the free space, the visibility graph.

Please login into the system with your student account.

1.Introduction

Building up on the previous approaches in robot motion planning, this implementation investigates the visibility graph algorithm. So similar to bug algorithms, the world that our robot lives in is a 2D environment with polygon based obstacles, in our case even rectangles. These obstacles in general can be convex or concave and the robot can be point or has a shape instead of a point. Considering this robot volume, visibility graph approach focuses on an important space transformation where workspace obstacles are expanded into configuration space obstacles. As its name suggests visibility graph approach forms a configuration space and finds visible node connections in this space. After finding the connections between obstacle vertices, the approach lets the robot to traverse a graph based search algorithm such as Dijkstra or A* search.

2. Visibility Graph Approach

As defined briefly above, visibility graph approach is applied in a 2D world with polygon shaped obstacles which aims to find the shortest path between the start and goal points by moving a point robot or polygon shaped robot. If the robot is considered to be a point moving translational, the workspace obstacles were the same as configuration space obstacles. If the robot has a shape, the configuration space obstacles are expanded from workspace obstacles because of the polygon shape of the robot. Another important point is the shape of the obstacles. Normally the visibility graph approach is limited to handle convex polygons as obstacles. However, by the help of triangulation methods, concave obstacles could also be used. *Delaunay Triangulation*, which will be explained in the algorithm section, will take concave polygons and convert them into many small triangles which are all convex polygons by themselves. The general method of converting workspace obstacles into configuration space obstacles is performed by the *Star Algorithm* which will also be explained in the next section.

The idea of forming a configuration space is leading the approach to create a graph by using the nodes, i.e. vertices and edges of configuration space obstacles, in the 2D world. Having formed a configuration space, the shape of the robot is then being included in the obstacles. This lets the algorithm accept the robot as a point from then on. At this point of the algorithm, all we have is vertices of obstacles, the start and goal points and edges of obstacles. Having only these elements, a graph can

easily be constructed by connecting the mutually visible elements. The name of the approach comes from this construction, where visible nodes are connected to create graph lines.

After creating a graph any graph based search algorithm becomes suitable for the aim of the approach. But before starting any search algorithm, the graph is required to be revisited. The result of visibility graph includes many unnecessary edges that will increase the search time while performing a graph based search algorithm. Therefore, reducing the graph nodes becomes an important operation in order to increase efficiency of the whole approach. The reference book [1] suggests an algorithm called *Rotational Plane Sweep Algorithm* and the idea of *Separating and Supporting Lines* in order to cope with the reduced visibility graph problem. For more information about these approaches the reader may consider revising the relevant chapters from the reference book [1] or the lecture slides.

The last step in the whole visibility graph approach is to apply a graph based search algorithm. Because of its completeness and efficiency *A* Search Algorithm* is preferred for this part. Given a graph as input, this algorithm produces a back track from the goal towards the starting point following each nodes visited. This path is generally unique and accomplishes the shortest path possible concerning the whole node probabilities. The details about the A* search algorithm are given in the next section.

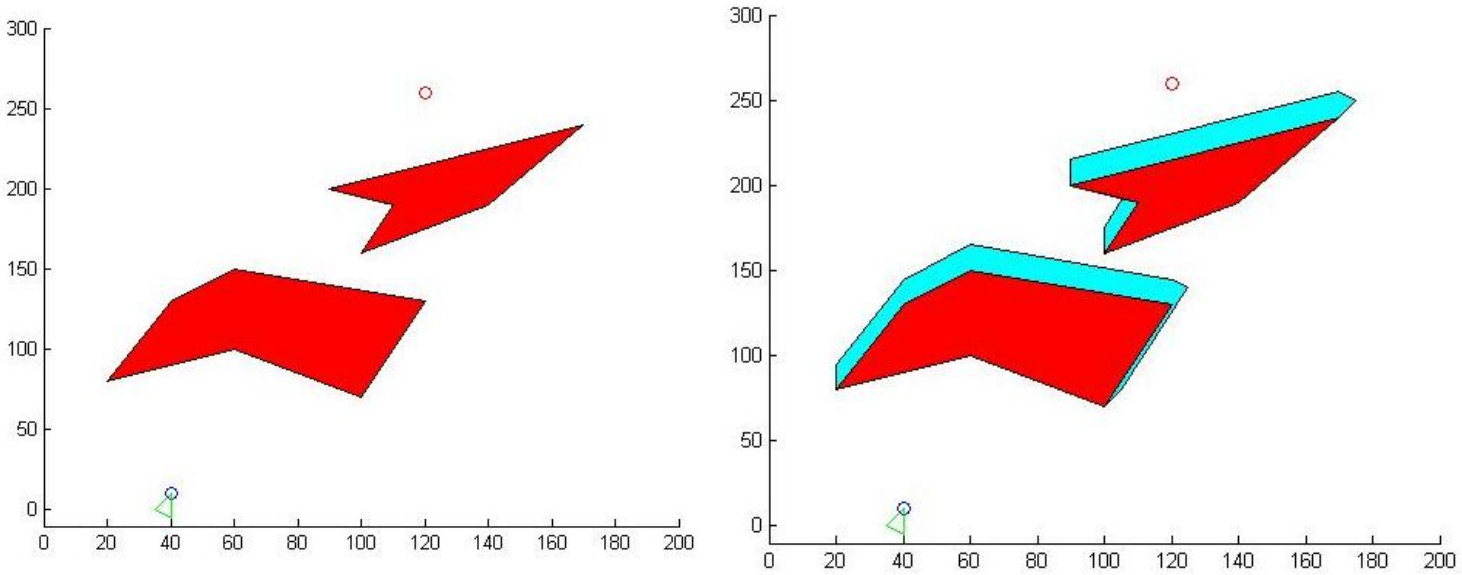

Figure 1 - Figure on the left shows an example 2D world configuration with 2 concave obstacles shown in red polygons. The robot is shown with a green triangle whose vertex is located on the starting point shown as a circle. The goal point is the other circle on top. The figure on the right shows the resulting configuration space formed concerning the workspace obstacles and the robot. This result of star algorithm is shown with cyan expansions to the workspace obstacles.

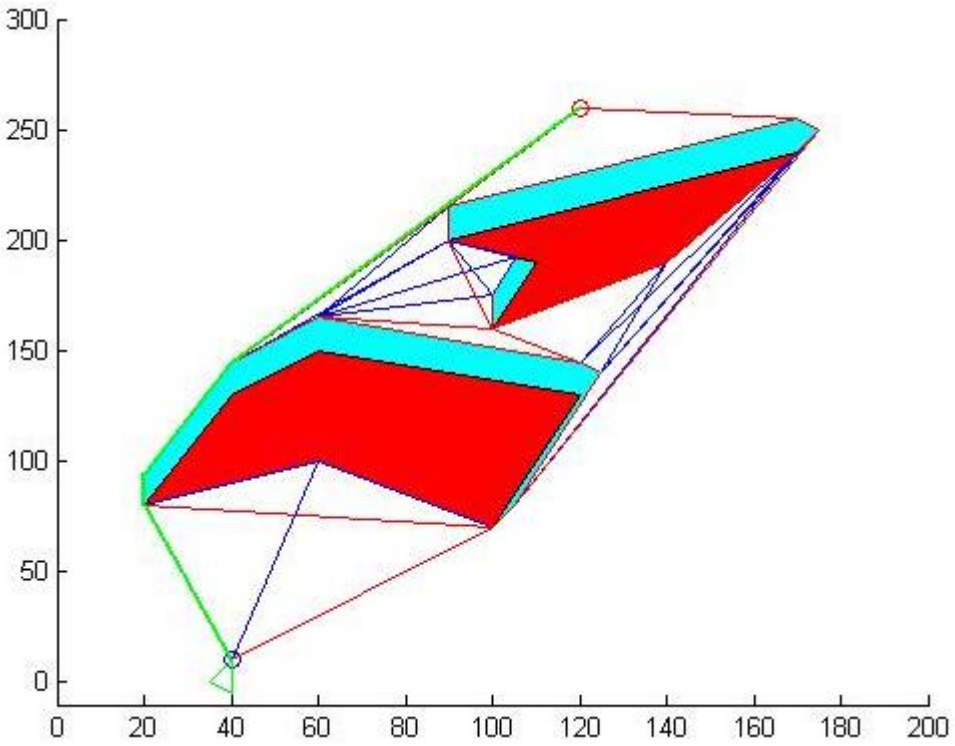

Figure 2 - The resulting graph nodes and shortest path. The blue lines represent the result of visibility graph, where brown lines show the reduced graph. The thick green line is the result of A* algorithm and shows the shortest path.

3. Algorithm

In this part, the algorithms that build up the visibility graph approach will be explained in required detail.

3.1 Star Algorithm

The star algorithm is based on finding the intersection points of two convex polygons in a 2D world.

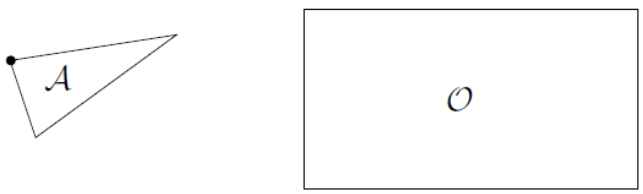

Figure 3 - A triangular robot and a rectangular obstacle.

The method is based on sorting normals to the edges of the polygons on the basis of angles. The key observation is that every edge of QO is a translated edge from either R or O. In fact, every edge from O and A is used exactly once in the construction of QO The only problem is to determine the ordering of these edges of QO. Let $\alpha_1, \alpha_2, \ldots, \alpha_n$ denote the angles of the inward edge normals

in counterclockwise order around A. Let $\beta_1, \beta_2, \ldots, \beta_n$ denote the outward edge normals to O. After sorting both sets of angles in circular order around S1, QO can be constructed incrementally by using the edges that correspond to the sorted normals, in the order in which they are encountered.

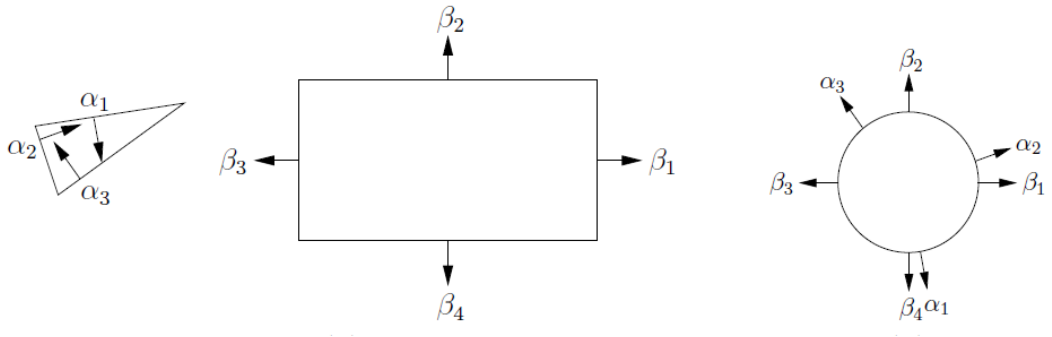

Figure 4 - (a) Take the inward edge normals of A and the outward edge normal of O. (b) Sort the edge normals around S1. This gives the order of edges in QO.

To gain an understanding of the method, consider the case of a triangular robot and a rectangular obstacle, as shown in Figure 3. The black dot on A denotes the origin of its body frame. Consider sliding the robot around the obstacle in such a way that they are always in contact, as shown in Figure 5a. This corresponds to the traversal of all of the configurations at the boundary of QO. The origin of A traces out the edges of QO, as shown in Figure 5b. There are seven edges, and each edge corresponds to either an edge of A or an edge of O. The directions of the normals are defined as shown in Figure 5a. When sorted as shown in Figure 5b, the edges of QO can be incrementally constructed.

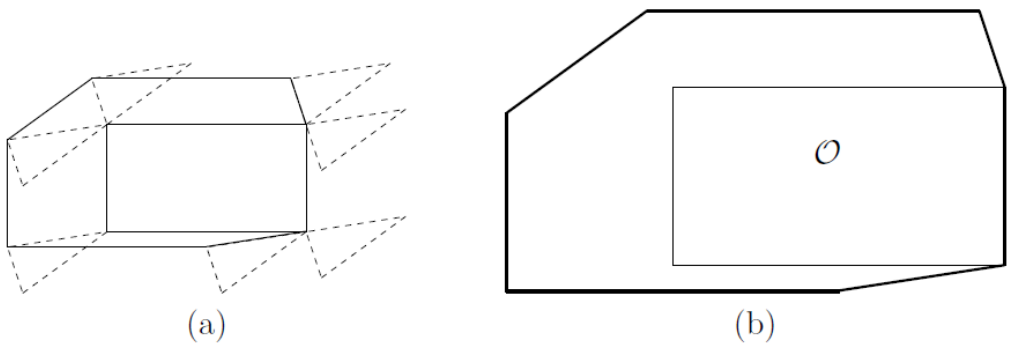

Figure 5 - (a) Slide the robot around the obstacle while keeping them both in contact. (b) The edges traced out by the origin of A form QO.

However, there is one drawback of the star algorithm; it works only

on convex polygons. In order to improve the visibility graph approach, concave obstacles should be included. In order to accomplish this, another important process is required. This process is called Delaunay Triangulation which aims to create smaller triangles out of convex and concave polygons.

3.1.1. Delaunay Triangulation

Basically the Delaunay triangulation is a process where for a given polygon the minimum number of triangles that make up this polygon are found. The idea is well suited to our problem because by this method concave obstacles can be turned into triangles, which are all convex shapes, and star algorithm can be applied. This is a very useful and commonly used method used in computational geometry.

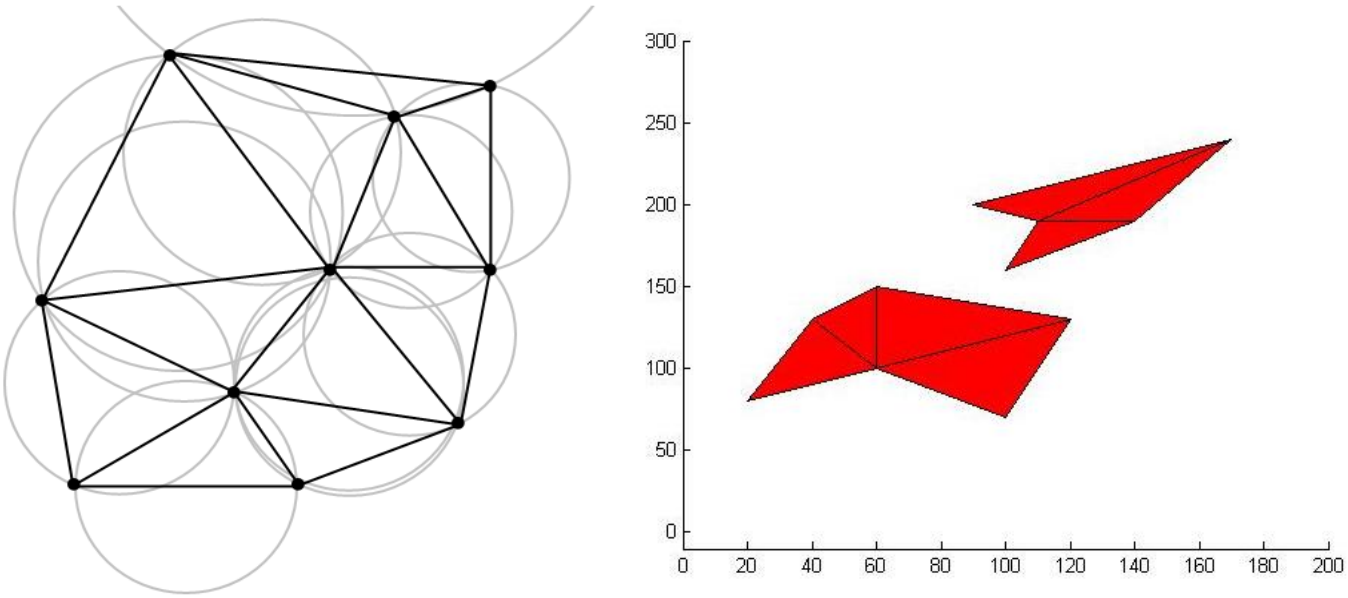

Figure 6 - The figure on the left adapted from Wikipedia, shows the idea behind triangulation of a polygon. The aim here is to find the minimum number of triangles with each having possibly large angles. The figure on the right is the triangulation results of the above example.

Delaunay triangulation can be done before applying the star algorithm to handle concave objects. The triangles found by this method is then given to star algorithm as unique convex obstacles. Subsequently these obstacles are transformed into configuration space obstacles. Any configuration space obstacle which has a connection with another obstacle is converted into a single obstacle.

3.2. Complexity constructing the Visibility Graph

A brute force method to construct the visibility graph has the complexity $O(n^3)$. Construction of visibility graph is also explained in the lecture with the rotational sweep line algorithm (see also reference book [1]). The main idea of this algorithm is creating a virtual axis for each vertex in the configuration space, including the start

and goal points, and create lines emanating from this vertex connecting to the other vertices in the space (see lecture).

For a given visibility graph, supporting and separating lines can be used for reduction. The main idea of these lines is to find tangent lines between and around the obstacles. Any lines that fall apart from these lines are eliminated. These lines actually represent the shortest path from one obstacle vertex to the other obstacle. One idea is to increase the lengths of the connecting lines found in the visibility graph. If an increased line still has 2 intersection points, these lines are suitable for reduced visibility graph. Other lines which fail to have only 2 intersection points are observed to be non-tangent lines between vertices.

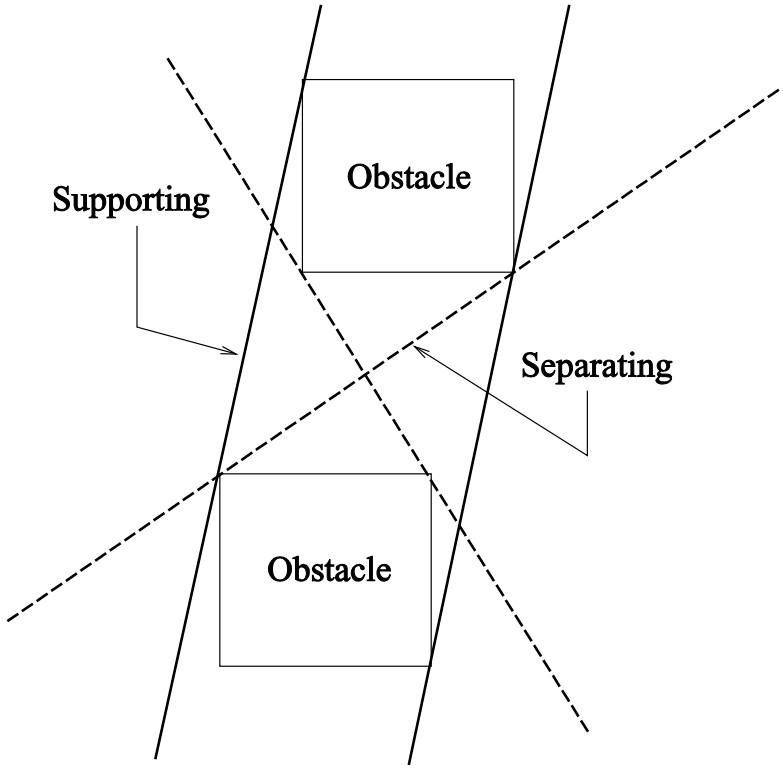

Figure 7 – Supporting and separating lines of two obstacles. Supporting: the obstacles lie on the same side of the line. Separating: the obstacles lie on different sides of the line.

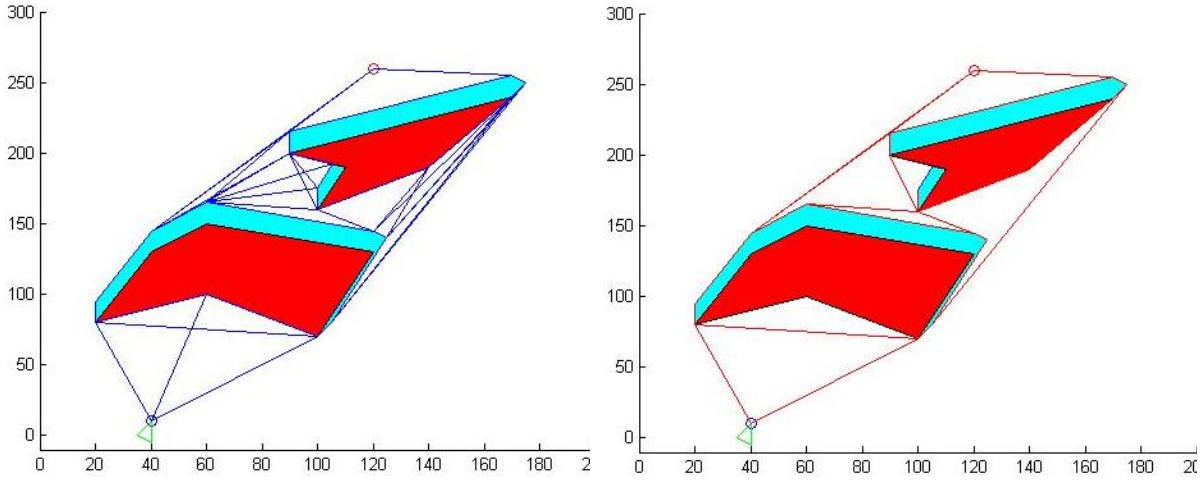

Figure 8 - The figure on the left shows the resulting visibility graph. The figure on the right shows the reduced graph.

3.3. A* Search

A* search is a best first graph based search algorithm that finds the shortest path between two nodes in the graph. An important function used in this search algorithm is a heuristic information that leads the search towards the goal. This heuristic information is an estimate about the distance to the goal in our case. Although various strategies may be used as heuristics, in visibility graph approach the Euclidian distance between the current node and the goal is preferred.

The search algorithm starts with an initial number of nodes that are neighbour to the starting point. In each iteration algorithm looks for the value of a function f(n) = g(n) + h(n) for each node n. This value is the sum of the distance to the goal and the real distance covered so far in order to reach that node. For each node with a value of function f(n), the algorithm selects the node with this minimum value and expands the neighbours of this selected node n. During the process, any node which has the minimum value of f(n) is selected.

A* search also remembers the nodes visited in each iteration. This memory both avoids the re-visiting of nodes and is used for back tracking when the shortest path is found between the start and goal nodes. With the notations given, the A* search algorithm can be described as below.

- O = Open set, or priority queue which includes the nodes that are subject to search at an iteration step
- C = Closed set, the visited nodes so far
- c(n1, n2) = the length of the edge connecting n1 and n2
- g(n) = total length covered so far in order to reach node n
- h(n) = heuristic cost function, or the Euclidian distance from node n to goal
- f(n) = g(n) + h(n) main criteria for search and selection evaluation

Algorithm 24 A^* Algorithm

Input: A graph

Output: A path between start and goal nodes

1: **repeat**
2: Pick n_{best} from O such that $f(n_{best}) \leq f(n), \forall n \in O$.
3: Remove n_{best} from O and add to C.
4: If $n_{best} = q_{goal}$, EXIT.
5: Expand n_{best}: for all $x \in \text{Star}(n_{best})$ that are not in C.
6: **if** $x \notin O$ **then**
7: add x to O.
8: **else if** $g(n_{best}) + c(n_{best}, x) < g(x)$ **then**
9: update x's backpointer to point to n_{best}
10: **end if**
11: **until** O is empty

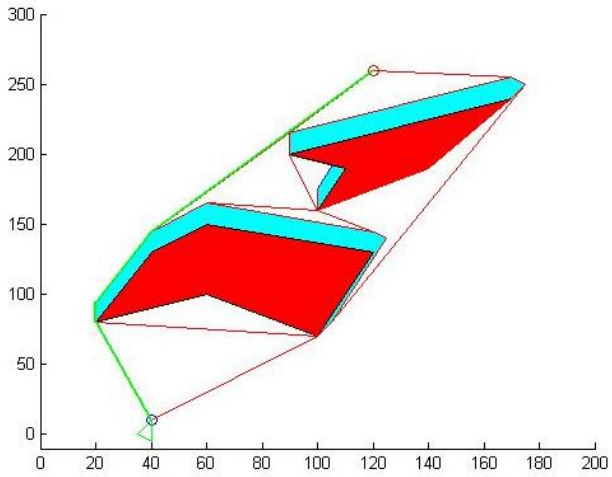

Figure 9 - A* search. The thick green line shows the result of the A* search.

6. References

[1] "**Principles of Robot Motion**" by *Choset H. et.al. Ch:5 Pg: 110-117*

6. Tasks

You should implement a visibility graph algorithm for a given robot work cell, were the robot is a **point robot**, which moves in two translational degrees of freedom. You might use (if you want) the Boost Graph Library (BGL) and are free to use any graph search algorithm, even though a Dijkstra or A* based search is recommended. If you use the given framework, the BGL is expected to be installed. You need not build a library, you can use the BGL functionality by including the appropriate BGL source files into your code.

In addition, with the framework provided you can use gnuplot to draw the visibility graph for documentation purposes.

The following example of a *gnuplot script* draws the graph:

```
set title 'Visibility Graph'
set size ratio 1.0
set xrange[0:1]
set yrange[0:1]
plot 'VisibilityGraph.dat' using 1:2 with lines
```

Boost header files as well as gnuplot binaries are provided as supplemental files. The VS2013 solution expects both archives to be extracted inside your solution directory!

Voltunteer Task

Improve your algorithm regarding **one** of the following aspects:

- Building a reduced visibility graph
- Handling polygonal robots
- Handling concave objects
- Improving the complexity of the algorithm by implementing the rotational plane sweep algorithm

Preparation after

You should prepare (pairwise) a document, which

- Describes the task
- Describes the solution in detail including pictures
- Comprises a well documented code
- Describes the lessons learned
- Is uploaded to Moodle in time.

  You check in your document and your code as a .zip file into moodle. Provide only the .sln solution and source files, so that the overall size keeps small.