International Journal of Mechanical Engineering and Robotics Research

*ISSN 2278 – 0149 www.ijmerr.com*
*Vol. 3, No. 2, April 2014*
*© 2014 IJMERR. All Rights Reserved*

***Research Paper***

# ALGORITHM FOR GRAPH VISIBILITY OBTAINMENT FROM A MAP OF NON-CONVEX POLYGONS

**J Crespo[1]*, R Barber[1], J G Victores[1] and A Jardon[1]**

**Corresponding Author: **J Crespo,** ✉ jocrespo@ing.uc3m.es*

Visibility graphs are basic planning algorithms,widely used in mobile robotics and other disciplines. The construction of a visibility graph can be considered a tool based on geometry that provides support to planning strategies in mobile robots. Visually, the method is used to solve that planning, which is quite extended due to the simplicity of operating with polygons, that represent obstacles in the environment. The cost of these algorithms tend to be quite low. The most sensitive issue of obtaining visibility between polygons is in cases in which the polygons are non-convex. In such cases, it is obligatory to know whether the area where one vertex of the polygon is found, is located in a convex or non-convex area, being desirable to distinguish between both situations in a simple way, issue that was not possible up to now. To obtain the visibility of non-convex polygons, the authors have developed a visual and intuitive method which gives the machine the ability to interpret the visibility with a simplicity similar to the human mind.

***Keywords:*** Path planning algorithm, Mobile robots algorithm, Visibility graph, Visibility in non-convex polygons

## INTRODUCTION

Visibility graphs, as can be seen in LaValle (2006), are tools for trajectory planning, widely used in navigation of mobile robots in works such as López (2002), but that are also extend to other disciplines such as the one discussed in Wein *et al.* (2005). In the case of robot navigation, it can be used to set the final path between two points as the library proposed in Obermeyer (2008) is expected to be used or as a tool for prior pre-processing for support or comparison with other techniques, obtaining a first solution to the problem, as suggested in Ortega *et al.* (2010). Currently, they still appear as complementary to other techniques such as the probabilistic one, which may be more efficient depending on the complexity of the environment. In the mobile robot movements

[1] System Engineering and Automation Department, Carlos III University, Madrid, Spain.

planning field, quick and useful solutions are offered. These solutions allow the robot's movement avoiding collision with the environment obstacles. They are a practical solution where the convergence to the solution prevails, against other techniques that can provide more optimal paths but may be oscillating or may not ensure the convergence or may require excessive time to reach it as the environment becomes more complicated.

In this paper, the concept of classic visibility graph is used as a basis. In LaValle (2006), a non-directed graphis defined as the pair ($N$, $\gamma$): where $N$ is a nodes set made up of the initial configuration $q_a$, the final configuration $q_f$, and the obstacles vertices. The $\gamma$ visibility function defined is not void if and only if the two referenced nodes are connected. Two nodes are connected if and only if a segment that joins them can be drawn, obtaining an edge that does not cross with any side of the obstacle, or being a side of an obstacle. In this way, two nodes are connected if and only if they are "visible". This is, when the second node can be reached from the first one (or vice versa) when the straight line that joins them is followed, without intercepting any environments obstacles.

### Motivation

Visibility graphs are a useful resource for robot navigation planning when therequirements of the movement do not require optimization of the trajectory followed by the platform and on the other hand, it is desirable to minimize the calculation time, looking for simplicity in the algorithm. In any case, a first approximation to a problem of navigation in an environment with obstacles may be useful. The problems that increase the complexity of the construction of a graph appear when dealing with non-convex polygons, so for cases in which there are only convex polygons, the visibility algorithm is considerably simple both to understand and to implement. To take advantage of using these graphs with obstacles represented by all kinds of polygons, whether they are convex or not, it is expected to obtain a visibility algorithm which, while keeping the simplicity of the concept and of the implementation, it works with any type of polygon.

In this paper a conceptually simple, fast, and robust algorithm that performs a visibility graph construction is presented. When the proposed algorithm is used, the computational cost is reduced. This is the computational cost due to the obtainment of the visibility of the obstacles of the environment when they are represented with polygons, convex and non-convex. In this way, an algorithm as efficient as possible that can construct the visibility graph is expected to be developed. A simple, low-cost and robust algorithm that can be applied to any environment, is proposed.

## TECHNIQUES AND PROBLEMS WHEN VISIBILITY GRAPHS ARE OBTAINED

Planning is defined as the search for a route free of obstacles from an initial position to a final one, through the working environment of the mobile robot. This task is accomplished using the information obtained of the environment in that moment, the description of the navigation task and of some kind of strategic methodology. Thus, the planner strongly depends on the model of the environment and the search algorithm used.

The simplest case is to consider a completely known environment, static and geometrically modeled using polygons. With these assessments it is possible to apply a search algorithm in graphs, using some cost function to obtain the path. However, the direct application of this methodology to use the resulting path as a path that must be followed, implies to carry out certain considerations about the vehicle. In this way, it is to be considered a specific vehicle, Omni-directional and that it goes over the assigned paths in a correct way.

## Visibility Graphs as Planners

There are different approaches to the definition of a path function path that leads to the robot from its initial position to the final position. Although the techniques based on visibility graphs, in recent years represent a slow progress in their studies, as the work of [?], they are still being used, for example by means of the Obermeyer (2008) library, in navigation techniques, therefore it is still necessary to contribute with new ideas in order to obtain some improvements. All approaches have the same goal, that is, the definition of a safe path for the vehicle that ensures there is no collision with an obstacle, and that does not fail to comply with the kinematic and dynamics restrictions imposed by the physical structure of the robot. A path function free of obstacles is searched for. This function must fulfill the condition of continuity in position. To solve this problem, a tool called visibility graphs can be used. The first work contributions can be seen in Nilsson (1969), that provide a geometric approach to solve the problem of planning. This method is very widespread since it operates with polygonal models of environment, so there are algorithms that construct this kind of graphs with a relatively low computational cost: O(n3) according to Lozano-Perez and Wesley (1979). This method needs models of environments defined with polygons, and it can both work in 2D/3D.

This approach has been preserved and extended at present. As the concept of topological graph with polygonal maps that present accessibility and connectivity seen in de Berg *et al.* (2000) and LaValle (2006) representation of obstacles. And coexisting with other algorithms such as Chazelle (1987) cell decomposition algorithm which have been based on principles of computational geometry plane-sweep seen in Boissonnat and Yvinec (1998), in Berg *et al.* (2000) and in Edelsbrunner (1987) and not forgetting the Voronoi diagrams described in O'Dunlaing and Yap (1982) in topological areas such as those seen in Hocking and Young (1988), with a cost higher than the basic visibility graphs(O(n4)), although there are algorithms that improve the execution time but that are considerably more difficult to implement as shown in Lee and Drysdale (1981), in Leven and Sharir (1987) and in Sharir (2004). For the visibility graphs, over time modifications of the algorithm appear. These changes provide improvements such as in Latombe (1991) and in Mitchell (2004), reaching algorithms much faster than the original ones but more complicated, as the library proposed in Obermeyer (2008) based on visibility in floating-point computation, where a visibility graph in non-convex polygons is also achieved, but it is much less intuitive than the solution proposed in this article.

## Problems Related to the Visibility Graphs

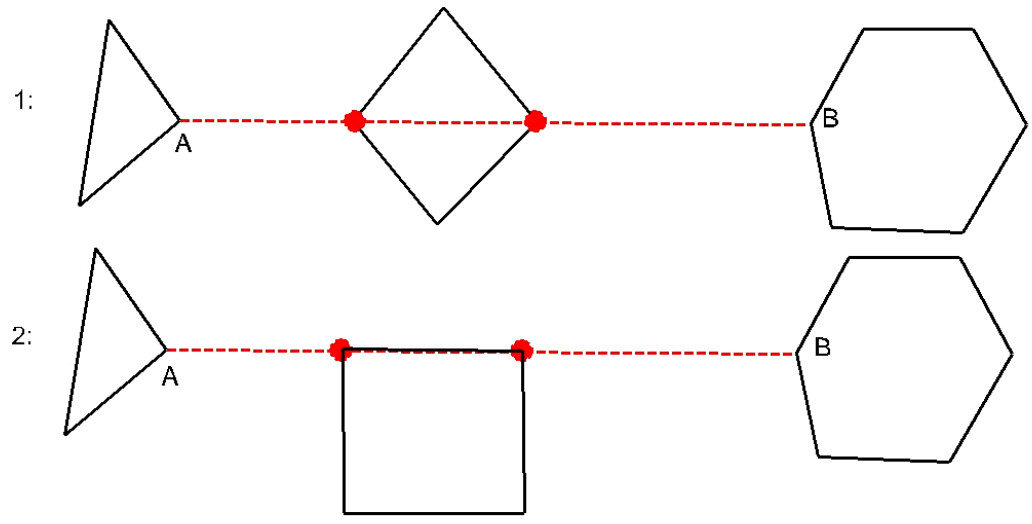
The main goal in obtaining a visibility graph is to find a way to generate the set of existing edges between the visible vertices of the elements that make up the environment, with the only information obtained from the map of polygons. This information consists essentially of geometric coordinates of vertices that form polygons and the information of which of them are connected together. Although, this idea can be applied to 2D or 3D maps, this work focuses on the first type.

Non-convex objects tend to be much more difficult to manage than convex objects, a fact proven in Bajaj and Kim (1988). Any analytic representation of contact surfaces or study of vertices is more complicated to formulate because the contact of two non-convex objects or a convex object and a segment can occursimultaneously in multiple discrete points. In addition, computational distance is much less complicated among convex polygons.

In order for convex polygons to find visibility is relatively simple, there are a few cases somewhat more difficult but equally resolvable. Taking into account that the way to obtain visibility is the calculation of breakpoints between two segments in two dimensions. Situations like the one shown in Figure 1 may appear.

In this situation, the first approach to check if two points are visible consists of drawing a segment between two points and seeing if it crosses a side of an obstacle. If there is a breakpoint, as shown in Figure 1, case 1, those points will be considered as non void breakpoints between both segments and therefore not visible. Figure 1, case 2, two breakpoints are shown, as in case 1. However, in case 1 both points are not visible and in case 2 both points are visible. With convex polygons this is not a problem, since if it is

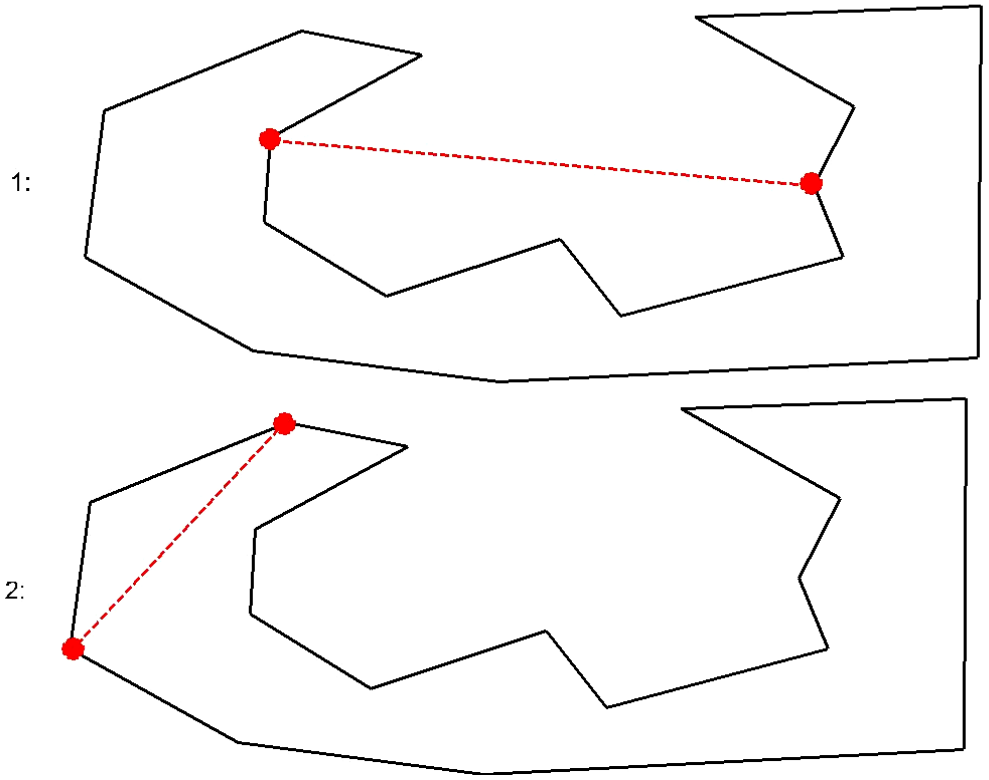
**Figure 1: Example of Visibility with Convex Polygons**




In case 1, there are two intersections, and no visibility between A and B.
In case 2, there are two cross points between A and B, but there is visibility between them.

**Figure 2: Example of Visibility with Convex Polygons**

In case 1, red points are visible because they are inside the convex zone but there is no obstacle between the two vertices. In case 2, there is no visibility. Differentiating all the possibilities of both cases can be done easily with the method described in this paper.




verified that the breakpoints are not part of a same edge of the polygon, an effective way of differentiating when there is visibility or not is obtained.

In the case of non-convex polygons, this way of reasoning is not adequate, and both cases cannot be distinguished, as shown in Figure 2. In a convex polygon, any vertex of a polygon has only visibility with the preceding vertex and the following vertex. However, in the non-convex polygons, any point belonging to the inner side can normally observe a large number of other vertices of that side. Generally solving the visibility of this type of polygon is a work of greater complexity and computational cost. This paper, in the line of other previous works searching for applications in the visibility graphs as in de Berg *et al.* (2000), presents a method of resolution which its goal is to provide more conceptual simplicity in an implementation that does not raise the computation time when identifying the visibility between two vertices of a non-convex polygon.

## ALGORITHM DESCRIPTION

The implemented method to solve the problem of visibility is based on a geometric transform. The main algorithm, the first level of this method, gives a general idea of the goal and the way of working of the algorithm.

The basic idea from the visibility graphs described earlier is based on the fact that if a

segment between two vertices of the graph can be drawn without being crossed by the edge of a polygon, then those vertices are considered visible vertices and the way of connecting them is added to the visibility graph as shown in the pseudo code below. Based on this idea, the flow diagram of this first level of the proposed algorithm can be seen in Figure 3.

for $v = 1 \rightarrow n_{vertices}$ do

for $w = 2 \rightarrow n_{vertices}$ do

$s \leftarrow$ createSegment($v$; $w$)

if $s$ is part of an edge of any polygon then add $s$ to the list of connections of the graph

else

if $s$ does not cross any edge then

add $s$ to the list of connections of the graph

; //it is VISIBLE

end if

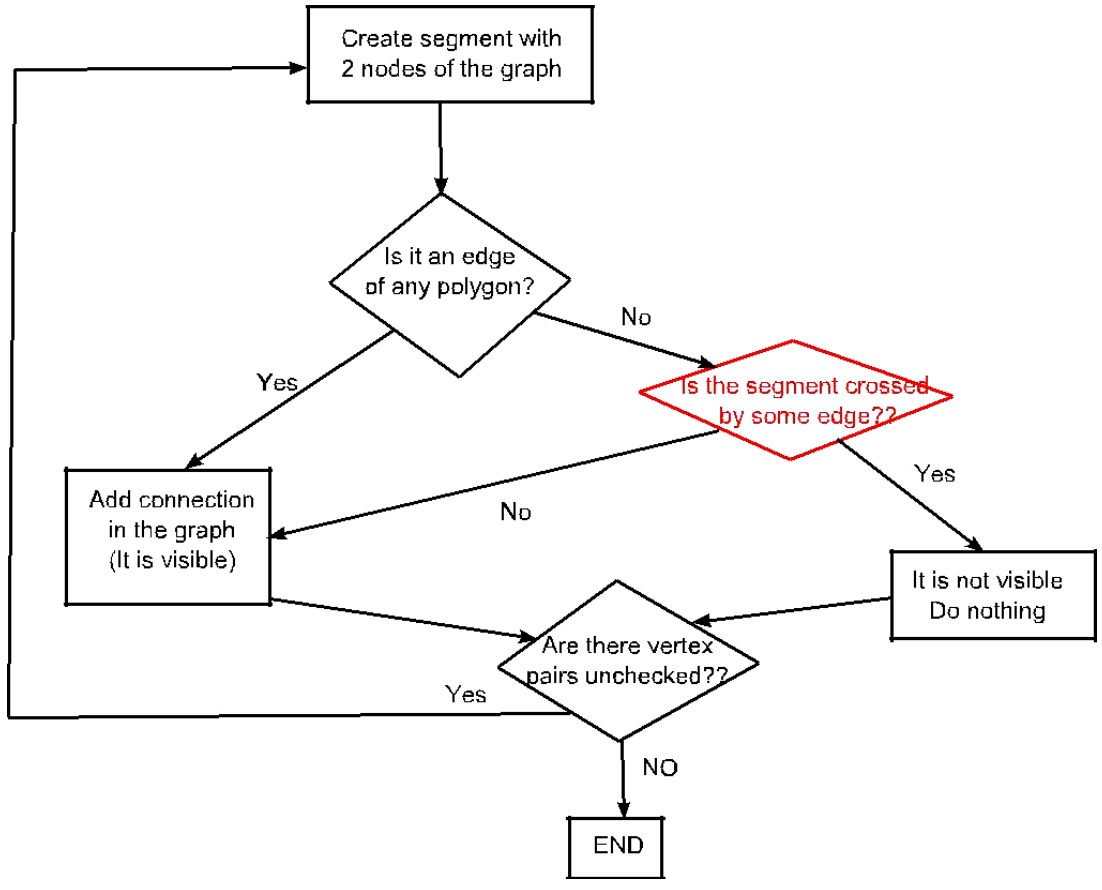
end if

end for

end for

However, with this simple consideration, it is not possible to find out whether the segment crosses or not with an edge. Therefore, a modification is included in the proposed algorithm, which is added in red (Figure 3): a module which checks if there is a breakpoint

**Figure 3: Flow Diagram of the First Level of the Algorithm**



between the segments and edges. To carry out this verification first, the breakpoint (BP) with the segment (S) is calculated. If BP matches with one of the vertices that form the edge, it is ignored. If BP is within the limits of the studied segment and edge, then it is returned as a valid breakpoint. If BP is also within the boundaries of the segment, it will be saved in a list of valid points.

For each point that is saved in the list, a segment with the rest of points from the same list is created. If the segment crosses the polygon, it returns the breakpoint, concluding that it is NOT VISIBLE. Otherwise, the algorithm indicates that it is VISIBLE.

The flow diagram that follows this level of the algorithm is shown in Figure 4.

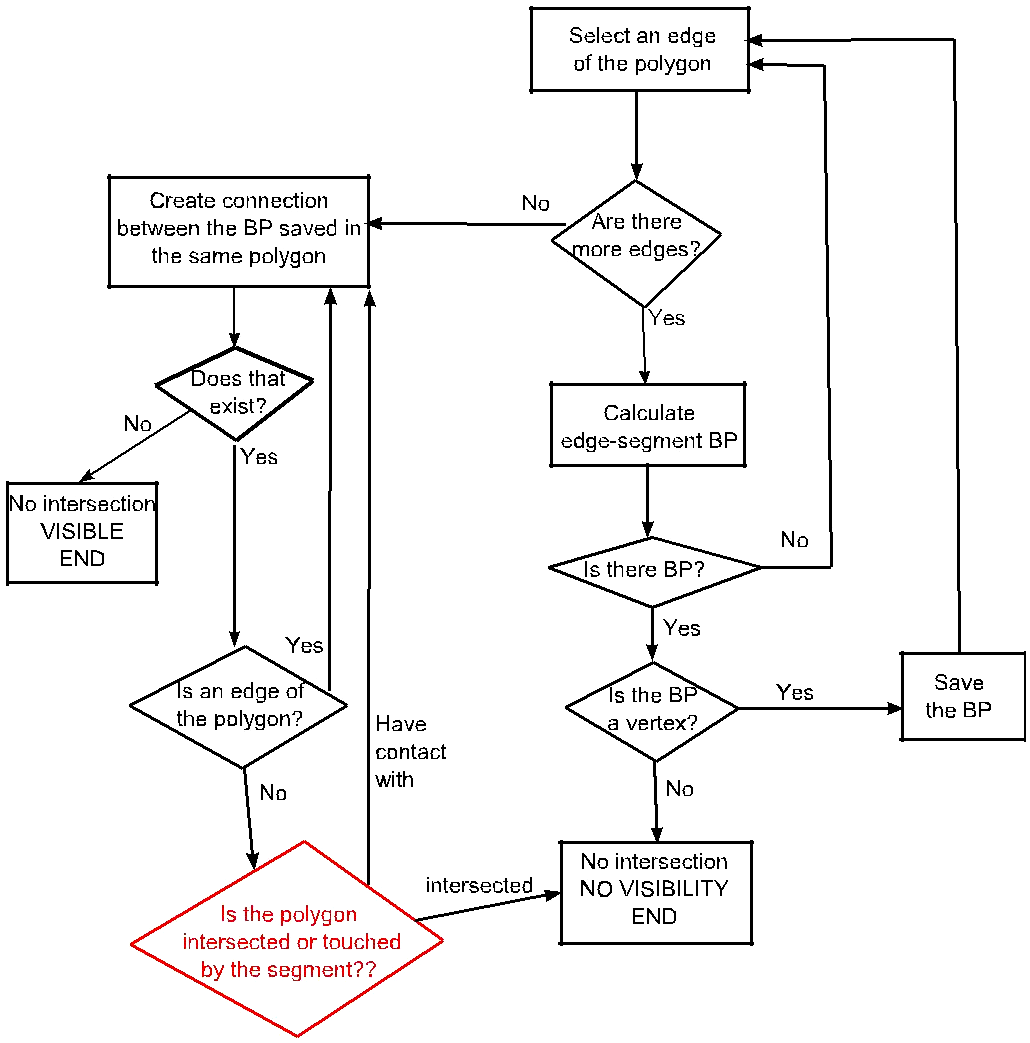
Below the pseudocode that performs this verification isshown:

while*edge*! = *NULL* do

BP = breakPoint(edge,S) if *bp* = *edgevertex* then

continue; end if

**Figure 4: Flow Diagram of the Second Level of the Algorithm**



if BP between limits of S and edge then return BP end if

if BP between limits of S then *listBP* ← *bp*

end if

*edge* = *nextEdgeOfTheGraph*end while

whilelistBP No empty do *p* = *pointofthelistBP*

whilelistBP No empty do

*q* = *pointofthelist*BP + 1 *segment* = *createrSegment* (*p*; *q*) if *crossesPolygon* (*segment*) then

return BP end if

end while end while

return NULL // Finally, it IS VISIBLE

## Visible Points in Convex Polygons

For visibility purposes, a breakpoint between a segment and a side of a polygon which matches with the end of that side, is considered that it does not cross and that, therefore, the point may be visible. This is because the segments are constructed by joining each vertex with the $n^{th}$ point that is the one where the iteration is run. The matching of the breakpoint with a vertex is a logical consequence, and it does not provide information about visibility, that is the reason why they are ignored in the following verification of the algorithm, as can be seen in Figure 5.

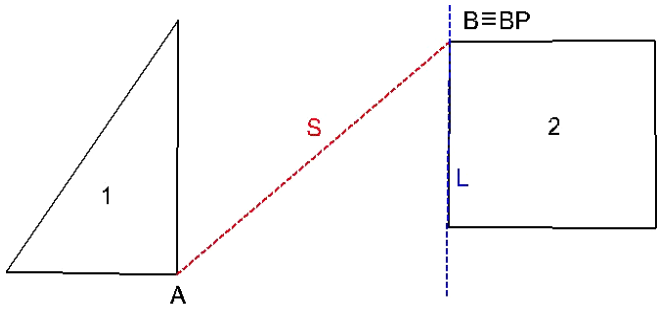
## Non Visible Points in Convex Polygons

In this case, it can be unequivocally concluded that the segment crosses the polygon, so, it is not possible that the segment points are visible among themselves. Figure 6 tries to show this fact.

## Visible Points in Non-Convex Polygons

An extra check is carried out due to a problem that arises only if what has been described up to now is considered. In this case, the algorithm saves in a list that tells, for each polygon, the breakpoints that match with vertices within the range of the segment. Figure 7a is an example of the BP that is NOTsaved (the BP is outside the range of the segment). Those of Figure 7b are saved in both of the described cases.

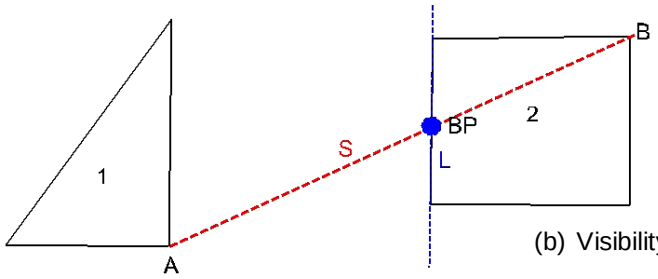
**Figure 5: Matching of the BP with a Vertex of its Own Side S**




S: Segment AB
L: Extended line of the edge of polygon 2

If the break point (CP) between S y L is a vertex of that edge => A and that vertex are VISIBLE

**Figure 6: When the BP is a Point of S that is Not One of its Vertices**

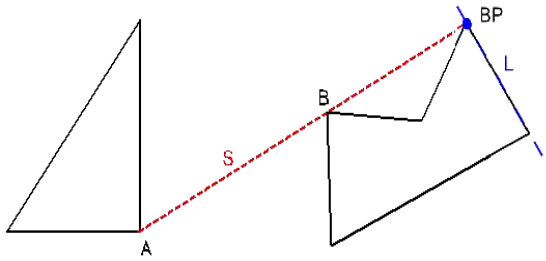



(b) Visibility with non-convex polygon

S: Segment AB
L: Extended line of the edge of polygon 2

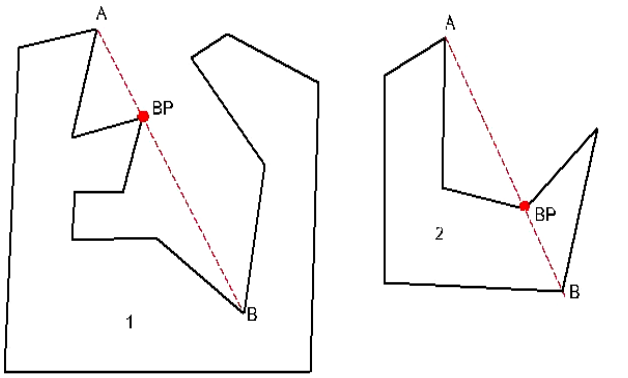
If S is crossed with a side of the polygon L, at a point that does not match a vertex of the edge
=> A and B ARE NOT VISIBLE

**Figure 7: BP Position Located Off the Limits of a Segment and Visibilities Cases**




The segment AB (S) is crossed with L in BP, but it is not within the limits of AB. It is a crossing point between the line associated with S and L, but not of S and L, then BP is not saved

**a) The BP is Off the Limits of S**




BP is within the limits (belongs to the segment) AB, is then saved in the two cases.

**(b) Visibility with Non-Convex Polygon**

## SOLUTION TO DIFFERENTIATE SEGMENTS THAT CROSS OR HAVE CONTACT WITH CONVEX AND NON-CONVEX POLYGONS

The main idea of this article has been to find a way to take advantage of the simplicity of the algorithm previously described and to obtain a method likewise easy, to distinguish between polygons that are crossed by a segment(and there is no visibility) and those that have contact with a segment (and there is visibility). Thus the visibility graph, without errors, is completed for a map made up of objects that can be represented by convex and non-convex polygons.

### Non Visible Points in Non-Convex Polygons

It must be checked if the polygon is crossed by the segment or it has simply have contact with the segment. The first case corresponds to what can be see on the right hand side of Figure 7b and is a case in which there is NO

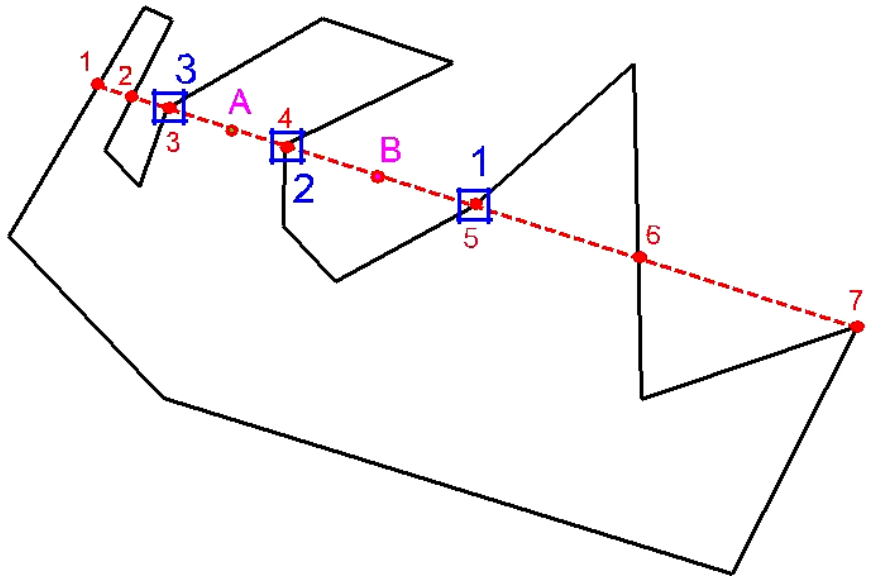
VISIBILITY between points A and B. The second case, which corresponds to where the segment has contact with the polygon, is represented on the left hand side of Figure 7b and if there is VISIBILITY. But the information that has been obtained up to now is not enough to tell apart both cases. There are two points that form a segment and a third point that matches with another vertex of the polygon and through this point crosses that segment. To discern between HAVING CONTACT WITH or CROSSING, another function has been implemented. That point in the algorithm is only reached when there is more than one point of crossing that matches with a vertex in a polygon. If it is not the case, the algorithm ends because no difficult cases of non convex polygons have been encountered.

The function that tells if a polygon is crossed by the segment that results from joining two of its vertices and therefore are not visible, or if instead it has contact with that segment and they are visible vertices, must carry out the following steps:

- Calculate all the breakpoints of the polygon with the line that the segment forms part.
- Delete the breakpoints that match with the vertices of the polygon that have contact with and that are not crossed.
- Choose a point belonging to the segment (the halfway point is a good candidate).
- Go over the line in both directions from the point. If in both tours, the number of breakpoints that has been obtained is even or void, then the vertices that made up the initial segment are VISIBLE between them. Otherwise, they are not visible.

This procedure allows to find out if both vertices were in a non-convex area of the polygon, where they can be visible. This function will return a void value if the segment had contact with the polygon. Therefore, it would be considered in the main function that they were visible vertices. In Figure 8 the idea described in this section is presented. Supposing that the visibility between vertices 1 and 2 (in blue) has been evaluated, the segment that joins them together is extended. This extension takes place until all the breakpoints in the polygon are obtained. A point between vertices 1 and 2, that is, in this example, point “B”, is selected. Now, among the 7 breakpoints (purple numbers), those that match with the vertices that have contact with them and that are not crossed are removed. It is shown that the original points 1 and 2 now correspond to points 4 and 5 of the set of intersections of the polygon with the extension, among them those that match with vertices are 3, 4, 5 and 7. Of all of them, only 7 has CONTACT WITH it, the rest of them are crossed. Therefore, 7 is removed from the list. In one direction, starting from point “B”, two breakpoints (5 and 6, since 7 was removed because it was not crossed) are obtained. In the other direction four breakpoints (1, 2, 3 and 4) are obtained. In both directions there are an odd number of points, so it can be stated that between vertex 4 and vertex 5 (the original 1 and 2 vertices) there is not included a portion of the area of the polygon to which they belong (is a non-convex area), being able to draw a segment between them without crossing with anything. Therefore, they are visible. As an example of non-visible vertices, instead of considering the original vertices 1 and 2, the case of vertices 2 (original) and 3

**Figure 8: Example of Points Crossed and that Have Contact with Segments**

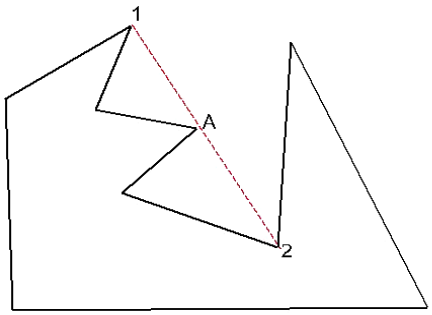



can be studied. They are aligned with vertex 1, therefore it is the same line and the same breakpoints, but this time, the chosen point between two vertices is point "A". In an address starting from point "A" three breakpoints (4, 5 and 6) are obtained and in the other one another three breakpoints (the 1, 2 and 3) are obtained, being both odd numbers. It can then be stated, that it is a convex area of the polygon, so both vertices are NOT VISIBLE.

## Vertices Contacted with Segments

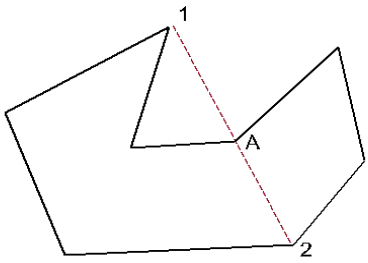
Finally, only an efficient method to check if a vertex has had contact with a segment or it has been crossed by a segment to be removed or not from the list of breakpoints of the previous function is needed. As it is

**Figure 9 Implication of Vertices Crossed by a Segment or that Have Contact with a Segment**




A: Vertex that have contact with segment, does not affect the visibility between 1 and 2

**(a) Result of Finding a Vertex that has had contact with a segment**




A vertex crossed affects visibility between 1 and 2

**(b) Result of Finding a Vertex that has been Crossed by a Segment**

shown in the sub Figure 9a and in Figure 9b, they are determinants facts regarding visibility.

The mathematical operations to check this out are greatly reduced in difficulty if a change of axes is carried out in order to represent these points. The other two vertices where the vertex under study is connected, must be checked. If a change of axis is performed in a way in which the new X axis matches with the crossed segment, there are two possible situations regarding the other two vertices:

- Both vertices having the same sign in the coordinate and, in that case the studied vertex should not be taken into account because it is a case of CONTACT between the segment and the polygon.
- Both vertices having different sign in the coordinate and, in that case the studied vertex is being crossed by the segment.

In the example of Figure 10, assuming that the visibility between 1 and 2 is being checked, A is a vertex that belongs to the breakpoints that the s segment creates with the polygon. It is being evaluated if A is being crossed by s or if it has had contact with s. Examining in the system of axes of coordinates for A, B and C, what it is shown in Figure 11a is obtained.

The change of axis consists of matching the new X-axis with the s segment. And the center, with A. The change consists of a translation of A to the center of coordinates and a rotation of angle that formed the s segment with the old X-axis. The result is shown in Figure 11b.

As C and B have the same sign in the Y coordinate (positive), it can be easily checked that the A vertex only had contact with the segment and the visibility will not be changed.

Instead, in this other case of Figure 12, with the change of axes, the B and C vertices will have different sign in its Y coordinate. Therefore, it is stated that the segment going over vertex A crosses the polygon.

**Figure 10: Example Checking Visibility Between 1 and 2**

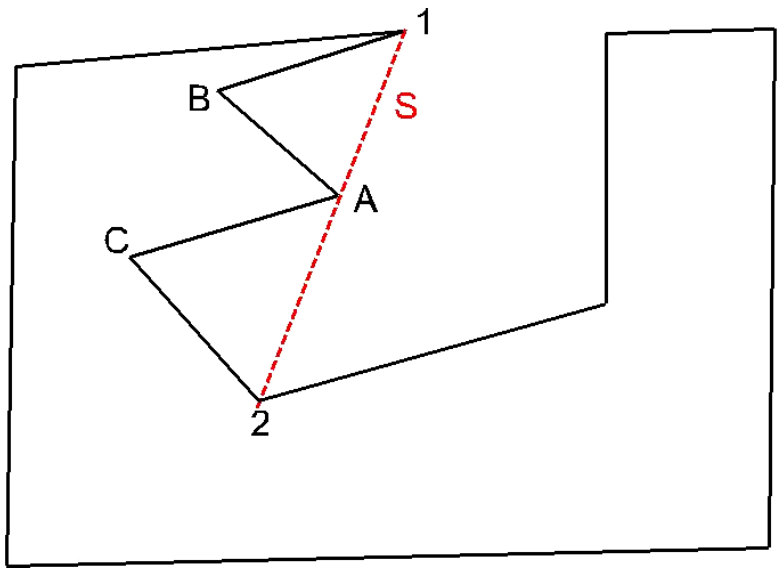
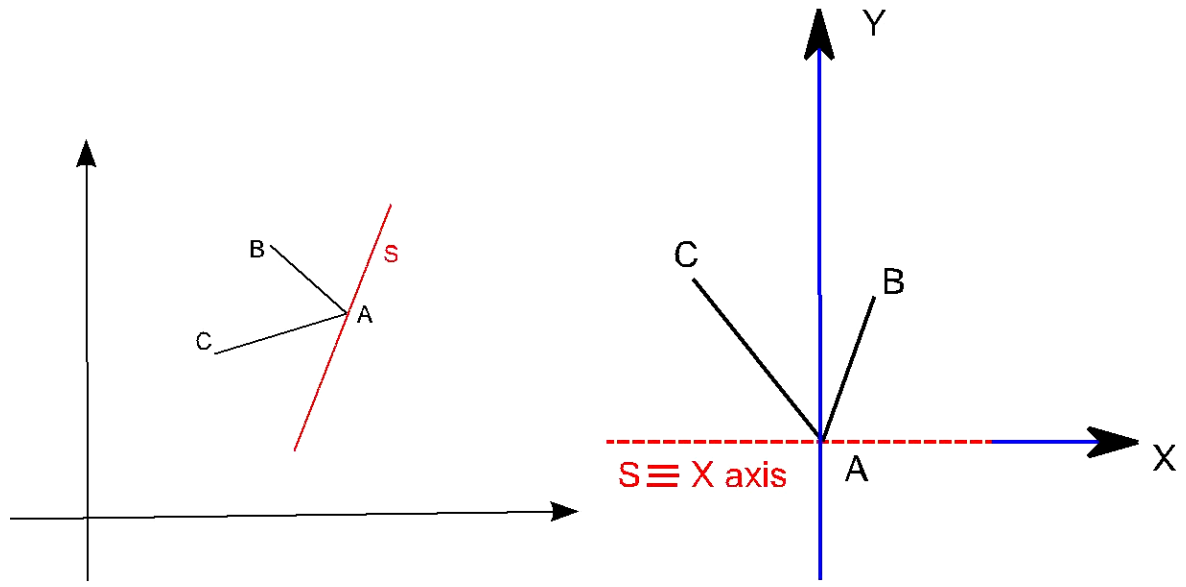


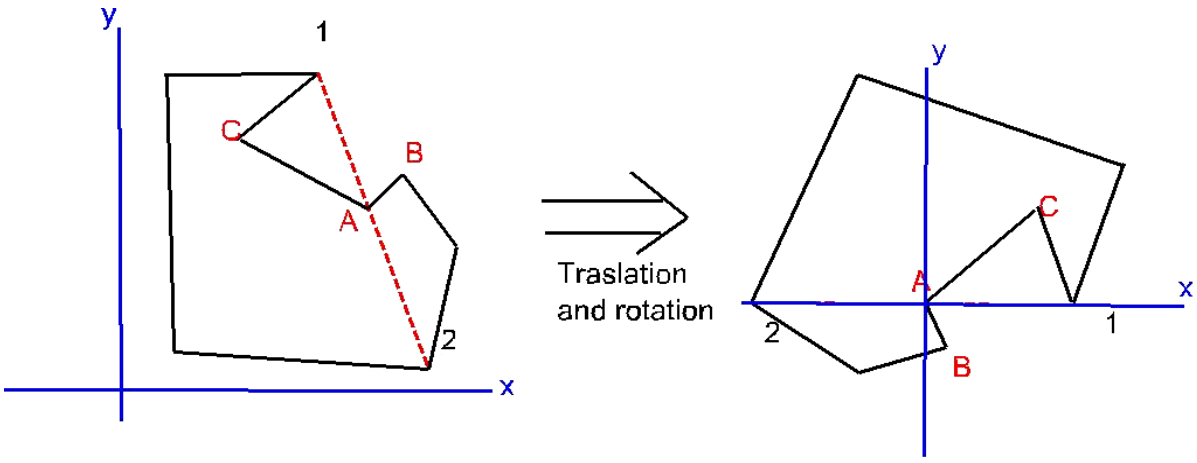
**Figure 11: Continuation of the Example Procedure**




(a) Example (Cont)

(b) Example Checking Visibility Between 1 and 2 (Cont II).

**Figure 12: Example Checking Visibility Between 1 and 2 (Cont III)**




If any of the vertices, to which the segment under study connects to, is also on the new X axis, the next vertex should be checked to verify and to evaluate if it is a case of CROSSING the segment or HAVING CONTACT WITH it.

The change of axis corresponds to a simple transformation matrix where two necessary changes will be carried out: a translation of the new center of coordinates, which will match the studied vertex, and a rotation of angle that formed the old axis with the segment. Applying the resultant matrix to the two vertices to which the studied vertex connects, the sign of the Y coordinate that these vertices would haveafter this transformation is obtained. In this case it can

easily be proved if the segment CROSSED the polygon with that vertex or if the segment HAD CONTACT WITH that vertex.

The first thing is to make a translation, placing the breakpoint as the center of coordinates. It is an inverse translation, the result shown in Equation (1) is sought:

$$(X_c, Y_c) \rightarrow (0, 0) \quad ...(1)$$

So each point in homogeneous coordinates, after the transformation, would correspond to the following Equation (2):

$$(X_{new}, Y_{new}, 1) = (X_{original} - X_c, Y_{original} - Y_c, 1) \quad ...(2)$$

And the translation matrix is shown in Equation (3):

$$T_{inverse} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -X_c & -Y_c & 1 \end{pmatrix} \quad ...(3)$$

Next, a rotation centered about the origin would have to be carried out. Working in polar coordinates ($r$, $\gamma$), points are obtained that correspond to ($x_1$, $y_1$) = ($r$*cos$\gamma$, $r$*sen$\gamma$). Therefore, this change is fulfilled in Equation (4):

$$(X_{new}, Y_{new}) = (r\cos(\gamma + \theta), rsen(\gamma + \theta)) \quad ...(4)$$

Obtaining this matrix of change Equation (5), typical of rotations:

$$R_{-\theta} = \begin{pmatrix} \cos(\theta) & -sen(\theta) & 0 \\ sen(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{pmatrix} \quad ...(5)$$

Multiplying the two previous matrices, being $\theta = M$, the final Matrix of Change (MC) is obtained in Equation (6):

$$MC = \begin{pmatrix} \cos M & -senM & 0 \\ senM & \cos M & 0 \\ -X_c \cos M - Y_c senM & X_c senM - Y_c \cos M & 1 \end{pmatrix} \quad ...(6)$$

**Being**

- $M$ the angle that segment s formed with the $X$ axis
- $X_c$ the original $X$ coordinate of the point that is now the center of coordinates
- $Y_c$ the original $Y$ coordinate of the point that is now the center of coordinates

It must be borne in mind that computational cost can be saved because only the sign of the coordinate and the new points (of $A$ and $B$) need to be verified, having only to carry out the operation of Equation (7) for each point.

$$Y'_A = -X_A sen(M) + Y_A \cos(M) + (X_c sen(M) - Y_c \cos(M)) \quad ...(7)$$

Once $Y'$ for $A$ and for $B$ is obtained, the sign can be compared in order to know if the segment crossed the polygon or if the segment had contact with the polygon.

In this way, the visibility for any type of polygon is obtained. The only thing needed is a multiplication of the coordinates of two points by a three elements vector. The rest of the matrix is not necessary, because only with the corresponding value of the new $Y$ coordinate the desired verification is run. It is important to note that this multiplication does not even have to be carried out for all the points of the graph. It only needs to be applied to those that form an edge, using the basic algorithm of visibility graphs (a quite less amount than the number of total vertices, $N$), it can not be tell if it is a convex or non-convex area. Therefore, the

computational cost of creating visibility graphs is not increased. The cost of these operations is insignificant because it does not affect the order of magnitude of the cost of the base algorithm, which remains the same. It must be considered that the cost depends therefore on the algorithm base on which the visibility graph is created, having several solutions as the ones described in Latombe (1991).

## EXPERIMENTAL RESULTS

This section shows the visibility obtained on a map of cases specially chosen for the problems they caused in basic methods of obtaining visibility. While using the proposed algorithm, the right visibility is obtained, without failure, and without increasing the computing time.

### Description of the Maps

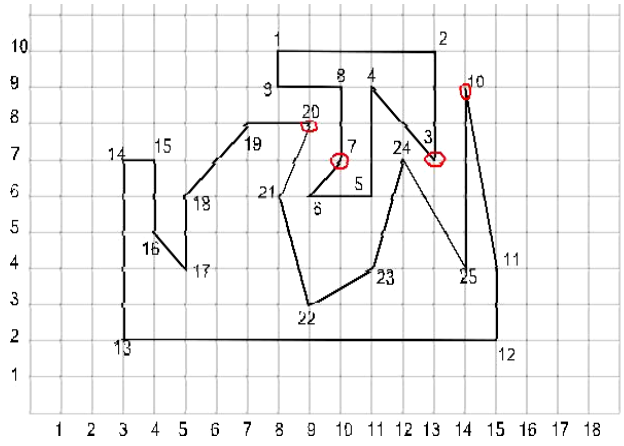
Two example of scenarios with obstacles represented by polygons are presented. Figure 13a shows one of the environments where the goal is to obtain the visibility of the polygons. The polygons are non-convex polygons and its vertices layout have multiple matches in coordinates that cause the borderline cases on which there could be errors, where all the situation referred throughout this work are tested. It is one of the maps designed to check the robustness of the algorithm.

Figure 13b shows a real environment corresponding to a polygonal simplification of obstacles of a laboratory of the RoboticsLab at the Universidad Carlos III de Madrid.

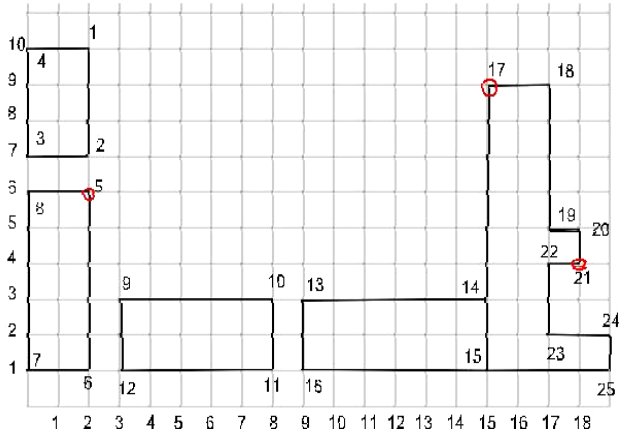
### Visibility Obtained

Following the visibility of the chosen points that the program has obtained, is shown. Only the visibility of the red points is shown in Figure 13a to avoid having a great amount of information. Theconnectivity of those nodes is shown in Figure 14a, directly built from the information obtained from the program and being considered representative enough of the performance of the algorithm. Figure 14b shows the connectivity of visibility of the red nodes on the map of the laboratory, being consideredequally representative of the overall performance. In both cases, it can be

**Figure 13: Test Maps**

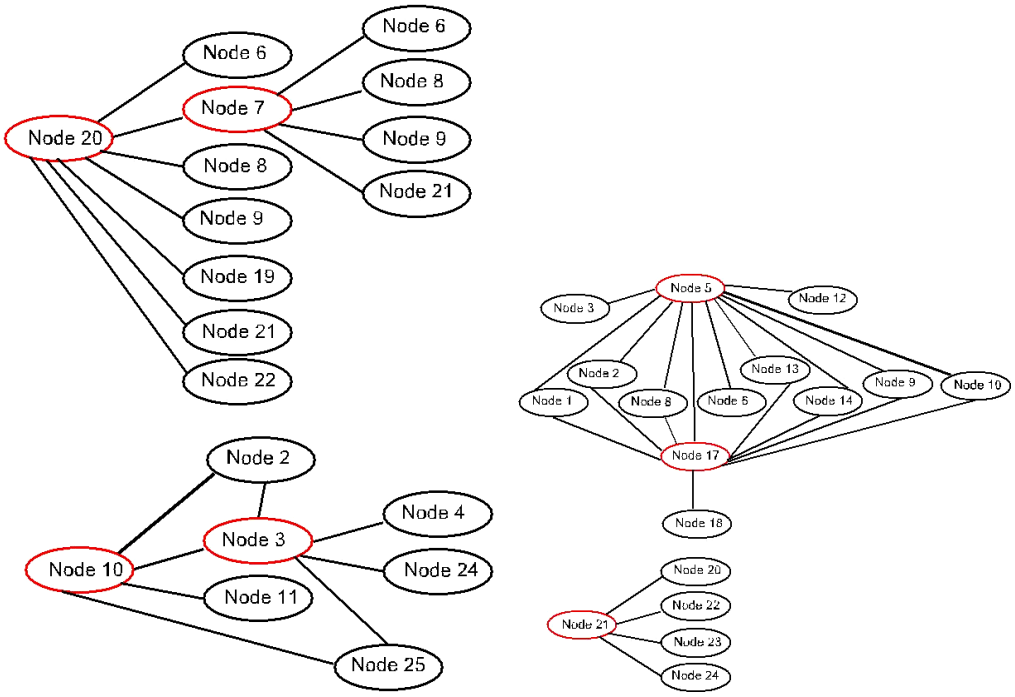


**(a) Map of a Complex Environment**



**(b) Map of the Robotics Lab's Laboratory**

**Figure 14: Visibility Obtained from Test Maps**




**(a) Visibility Obtained from the Complex Map** **(b) Visibility Obtained from the Map of the Laboratory**

verified that the visibility obtained satisfies its expectations, working without any problem. If a segment can be drawn, from one node to another, without crossing any polygon, a connectivity of visibility is created. The performance is correct, identifying the crossed vertices and those that have had contact with the segment.

## COMPARATIVE STUDY WITH OTHER METHODS OF CALCULATION OF TRAJECTORIES

It has been considered important to compare this method of obtainment of trajectories with the RRT method (Rapidly-exploring Random Tree), as it is described in Lpez and Gmez-Bravo (2006) being a widely used algorithm, known and being based on a different concept in several ways. The goal of the RRT method is to build a tree of exploration that uniformly covers all the space of collision-free configurations mentioned in LaValle (1998). The idea is to spread the tree, creating random configurations and checking if it can be reached (that is, that there are no obstacles between the points), in that case it is added to the tree.

### Difficulties of the Algorithms Based on RRT

The main problem of this method is the dependence of its effectiveness regarding the environment and the shape of the obstacles. This relationship is non-existent if a deterministic planner is used. The deterministic planner is based on visibility graphs as the one proposed in this paper, since the graph is generated with the

information in the form of obstacles. The RRT algorithm may not be able to obtain the right solution if during the creation of the tree a configuration or node which covers the free space on a specific point in a specific iteration does not arise. The randomness of the method, its stochastic nature, makes this occur. This problem appears in a more pronounced way with environments that contain obstacles with narrow passageways.

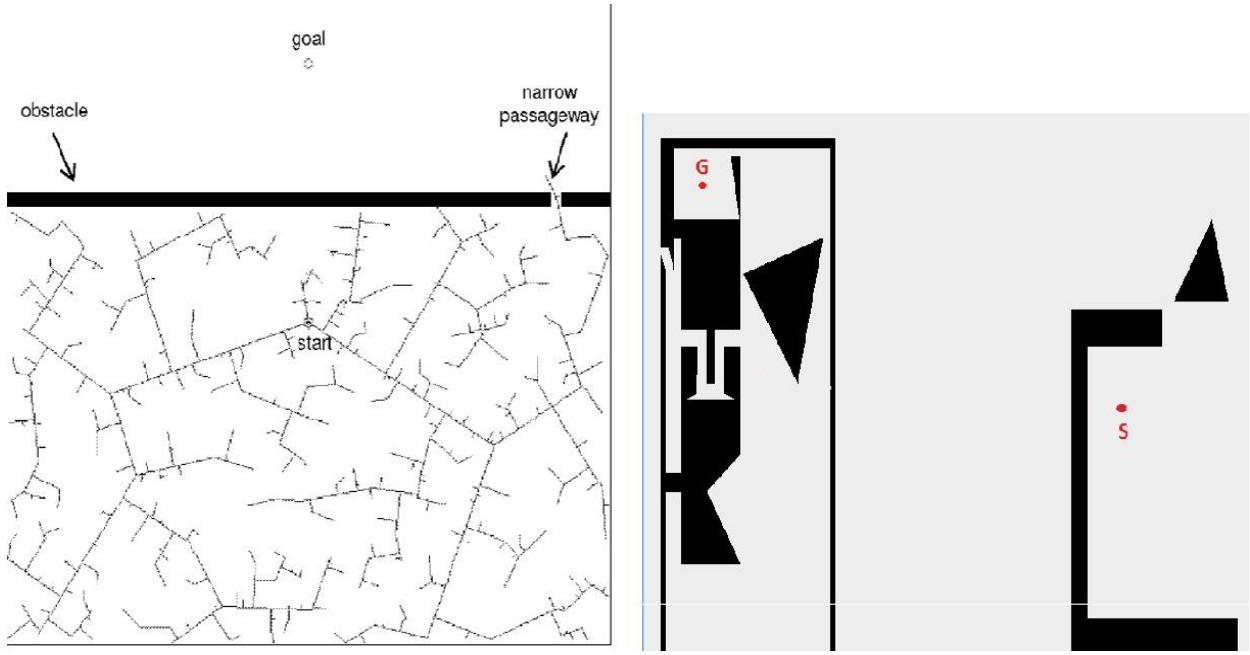
In addition to environments with narrow passages that lead to unreasonable computation time of the trajectory, RRT returns occasionally a wrong path with some oscillations. Another problem is the way to check whether if a connection between the two configurations is collision-free or not. The method used in the implementation of Benkmann (2001) divides the line that joins both points in N configurations. It virtuallyplaces the robot in each configuration and it checks if in any of the iterations, the robot is inside an obstacle. This seems to be a problem, because if the N number is small, the possibility of finding a case for a narrow obstacle (a very thin wall) from a configuration to the next, exists, thinking that there is a free path when there it is not because there are not any configurations inside the obstacle, but there is one between both of them. The solution would be to take a large N, but this increases the computational cost. In the implementation cited above, N had a default value of 10, that caused that certain obstacles were not detected. A value of 100 was enough to perform the experiments, but it therefore implies 10 times more iterations for each pair of nodes created. Note that this value does not assure that this problem may never occur.

## Experimentation with the Algorithms

This work has used two implementations of the RRT algorithm, being tested in environments also tested with the visibility graph based algorithm proposed in this article. To carry out the experiments it has been considered that the obstacles have been increased when working with robots considered as specific particles. In all cases, the proposed algorithm obtained an initial trajectory between the initial and final points in an insignificant time (less than 1 second). It was implemented in C++ and Java, to objectively be compared with the RRT. Among the RRT implementations, one uses Matlab programmed by Gavin Paul and Matthew Clifton, described in Clifton (2008) which is a Multiple RRT, where several test can be run, it will perform tests in 2D and 3D, with three-dimensional obstacles. For the comparative tests with the visibility graph algorithm, a 2D environment was used, ignoring the graphical part.

As the authors of that RRT implementation describe in their work, the level of convergence to a solution of the algorithm is strongly determined by the shape of the environment. Figure 15a very well illustrates this idea. It is evident that the algorithm is very efficient when exploring a particular area of the environment, but it has great difficulties to reach another region separated by a small opening of an obstacle, since whenever a point is chosen on the other side of the barrier, it is very likely that a collision occurs. Just close to the small opening is where the tree can be expanded, which increases the computational cost of finding a path.

**Figure 15: Difficulties of RRT and Test Map**




**(a) Example of the Difficulties that May Appear with RRT**

**(b) Map of Test for the RRT and the Visibility Graph. The Final Point (Goal G) Must be Reached Beginning at Theinitial Point (Start S)**

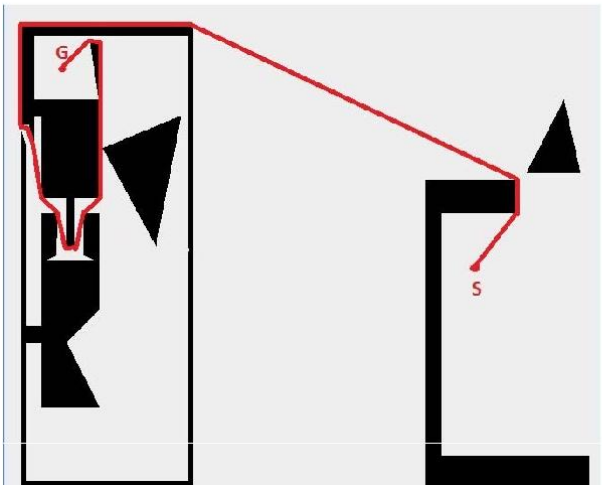
However, in the tests performed in the environment of Figure 15b the implementation of Clifton (2008) reachedwith difficulty the solution, but with a time too high (between half an hour and one hour of execution). This time difference was not taken into account because it was implemented in a different language, but the way between some areas was narrowed, specifically in the U-shape way, next to the point aimed for, where two protrusions that left a more confined space were added. And thus, the RRT algorithm does not converge, it reaches the maximum number of iterations set and the solution is no longer found.

An implementation in Java from Benkmann at (2001) under GNU General Public License (GPL) has also been used. In this implementation the number of nodes that the algorithm will create can be changed by the user. Initially 500 nodes, are recommended in order to not slow down the calculation of the trajectory. However, in the environment of the map of Figure 15b the tree did not expand inside the narrow passageways, which were not reachable destinations due to the absence of nodes close to it.

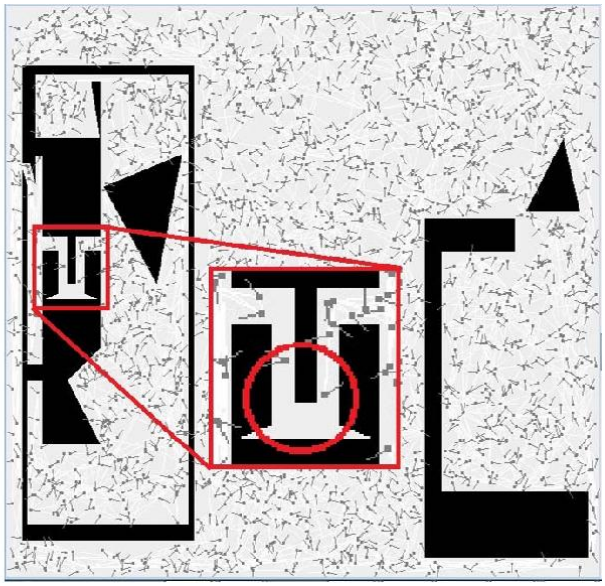
It must be taken into account that for each node that it is created, the connectivity should be checked, and that it can join the graph with a line without colliding with any obstacle. While in the visibility graphs the cost depends on the edges (that is, of the pairs of vertices that form the obstacles) which is a certain number, in the RRT depends on the number of nodes that are created and on the iterations that are taking place in the algorithm. Therefore, in certain environments, the computation time is lower with visibility graphs than with RRT based

algorithms. Using 2500 nodes, the image of Figure 16b is obtained. It is important to note that the stochastic nature of the algorithm makes difficult to tell whether there will be problems or not. This last figure, shows that inside the area surrounded by a red circle,

**Figure 16: Test Maps**

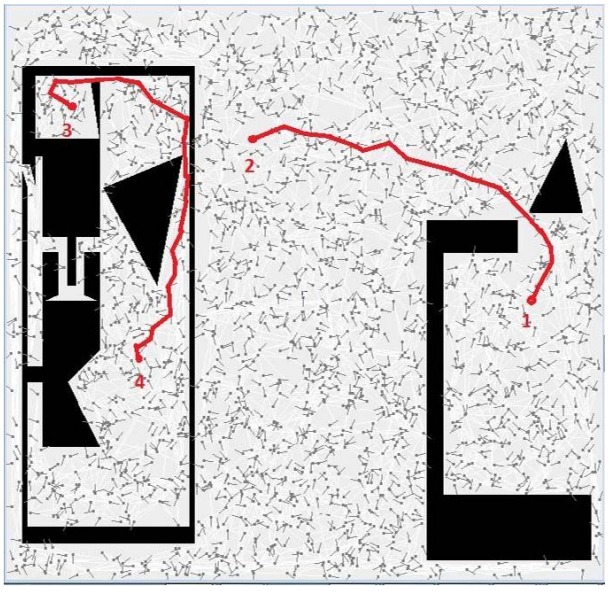



**(a) Map of the Tests Where the Path that Returns the Proposed Algorithm is Shown. It is the Same Problem that was Proposed for the RRT.**



**(b) Map of the Nodes Created with the RRT**

**Figure 17: Map of the Tests Where Two Posible Paths, with the Graph Created by the RRT, are Shown**



there is a dead zone that is left disconnected because there are no nodes between areas that cannot be connected without having to cross an obstacle. The reason is that the nodes that are inside the passageway (those that are inside the red circle) cannot be connected together. Not being able to communicate among them, both zones of the map are divided. Therefore, there are no possible paths between areas that have to go through this passageway, as it is shown in Figure 17, where the algorithm approximately creates thepath of the figure between points 1 and 2, in the same way between points 3 and 4. But no path is generated between points 1 and 3. The experimental results show that the algorithms based on RRT does not always reach the solution, while the algorithm proposed in this paper always reaches a solution (if it exists, as shown in Figure 16a), and the algorithms can be much slower that those based on visibilitygraphs.

## CONCLUSION

The method described in this article allows to obtain the visibility of a map of polygons, to easily build the related visibility graph. The calculation of the visibility in convex polygons is easy, but the case of non-convex polygons is the main problematic of the visibility graph construction. In this article, a method that is based on an idea that is simple and very visual on how to interpret the vertices that are crossed along the way between another pair of vertices is presented. For the human mind, it is a task that is solved with a knockout view. However, discern with geometric calculations, the visibility between two points can be very tedious for an algorithm. The idea of the change of axis, allows that with a small size matrix (3 x 3) the visibility between vertices that make up a segment can be checked. This operation with the matrix is only necessary when the situation of the segment matches with several vertices of a polygon and it needs to tell apart whether they are crossed vertices or vertices that have had contact with the segment, because it is a determinant fact for visibility. This way, a solution that solves the problem of visibility on maps with non-convex polygons is obtained. The way of solving the problem is intuitive, easy to understand and efficient since the computation time of the base algorithm is not increased. It has also been demonstrated that this algorithm always finds a solution (if it exists) regardless of the form of the environment, problems that arise with other algorithms such as the RRT. Therefore, this algorithm results reliable and an interesting option when the environment contains narrow passageways or complicated areas.

## ACKNOWLEDGEMENT

The research leading to these results has received funding from the RoboCity2030-II-CM project (S2009/DPI-1559), funded by Programas de Actividades I+D en la Comunidad de Madrid and cofunded by Structural Funds of the EU.

## REFERENCES

1. Bajaj C and Kim M (1988), “Generation of Configuration Space Obstacles, the Case of a Moving Sphere”, *IEEE Journal of Robotics and Automation*, Vol. 4, pp. 94-99.
2. Benkmann M S (2001), *Motion Planning Using Random Networks*.

3. Boissonnat J D and Yvinec M (1998), *Algorithmic Geometry*.
4. Chazelle B (1987), "Approximation and Decomposition of Shapes", *Algorithmic and Geometric Aspectsof Robotics*, pp. 145-185.
5. Clifton M (2008), "Evaluating Performance of Multiple RRTs", *Mechtronic and Embedded Systems and Applications*, MESA, IEEE/ASME International Conference.
6. de Berg M, van Kreveld M and M O Y O S (2000), *Computational Geometry: Algorithms and Aplications*.
7. Edelsbrunner H (1987), *Algorithms in Combinational Geometry*.
8. Hocking J G and Young G (1988), *Topology*.
9. Latombe J C (1991), *Robot Motion Planning*.
10. LaValle S M (1998), "Rapidly-Exploring Random Trees: A New Tool for Path Planning".
11. LaValle S M (2006), *Planning Algorithms*.
12. Lee D T and Drysdale R L (1981), "Generalization of Voronoi Diagrams in the Plane", *SIAM Journal on Computing*, Vol. 10, pp. 73-87.
13. Leven D and Sharir M (1987), "Planning a Purely Translational Motion for a Convex Object in Two Dimensional Space Using Generalized Voronoi Diagrams", *Discrete and Computational Geometry*, Vol. 2, pp. 9-31.
14. López S (2002), *Robots móviles: técnicas de navegación, grafos de visibilidad*.
15. Lozano-Perez Y and Wesley M (1979), "An Algorithm for Planning Collision-Free Paths Among Polyhedral Obstacles", *Communications of the ACM*, Vol. 22, pp. 560-570.
16. Lpez D and Gmez-Bravo F (2006), "F.A.O. Planificacin de trayectorias con el algoritmo RRT", *Aplicacin a robots no holnomos*, RIAI, Vol. 3, pp. 56-67.
17. Mitchell J S B (2004), "Shortest Paths and Networks", *Handbook of Discrete and Computational Geometry*.
18. Nilsson N (1969), "Searching Problem-Solving and Game-Playing Trees for Minimal-Cost Solutions", *Information Processing 68*, in Morrell A (Ed.), Vol. 2, pp. 1556-1562.
19. O'Dunlaing C and Yap C (1982), "A Retraction Method for Planning the Motion of a Disc", *Journal of Algorithms*, Vol. 6, pp. 104-111.
20. Obermeyer K J (2008), "Contributors", *The VisiLibity Library*, http:// www.VisiLibity.org
21. Ortega L M, Rueda A J and Feito F R (2010), "A Solution to the Path Planning Problem Using Angle Preprocessing", *Robot. Auton. Syst.*, Vol. 58, pp. 27-36.
22. Sharir M (2004), "Algorithmic Motion Planning", *Handbook of Discrete and Computational Geometry*, pp. 1037-1064.
23. Wein R, van den Berg J P and Halperin D (2005), "The Visibility-Voronoi Complex and its Applications", Proceedings of the 21st Annual Symposium on Computational Geometry, pp. 63-72, ACM, New York, USA.