Proceedings of 2004 IEEE/RSJ International Conference on
Intelligent Robots and Systems
September 28 - October 2, 2004, Sendai, Japan

Dynamic Visibility Graph for Path Planning

Han-Pang Huang* and Shu-Yun Chung+
Robotics Laboratory, Department of Mechanical Engineering
National Taiwan University, Taipei, 10660, TAIWAN
Email: hanpang@ntu.edu.tw
*Professor and correspondence addressee, + Graduate student

Abstract* — *In this paper, we propose a fast Dynamic Visibility Graph (DVG) for constructing a reduced roadmap among convex polygonal obstacles. DVG is extracted from the global environment with the simple geometric method and rules. Moreover, the data preprocessing is based on the concept of V-circle. Through V-circle, the process is speeded up greatly. Finally, DVG is extended to deal with multi-target problems that traditionally require a lot of time for reconstructing configuration space (C-space).

I. Introduction

A study of finding the shortest path of a point among polygonal obstacles has been discussed for a long time. Visibility graph (V-graph) proposed by Lozano-Pérez and Wesley [1] is the classical method to deal with that problem. V-graph is a compact, undirected graph that registers visibility among vertices of obstacles. The V-graph algorithm is complete and easy to implement. However, it is difficult to realize an efficient path planning for the environment with complicated obstacles. It usually takes $O(N_v^3)$ computation time [1], where N_v is the total number of obstacle vertices.

In this paper, we propose a new method to improve the efficiency of V-graph with only considering the local region. It will be shown that the path found from the local region is identical to that computed from the entire visibility graph. In other words, the computational efficiency is improved enormously. In particular, it is very suitable for multi-targets path planning due to the superior efficiency.

This paper is organized as follows: the preprocessing will be described in section 2. In this section, we will explain the steps of preprocessing briefly and the relationship between preprocessing and dynamic visibility graph (DVG). The principle of dynamic visibility graph will be introduced explicitly in section 3. Finally, the results of single-path planning and multiple-path planning will be shown in section 4 and section 5, respectively.

II. Preprocessing

Reducing computation time is the major purpose of preprocessing for visibility graph. Several algorithms [2][4][5][6][8] were proposed for improving the efficiency of V-graph and implemented in FPGA [7] recently. The most famous example is the reduced visibility graph defined in [2]. Its major difference from visibility graph is the way to choose vertices of polygonal obstacles. It only keeps the convex vertices so that given a pair of convex polygonal objects A and B, the reduced visibility graph consists of at most four distinct segments. However, this method still needs to re-compute its configuration space when the radius of a circular robot is changed. To avoid the re-computation of configuration space, Liu proposed ETG (Extended Tangent Graph) in 1991 [9]. The size $O(M \times N)$ is developed by giving a threshold interval $[l,h]$ to the edge such that it is collision-free if, and only if, $l < r < h$. Here M and N denote numbers of convex components and convex vertices of obstacles. Although this method solves re-computed problem successfully and has excellent efficiency, it is hard to be realized due to its complexity. However, it brings up an important idea, the concept of V-circle (Vertex circle).

In this paper, we extend the concept of V-circle and develop a fast dynamic visibility graph (DVG) for constructing a reduced roadmap among convex polygonal obstacles. Since only the tangent point between convex polygonal obstacles is considered in a V-circle, the radius of a V-circle depends upon the minimum safe distance. We can construct C-space (configuration space) easily through V-circle and find the shortest path, as shown in Fig. 1. If the size of the circular robot is changed, the only part for re-computation is the tangent line between two V-circles. Sometimes the paths with different safe distance are diverse even if their own start points and goal points are identical, as shown in Fig. 2 and Fig. 3.

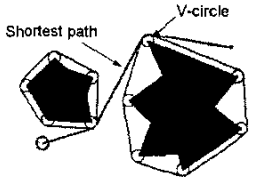

Fig. 1 C-space constructed by V-circle and the shortest path.

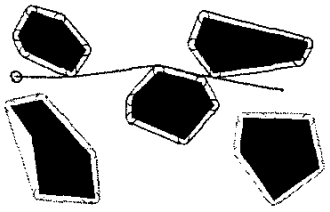

Fig. 2 The original shortest path.

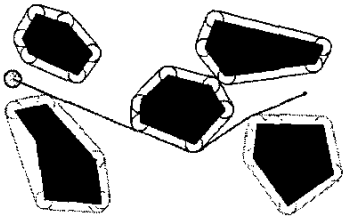

Fig. 3 New path

The original path shown in Fig. 2 is not adaptive when the robot size is changed. However, it will find another way to keep the path shortest in this configuration space, as shown in Fig. 3.

In order to realize V-circle, we need to find the convex vertices of the obstacle. The preprocessing for finding the

0-7803-8463-6/04/$20.00 ©2004 IEEE

convex vertices is shown in Fig. 4. The first step is connected component. Through this step, we can recognize the obstacles and label them. Then we can extract the boundary of obstacles and record those vertices in counterclockwise direction. The convex vertices can be obtained by quickhull algorithm [10]. After the process we described, the C-space can be easily constructed.

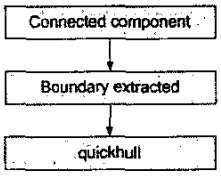

Fig. 4 The preprocessing for finding convex vertices.

III. Dynamic Visibility Graph

So far, we have seen that the shortest path is the line connected from the start point to the goal point (called S-G line) if it does not cross any obstacles. If the line crosses obstacles, the shortest path will be the path along the tangent line between obstacles, as shown in Fig. 5.

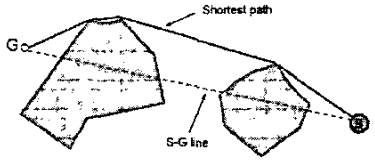

Fig. 5 The shortest path, where S and G denote start and goal.

In other words, ideally only the obstacles crossed by the S-G line need to be considered. However, this view is quite unsatisfactory. Once any obstacles which are not lied on the S-G line get into the tangent, the idea does not work. The situation is shown in Fig. 6. To resolve this difficulty, the shortest path will consist of the tangent lines connected the vertices of these obstacles, as shown in Fig. 7.

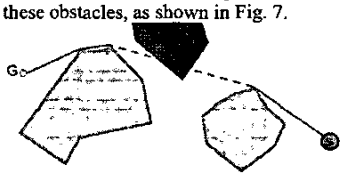

Fig. 6 Some obstacles need to be considered.

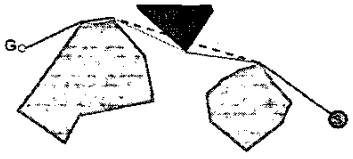

Fig. 7 The shortest path will be changed if any obstacles get into the original path.

The idea fails in that we cannot predict the location of the remaining obstacles by only considering local environment. However, it reveals an interesting phenomenon that the vertices on the shortest path have common characteristics: most of them are located inside the region (called active region). The boundary of this region is limited to the vertices which belong to the obstacles crossed by the S-G line and have maximum distance from the S-G line. For simplicity, we denote the vertices which have maximum distance from the S-G line as M_P. The active regions are shown in Fig. 8, Fig. 9, and Fig. 10.

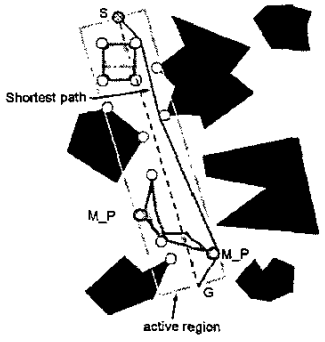

Fig. 8 Example for active region

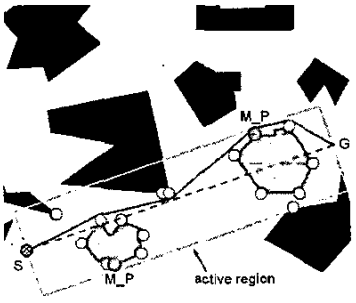

Fig. 9 Example for active region

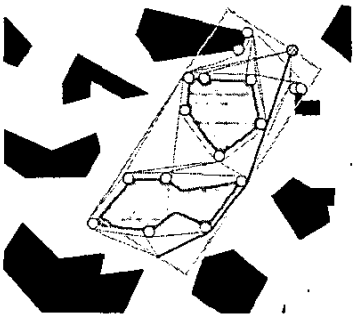

Fig. 10 Local roadmap

However, there are still some exceptions with the foregoing concept. If the obstacle is close to the S-G line and slightly oversteps the boundary of an active region, the shortest path will be out of the active region. The situation is shown in Fig. 11.

Considering the situation, we propose the following decision rule to modify the range of the active region.

Decision Rule:

The shortest outer path is longer than the longest inner path

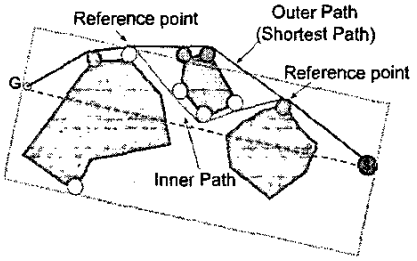

Fig. 11 The situation that shortest path is out of the active region.

The outer path means the path which is out of the active region. On the contrary, the inner path means the path is located inside the region, as shown in Fig. 11. If it is confirmed that all the outer paths are longer than the inner paths, the shortest path must be inside the active region. The range of the active region will be determined after all obstacles obey the rule. On the other hand, the active region must be expanded if any outer path is shorter than the inner one.

It costs lots of time for finding the outer path and the inner path of obstacles because the reference points are different each time. The reference point means the separated point between the outer path and the inner path, as shown in Fig. 11. In fact, it would be very complicated to search for every reference point, especially we cannot predict its location. To simplify and speed up the process, we only consider some necessary conditions. If the outer path is shorter, then the shortest outer path will be considered. On the contrary, if the inner path is longer, then the longest inner path will be presented. It can ensure that the shortest path is inside the active region if the shortest outer path is still longer than the longest inner one. The procedure to determine the shortest outer path is shown below.

1. Find the farthest outer vertex of the obstacle from the active region boundary, denoted as F_P in Fig. 12. It must be considered because all the outer paths have to pass through it.
2. In order to roughly predict the shorter outer path, the reference points are moved vertically to the boundary of the active region. By this way, the outer path would be shorter than before. On the contrary, the inner path will be longer. In addition, the reference points are moved horizontally to the original active region vertexes. It is not only easy to define the reference point location but also to present a more reasonable outer path because the father reference points from the F_P are easier to cross it.
3. The path connected from the active region vertex to the F_P will be the shortest outer path.

On the other hand, we determine the longer inner path by the following steps.

1. Selecting the rectangle vertexes of the obstacle which is determined by preprocessing instead of F_P. It is the longest inner path to cross the obstacles because any boundaries of obstacles cannot overstep the rectangle.
2. Connecting the active region vertexes and rectangle vertexes.

Finally, we compare the length of the shortest outer path and the longest inner path. If the outer path is shorter than the inner path, the active region is expanded to the F_P. The final range of the active region will not be determined until all obstacles obey the decision rule. The entire DVG algorithm flow is shown in Fig. 14.

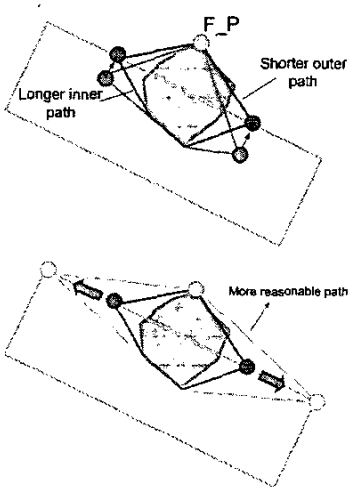

Fig. 12 The reference points motion

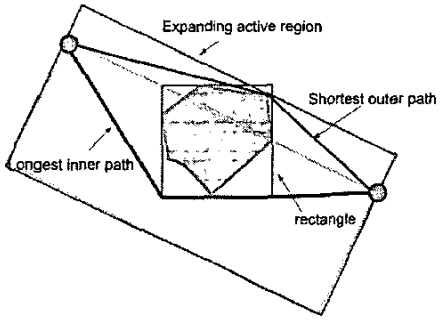

Fig. 13 Expanding active region

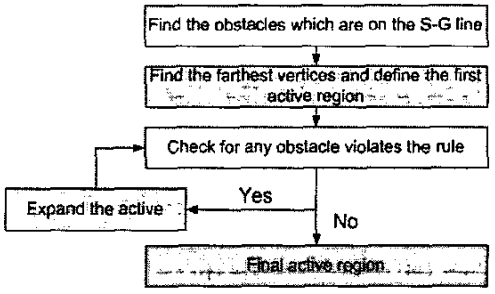

Fig. 14 DVG algorithm flow chart

As mentioned above, it is very useful to find the shortest path without considering all the convex vertices. The vertices need to be concerned are those which lie inside the active region. These vertices are used to construct the local roadmap, as shown in Fig. 10. In other words, the efficiency is enormously increased, specifically in the environment which has a lot of obstacles and vertices. Although only the situation of a robot-disk is considered in this paper, the concept of V-circles can be applied to any shape of robots in a 2D environment. The proposed method chooses the half maximum size of a robot as the radius of the V-circle. Moreover, DVG still works, but with less efficiency, even though the concept of V-circle is not utilized.

Since different initial conditions result in different visibility graphs and the active region range changes dynamically, we call the method Dynamic Visibility Graph (DVG). In the following section, the difference between a visibility graph (V-graph) and a dynamic visibility graph (DVG) will be discussed in detail.

IV. Performance Analysis

To compare the difference in efficiency, we choose a 2D environment with lots of obstacles and vertices. Besides, Dijkstra algorithm is utilized to find the shortest path after constructing a roadmap. In the following experiments, we will analyze the efficiency between V-graph and DVG. Because the preprocessing are the same, we only compare the processing time from constructing the roadmap to finding the path. The computation time of this part is denoted as A and the total processing time is denoted as B in Fig. 15. The entire process is shown in Fig. 15.

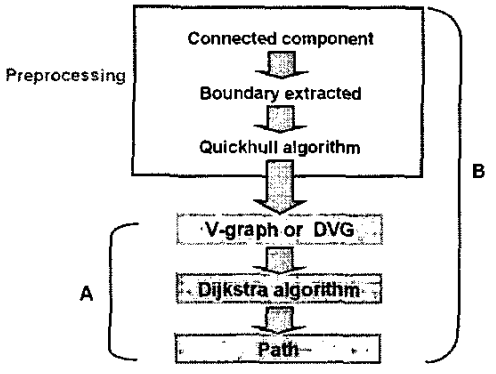

Fig. 15 Entire process for a single path planning

At first, we assign the identical start and goal points to keep the same initial condition. For easy recognition, the roadmap and V-circle considered in the environment will be drawn. Finally, the path will be shown for both DVG and V-graph. If the paths are identical, the algorithm with less computational time is more efficient than the other one. The experimental results are shown in Fig. 16 to Fig. 19 and Table I and Table II. Note that V_num denotes the number of vertices that are considered. T_roadmap is the period from constructing the roadmap to generating the path, and is denoted as A in Fig. 15. T_total represents the total processing time, and is denoted as B in Fig. 15.

From those figures and tables, the dynamic visibility graph (DVG) clearly surpasses the V-graph. Comparing the roadmap construction, DVG costs less time to define the range of the active region. In most cases, the active region is determined first without expanding. The efficiency of a dynamic visibility graph only depends on the complexity of the environment between the start point and the goal point. On the contrary, a visibility graph still needs to calculate all environment roadmaps even though the path is simple.

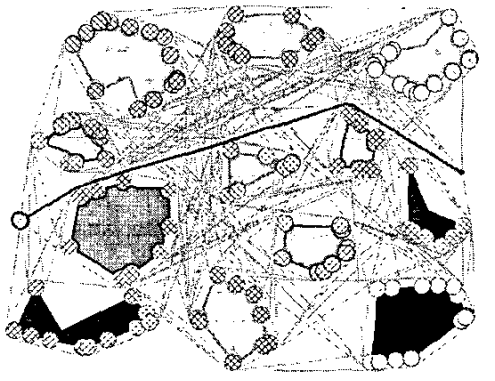

Fig. 16 Visibility graph (V-graph).

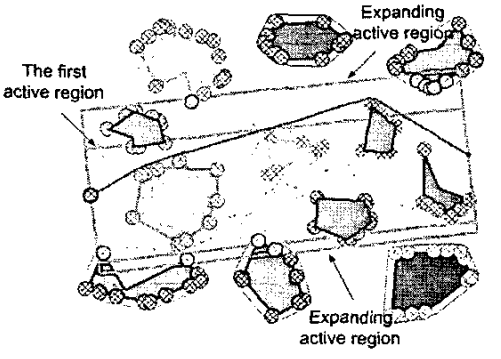

Fig. 17 Dynamic visibility graph (DVG).

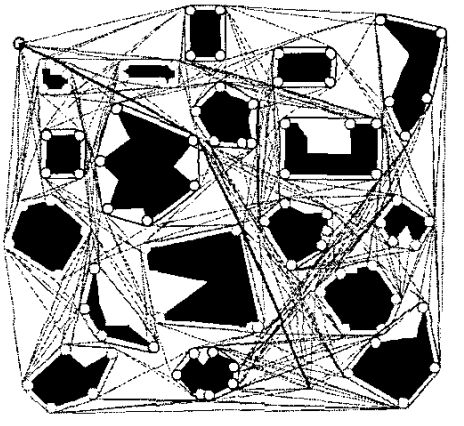

Fig. 18 Visibility graph (V-graph)

V. Multi-Target Path Planning

A dynamic visibility graph is very adaptive and can deal with multi-target path planning problem by its superior efficiency. Because all the convex vertices are found during the preprocessing, it only needs to reconstruct the dynamic visibility graph when the size of a robot is changed or another path is planned. The process is shown in Fig. 20.

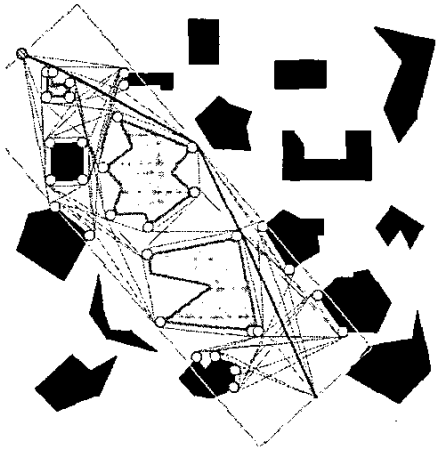

Fig. 19 Dynamic visibility graph (DVG).

Table I Comparison between Fig. 16 and Fig. 17

	Fig. 16	Fig. 17
V_num	129	70
T_roadmap(s)	0.762	0.20
T_total(s)	0.8	0.27

Table II Comparison between Fig. 18 and Fig. 19

	Fig. 18	Fig. 19
V_num	96	34
T_roadmap(s)	0.531	0.050
T_total(s)	1.631	1.131

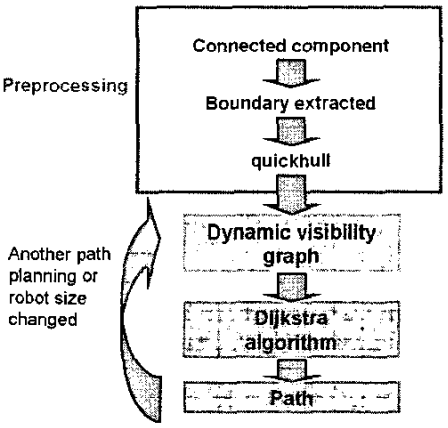

Fig. 20 The complete process for multi-target path planning.

We assign two robots (robot1, robot2) with different sizes in the environment and try to find their paths after giving their own start points and goal points. First, it will find the path of robot1, and the path is denoted as path1. Then in order to find the path of robot2, we need to reconstruct the DVG or V-graph, and reuse the Dijkstra algorithm. The process is shown in Fig. 20. The experimental results and data are shown in Fig. 21, Table III and Table IV.

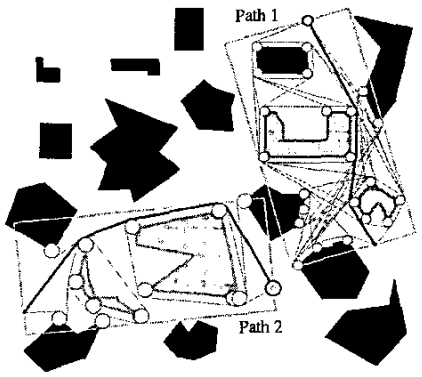

Fig. 21 Two-path planning.

The total number of obstacles and convex vertices are 18 and 96. The resolution of the picture is 860 x 860 pixels. The experimental results of the V-graph and DVG are listed in Table III and Table IV. Clearly, DVG is superior to V-graph.

Table III Two-path planning with V-graph

	Path 1	Path 2
V_num	96	96
Robot size	20	30
T_roadmap(s)	0.530	0.531
T_total(s)	2.131	

Table IV Two-path planning with DVG

	Path 1	Path 2
V_num	24	15
Robot size	20	30
T_roadmap(s)	0.012	0.010
T_total(s)	1.102	

Similar to the two-path planning, we also conduct experiments for three-path planning and four-path planning, as shown in Fig.22 and Fig.23. In the three-path planning, the total number of obstacles and convex vertices are 20 and 108. The resolution of the picture is 960 x 870 pixels. The experimental results of the V-graph and DVG are listed in Table V and Table VI. In the four-path planning, the total number of obstacles and convex vertices are 20 and 108. The resolution of the picture is 960 x 870 pixels. The experimental results of the V-graph and DVG are listed in Table VII and Table VIII.

Consider the experiments mentioned above. It should be noted that the processing time does not depend upon the number of paths. On the contrary, it depends upon the total number of the vertices that are used. There is a good evidence to prove this viewpoint from Table VI and

Table VIII. We can see that the processing time of the four-path planning is shorter than that of the three-path planning. From another viewpoint, a visibility graph needs to reconstruct whole roadmap whenever the robot size is changed. For example, it takes about 0.75 seconds to complete the roadmap and Dijkstra search each time in Table V. In other words, we need to take 0.75s to complete another path while the robot size is changed. In contrast with the visibility graph, the dynamic visibility graph is more adaptive for dealing with multi-target path planning.

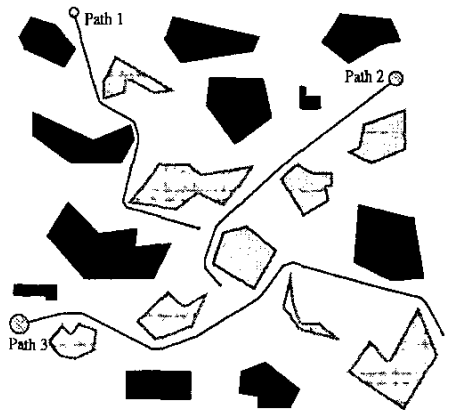

Fig. 22 Three-path planning.

Table V Three-path planning with V-graph

	Path 1	Path 2	Path 3
V_num	108	108	108
Robot size	20	30	40
T_roadmap(s)	0.751	0.753	0.750
T_total(s)	3.482		

Table VI Three-path planning with DVG

	Path 1	Path 2	Path 3
V_num	22	21	43
Robot size	20	30	40
T_roadmap(s)	0.020	0.010	0.060
T_total(s)	1.332		

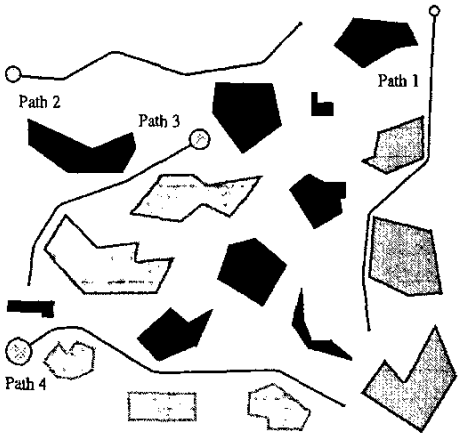

Fig. 23 Four-path planning.

Table VII Four-path planning with V-graph.

	Path 1	Path 2	Path 3	Path 4
V_num	108	108	108	108
Robot size	20	30	40	50
T_roadmap(s)	0.749	0.751	0.750	0.751
T_total(s)	4.213			

Table VIII Four-path planning with DVG

	Path 1	Path 2	Path 3	Path 4
V_num	19	21	19	21
Robot size	20	30	40	50
T_roadmap(s)	0.010	0.010	0.010	0.010
T_total(s)	1.282			

VI. Conclusions

The visibility graph has been used for path planning of mobile robots among polygonal obstacles for a long time. Furthermore, lots of algorithms were proposed to improve the efficiency. But few of them can solve re-computed C-space problem when the robot size is changed. In this paper, we proposed a simple method named dynamic visibility graph to enormously decrease the computation time of re-constructing the roadmap. Based on its superior efficiency, we also applied DVG to dealing with multi-target problem and got excellent performance.

References

[1] T. Lozano- Pérez and M. A. Wesley, "An algorithm for planning collision-free paths among polyhedral obstacles," *Commum. ACM*, vol. 22, pp. 560-570, 1979.

[2] H. Rohnert. "Shortest paths in the plane with convex polygonal obstacles," *Information Processing Letters*, 23, pp.71-76, 1986.

[3] Yun-Hui Liu, "Finding the shortest path of a disc among polygonal obstacles using a radius-independent graph," *IEEE Tran. On Robotics and Automation*, vol. 11, no. 5, pp.682-691 October 1995.

[4] Ta, Asano, Te Asano, L. J. Guibas, J. Hershberger, and H. Imai,"Visibility of disjoint polygons," *Algorithmica*, vol. 1, no. 1, pp. 49-63, 1986.

[5] E. Welzl, "Constructing the visibility graph for n-line segments in $O(N^2)$ time," *Inform. Process. Lett.*, vol. 20, pp. 167-171, 1985.

[6] J. Mitchell, "Shortest path among obstacles in the plane," *Proc. ACM Symp. Computational Geometry*, pp. 308-317, 1993.

[7] T.K. Priya and K. Sridharan, " An efficient algorithm to construct reduced visibility graph and its FPGA implementation," *Proceedings. 17th International Conference on VLSI Design*, pp.1057-1062, Jan. 2004

[8] Jason A. Janćt, Ren C. Luo and Michael G. Kay, "The essential visibility graph: an approach to global motion planning for autonomous mobile robots," *IEEE ICRA*, pp. 1958-1963, 1995.

[9] Yun-Hui Liu and Suguru Arimoto, "Proposal of tangent graph and extended tangent graph for path planning of mobile robots," Proc. 1991 IEEE ICRA, vol. 1, pp. 312-317, 1991

[10] C. Bradford Barber, David P. Dobkin, Hannu Huhdanpaa, "The quickhull algorithm for convex hulls," *ACM Transactions on Mathematical Software (TOMS)*, Vol.22, Issue 4, pp. 469-483, 1996