Path Planning by Using Configuration Space

Zhang, Lei Cai, Wenchao
Computer Science Dept., Tsinghua Univ. 100084
{thunderz99, xiaoxiao99}@mails.tsinghua.edu.cn

1. Abstract:

This article presents methods of path planning for a polyhedral object (called a robot) in 2D space to move to its destination without collision with obstacles. These obstacles are convex or non-convex polyhedra. The robot to be moved can be either convex or non-convex. Using configuration space, this problem can convert into path planning for a point. This is done by shrinking the robot to a point and enlarging the obstacles correspondingly. Then the path can be generated by visibility graph methods. But if you allow the robot to rotate, the Cspace method transforms the workspace into Configuration space with higher degrees of freedom, e.g. into 3D(2 translation plus 1 rotation) for two dimension problem and 6D(3 translation plus 3 rotation) for three dimension problem. So we restrict our work to pure motional problems. Our work is consisted of 2 parts, first to create the configuration space, and then solve the point path planning in the configuration space.

2. Introduction:

A robot works on a plane in the presence of obstacles. Finding a collision free path is of importance for a robot work cell with artificial intelligence.

A popular modeling of the robot is to consider the robot as a point, which can be obtained by transforming the workspace into C-space, then a 2D robot whose outline is a polyhedron can be considered as a point in the 3D C-Space.

The shortest path is widely pursued objective in path planning. Although the 2D shortest path planning was studied in 60s, the 3D shortest path (3DSP) planning was proposed in the later 70s, motivated the by path planning for manipulators. Lozano-Perez and Wesley proposed the general path planning

algorithm for a robot among polygons and polyhedral obstacles in 1979. In 1980, Lozano-Perez used A configuration Space Approach[LP] to solve the problem.

Sharir and Schorr presented their studies on the 3DSP finding in [SS] in 1984. They included some basic but important observations as lemmas, such that the shortest path on an edge forms two equal diagonal angles with the edge, and the existence of ridge lines on polyhedra.

The general 3D shortest path planning problem is quite complex, its NP-hard complexity makes people believe that its application prospects are gloomy. However, the robot's moving envelope is limited and the number of work pieces can be a few. Thus in the light of n^O(k) complexity, where k is the number of obstacles, the shortest path planning algorithms may still have potential for use.

3. Obtain the Cspace of purely translational motion robot in 2D

Let B be a rigid k-sided polygonal object free to translate in the plane amidst a collection of polygonal obstacles $A_1, \ldots, A_m$, having n sides altogether. Our goal is to calculate the free configuration space FP of B, consisting of all free placements of B (i.e. placements in which B does not intersect any obstacle). Having calculated FF, we can then decompose it into its arcwise connected components, so that, given any pair of free placements Z_1, Z_2 of B, we can determine whether they lie in the same connected component of FP, in which case collision-free translational motion of B from Z_1 to Z_2 is possible.

Note that FP is two-dimensional, as B has only two degrees of freedom. In fact, FP can be represented in the following way, initially proposed in [LP]. Take a reference point $b \in B$, which, without loss of generality, we assume to lie at the origin when B lies in some given standard position. For each obstacle A_i, calculate the Minkowski (vector) difference

$$K_i = A_i - B = \{x - y \mid x \in A_i, y \in B\}$$

Here, we define the – and + for the set of point as follows:

$$A+B=\{a+b \mid a \in A, b \in B\}$$

$$A-B=\{a-b \mid a \in A, b \in B\}$$

$$-A=\{-a \mid a \in A\}$$

A+B is shown in Fig.1.

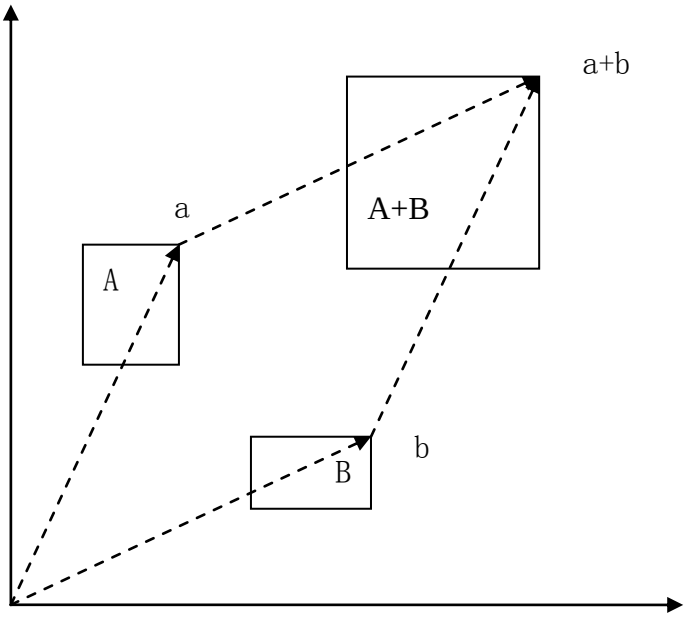


Fig .1 Illustration for A+B

Thus, if there is an obstacle A, and a robot B, then K=A-B will be the expanded A in CSpace, as shown in Fig.2.

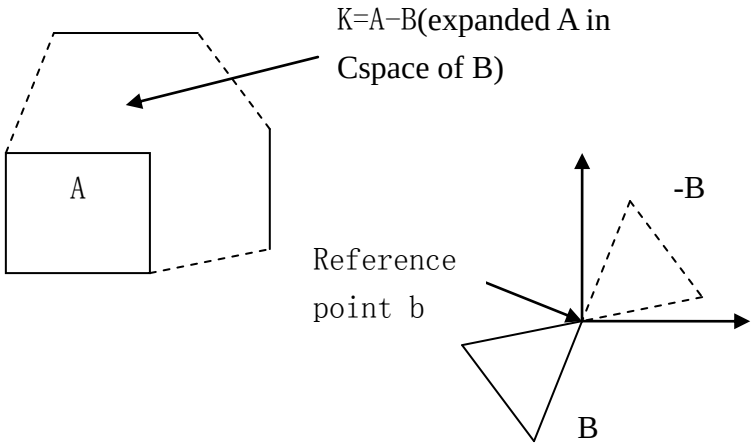


Fig.2 Illustration for K=A-B

Amidst m obstacles $A_j\,(j=1,...,m)$, It is clear that B lies at a free position if and only if the reference point b does not lie in $K=\bigcup K_j$. We can thus represent FP as the complement K^c of K, and our task thus reduces to that of calculating the polygonal "forbidden space" K. Once we get the FP, then we can solve the problem by planning a path for a single point robot. This is shown in Fig.3.

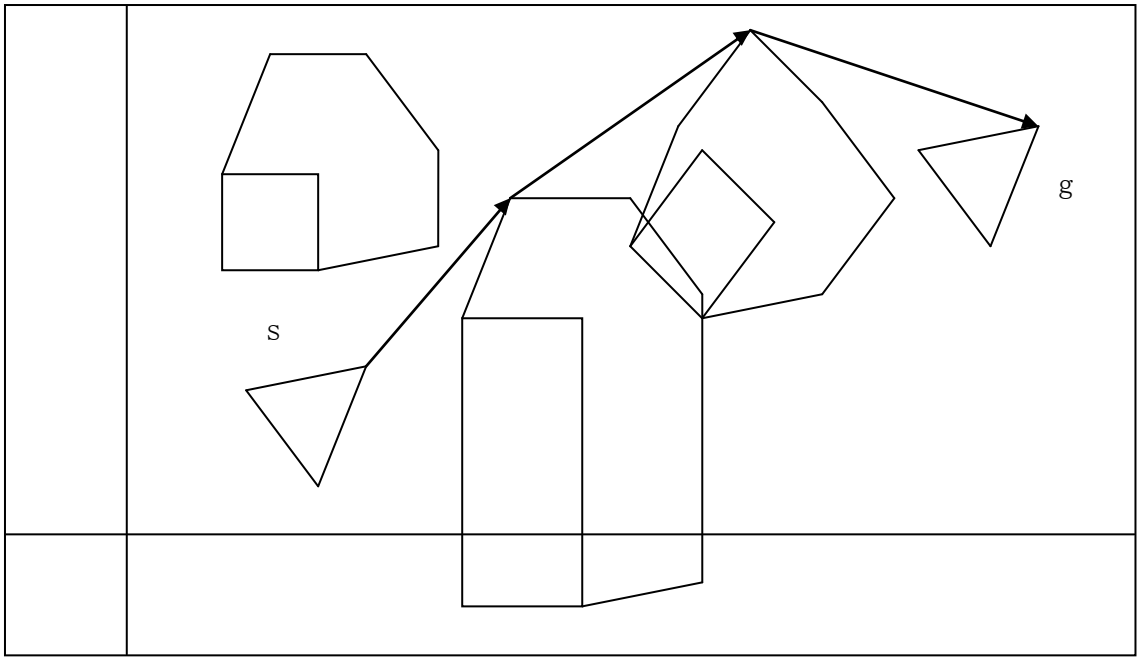


Fig.3. The FindPath problem and its solve using Cspace. The shortest path from Source to Goal are shown in arrows

4. Shortest path among polygonal obstacles in the plane

The classic motion planning problem is: Given a source point and a destination point along with a set of polyhedral obstacles in two or three dimensional Euclidean space, find out the shortest path from the source point to the destination point without coming in contact with any of the obstacles.

Due to the limit of time and ability, we just pay our attention to the movement of a single point in the plane.

Through out this report, we let *v* denote the number of the vertex and *n* denote the number of the edges and *k* denotes the number of the obstacles. In the practice problem, it may that *k<<n and k<<v,* for example, the layout of a particular floor in an office building may be such that the hallway divide the layout into only a few distinct connected components even though it requires hundreds of thousands of edges/vertex to accurately describe the layout.

On this problem, we present a straightforward algorithm, the output of the algorithm is the visibility graph of the obstacles and the Shortest Path Map corresponding to a given start point. The SPM is a planar subdivision of size *O(n)*, which allows one to find the length of the shortest path to a query point in time *O(log n)*(by point location), and to produce a shortest path in time *O(log n + m)*, where *m* is the number of bends in the path.

The basic idea behind the approach is to make the visibility graph of the obstacles. The visibility graph is the lists of the visible points for every vertex. Given a source point, compute its available vertex directly, and find the shortest path from the source point to every vertex based on the visibility graph, that is Shortest Path Map. In light of SPM, given a destination point, the algorithm just finds out the visible vertex for the destination.

1 · Visibility graph

Visibility graph is the basis of the algorithm, which is a table recording the visible vertex for every vertex. An array of two dimensions is applied for it. Fig.4 gives the simple example.

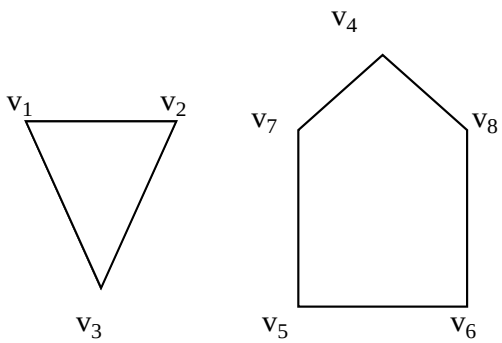


Fig 4: two obstacles in plane
v_1 just can see vertex v_2, v_3, v_4.

vertex	Seen vertexes
V_1	V_2,V_3,V_4
V_2	V_1,V3,V_4,V_5,V_7
V_3	V_1,V_2,V_5,V_7
V_4	V_1,V_2,V_7,V_8
V_5	V_2,V_3,V_6,V_7
V_6	V_5,V_8
V_7	V_2,V_3,V_4,V_5
V_8	V_4,V_6

Table 1: visibility graph for left plane

For a short, one vertex can be seen by another vertex if the segment of these two vertexes doesn't intersect the interior of any polygon. Because pair vertexes can/cannot see each other, we can use only the pair points (*vi, vj*) for (*vi, vj*) and (*vj, vi*) (where *i<j*). Thus, Table1 can be simplified as follow Table2.

2. Visible vertex

The pair visible vertexes have been described above. We can compute the visible vertexes pairwise according to the definition. But there is too much useless computation. First, the two adjoining vertexes in the same polygon can see each other, we don't need any computation, if the polygon is convex, every vertex will only see the adjacent vertexes in the polygon (see Fig.5); secondly, for a vertex on another polygon, the field of view on this polygon will less than 180 degree (see Fig.6), we just need find out the right most and left most vertex for this view vertex, if the polygon is convex, all the vertexes between leftmost and rightmost can and only can be seen. Fig.5 and Fig.6 give the description.

vertex	Seen vertexes
V_1	V_2,V_3,V_4
V_2	V_3,V_4,V_5,V_7
V_3	V_5,V_7
V_4	V_7,V_8
V_5	V_6,V_7
V_6	V_8
V_7	NULL

Table 2: simplified visibility table

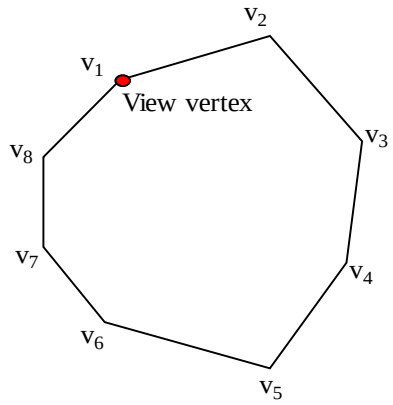


Fig. 5: In the convex polygon, v1 only can see v2 and v8.

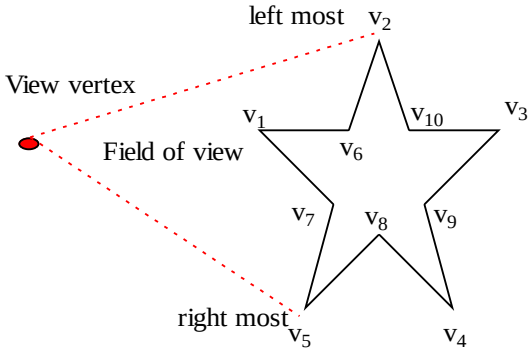


Fig.6: Judge the visibility of vertexes on the polygon, we only consideration the vertexes between left most and right most.

3. Leftmost/rightmost vertex and intersection

Based on the analysis above, we can make the visibility graph in three steps. In first step, compute the visibility graph in every polygon; in second step, for every vertex, find out the leftmost and rightmost vertexes in other

polygons; in last step, judge the visibility of this vertex and the vertex between leftmost and rightmost vertexes by judging intersection.

First of all, we introduce the method to judge which side a point *p3(x3, y3)* locates on a segment defined by *p1(x1 , y1)* and *p2(x2,y2)*. Determinant of the following matrix gives the answer.

$$\det(A) = \begin{bmatrix} 1 & x_1 & y_1 \\ 1 & x_2 & y_2 \\ 1 & x_3 & y_3 \end{bmatrix} \qquad (1)$$

If det(A)>0, p3 is on the right of the vector $\vec{P}_1\vec{P}_2$, vice versa. We don't care about the case that three vertex are collinear, that is det(A) = 0. Using this judgment, we can get not only the rightmost and leftmost vertex on a polygon, but judge the intersection of two segments.

We give the formal description on the algorithm for leftmost and rightmost vertex on a polygon:

Given a vertex *Vs*, and a polygon *P*(V_0,V_1,…,V_d),
Initial *leftmost* = *rightmost* = P_0 ;
For *I*:=*1* to *d* do
    If (*det*(*Vs*,*leftmost*,V_I)<=*0*)
        ***leftmost* := V_I ;**
    else If (*det*(*Vs*,*rightmost*,V_I)>=*0*)
        ***rightmost* := V_I ;**

Now, we just have to judge whether a segment defined by two given vertexes intersect a polygon.

We first compare the location of the segment and polygon simply, which will discard many polygons immediately. Then we can further give the result by computing the segment and every edge of the polygon.

Intersection of two segment can be deal with the det(A) as mention before. The detail just likes this, if two segments AB and CD have intersection (not include A,B,C,D), it must be A and B locate different sides of segment CD and

C and D locate different sides of segment AB. A segment don't intersect a polygon, we must have that it don't intersect any edge of the polygon.

4. Shortest path map

Given a source point S, we can generate a shortest path map based on the visibility graph. The data structure of SPM is: associated with each vertex *v* in the plane are the values

d(v) --- length of a minimal length path between A and *v*,

b(v) --- a vertex adjacent to *v* in the minimal path between S and *v*.

Hence, after SPM has been constructed, for a vertex *v* it is possible in O(1) time to determine the length of a minimal path between S and *v*, and by using the *b(*)* pointers, we can find the shortest path in time proportional to the number of edges it contains. Furthermore, for arbitrary destination point *x* of the plane that is not a vertex of any obstacle, a shortest path between S and *x* can be obtained by placing a straight line segment between *x* and the vertex *v* which can be seen by *x* such that the *d(x ,v) + d(v)* is minimum, and then following the *b(*)* pointers back to S and get the shortest path.

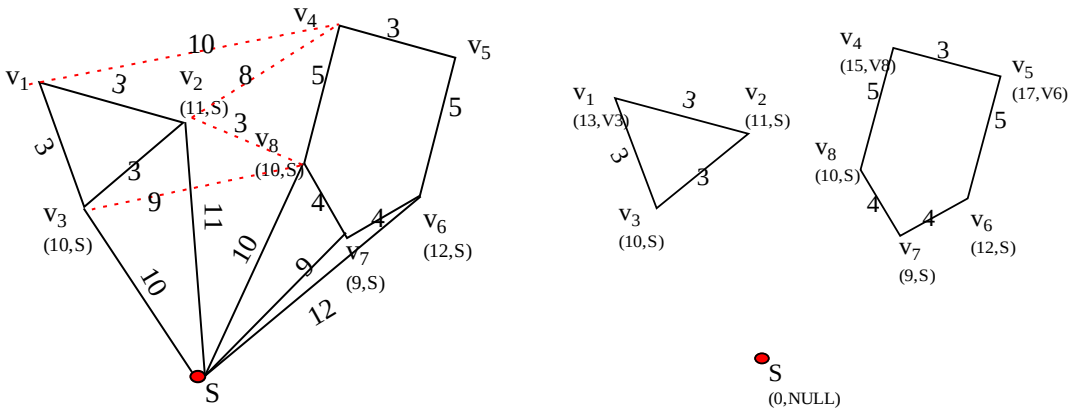
Following is the algorithm for generating SPM:

Initial *d*(S):= 0, for arbitrary vertex *v*, *d(v)* := $+\infty$**;**
Compute the list of visible points of source point S, and for every vertex *v'* in this list we have that *d(v'): = Distance(S,v')* (the distance of two points in the Euclidean space), and *b(v):=S*
Set Boolean variable *Change*: = TRUE;
While (*Change*) do
Begin
Change: =FALSE;
For each vertex *v* not in the visible list of S
For each vertex *t* can be seen by *v*
If (d(t)+Distance(t,v)<d(v))
Begin
d(v) := d(t)+Distance(t, v); b(v) := t ;Change := TRUE ;
End
End.

Thus, we will get SPM, and also we can consideration the destination as a whole. To consider the robust, the algorithm deal with the destination point separately, we only need to get the list visible points of the destination point.

Fig.7 gives a simple example to further explain the process of generating SPM.

Fig.8 give some results of our algorithm.



(a) (b)

Fig.7: (a) the mid-result just before the iteration. (b) final result of SPM.

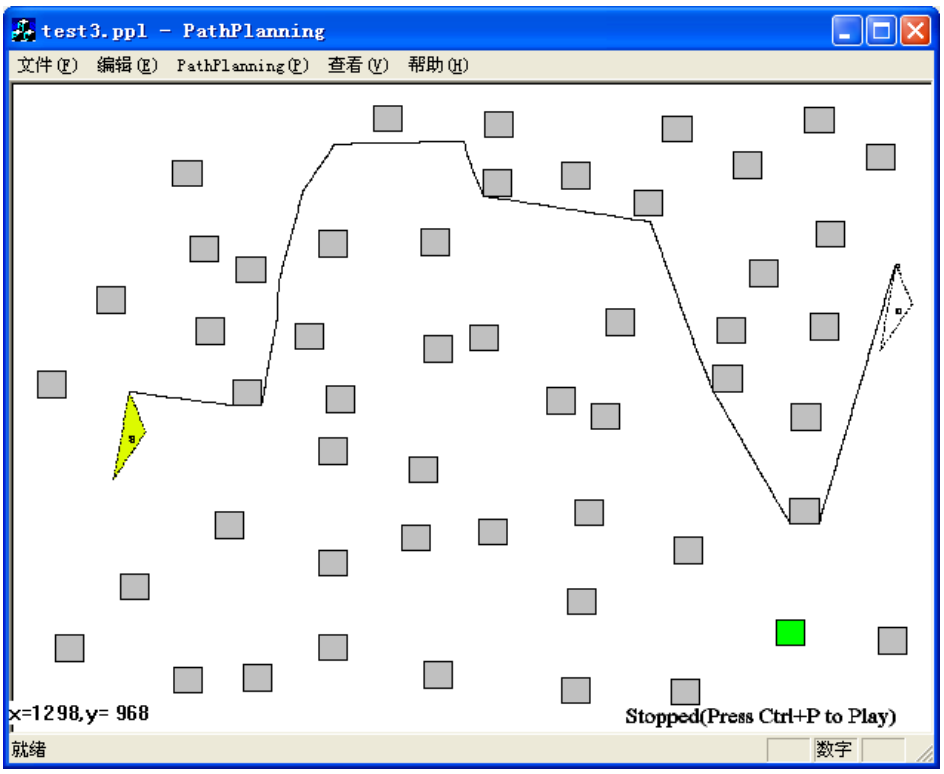


Fig.8 PathPlanning for 40+ obstacles costing 1~2 second

5. Conclusion:

This project is aimed to develop a path planning system for a translational robot using configuration space and extended visibility graph. The process will be divided into 2 parts. First, generate the Cspace of 2 degrees of freedom for the robot. Second, find the shortest path in the Cspace.

6. References

[LP] IEEE TRANSACTIONS ON COMPUTERS. VOL. C-32 NO.2 FEBRUARY 1983 LOZANO-PEREZ: SPATIAL PLANNING.

[SS] M. Sharir and A. Schorr, “On the shortest path in polyhedral spaces”, Proc, 16th ACM Symposium on the Theory of Computing, Washington DC, 1984, pp. 144-153

[SR] J. A. Storer, J. H. Reif. Shortest paths in the plane with polygonal obstacles. J. Assoc. Comput. Mach. 41 (1994), no. 5, 982-1012

[MS] M. Lanthier, A. Maheshwari and J.-R. Sack, "Approximating Weighted Shortest Paths on Polyhedral Surfaces", Technical report, School of Computer Science, Carleton University, 1996.