Guillaume Stamm

Local motion planing for robotics

Find a path for a simple polygon
through the smallest door

Éditions des Nik's News
www.niksnews.com/editions/

1998

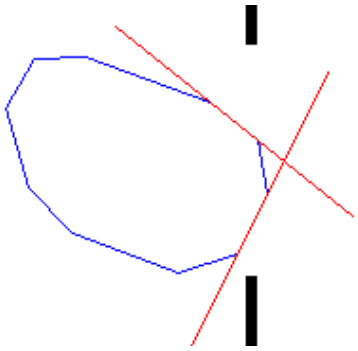
Presentation of the problem

The problem of motion planing is very large, and at this time widely open. There are just results in some very particular cases, like working only with convex objects, avoiding the rotations or no obstacles. I will present you here, some results on convex polygons motion planing, with as obstacle a simple door. But I will also give you the opportunity to look at what is possible with simple polygons. The given applet is a demo of some algorithms, and by manual motion you will be able to see the complexity of the problem for simple polygons.

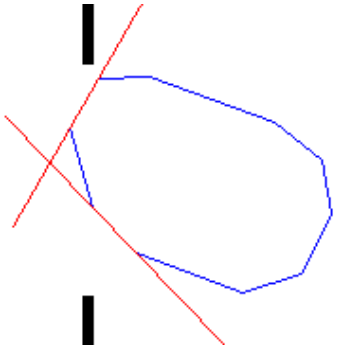
Working with convex polygons

For convex polygons the problem "the width of a chair" is completely soluble. And I'm presenting here the base principle and the algorithm.

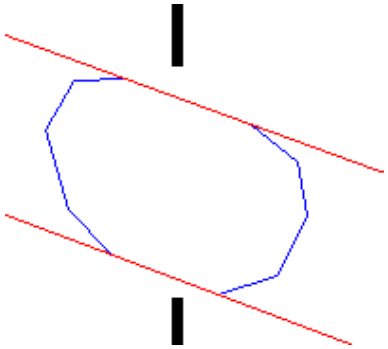
If we observe the two tangents of the polygon, where the imaginary line of the door cross the polygons, we can observe the following:



Before the polygon goes through the door,
the tangents are crossing on the right side of the polygon.



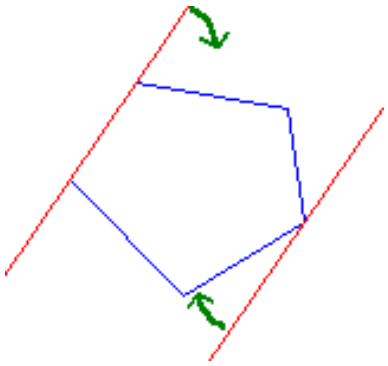
And after, the 2 tangents are crossing on the left side of the polygon.



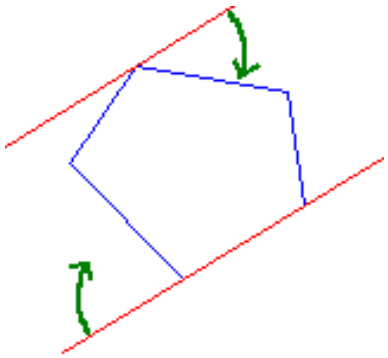
Therefore we are sure that at one time the two tangents are parallels.

If we can find these two tangents, it is then obvious, that with a simple translation parallel to these tangents, we will be able to lead the polygon through the door. Of course, only if that door is large enough. That's in fact the question: is the door large enough? or in other words is the polygon small enough? To answer this question we need to find two parallel tangents on the polygon, such that the distance between them is minimum: the rotating calipers algorithm will do that easily:

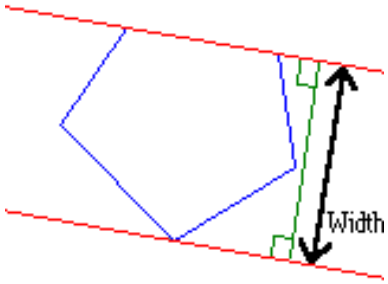
The rotating calipers is a very simple linear algorithm to compute the width of a convex polygon. The starting point of the algorithm is the most left and right points. Then the two straight edges will make a PI radian rotation around the polygon. At each step we check which side will "hit", and we keep trace of each distance between these two straight edges.



We start the rotating calipers algorithm on extreme vertex.



Then the two straight edges are rotating till one hits again the polygon.

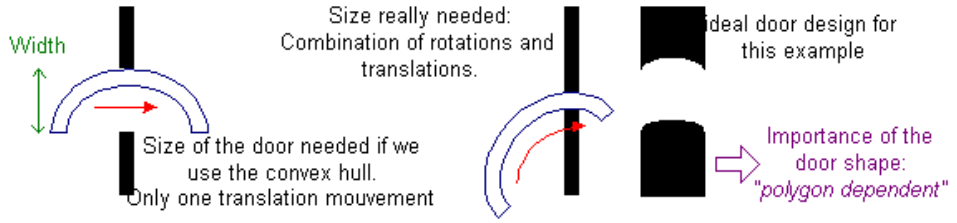


At each step we keep trace of the width founded.

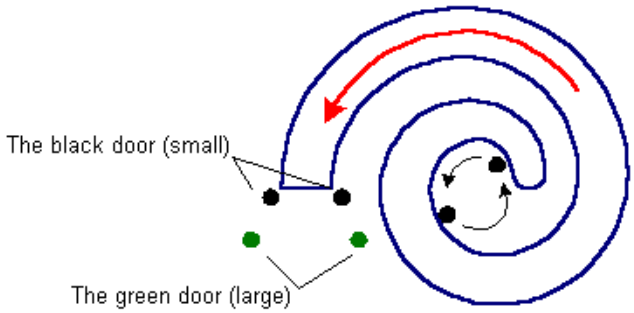
To the width of the polygon corresponds an angle *alpha*, angle that the two tangents are making with the perpendicular of the door. It is so recommended to make a rotation of *alpha* to reduce the size of the projection of the polygon on the door. Then a simple horizontal translation will lead the polygon through any door larger than the width of the polygon.

Working with simple polygons

What's the problem? Of course we can always approximate a polygon by its convex hull and do the same as before. That's right but in fact sometimes, if we consider the simple polygon it can go through a really smaller door (infinitely smaller at the limit). Look at this simple example:



This example also shows a new problem. The shape of the door becomes important. And there are many types of doors: we can think of the entrance in a long tunnel, two thin or large half lines, two points... Think for a while of the entrance in a tunnel. In this case, we know that once the door passed, the polygon will be entirely in the tunnel. Then the convex hull also has to fit with the size of the tunnel. So we can use previous methods. But that's the simple case, because it is also possible that a lager door will not let the polygon pass through it any more! Look at this example:



Here we are working with a two point door and surprisingly the smallest door let the polygon go through and not the larger one. Let me move the door instead of the polygon (same problem), you can see that the black door is able to wind itself on the polygon, then make a half turn in the center of the polygon, and finally unwind itself in the other direction. The green door is too large to make the half turn in the middle so it can not unwind itself in the other direction.

As you can see the problem is really complex if you don't make some approximations. To my knowledge there is no real geometrical approach of the general problem, even with a very high complexity. A good approach for this problem is not the geometric one, but the "robotic interaction" one. I mean trying to make any movement in the degrees of liberty we have and see what's happening. If it doesn't work try to change a little the position,... and so on. This approach works on a discreet set of movements, to have a finite number of them, that's again an approximation but which can give better results than the perfect geometric approach.

Presentation of the other algorithms

The single polygon data capture

In the applet you are asked to enter a simple polygon. In fact you don't have the choice because an algorithm checks that. For each click of the user it takes

a linear amount of time to verify that the actual chain is simple, it is difficult to think of something better. To know if two segments are crossing, I just compute the area of the four possible triangles and look at the sign of these areas. The problem consists in all the particular cases, when one of these areas is null. I have at first treated all these particular cases, but the algorithm began then to be very long, incomprehensible and with a high probability of being incorrect for some untestable cases. In an other project I'm making in computational geometry, boolean operation on simple polygons, there is an algorithm in witch I had to study 72 particular cases and each of them quite different from the others. Hence I decided to use an approximation principle.

The approximation principle

On the one hand, in physics we always use tolerance and precision notions, because in the real world physical objects cannot have a perfect metrics. On the other hand mathematics has the power of continuity, and can use perfect numbers. In all my algorithms I used two levels of precision: the precision needed for the IO, and another precision 100 times negligible in comparison with the IO precision. By IO precision we can think of whatever we want: the precision of one map as input, the size of a pixel on the screen as output, the minimum possible rotation for a motor driving a robot...

The idea is then the following: take all the inputs and disturb them randomly with your second order precision. Input and output will have the expected precision, and internally we are working with more precise data. This can, in the ideal case, remove all the particularity of any set of points. Of course if input and output are integers, we will loose a little time to work on floating point numbers (typically such number: xxx.00xxx), but in my case I was already working in float (needed for rotation abilities).

Of course the computers are working with bits, and a random number can give the same number twice. But look at this example: consider IO as the integer [0..1023] (typically pixels on a screen) we need 10 bits to code them, and the microprocessor abilities today allows us to work easily with 32 bits numbers. It means that there are 22 bits not used. Hence your random noise will have the probability of $1/2^{22} = 0.00000024$ to be the same for two points, (that's in dimension 1 because in dimension two the probability is $(1/2^{22})^2$) and then we will just have very rarely the creation of a particular position. I think that the probability to make an error in treating all the particular cases is higher than this one, and this method has the advantage of giving simple clean and comprehensible code for the algorithms.

And the last point is that by treating all the particular cases you have to introduce a huge number of tests. Theses tests are translated in assembly

language as branches, and the branches are difficult to handle actually for pipelined microprocessors. Hence I think that by treating all the particular cases, the speed of the algorithm will slow down.

Computing the convex hull in linear time

As the user enters a simple polygon, and the computer needs a convex one two be able to work alone, I have used a simple implementation of the very fast linear algorithm from Avraham A. Melkman. This algorithm doesn't take into account the particular cases I have just spoken about. If you are interested in you can see the project from Pierre Lang on it.

Further developments and improvements

There are lots of interesting things to do now, but I'm unfortunately short of time. Especially the study of the 3D case where there are again some good results on convex polyhedron. But it is obviously more difficult than the 2D case that is already as we have seen quite difficult. For example there are surprising polygons whose shadows doesn't fit to the door, but that can go through it with only two translations.

The second interesting development is to implement in 2D the interacting approach of the problem to see if it can works well. Because then there are direct applications in robotics, if it doesn't work too slowly. It is also possible to think of a combination with the method for convex polygons as a first approach.