Homework 2 Report

(CS6244 Robot Motion Planning & Control (Prof David Hsu))

Irvan Jahja
A0123879R
dolphinigle.mailbox@gmail.com

October 8, 2014

1 Preliminaries

I assume the polygons do not overlap – my algorithm perform a precomputation that merges overlapping or touching polygons together (in particular it may form a polygon with hole(s)).

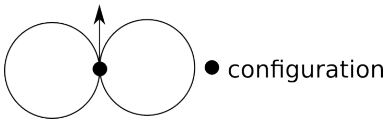
2 Algorithm idea

The algorithm is based on extending the *visibility graph* [1] into non-holonomic constraints using the following two (not necessarily correct) observations:

- The shortest possible path from start to goal must consist of straight lines and arcs whose radius is exactly the turning radius T of the robot.
- There always exists a path such that all of its arcs touches a vertex of the obstacles (with few exceptions – later on this).

The first observation can be proven. The second is not complete (instead of touching a vertex, the arc may touch two edges instead. However incorporating this into the algorithm is too time expensive).

With these two observations, I modify the visibility graph as follows. For each vertex of the polygon, I create circles that touches the vertex (we will explain how I choose the circles later). In addition, for both the starting and goal positions, I create circles that touches the positions as well. In particular, we create the four circles that are tangent to either the starting or goal configurations. The following figure shows the two circles that are tangent to a particular configuration:



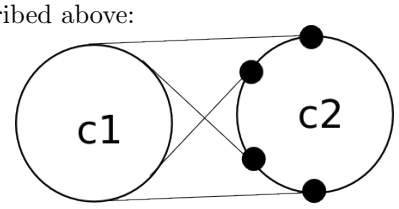
The second group of circles are the exception I mentioned in the observation – it may be required for the robot to adjust its orientation in the beginning and at the end of the route.

I now define our modified visibility graph $G = (V, E)$. A node in V is a position and orientation of the robot in the plane – in particular, the robot will always be positioned and oriented on the boundary of a circle (notice that this is true for the starting and goal since I created circles that are tangent to both of them).

However, I restrict the set of nodes to be a subset of all possible such configuration:

- The starting and goal configurations are in V.
- For each ordered pair of different circles c_1 and c_2, the four points on c_2 tangent to the four possible different tangent lines connecting c_1 and c_2 are in V.

The following picture illustrates the four points as described above:

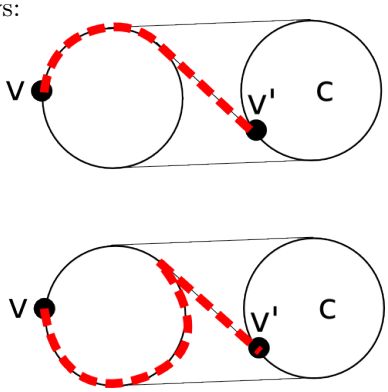


Thus, if I have C circles, we will have $2+C(C-1)$ nodes.

For every node $v \in V$, consider another circle c. Now, consider a particular tangent line connecting

the circle of v and circle c – the point on circle c on which this tangent line touches is a vertex v' in V. There is an edge from v to v' if it is possible to go from v to v' following the curvature of the arc of the circle of v, followed by the following the tangent line connecting the two circles (that is, both of them must be clear of obstacles). The weight of the edge is equal to the sum of the lengths of the arc and the tangent line. The two possible such paths are illustrated as follows:

For a particular tangent line, there are two choices of arc. If both of them are free of obstacles, the shortest is selected. They are illustrated as follows:



3 Picking circles

I iterate the algorithm in levels – higher level will run slower but produce more circles. In level i, I produce approximately 2^i circles for each vertex of the obstacles. Consider a vertex p of the obstacle, and consider the circle c centered at p with radius T. For a circle to touch p, its center must lie in the circle c. Thus, I pick 2^i points on c and let them be the centers of the circles. I select equidistant points for the centers.

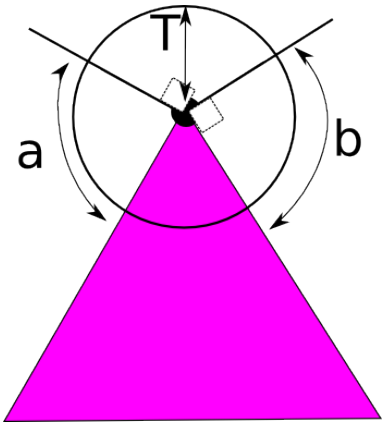
4 Complexity

The overall complexity of the algorithm is at most $O(N \times C^3 \log C)$, where C is the number of circles and N is the number of vertices on all obstacles. Thus, at level i, the complexity is at most $O(N^4 \times i \times 2^i \log N)$.

5 Optimizations

5.1 On picking circles

I do not pick circles touching reflex vertices, following the intuition from Homework 1. In addition, I only pick circles from a specific range as illustrated below:



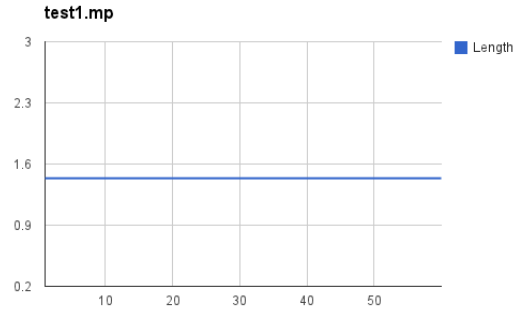
I do not pick circles centered in the arc marked by a and b. This is since it is impossible for an arc belonging to the circle to touch the vertex.

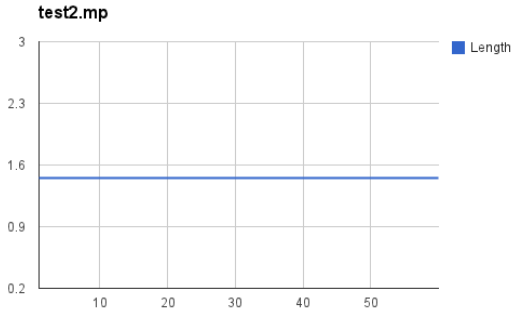
5.2 A*

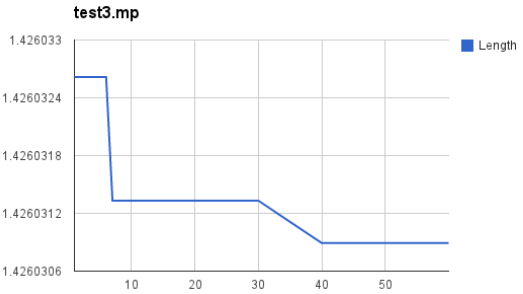
I use A* to search for the shortest path in G. I used the distance according to the normal visibility graph for free-flying object as the heuristic. In the given test cases, it usually narrows the number of node expansion to one tenth the number of expansion were I to use the Euclidean distance between the configuration and the goal position as the heuristic.

6 Experimental result charts

The following charts show the results of the experiments on the six test cases. The horizontal axis denotes the amount of computation time, and the vertical axis denotes the length of the path found.







For the fourth and fifth test cases, the algorithm is not able to find any path within a very small amount of computation time – this is the reason of the missing line in the beginning of the chart.

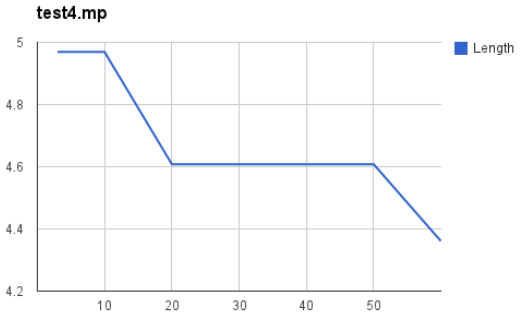
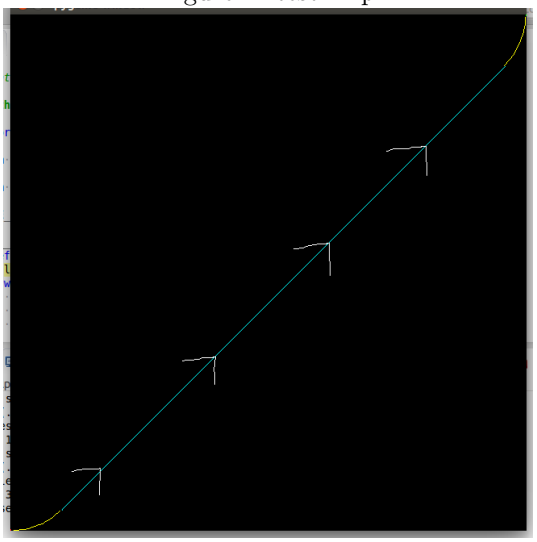
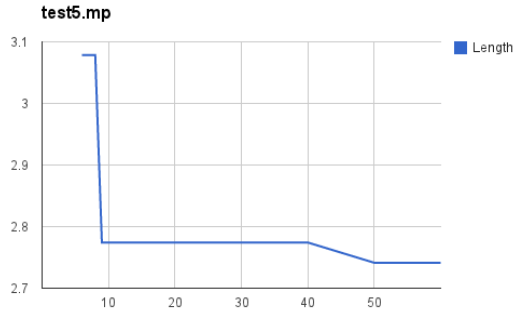
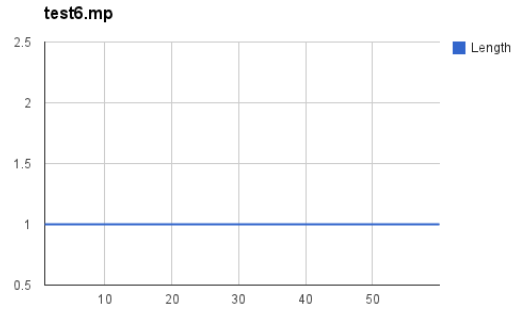


Figure 1: test1.mp







7 10 seconds results

Figures 1 through 6 show the found paths when given 10 seconds of computation time. The path are colored blue and yellow – blue paths are straight lines, while yellow paths are arcs with radius equal to the turning radius.

Figure 2: test2.mp

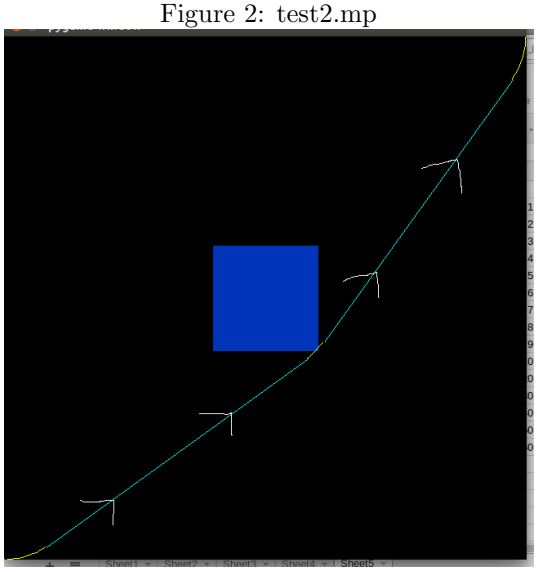


Figure 3: test3.mp

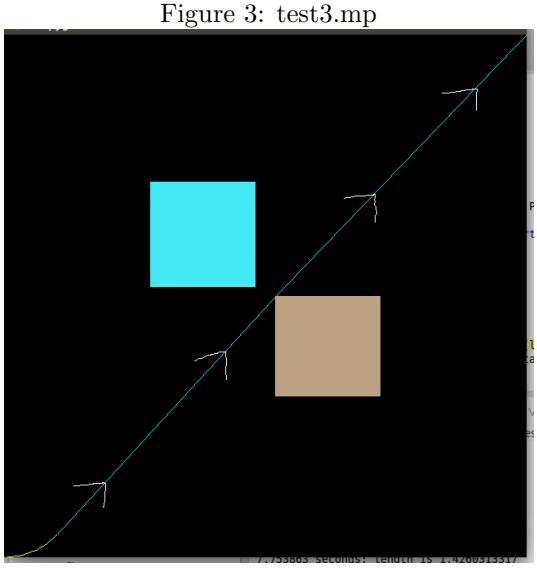


Figure 4: test4.mp

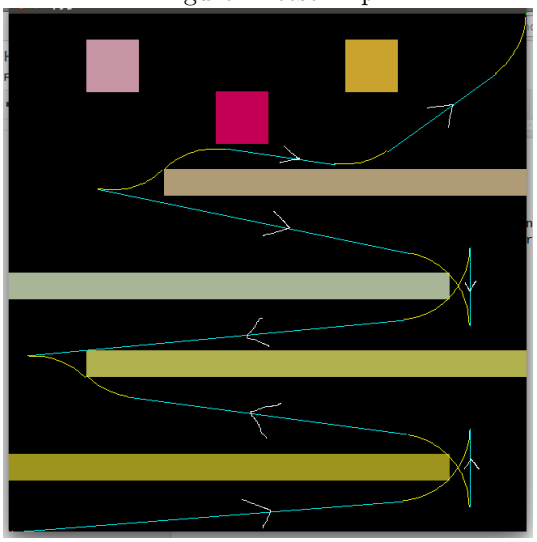


Figure 5: test5.mp

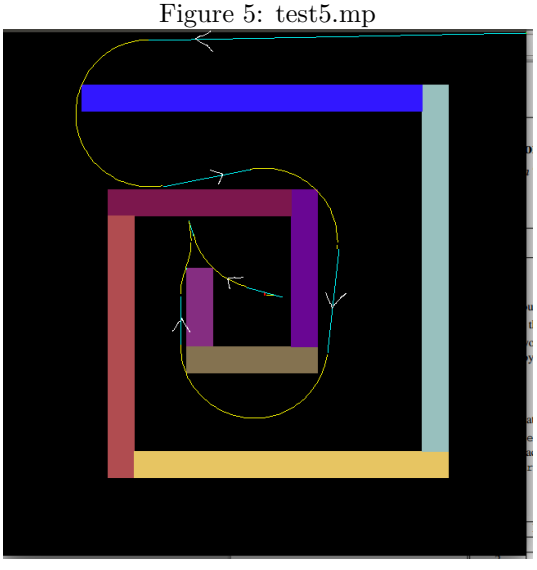
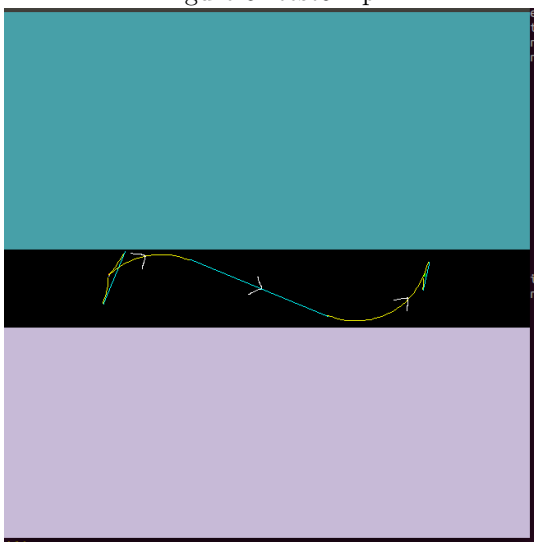


Figure 6: test6.mp



References

[1] T. Lozano-Pérez and M. A. Isley, An algorithm for planning collision-free paths among polyhedral obstacles, *Commun. ACM* **22** (1979), 560–570. Addison-Isley, 1974.