1.Problem description

From the requirements of this homework, we are required to setup a 10*10 environment and a 2*2 obstacle in the environment and then move our robot from the starting points to point1 and point2.

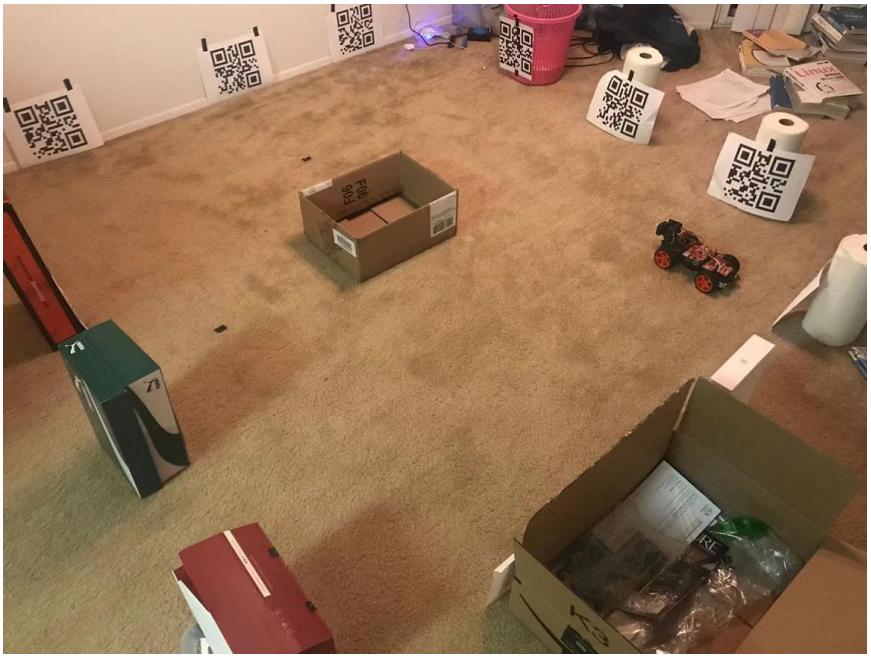


Figure 1. Environment

From the above image, you could see how I place different QR codes and the obstacles. The two pieces of black tape represents the two stop points.

2.Configuration space

As the professor said in piazza, I set (8,1.5) to the starting point, (5,8) to the point1 and (2,5) to the point2. I use Matlab to draw a diagram as follow:

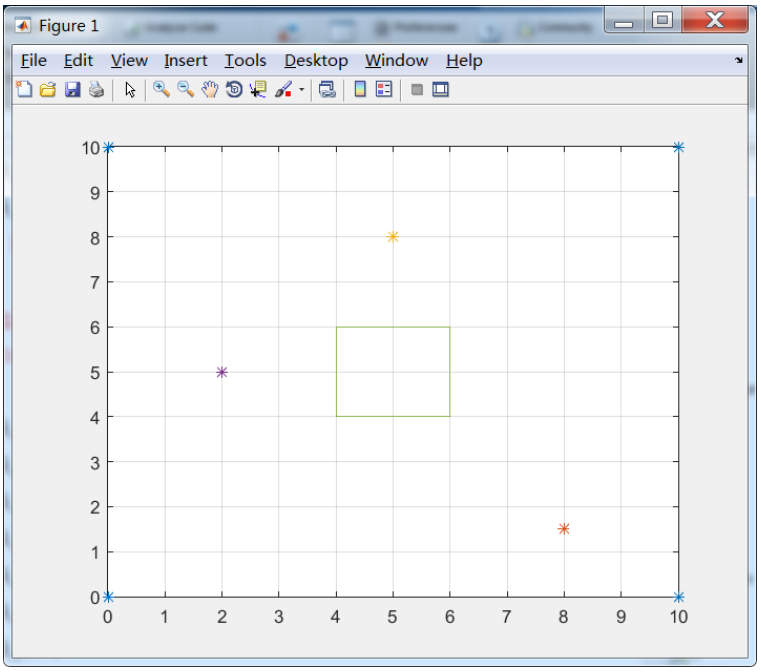


Figure 2. Diagram of workspace(only points)

However, in real world, our robot could not be considered as a point. Hence, I introduce the concept of configuration space which makes it possible for us to consider our robot as a point, greatly simplifying the mathematics model.

Configuration space contains all the points that are reachable for our robot.

In order to find the configuration space, I need to know the physical size of our robot: 20cm*12cm, which is equivalent to 0.65ft*0.39ft.

First, I assume that our robot is being placed in such a way that it heads forward, and its orientation is along the positive direction of axis y(the left image).

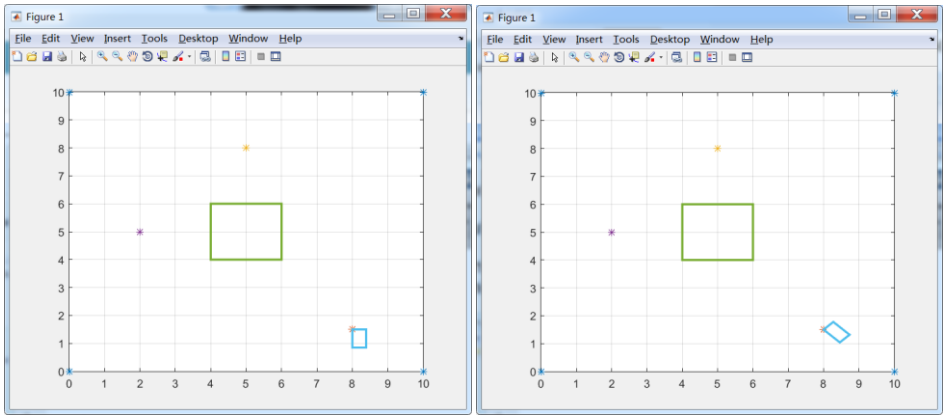


Figure 3. Diagram of workspace(with real robots)

The left image shows the initial conditions when the orientaion of the robot is 90°

and the right one is for 135°.

I assume that the orientation of our robot is unchanged and it is easy for me to find the configuration space:

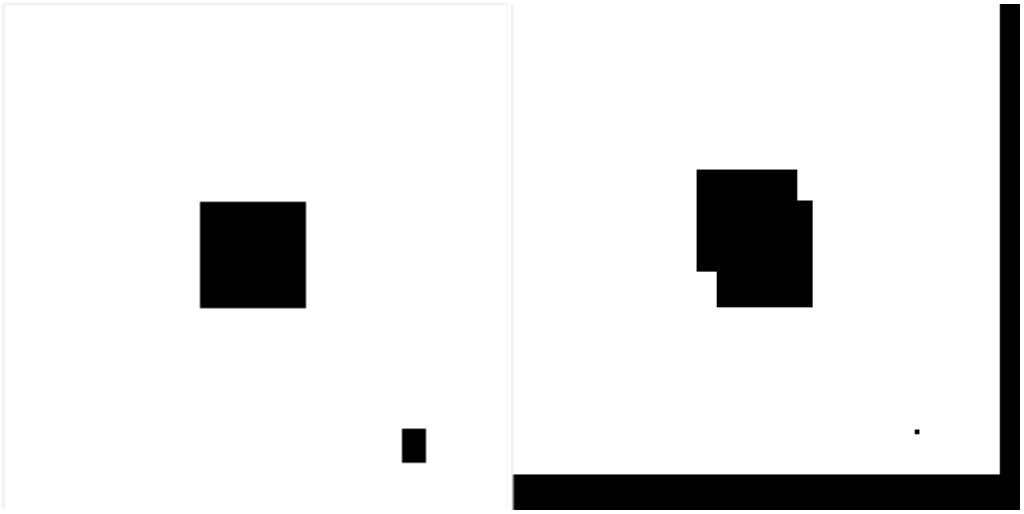


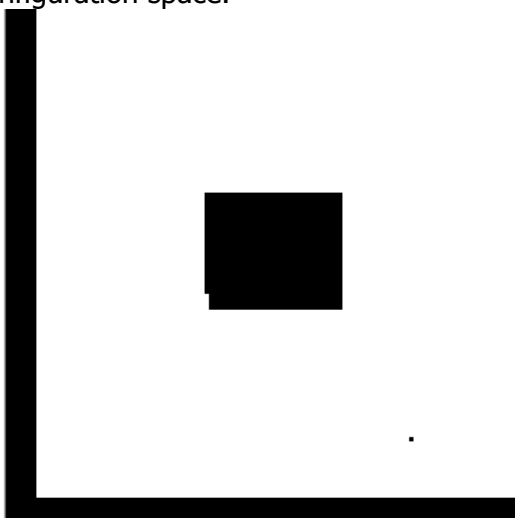
Figure 4. Diagram of configuration space(90°)

The left image describes the workspace and the blace parts represent the obstacle and our robot respectively.

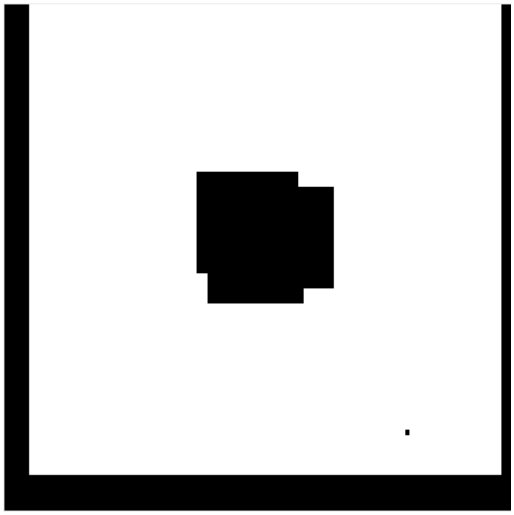
The right one describes the configuration space. And In configuration space, our robot could be considered as a point(the black point).

As you can see in Figure 2, each pixel in the images corresonds to the left-upper corner of our robot. Hence, the configuration space(white parts in the right image) makes sense.

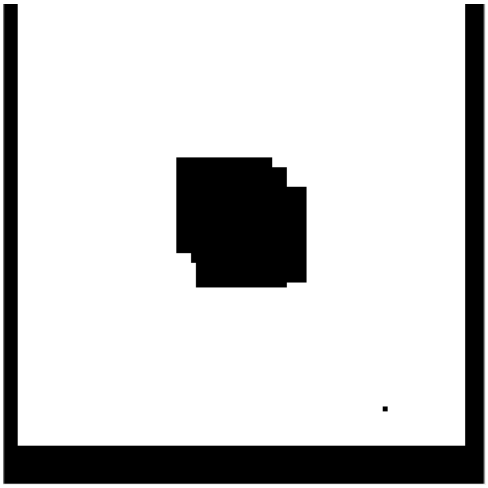
However, the orientations of our robot could also limit where it could reach. In order to take the orientations into account, I discrisize it and choose 0,30,60,90,120,150, 180 degrees. And for each orientation, I could get an image representing the configuration space.



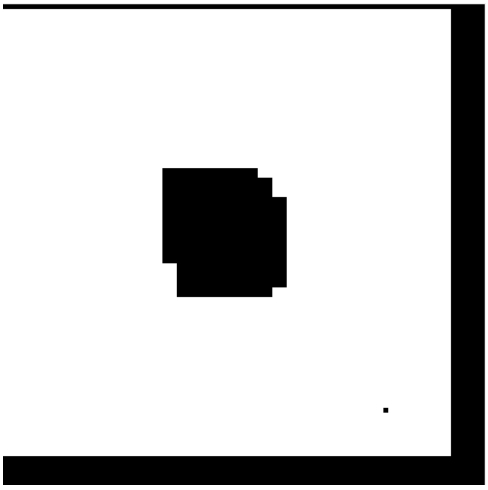
0°



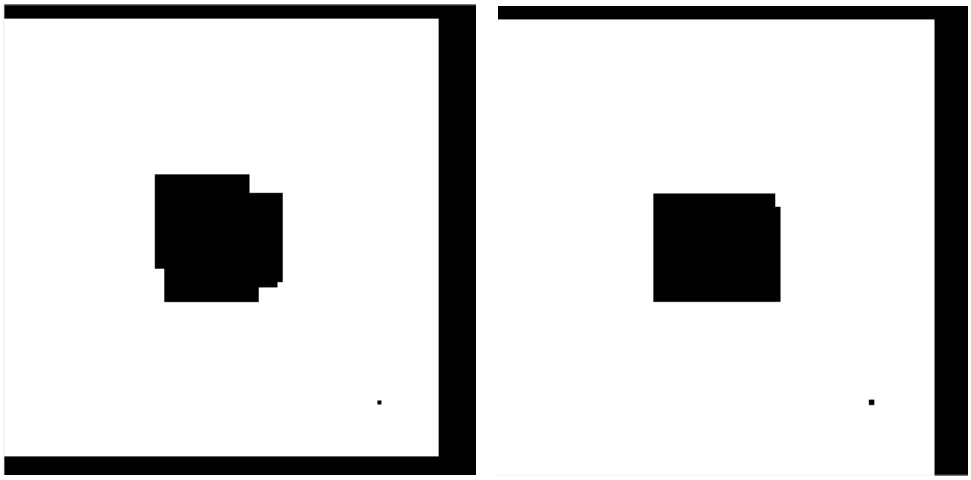
30°



60°



120°



150° 180°

Figure 5. Diagram of configuration space(various orientations)

Right now, I have seven images for seven different orientations. What I do next is to find a way to combine all these configuration spaces into one image in which points are reachable for our robot for any orientations.

I assume that it seems a little stupid because our robot could only have one orientation at a time, so it is unnecessary to combine all of these images and the right way should be finding the configuration space when our robot is moving(This algorithm should be running in real time). However, the main purpose of this homework is to get us familiar with the basic motion-planning methods and the processor on Picar is not good enough to run it in real time(Also, from the previous three homeworks, I know that it is hard to get the precise positions and orientations when Picar is running).

With all of these reasons, I combine the seven configuration spaces and search for a feasible path in that configuration space before our robot begins its journey.

The following image represents the the configuration space after I combine these individual ones. The three black points represents our robot, 1st stop and 2nd stop.

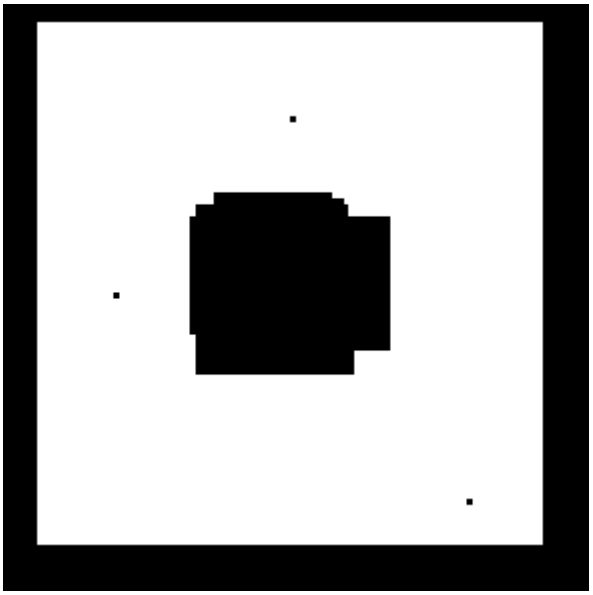


Figure 6. Diagram of combined configuration space

Right now, I will show how I design a voronoi based graph planner.

3.voronoi based graph planner

Voronoi graph could help us find the maximum distance between any point and its nearby obstacle(or its boundary). Hence, from my perspective, the path along the voronoi graph should be the safest way.

First, I need to extract all the vertices of obstacles and boundaries from Figure 6 for further use. Figure 6 is used for visualization but not for computation.

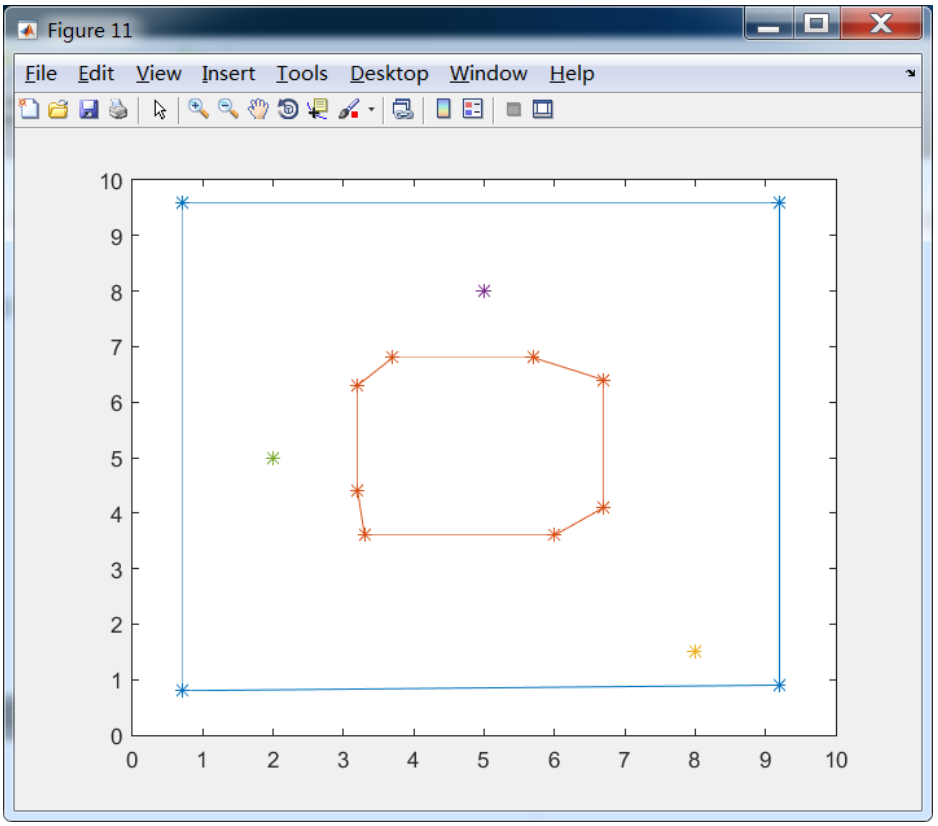


Figure 7. Vertices of obstacles and boundaries

Compare Figure 7 and Figure 2, you can see how configuration space works.

Because in this case, there is only one obstacle in the center, the voronio space could be easily determined. I use Matlab to find the voronio space and the intersection lines of the five parts with different colors are the voronio graph in this case.

I explain the different parts in Figure 7 here:

1.The points in the red part are closest to the left wall in the configuration space.

2.The points in the green part are closest to the right wall in the configuration space.

3.The points in the yellow part are closest to the upper wall in the configuration space.

4.The points in the blue part are closest to the lower wall in the configuration space.

5.The points in the pink part are closest to the obstacle.

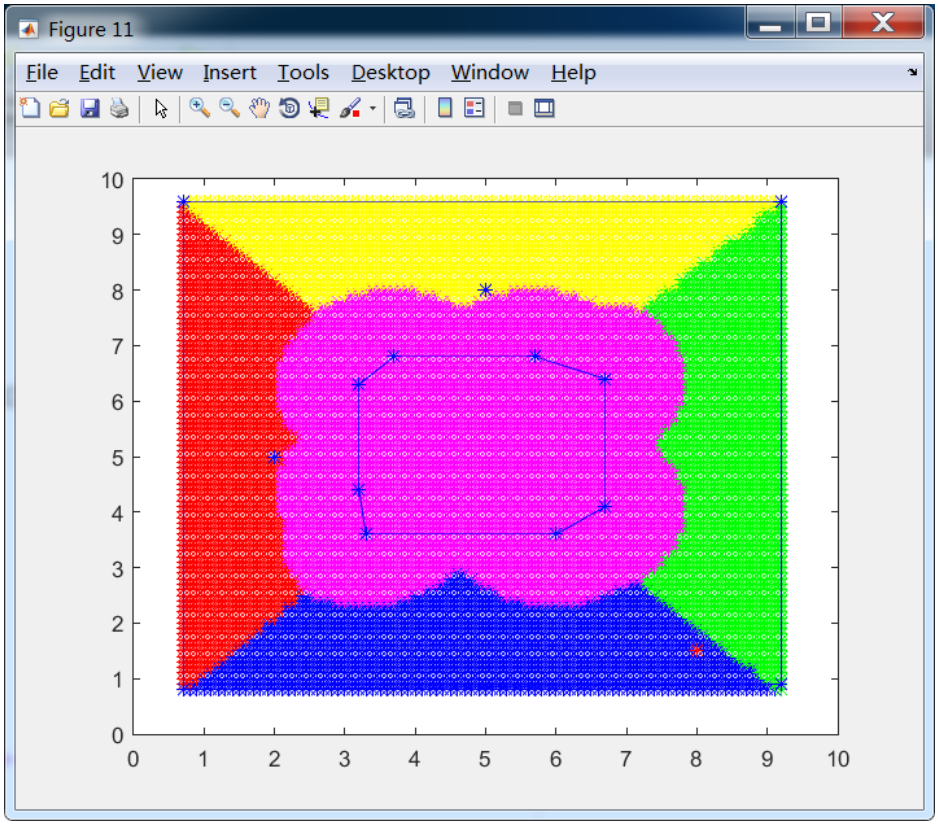


Figure 8. Voronoi graph

Then, I extract all the points which are on the voronoi graph.

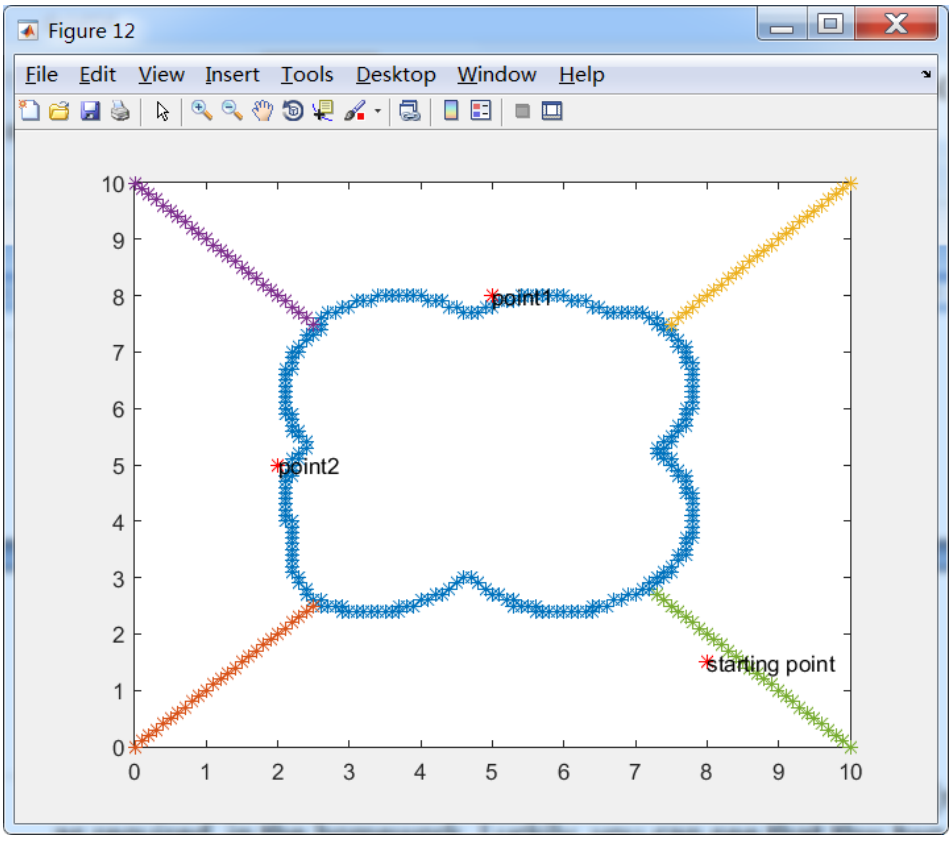


Figure 9. Points on the voronoi graph

With these points, Picar could move from (8,1.5) to two points(red points in Figure 8) as required in the homework. Luckily, you can see that the two points are really close to the extracted edges.

Finally, I only have to find the path from the voronoi grapg shown in Figure 9. For this part, It is very easily because I know all the points on the graph and if I implement a greedy searching strategy, there must be a feasible for which our robot is approaching the target point gradually.

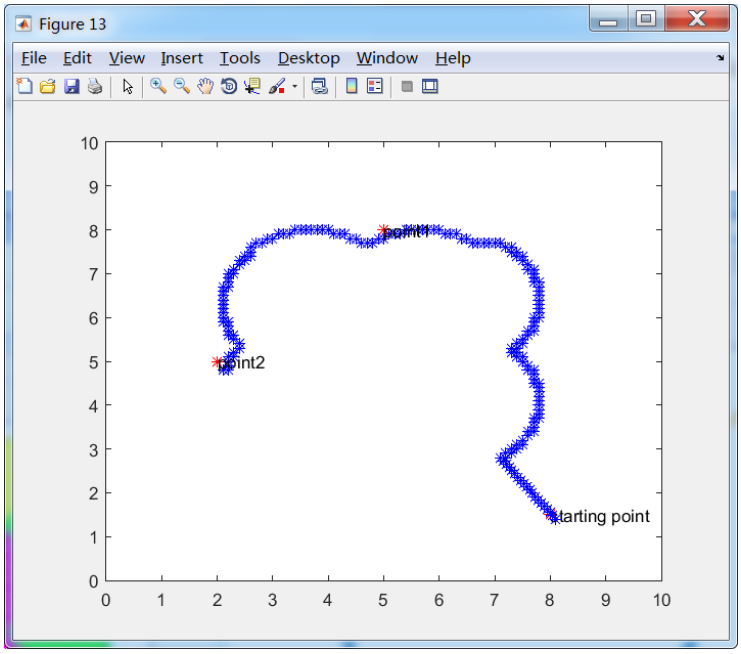


Figure 10. Motion path

However, there is still one significant drawback. You can see that there are three pieces recessed in which makes physically impossible for our robot to follow the path at those points. So, indeed, the real motion path should be more like:

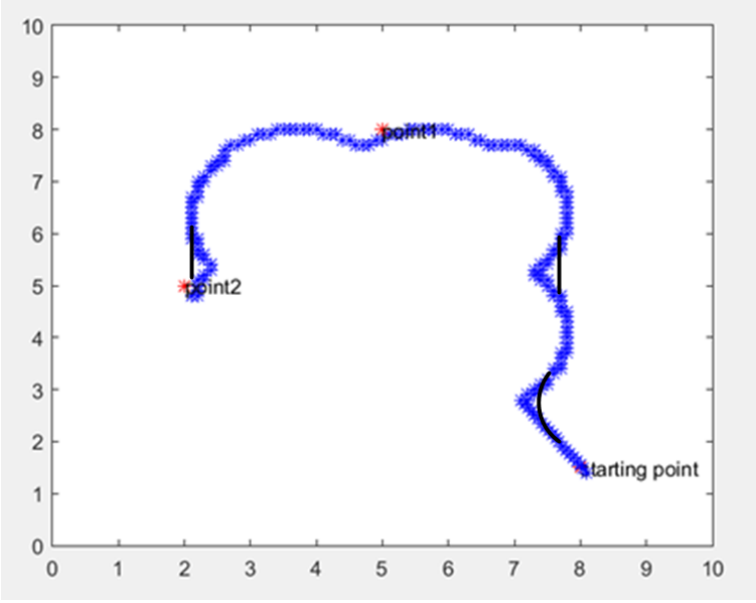


Figure 11. Real motion path

After I enter all the points into Picar, I could use the camera and the QR codes for localization, which means I want Picar keep moving on the path consisting of these points(It is much easier than what I did in homework3 because I know exactly where I place these QR codes).

I save these points and write them into a txt.file named path_voronoi.txt.

4.Visibility graph based path planner

Visibility graph is defined for polygonal obstacles. In this graph, nodes correspond to vertices of obstacles and there are two requirements if we want to connect two nodes:

1. They are already connected by an edge on an obstacle.
2. The line segment joining then is in free space(configuration space).

If I include the start and goal nodes, they could also be connected if the two above requirements are satisfied.

Besides, I find a lecture about motion planning from CMU that concludes not only is there a path on the roadmap, but it is the shortest path.

With the instructions from that lecture, I try to draw a visibility graph based on the configuration space shown in Figure 12.

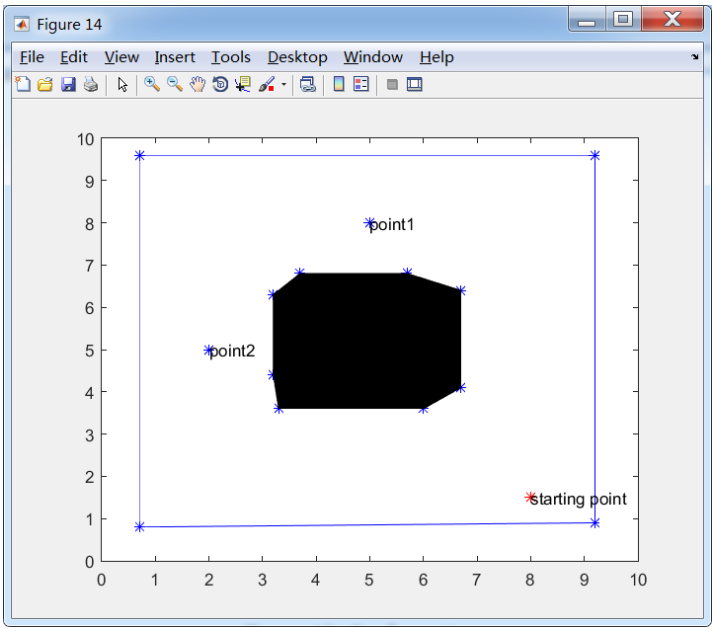


Figure 12. Configuration space

1. Draw lines of sight from the starting point and the other two points to all visible vertices and corners of the world.

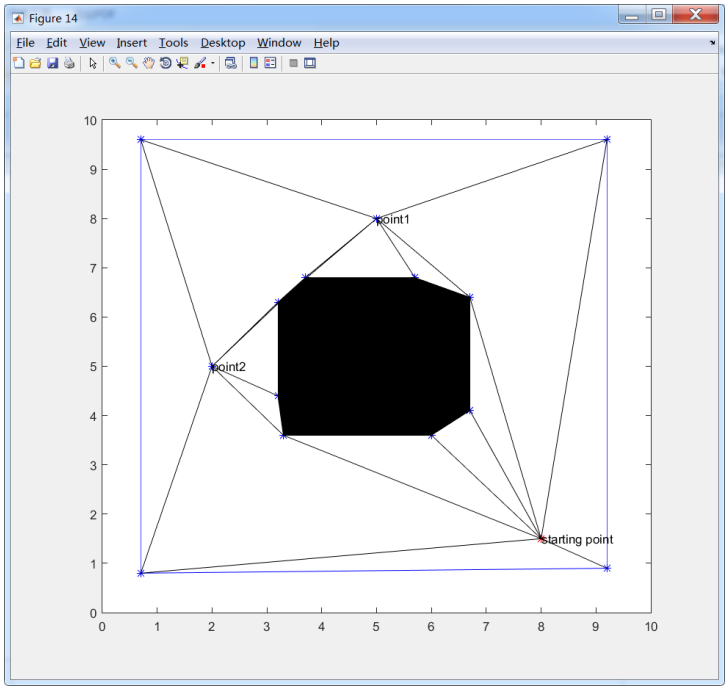


Figure 13. Visibility graph after step 1

2. Draw lines of sight from every vertex of the obstacle and corners of the world. And lines along edges are also lines of sight.

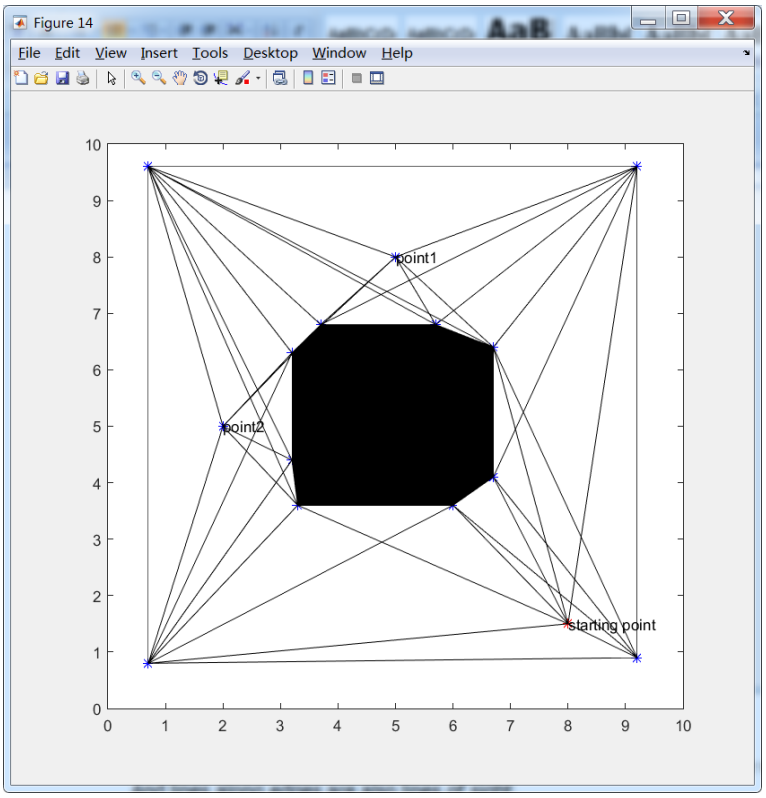


Figure 14. Visibility graph after step 2

After step 2, the visibility graph has been successfully built as shown in Figure 14.

Then, I need to find a path which connects the start point, point1 and point2.

I divide the path into two subpaths connecting start point and point1 and point1 and point2, respectively.

For the first one, you can see there are multiple feasible paths connecting start point and point1, for example:

- starting point->lower-left corner->upper-left corner->point1
- starting point->vetrices of the obstacle->point1
- starting point->vetrices of the obstacle->upper-left corner->point1
- some other paths

In order to simplify the problem, I introduce an assumption that the path connects the fewer points, the length of it is smaller.

From Figure 14, you can see that the minimum number of such points are three and there are two such paths.

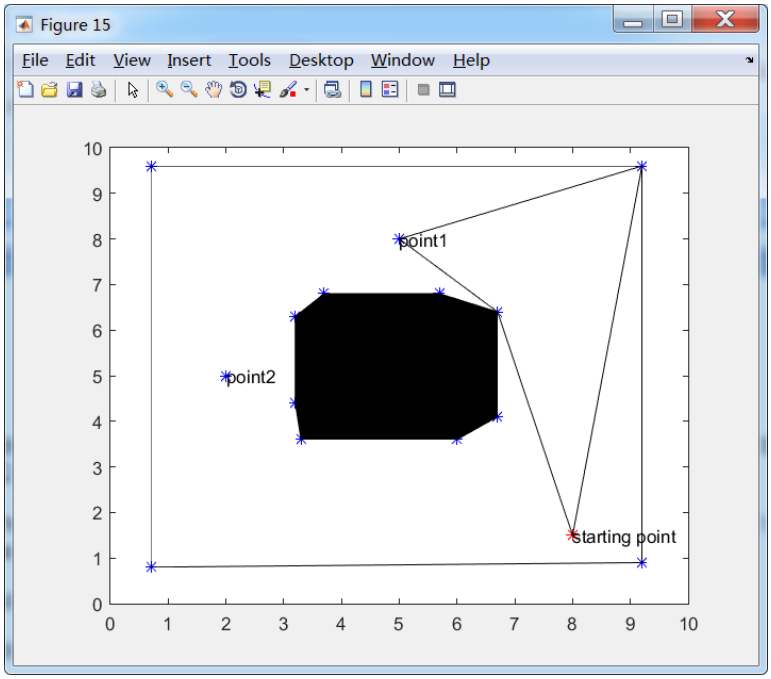


Figure 15. Two selected paths

Since I know everthing about these two paths, I could compare the length of each path to find one with smallest length. With the assumption I introduce before, the path that connects the fewset points and has smallest length is the optimal solution for this problem. In other words, I find the best path connecting start point and point1. It is obvious that the best path is the left one shown in Figure 15.

I could use the same method to find the best path connecting point1 and point2. From Figure 14, you can see that the minimum number are three and there are two such paths. It is obvious that the best path is the lower one shown in Figure 16.

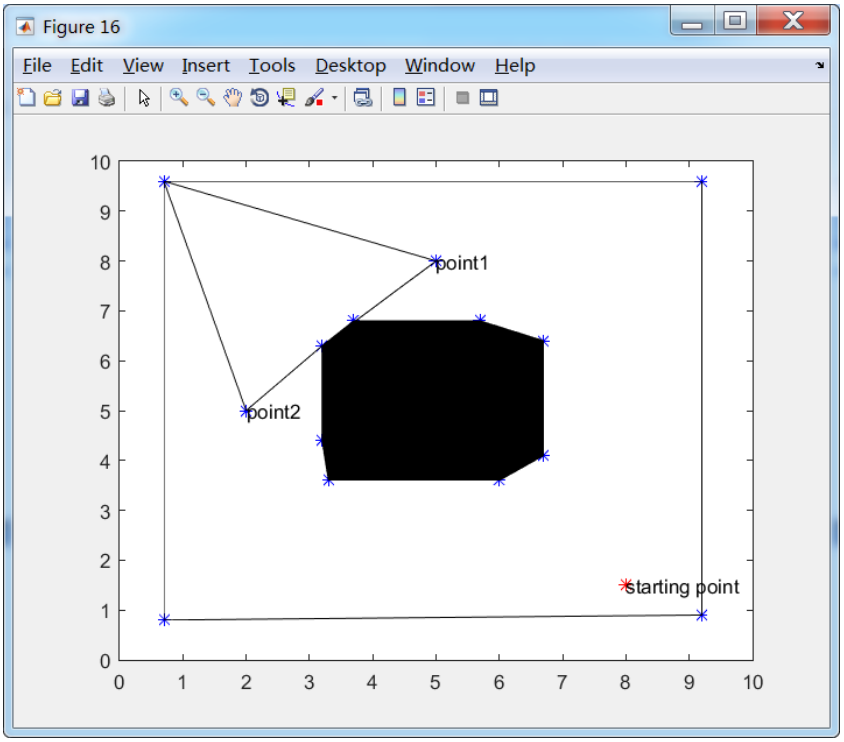


Figure 16. Two selected paths

Finally, I combine the two subpaths into one full path and this path is the optimal one connecting starting point and point2.

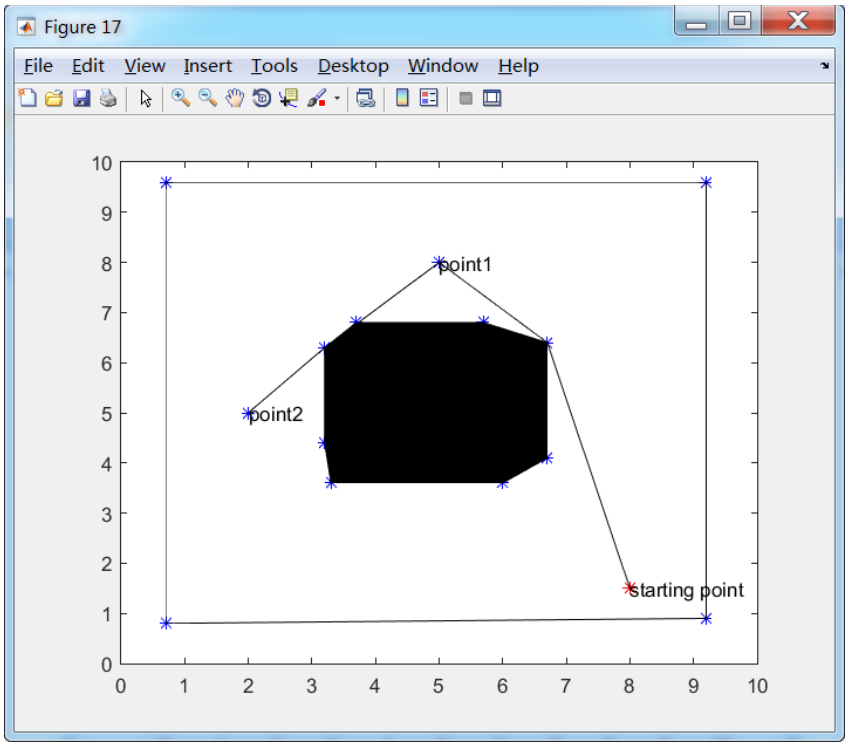


Figure 17. The optimal path

I extract the points on the path and write them in a txt file named path_visibility.txt.

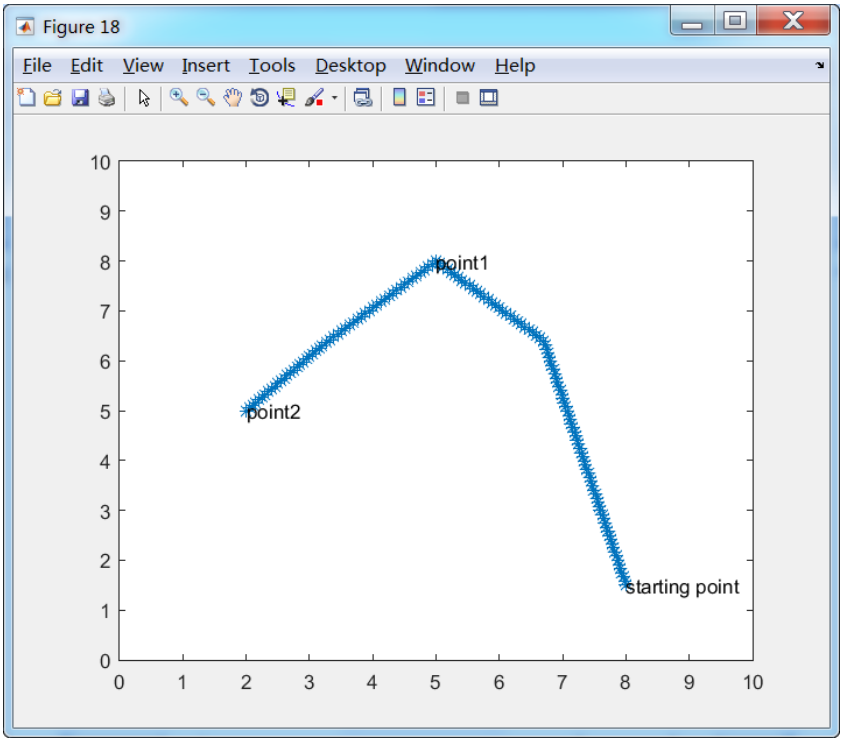


Figure 18. Extracted points from the optimal path

Due to the same reasons I mentioned before, the real motion path could be more smooth.

Traditional, if I don't introduce the assuption which makes the problem easier, I could use some other algorithms like Dijkstra's algorithm or Floyd's algorithm or simulated annealing algorithm(one of heuristic algorithms). Because this problem could be converted into TSP(Travelling salesman problem) with small scale and Dijkstra's algorithm, Floyd's algorithm or Floyd's algorithm could deal with TSP efficiently.

For TSP, it is a common practice to use a matrix to represent the connectivity between different nodes. If point1 is connected to point2, the corresponding element of that matrix is one and this kind of matrix is called unweighted graph. If the corresponding element is not only endowed with a number but also the number is the distance between the two nodes, this kind of matrix is called weighted graph.

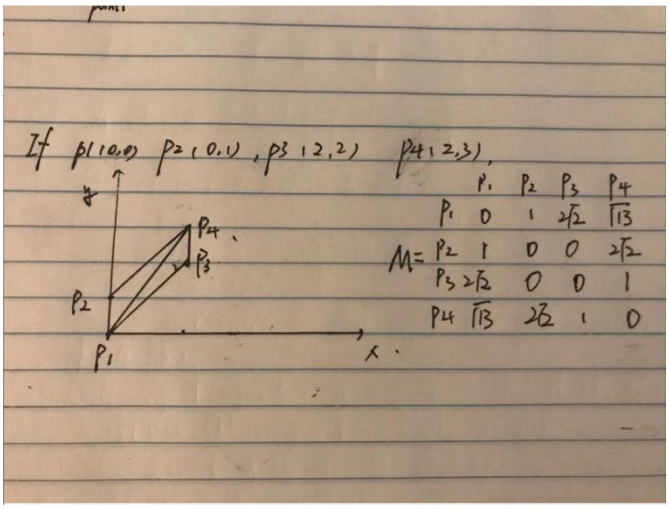


Figure 19. demonstration of weighted graph

After I get the matrix M for this problem, I could use the the algorithms above to find the optimal path and the principles behind these algorithms are well studied in the graph theory.

In practice the distance between two unconnected nodes is infinity(inf in Matlab).

17.1400	16.8400	32.8400	46.2400	Inf	Inf	42.7600	33.2900	Inf	Inf	Inf	Inf	Inf
36.5000	Inf	Inf	Inf	Inf	35.9300	14.6000	19.2100	Inf	Inf	Inf	Inf	53.7800
Inf	Inf	Inf	36.5000	16.4900	17.5300	Inf	Inf	Inf	Inf	Inf	Inf	1.8000
Inf	38.0900	20.0900	16.4900	36.5000	Inf	Inf	Inf	Inf	Inf	Inf	Inf	67.0500
Inf	Inf	Inf	25.7000	8.4500	8.4100	26.5000	Inf	Inf	53.7800	1.8000	67.0500	Inf
6.1300	3.1300	1.9300	5.4500	Inf	Inf	Inf	Inf	21.0500	Inf	Inf	20.2000	Inf
3.1300	6.1300	Inf	Inf	Inf	Inf	3.6500	1.8000	22.8500	19.3300	Inf	Inf	Inf

Figure 20. Part of matrix M in Matlab

I use Dijkstra's algorithm to find the optimal solution and you can see that the results are the same. And this justifys the assumption I introduce before.

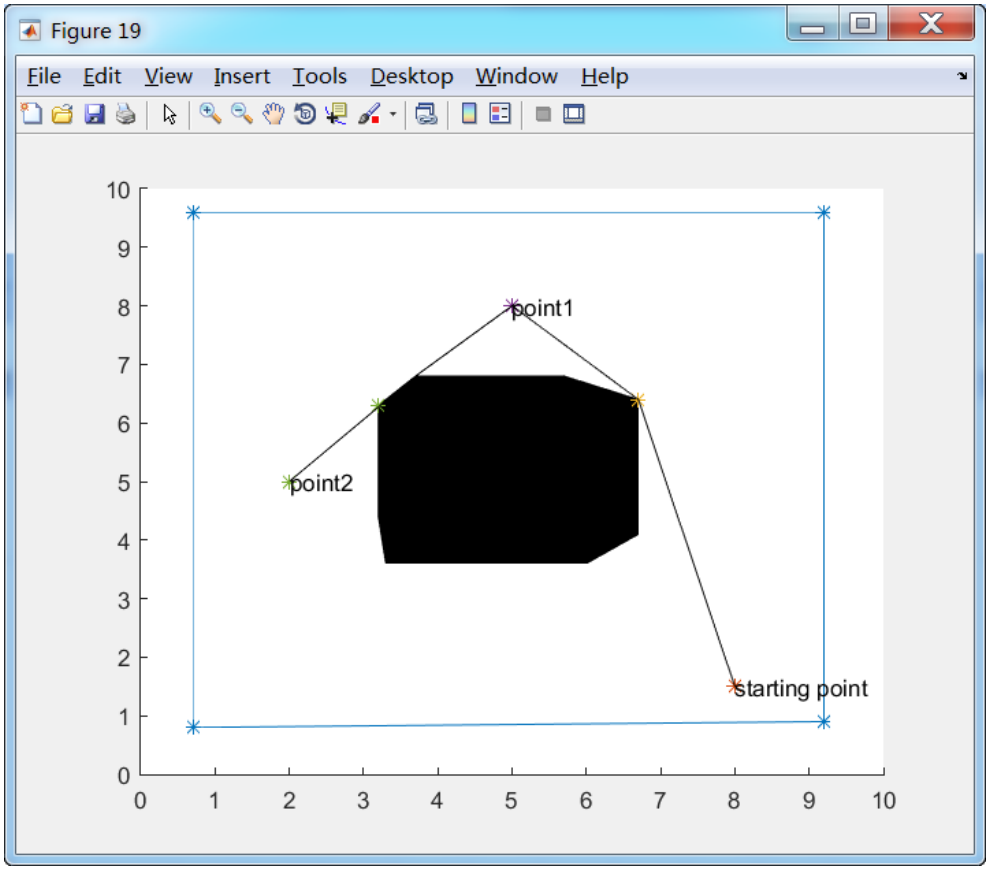


Figure 20. The optimal path(Dijkstra's algorithm)

The length of this path is(ft):

```
The smallest length:
    40.4100
```

Figure 21. The length of the optimal path

There are in total five files containing all the codes and results for part2,part3 and part5. Here is a list of them:

- cse276A_hw4.m
- cse_276A_robot_ori.m
- Dijk.m
- path_visibility.txt
- path_voronoi.txt

5. Difference in execution between the two methods

As you can see in the video, the performance of our robot with voronoi based path planner is better than that with visibility graph planner. Besides, when I was programming, It took me more time to tune some parameters for it. Sometimes, our robot just crashed into the obstacle while the right action should be moving along the edge of the obstacle and turn right/left at the corners. For the first one, with voronoi based path planner, our robot would not get close to the obstacle. Instead, it would always try to avoid any contact with any obstacles(in this case, they are the walls and the obstacle). Hence, although the motion path is not totally the same as the path I get from Matlab, our robot would not crash into the obstacle. To see a robot crashing into the obstacle many times is really a disaster.

The main reason behind this problem is that our robot(Picar) can not reach any points as we want. The precision of its movement is not good enough for us to expect it following all the points all the time. And since visibility graph planner could provide us with a motion path that at some points, our robot is really close to the obstacle(vertices or edges), it could make lots of troubles. And, in this case, what could help me measure the magnitude of the troubles is that the time I spent on tunning the parameters.

Hence, I could tell it is more easy for me execute this mission with the voronoi based graph method.

6. The applicability of each method

Just as what I say in part 5, voronoi based graph planner is more suited for the situation when the robot is not good enough to follow all the points as we wish. Because even if it deviate from the right path, it could not crash into any obstacles (assume there are not many obstacles in a narrow space) and that makes this planning algorithm acceptable for most cases.

Visibility graph based planner requires higher accuracy of robot's acuators because sometimes the robot needs to move along the edges of an obstacle and turn right/left at a vertex. However, visibility graph has a really good mathematics structure, which means we could take advantage of any existed algorithms in the graph theory to find the optimal solution. This property becomes more and more important and useful when the scale of the problem grows larger. For example, if the environment is set in a 1000ft*1000ft area with 100 obstacles in it, it really takes a great amount of time to get the voronoi graph. On the other hand, for visibility graph

based planner, it only needs to find the vertices of each obstacles and walls and then check the connectivity between each nodes. After we obtain the weighted graph matrix M, we could use Dijkstra's algorithm or Floyd's algorithm to find the optimal solution from any start points to any end points. Also, there are some data structures like tree could even accelerate the searching procedure. Hence, motion planning with visibility graph based planner is much more efficient especially when the scale of problem is large.

For this mission, motion planning with voronoi based graph planner is better(cheap robot with low accuracy and small-scale probem).