Path Planning

FINAL PROJECT REPORT

For course:

Projects in Computer Science and Engineering

(June 10th, 2017)

HALMSTAD UNIVERSITY

Lu Han

SUPERVISOR

Jennifer David & Hassan Nemati

TABLE OF CONTENTS

1. INTRODUCTION

Path planning is an important primitive for autonomous mobile robots. The typical path planning problem is to find a collision-free path from the start point to a specific target point [1] in an environment with obstacles, according to certain evaluation criteria (the shortest route, the least time, etc.)[2]. The robot should be able to move safely, without collision through all the obstacles. The path planning problem can be called collision avoidance planning problem as well [3].

The problem of path planning is finding a path in a static or dynamic environment, for example, a maze, graph and road. Path planning lets the robots plan in the environment with obstacles, then find the optimal path to avoid collision [4]. Researchers distinguish between various methods used to solve the path planning problem according to two factors, the environment type (i.e., static or dynamic) [5] the path planning algorithms (i.e., global or local) [6]

In this report, I will use simulation tools to analysis different path planning algorithms under different environmental scenarios and do a comparative study with a user interface developed in Matlab.

2. METHOD

Path planning can be divided into two types; one is global path planning in a static environment and the other is local path planning in a dynamic environment.

In global path planning, the static environment is defined as an environment which contains static objects only with the moving robot itself, while in local path planning, the dynamic environment is defined as an environment that contains moving objects like human beings, moving machines, or other moving robots. [7]

In global path planning, the robot needs to know all the environment information and all terrain should be static, then according to the corresponding path planning algorithm, the shortest path is found, while in local path planning, the robot needs to collect real-time (online) environment information by sensors, then determine the location of the robot in the map and the distribution of obstacles (localization), so that the robot can find the optimal path from the current location to the goal point. In other words, the algorithm is capable of producing a new path in response to environmental changes [8] at every instant.

The path planning problem includes environment or map representation, path searching (planning) and path smoothing in general.

1) Environment modelling or map representation.

The robot interprets the map of the environment in different ways. In this report, I have two kinds of map, one is a logical map (which is hand drawn by myself) and the other map is a .png image of the real environment.

2) Path searching or path planning

Path searching or path planning is finding the optimal path by different algorithms based on the map representation.

3) Path smoothing

The path we find through corresponding method might not be a viable path that the robot can pass through, it looks blocky and is not suited if you move your units with floating point datatypes. So the path requires further processing and smoothing. In this case, we can use inflated function to inflate the map with the robot radius first, after

getting the path, we choose some points in the path and generate more points between two points, then the robot doesn't need to cover all the point in the path and can avoid sharp point location.

The robot's hand, arm, or body travels through the external world of the obstacle, and when it reaches a certain target position, it is necessary to determine an optimized travel path that does not collide in space. The measure of a good or bad planning process mainly refers to two factors: collision free and efficiency.

Collision free: The main consideration is to avoid collision problems, also known as collision-avoidance path planning.

Efficient: The main consideration is the path optimization problem. Try to make the robot's hand, arm, or body in the possible shortest time reach the target location.

2.1 STATIC SITUATION

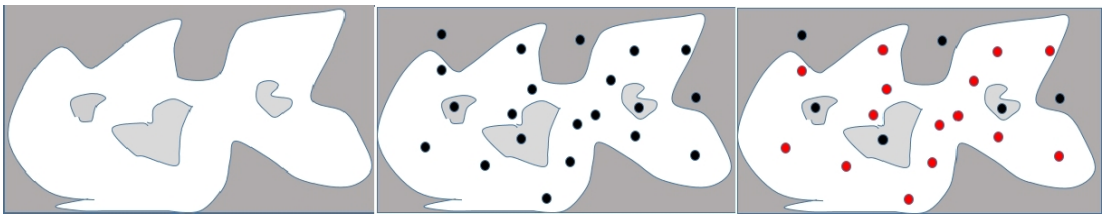
According to the feature of environmental information, path planning can be divided into discrete domain path planning and continuous domain path planning. The problem of path planning in discrete domain situation is roughly divided into global planning and local planning. Due to the discrete domain path planning doesn't care much about the changing environment in local planning, so local path planning with discrete domain can be regarded as global path planning in static situation.

PRM is a discrete domain path planning method, whereas Dijkstra and A star are continuous domain path planning methods.

2.1.1 PRM

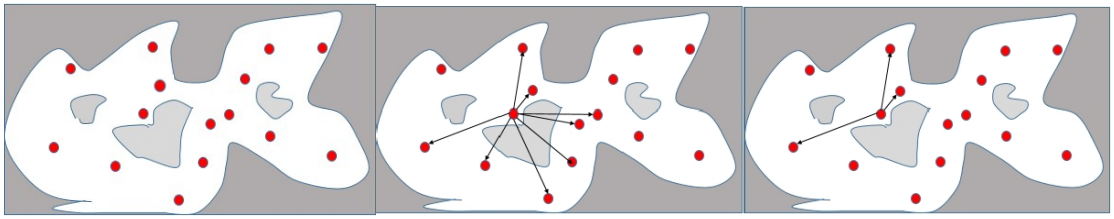
The probabilistic roadmap planner [9] is one of the mobile robot path planning method which is based on random search to solve the problem of finding a path from start location to a goal location without collision under static environment.

The basic idea of PRM is to generate specific number of random points in the map, check if the linear segment between two points is within the distance of connection and there is no obstacle in the line segment, and then connect these two points. After connecting all the segments in the map, we will get a connected network. The iteration ends when the start and the end point are selected in the randomly generated network. Then a simple search is run in this network from the start point to the goal point to get an obstacle-free path without any collision. If the start point or the goal point is not in the connect network, then the algorithm returns an empty array. Also increasing the number of nodes and connection distances can raise the possibility of finding path [10]. Hence, this method is probabilistically complete. This algorithm is illustrated in Fig 2.1.1.1-2.1.1.4



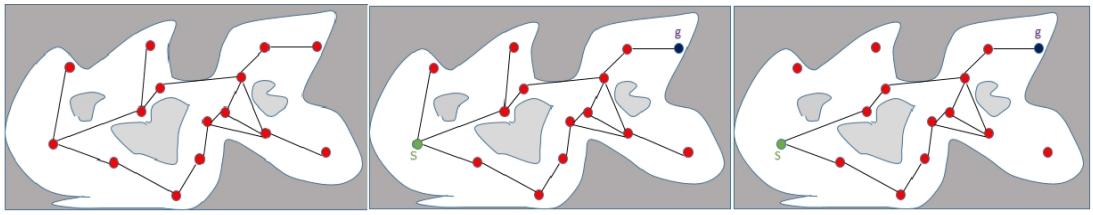
(a) (b) (c)

Figure 2.1.1.1 Pick coordinates randomly and get available points.



(a) (b) (c)

Figure 2.1.1.2 Connect collision-free neighbors.



(a) (b) (c)

Figure 2.1.1.3 Get all collision-free paths from start point to goal point.

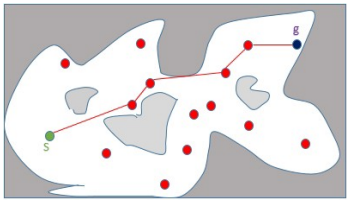


Figure 2.1.1.4: Get the shortest path from start point to goal point.

2.1.2 DIJKSTRA

The Dijkstra algorithm is a typical shortest path algorithm which is used to calculate the shortest path of a node to all other nodes. The main feature is in expanding from the starting point to the next layer, until reach the goal point. Dijkstra algorithm calculates the optimal shortest path always, but it traverses to all the nodes around which makes the Dijkstra algorithm has a low efficiency [11].

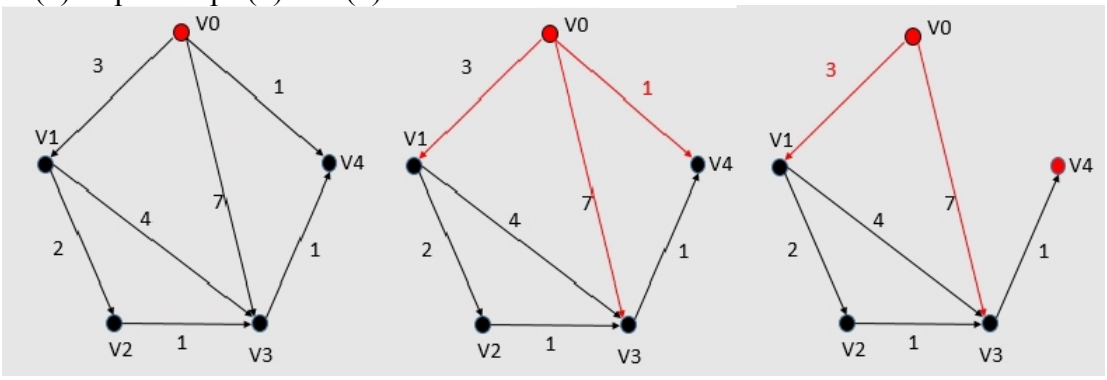
The basic idea of Dijkstra algorithm is: Assuming G=(V,E) is a weighted directed graph with E edges and the vertices as V. The vertex set V is further divided into two groups, the first group has the list of all the shortest path in the vertex set, which is S, the initial S has only one source point, and each time a shortest path is obtained, this new vertex will be added to the set S until all the vertices are added to the S; the second group is the set of vertices that do not determine the shortest path, which is U. Then add all the vertices from U to S is sorted in the order of their shortest path length. During this process, the shortest path length of the vertices from the source point v to the S is always smaller than the shortest path length from any of the vertices from the source point v to U. It should be noted that each vertex corresponds to a distance, the distance from the vertex in S is the shortest path length from v to the vertex, and the distance from the vertex in U is current path length from v to the vertex. This is explained in figure 2.1.2.1

(1) Initially, S only contains the source point, S =[v], the distance of v is zero. U contains the other vertices except v, the vertex u in U is the weight for the edge.

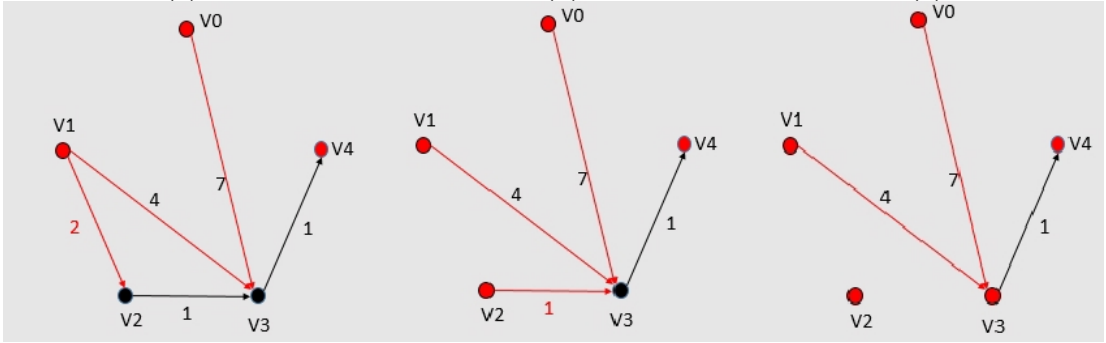
(2) Select a vertex k with the smallest distance from U, and add k to S, the selected distance is the shortest path from v to k.

(3) Modify corresponding vertex in U with the consideration of vertex k. If the distance from the source vertex v through point k to vertex u is shorter than the original distance without go through k, then modify the distance value of the vertex u, the modified distance value is the distance of k plus the weight of the edge.

(4) Repeat steps (2) and (3) until all vertices are included in S.



(a) (b) (c)



(d) (e) (f)

Figure 2.1.2.1 Steps in finding path with Dijkstra algorithm.

(a) Given initial graph G=(V, E), source node is v0.

(b) Initialize d[v0] to 0, add v0 to S. Relax all nodes adjacent to source v0. Update predecessor (see red arrows) for all nodes updated.

(c) Choose the closest node v4. Relax all nodes adjacent to node v4. Update predecessors for nodes in U.

(d) Node v1 is the closest node, so add it to S. Relax all nodes adjacent to node v1. Update node v2 and v3.

(e) Now node v2 is closest. Choose this node and adjust its neighbor node v3.

(f) Finally, add node v3.

	S	Dist[v0]	Dist[v1]	Dist[v2]	Dist[v3]	Dist[v4]
Step 1	{v0}	0	3	maxint	7	1
Step 2	{v0,v4}		3	maxint	7	
Step 3	{v0,v1,v4}			5	7	
Step 4	{v0,v1,v2,v4}				6	
Step 5	{v0,v1,v2,v3,v4}	0	3	5	6	1

Figure 2.1.2.2: The shortest path from each node to the source node v0.

2.1.3 A STAR

A-Star algorithm is one of the most efficient direct search methods for static environment to find the shortest path. It is a heuristic based method and so is faster.

The formula is expressed as: f (n) = g (n) + h (n) [12],

n is the last node on the path,

f (n) is the estimate cost from the initial node via node n to the target node,

g (n) is the actual cost from the initial node to node n,

h (n) is the estimated cost of the best path from node n to target node.

For the path search problem, the cost is distance. Here are many different ways in calculating distance, for instance, Manhattan distance, Euclidean distance, diagonal distance, etc.

Manhattan Distance and Euclidean Distances are metrics used for distance calculation. They are the most commonly used heuristic functions in A star algorithm.

Manhattan Distance: It's a standard heuristic for square grid that allows four directions of movement, the distance between two points measured along axes at right angles, in a plane with P1 at(x1,y1) and P2 at (x2,y2), it is |x1-x2|+|y1-y2|.

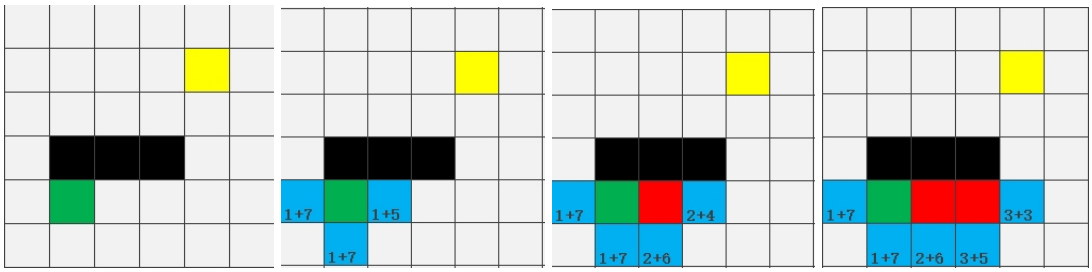
Euclidean distance allows eight directions of movement, deriving the Euclidean distance between two data points involves computing the square root of the sum of squares of the differences between corresponding values.

There are two lists in A-star algorithm: the open list and the closed list. The open list contains nodes potential best path nodes that are not considered yet, if the open list is empty, means that there is no possible path. The closed list contains the nodes that are already considered or visited, the close list is empty at the beginning.

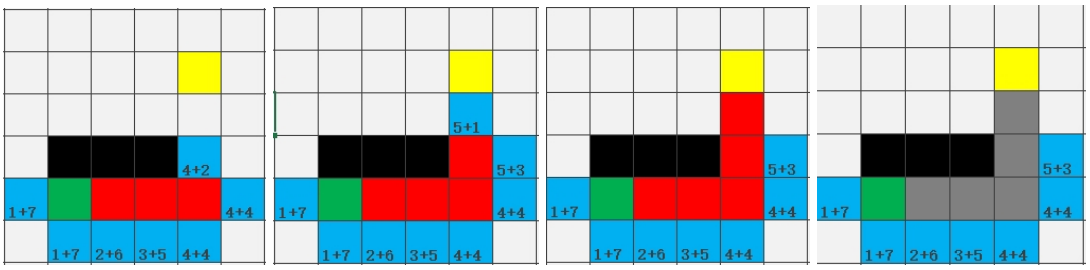
The main idea is to select the node with the lowest estimate cost to reach the goal. If this node is not the goal node, then add all neighbor nodes to the open list and add this node to close list, repeat this step until reach the goal node.

It also creates nodes to keep a reference to their parent, this means that it's possible to backtrack back to the start node from any node created by the algorithm [13].

Here are examples of A star algorithm with Manhattan Distance and Euclidean Distances as heuristic functions in Fig 2.1.3.1: The green cell is start point, yellow cell is goal point, black cells are obstacles, white cells are free way, blue cells are open list, red cells are closed list, grey cells are path.



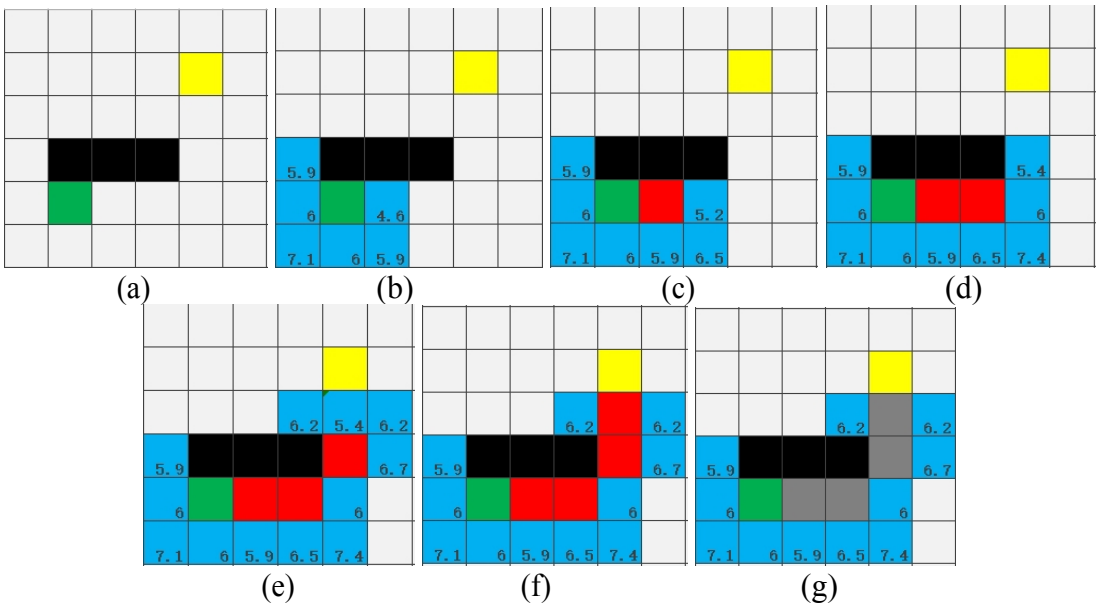
(a) (b) (c) (d)



(e) (f) (g) (h)

Figure 2.1.3.1 Steps in finding path with A star algorithm.(Manhattan Distance)

(a) Original map with start point, goal point and obstacles.
(b) From the start point calculate f for each neighbor, add these nodes to the open list.
(c) Choose one of the smallest estimate cost node, add the node to closes list, and calculate f for each neighbor, add these nodes to open list.
(d)(e)(f)Repeat the same step as (c).
(g) The result path of A star algorithm.



(a) (b) (c) (d)

(e) (f) (g)

Figure 2.1.3.2 Steps in finding path with A star algorithm.(Euclidean Distances)

2.1.4 COMPARE DIJKSTRA AND A STAR

As we mentioned before, Dijkstra and A star both are continuous domain path planning methods, they are essentially the same, the only difference is that A star algorithm has a heuristic. Since Dijkstra is not heuristic, it searches all directions in an average, so the Dijkstra algorithm usually explores a larger area before finding the target, which cause Dijkstra algorithm slower than A *.

Dijkstra algorithm is better than A star in some situations that we don't know the specific target. For example, you want your robot to get a ball for you. And it may be known that there are several balls available in the environment, but you want to get the ball as soon as possible, which means the robot better get the nearest one. In this

case, the Dijkstra algorithm is more suitable than A star, because we do not know which is the most close one. With A star, you need to sequentially pass on each target and calculate the distance, and then select the nearest path. We are looking for a target that may have countless positions, we just want to find the nearest one, but we do not know which is the nearest, so in this situation we should use Dijkstra algorithm rather than A star.

2.2 DYNAMIC SITUATION

Path planning in dynamic environment is the problem of mobile robot navigation to determine shortest feasible path to move from any current position to target position in unknown environment with moving obstacles. In dynamic environment, in order to solve the problem of the dynamic nature of obstacles, the robot should predict the future trajectories of these obstacles to plan its own path accordingly.[14]

2.2.1 D STAR

The original D* was introduced by Anthony Stentz in 1994. The name D* comes from the term "Dynamic A*", because the algorithm behaves like A* except that the arc costs can change as the algorithm runs.

The mainly method of D star is using Dijkstra algorithm searching from goal point to start location, save the shortest path and distance from each point to target point *h,k* , *k* is the smallest value of *h*, in current situation *k*=*h*. A robot follow the shortest path until detect the next location X, which case the robot needs re-plan, then the robot update the value of current location Y to the goal location G to

h(Y)=weight value from X to Y + original value of *h*(X).

Also, we can use other method to implement D star algorithm.

3. EXPERIMENT AND RESULTS

In my experiment, the performance of the approach is evaluated in Robotics toolbox for Matlab. Matlab can simulate the experimental environment by creating logical map or read an image map from files, with different algorithms the robot can find the best path based on the information of map and follow the best path until reach the goal point.

3.1 MAPS

A robot needs to know 3D environment information in real world, but in this simulation experiment we only need 2D environment information. Here are two kinds of maps in 2D environment, one is created to test algorithms, the other is 2D image map based on reality world.

In order to get a smooth path, before finding the best path, we need inflate the map.

3.1.1 LOGICAL MAP

In logical map part, I created a function named createMap(), when call this function, the user can get an initial map with arranged obstacles, the user can choose the shape, size, and number of the obstacles in random location as well.

Here are the steps in create map function:

(1)Create a specific size map with ones, map=ones(row,column).

(2)Create a sub-map with zeros, which used to add obstacles.

(3)Add some obstacles in the sub-map.

(4)Create some different size and shape obstacles as randomly generate obstacles

(5)for(number of obstacles){

Generate random location;

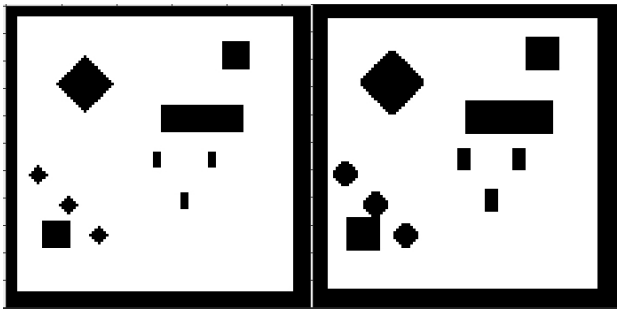
Add the obstacle to this location in sub-map;

}

(6)Add sub-map to map.

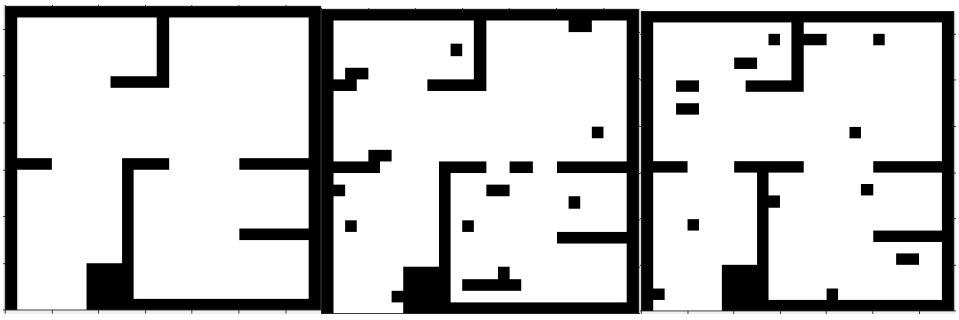
(7)Change the map type from double to logical

Here are some maps created by this function.



(a) (b)

Figure 3.1.1.1 Original logical map and inflated logical map.



(a) (b) (c)

Figure3.1.1.2 Original logical map and logical map with random obstacles.

3.1.2 MAP FOR SMART HOME

Imread() function can read image from graphics file [12] , the result of imread() function is a three - dimensional matrix, then use a(:,:,1) get all the rows and columns in this matrix, which is two-dimensional map.

(a) (b) (c)

Figure 3.1.2.1 Maps for smart home with inflation and random obstacles.

3.2 STATIC SITUATION

In Static situation, we mainly discuss PRM method, Dijkstra algorithm and A star algorithm. When it comes to algorithm, there are two main ideas for path planning:

Exact Arithmetic: mainly used to find a solution or prove that there's no solution, the main consideration is the existence of solution and find the optimal solution. This algorithm determines the complexity of time.

Heuristic Arithmetic: focus on a short period of time finding a solution, there might be no solution or not optimal solutions, it emphasis the efficiency.

3.2.1 PRM

The basic idea of Probabilistic Roadmaps Method is taking randomly samplings with a certain probability in areas without obstacles, and generates the Probabilistic Roadmaps by connecting these points according to certain rules, then transforms the path planning problem in space into the searching problem on a probabilistic map.The PRM algorithm belongs to the category of Exact Arithmetic algorithm. The basic theory is Probabilistic Completeness. Because the pots are randomly taken, so as the number of points increase; the probability of solving this problem tends to be 1.

3.2.1.1 PRM IN LOGICAL MAP

Steps in path planning with logical map:

(1) Create logical map

(2) Upload the map to robotics

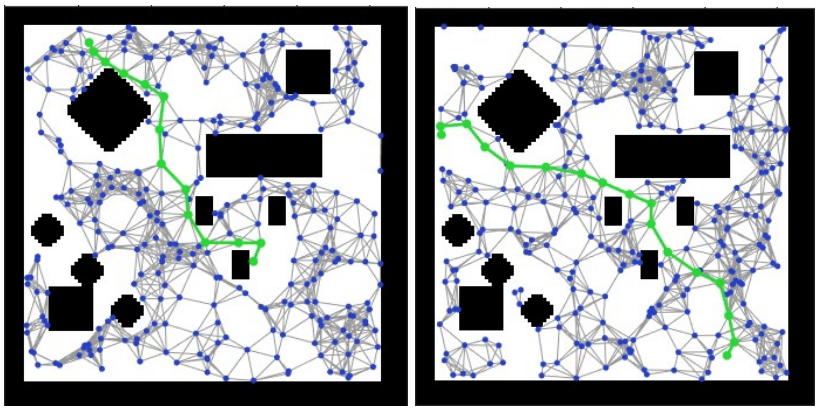
(3) Inflate the map with robot radius

(4) Configure robotics.PRM with number of random samplings and connection distance.

(5) Set the start point and goal point, or generate these two point randomly.

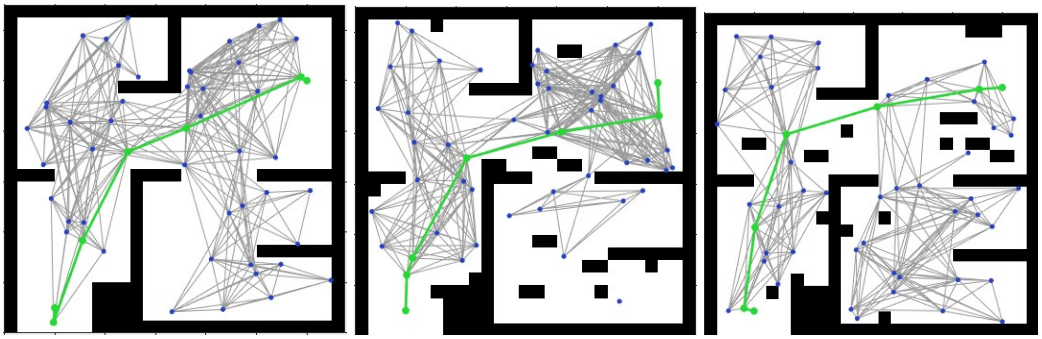
(6) Using findpath() function get the optimal path.

Here are some results:



(a) (b)

Figure 3.2.1.1.1 PRM algorithm in logical map with random start point and goal poin.



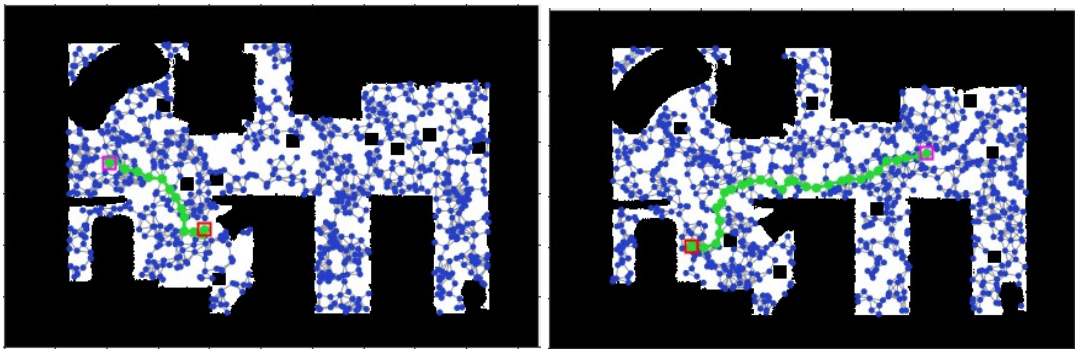
(a) (b) (c)

Figure 3.2.1.1.2 PRM algorithm in logical map with random obstacles.

3.2.1.2 PRM IN SMART HOME MAP

Steps in path planning with image map:
(1) Use imread() function read the image.
(2) Change the three-dimensions image to 2-dimensions map.
(3) Convert the 2-dimensions map to logical map.
(4) Same as step(2) to (6) in path planning in logical map.

Here are some results:



(a) (b)

Figure 3.2.1.2.1 PRM in map of smart home with random start point and goal point.

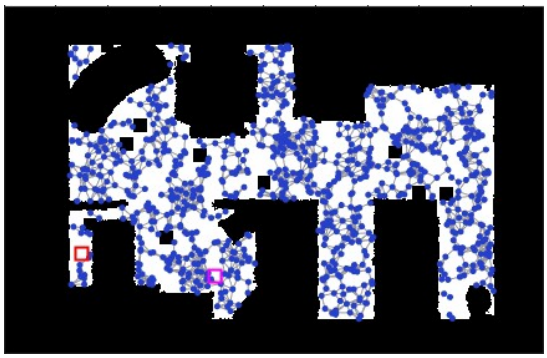


Figure 3.2.1.2.2 No solution from start point to goal point.

As we can see from Figure 3.2.1.2.2, sometime we may not find a path from start point to goal point, which means we can't always find an effective solution, this was because we may not generate enough points or the connect distance is too small, so we can solve this problem by generate more points or increase the connect distance. It's time consuming when generate more random points and may cause collision when connect two points far away, so we need to do more experiment to get feasible number of node and distance.

3.2.2 DIJKSTRA

The Dijkstra algorithm is a typical shortest path algorithm; it is used to calculate the shortest path from a specific node to any other nodes. The main feature of Dijkstra Algorithm is extending outward from the start point to the goal point, based on greedy algorithm, each time it will find the nearest point from start point. By iterating the path length in the order of the increasing shortest path length, we can get the shortest path from the source point to the other target node.

3.2.2.1 DIJKSTRA IN LOGICAL MAP

Steps in path planning with logical map:
(1) Give a logical map, start point and goal point.
(2) Set up color for display and judgement
(3) Read the map
(4) Create a table keeps track of the state of each grid cell
(5) Initialize distance array, parent array and visited array.
(6) while loop:
(6.1) Find the node with the minimum distance
(6.2) Update map information with distance, parent and visited array.
(6.3) Visit each neighbour of the current node and update the map, distances and parent appropriately.

```
if(neighbor node is in the map){
    if(neighbor node is the target node){
        update map information and exit while loop;
    elseif(neighbor node is not obstacles and haven't been visited)
        update map information;
    end
```

end

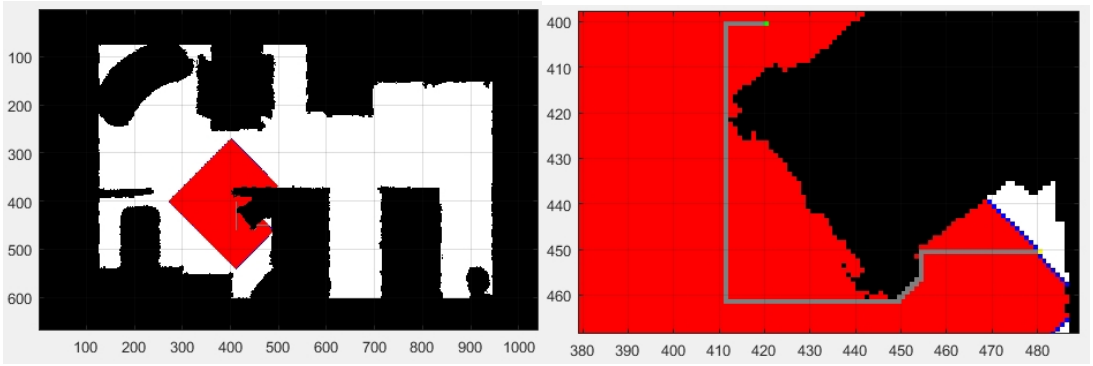
(7) Construct route from start node to goal node by following the parent links

(a) (b) (c)

Figure 3.2.2.1.1 Results of Dijkstra Algorithm in logical maps.

3.2.2.2 DIJKSTRA IN SMART HOME MAP

Steps in path planning with image map is the same as logical map, here is a shortest path calculated by Dijkstra algorithm in a image map.



(a) (b)

Figure 3.2.2.2.1 Results of Dijkstra Algorithm in smart home map.

3.2.3 ASTAR

Astar algorithm is a popular graph search algorithm that finds the path from source node to goal node with lowest cost. It is similar to other heuristic methods, such as the best search (BFS) and Dijkstra algorithm and other formal methods.

3.2.3.1 A STAR IN LOGICAL MAP

Steps in path planning with logical map:

(1) Give a logical map, start point and goal point.

(2) Set up color for display and judgement

(3) Read the map

(4) Create a table keeps track of the state
of each grid cell

(5) Evaluate Heuristic function, here I use Manhattan distance value from current to destination.

(6) Initialize parent array, f array and g array. f array is the estimate cost from the initial node via node n to the target node, g array is the actual cost from the initial node to node n.

(7) while loop:

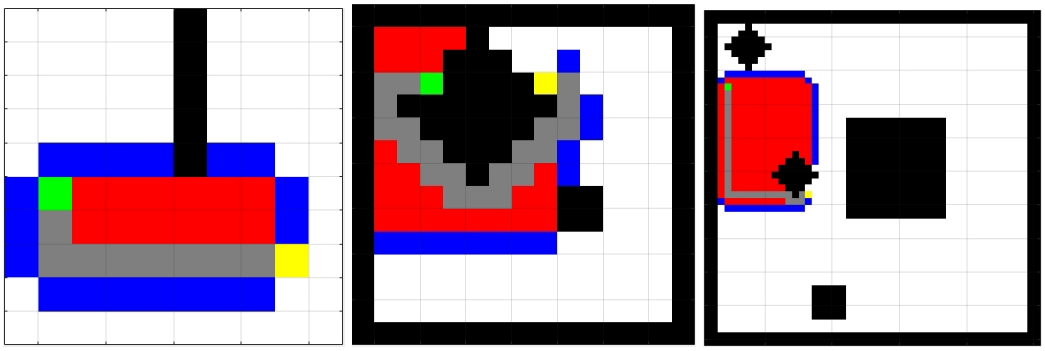
(7.1) Find the node with the minimum f value

(7.2) Update map information with f.

(7.3) Visit each neighbor of the current node and update the entries in the map, f, g and parent arrays.

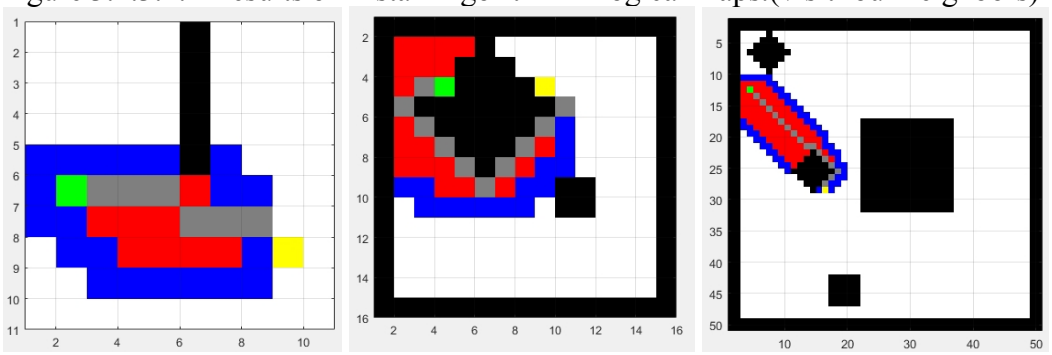
```
if(neighbor node is in the map){
    if(neighbor node is the target node){
        update map information and exit;
    elseif(neighbor node is not obstacles and haven't been visited)
        update map information;
    end
end
```

(8) Construct route from start node to goal node by following the parent links



(a) (b) (c)

Figure 3.2.3.1.1 Results of A star Algorithm in logical maps.(visit four neighbors)



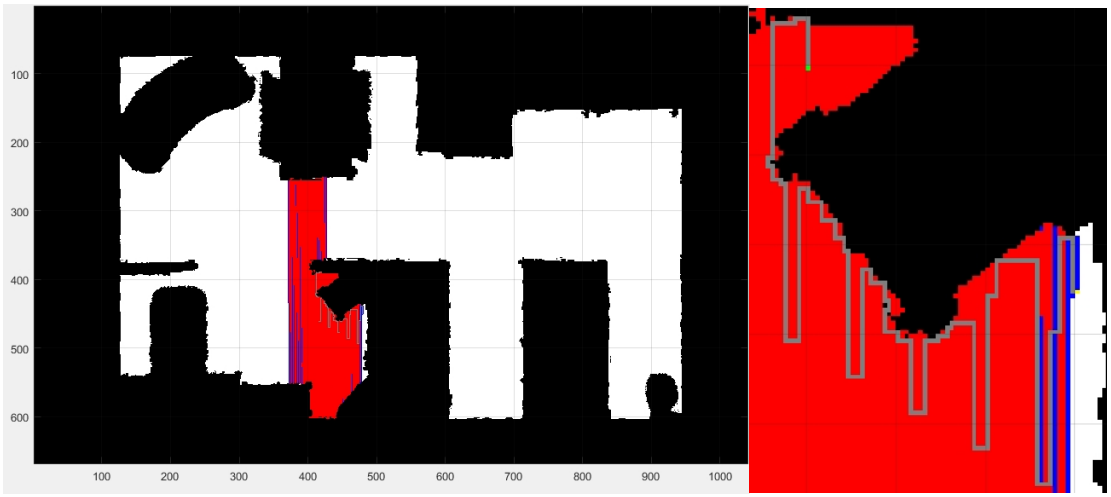
(a) (b) (c)

Figure 3.2.3.1.2 Results of A star Algorithm in logical maps.(visit all neighbors)

3.2.3.2 ASTAR IN IMAGE MAP

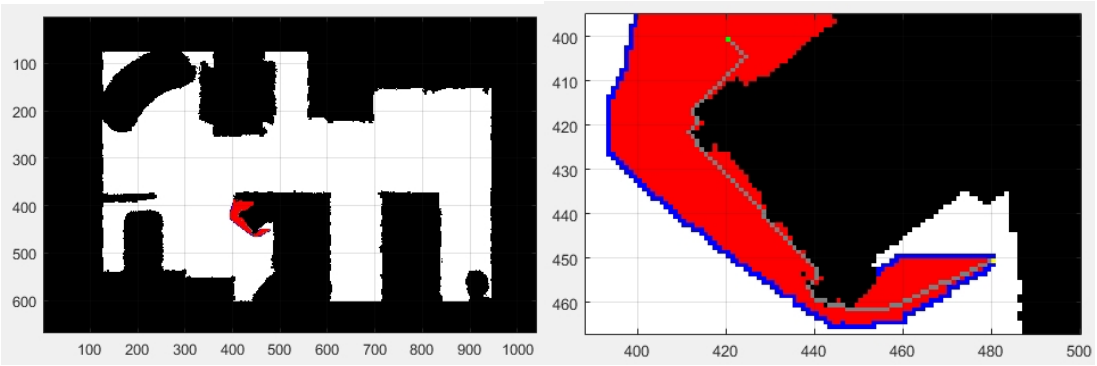
Steps in path planning with image map is the same as logical map, the figures show a shortest path calculated by Astar algorithm with Manhattan Distance (visit four neighbors) and Euclidean Distances (visit eight neighbors) in a image map. Here we

can see that A star calculate distance with Manhattan Distance cost more than Dijkstra in this image map.



(a) (b)

Figure 3.2.3.2.1 A star Algorithm in smart home map with Manhattan Distance



(a) (b)

Figure 3.2.3.2.2 A star Algorithm in smart home map with Euclidean Distances

Astar algorithm uses a Heuristic Function f to estimate the distance of the current point to the target point and determine the search direction. Therefore, we can find that the key to success of the Astar algorithm lies in the correct choice of the evaluation function. In theory, a correct valuation function can find the best path very quickly, but the correct valuation function is not available, so the Astar algorithm does not guarantee that it will get the correct answer every time.

An unsatisfactory evaluation function may make it work very slowly and even give an incorrect answer.

3.3 DYNAMIC SITUATION

In dynamic environment, we mainly use D star algorithm.

3.3.1 D STAR

D star algorithm was introduces by Anthony Stentz in 1994, it comes from the term "Dynamic A star", so D star is based on A star in a Dynamic environment, here we use Dynamic A star to re-plan.

3.4 GUI DEVELOPMENT IN MATLAB:

This is the GUI interface for users developed in Matlab as a part of this project work. The user can upload maps of their own and run different path planning algorithms and study them.

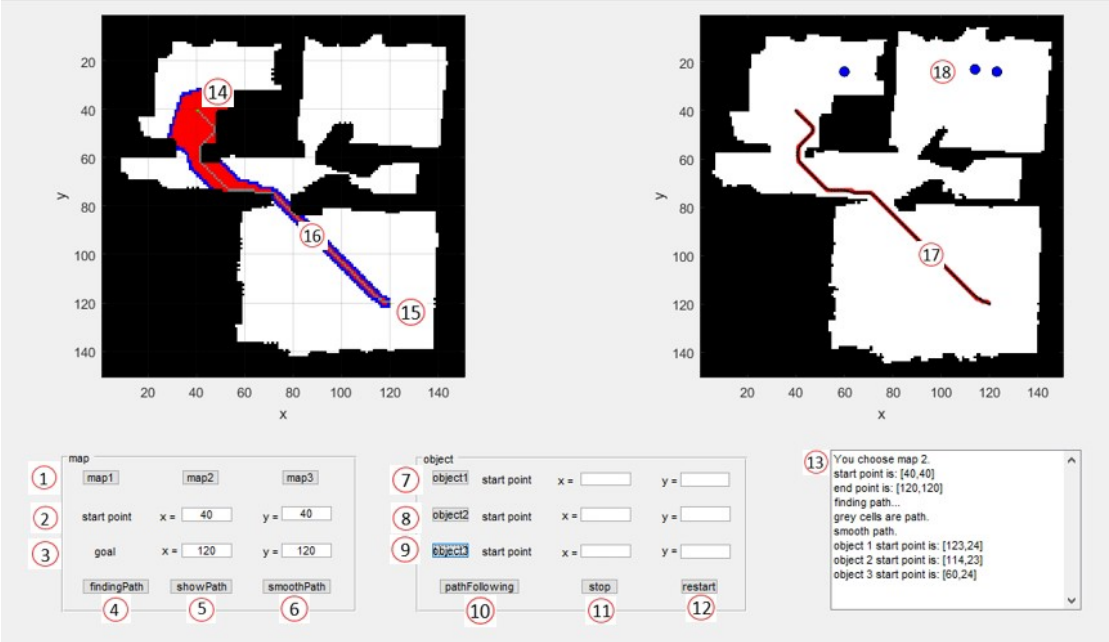


Figure 3.3.1.1 GUI for dynamic environment

1:choose different maps.
2:set start location.
3:set goal location.
4:show finding path process.
5:show find path result.
6:smooth path.
7,8,9:generate objects, users can set specific location or generate randomly.
10:show the process of following path.
11:stop current result.
12:restart the application.
13:show the knowledge of robot.
14:start location.
15:goal location.
16:grey cells are shortest path, red cells are visited cells, blue cells are considerated cells.
17:red cells are the original path we get from Astar algorithm, black points are the result of after smoothing the path.
18: objects genarated

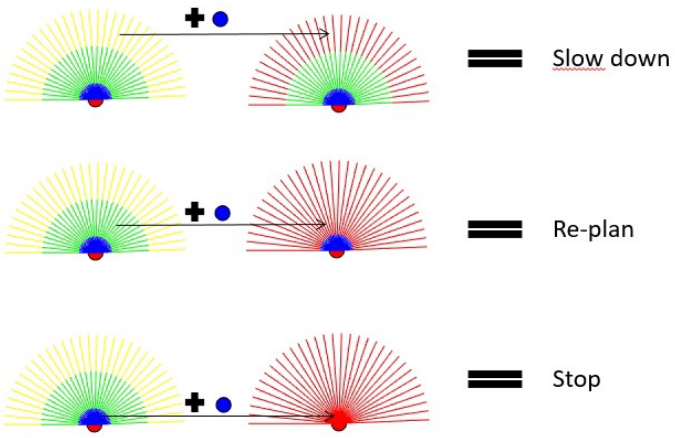


Figure 3.3.1.2 Performance for robot in different situations

In the simulation environment, I create a red circle with radius 0.2 meters as my robot, and set three half circles as lasers that can detect the obstacles.
The blue circle is the danger zone, if an object shows up in this zone, the robot will stop immediately; the green circle is safety zone, if an object shows up in this zone, the robot will re-plan the path; the yellow circle is the sense distance zone, if an object shows up in this zone, the robot will slow down.

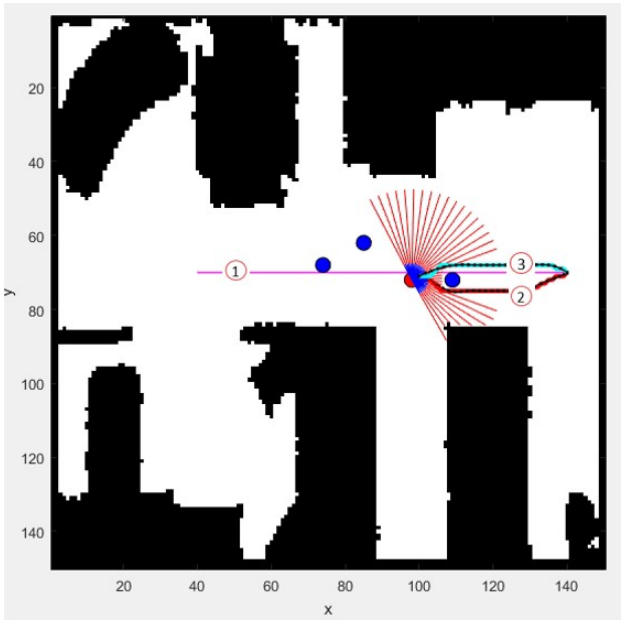


Figure 3.3.1.3 Result of re-planning

1: original path
2: current path
3: new path after re-planning

4. CONCLUSION

	Algorithm	Environemnt	Time	Optimal path
Discrete Domain	PRM	Static / Dynamic	Fast(good for huge maps)	Can't guarentee get optimal path
Continuous Domain	Dijkstra	Static	Very slow	Guarentee get optimal path
	A star	Static	Fast	Always get optimal path
	D star	Dynamic	Fast	Always get optimal path

Table 4.1 Comparison of the different algorithms

PRM is a discrete domain algorithm, because it generate nodes randomly in current environment, so it doesn't matter whether the environment is dynamic or static, PRM always generate nodes according to the current environment, so we can regards dynamic environment as static environment, which means PRM can use both in static and dynamic environment. Because PRM doesn't need cover all cells in the environment, so it's always fast getting a path in very huge maps, but it can't guarantee get the optimal path.

Dijkstra, A star and D star are continuous domain algorithms, while Dijstra and Astar are used in static environment and D star is used in dynamic environment, because Dijkstra algorithm extends from the start point to out layer in all directions, so it's very slow when finding a path, but it can always guarantee get an optimal path.

Astar algorithm uses heuristic algorithm when finding a path, it only extend the direction with smallest value of current cost plus the estimate value of current point to goal point, so A star algorithm is fast then Dijkstra. Because the path we get is depending on the heuristic function, sometimes we can't get a best result with a heuristic function, like Figure 3.2.3.2.1 shows, but we can always get an optimal path with a correct heuristic function.

D star is based on A star, so the feature of D star is almost the same as A star.

During the evaluation process of different scenarios, we can always find the optimal path with in different maps, the robot can avoid obstacles by slowing down, re-planning and immidiately stop when an object reach the danger zone.

These are the links for results:

https://www.youtube.com/watch?v=E57Ov2gdYPk&feature=youtu.be

https://www.youtube.com/watch?v=Oxw0Au8O4GE&feature=em-share_video_user

https://www.youtube.com/watch?v=ItY_qhCDUAU&feature=em-share_video_user

https://gist.github.com/luhan2016/4cff83d4f1705518cde3e6f1d65ba920

REFERENCES

[1] Vacariu, L., Flaviu Roman, Mihai Timar,Tudor Stanciu, Radu Banabic,Octavian Cret. Mobile Robot Path-planning Implementation in Software and Hardware. in 6th WSEAS International Conference on Signal Processing, Robotics and Automation. 2007. Corfu Island, Greece

[2] K. H. Sedighi, K. Ashenayi, T. W. Manikas, R. L. Wainwright, and H. M. Tai, "Autonomous Local Path Planning for a Mobile Robot Using a Genetic Algorithm," in Congr. Evol. Comput. 2004. CEC2004., Vol. 2, pp. 1338-1345.

[3]: Ismail AL-Taharwa, Alaa Sheta and Mohammed Al-Weshah,"A Mobile Robot Path Planning Using Genetic Algorithm in Static Environment" Journal of Computer Science 4 (4): pages:341-344, 2008, ISSN 1549-3636, © 2008 Science Publications

[4] Buniyamin N., Wan Ngah W.A.J., Sariff N., Mohamad Z. A Simple Local Path Planning Algorithm for Autonomous Mobile Robots

[5] A. Ramirez-Serrano, H. Liu and G. C. Pettinaro. (2008). Mobile robot localization in quazi-dynamic environments. Int. J. on Ind. Robot, pp.246-258.

[6] K. H. Sedighi, K. Ashenayi, T. W. Manikas, R. L. Wainwright, and H. M. Tai, "Autonomous Local Path Planning for a Mobile Robot Using a Genetic Algorithm," in Congr. Evol. Comput. 2004. CEC2004., Vol. 2, pp. 1338-1345.

[7] huozhe A Mobile Robot Path Planning Using Genetic Algorithm in Static Environment Ismail AL-Taharwa, Alaa Sheta and Mohammed Al-Weshah Department of Information Technology, Faculty of Science and Information Technology, Al-Balqa Applied University, Al-Salt, Jordan

[8] Ismail AL-Taharwa, Alaa Sheta and Mohammed Al-Weshah Department of Information Technology, Faculty of Science and Information Technology, Al-Balqa Applied University, Al-Salt, Jordan.

[9]:Kavraki, L. E.; Svestka, P.; Latombe, J.-C.; Overmars, M. H. (1996), "Probabilistic roadmaps for path planning in high-dimensional configuration spaces", IEEE Transactions on Robotics and Automation, 12 (4): 566–580, doi:10.1109/70.508439.

[10]: T. SIMÉON, J.-P. LAUMOND and C. NISSOUX, "Visibility-based probabilistic roadmaps for motion planning", LAAS-CNRS, 7 avenue du Colonel-Roche, 31077 Toulouse, France Received 25 January 2000; accepted 10 April 2000.

[11]: Moshe Sniedovich, Dijkstra's algorithm revisited: the dynamic programming connexion, Control and Cybernetics vol. 35 (2006) No. 3.

[12]: Zeng, W.; Church, R. L. (2009). "Finding shortest paths on real road networks: the case for A*". International Journal of Geographical Information Science. 23 (4): 531–543. doi:10.1080/13658810801949850.

[13]: PETER E.Hart,NILS J. NILSSON, BERTRAM RARHAEL,"A Formal Basic for the Heuristic Determination of Minimum Cost Paths", IEEE TRANSACTIONS OF SYSTEMS SCIENCE AND CYBERNETICS, VOL. ssc-4, NO.2, JULY 1968.

[14]: Anirudh Vemula and Katharina Muelling and Jean in Robotics Institute Carnegie Mellon University avemula1, Path Planning in Dynamic Environments with Adaptive Dimensionality