Voronoi Diagram Based Roadmap Motion Planning

Pratap Bhanu Solanki
Y9429
Electrical Engineering
IIT Kanpur

G Harsha Vardhan Reddy
10271
Computer Science and Engineering
IIT Kanpur

Mentor: Amitabha Mukerjee
Professor
Computer Science and Engineering
IIT Kanpur

***Abstract*—Robot Motion planning is one of the fundamental problem in robotics. It has various applications such as in Ware house automations, Driverless cars, Robotic surgery etc. The path planning problem is described as: to find a shortest or optimized path between start point and goal in a spatial configuration consists of obstacles of various types. There are many fundamentally different approaches suitable for different environmental configurations. The various methods are Cell Decomposition, Sampling method, Probabilistic Roadmap methods, Generalized Voronoi diagrams etc. In this project we will be exploring for Generalized voronoi diagrams in Robot motion planning.**

I. Related Work

A lot of research work is being done in the field of computation of Voronoi Diagram. Steven Fortune[1] has introduced a sweepline algorithm which can compute Voronoi Diagram for n point sites in O(nlogn) time. This algorithm become famous as Fortune's algorithm. Paul Chew [2] presented "expanding wave" analogy for computing Voronoi Diagram based on convex distance functions. Colm [4] have extended this work to motion planning of disc amid polygonal obstacles with an algorithm of O(nlogn) time complexity. A linear time algorithm for computation of Voronoi Diagram has also been introduced[3] for point sites lying on the vertices of convex polygon. Howewer in our project we are using Matlab function 'voronoi' for computation of Voronoi Diagram for point sites which we are using in computing Voronoi Diagram for polygonal obstacles.

II. Introduction to Voronoi Diagram

Voronoi Diagrams: "*A Voronoi diagram of a set of sites in the plane is a collection of regions that divide up the plane. Each region corresponds to one of the sites and all the points in one region are closer to the site representing the region than to any other site.*" [6] The path generated by Voronoi diagram ensures minimum collision risk if any possible. Figure 1 shows voronoi diagram for point sites. Here we can see that all the points lying in the yellow region are nearest to the point site corresponding to yellow region.

The voronoi diagram for a line site can be generated by considering line as a linear array of point sites. Similarly Voronoi diagram of a polygonal object can be drawn by considering the polygon as a set of line segments. Figure 2 shows voronoi diagram of a triangle where first it is considered as a set of three line-segments then each line-segment is considered as a set of point sites.

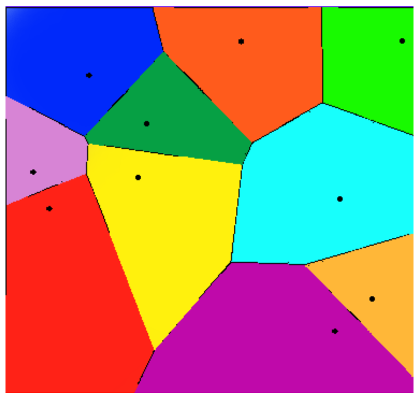
Fig. 1. Voronoi Diagram (Image courtesy [8])

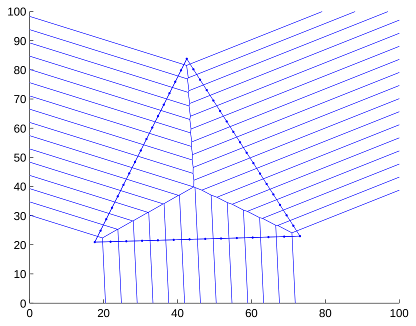

Fig. 2. Voronoi Diagram for Triangle by considering it as set of point sites

III. Problem Statement Formulation

In our problem statement an environment configuration consisting of polygonal obstacles with Start and Goal points is given. Our goal is to find the 'shortest' collision-free path from Start point to Goal point. For example a configuration is shown in figure 3 where there are two obstacles and Start point and Goal point are specified as green dot and red dot respectively.

Then our goal will be to find the 'shortest' collision free path from Start to Goal as shown in color magenta in figure 4. The green colored path shows the Voronoi Diagram for the given configuration. It is to be noted that the final path is a subset of Voronoi diagram.

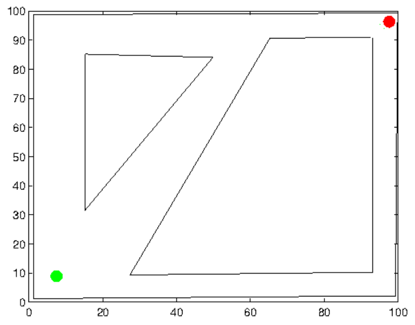
Fig. 3. Configuration of obstacles, Start and Goal points

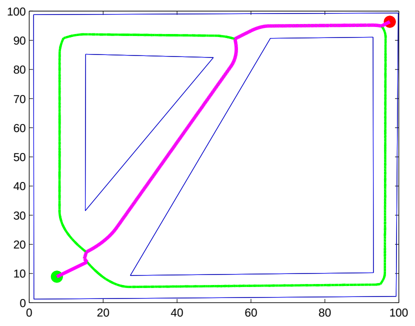
Fig. 4. Final path for obstacle configuration

IV. Methodology

This section explains how to go for solving the problem using an example. First consider the obstacle configuration given in figure 7. In this configuration we have six obstacles of different shapes and a bounding rectangle which will also be considered as an obstacle.

Now, the next step would be to divide the polygonal objects in number of point sites and then calculate the Voronoi Diagram for the whole set of points. The distance between consecutive point sites is governed by a parameter Epsilon. The less the Epsilon the more number of vertices will be which results in densely spacced voronoi edges. Less Epsilon gives more smoother output path but computation time increases as number of points are increasing. figure 6 shows Voronoi Diagram for the given obstacle configuration with Epsilon = 1. We used matlab function 'voronoi()' to compute the Voronoi diagram.

In figure 6 we can divide Voronoi edges in two categories: first are those Edges which are generated by point sites of same objects. And others are those which are generated by point sites of two different objects. It is to be noted that each of the voronoi edge corresponds to two point sites and the edge is perpendicular bisector of the line segment joining the

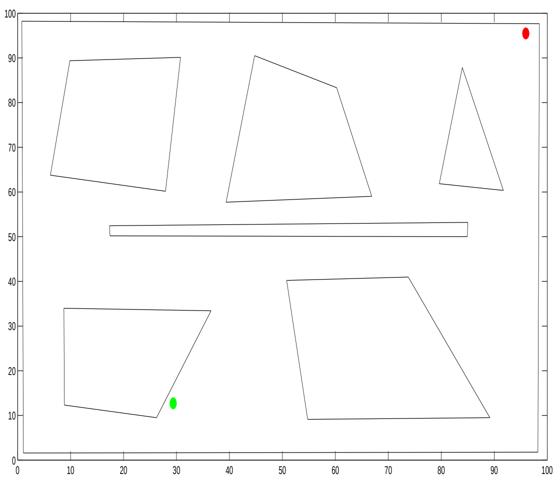
Fig. 5. Environment Configuration

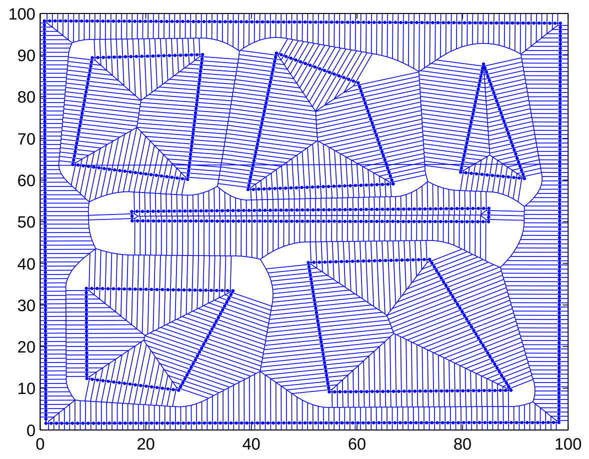
Fig. 6. Voronoi Diagram with Epsilon = 1

points. Now our task is to remove all those edges which are generated by point sites of same objects and retain the edges of second category. The information regarding the edges corresponding to sites is not directly given by Matlab. We used the function 'voronoin()' as follows:
[Voro_Vertex,Voro_Cell] = voronoin([X _Total_points' Y_Total_points']);
Here Voro_Vertex contains array of all voronoi vertices and Voro_Cell contains the information of edges corresponding to each point sites. So the voronoi edges of second category can be separated using the information in Voro_Cell. Figure **??** shows the separated voronoi edges of second category in green colour . This is infact the Voronoi Diagram of the given obstacle configuration.

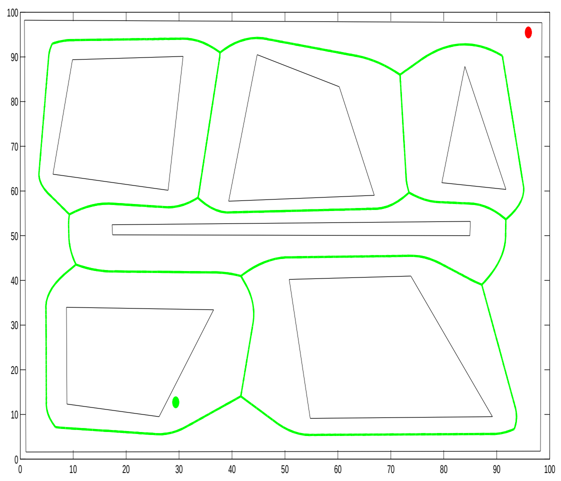
Fig. 7. Environment Configuration

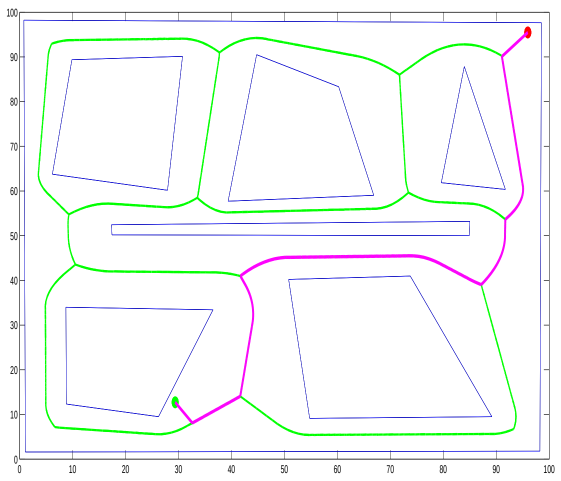
Fig. 8. Environment Configuration

After getting the Voronoi Diagram of the obstacle configuration. Our next task is to find the shortest possible route from Start and Goal. It is not necessary that Start and Goal lies on Voronoi Diagram. So for that we find the nearest Voronoi vertex from Start and Goal and named it Start' and Goal' respectively. The voronoi diagram can be represent in the form of acyclic Graph called Voronoi Graph. The weights in the adjacency matrix are the euclidean distances between the connected vertices. It is to be noted that Start' and Goal' both are Vertices of the Voronoi Graph. Now we can find shortest route between Start' and Goal' using any shortest route finding algorithm. We have used Dijkstra algorithm code by [7]. So using that algorithm we are able to get the final path from Start' to Goal'. Now combining the three paths Start to Start',Start' to Goal' and Goal' to Goal we can get final path from Start to Goal. Figure 8 shows the final path in magenta colour from Start (green dot) to Goal(red dot).

V. Description of Matlab Files

There are 4 Matlab files in main project directory: Obstacle_Draw.m, Get_Voronoi.m, Final_Path.m and dijkstra.m. Using Obstacle_Draw.m any polygonal obstacle configuration can be drawn. For drawing obstacle Obstacle_Draw.m needs to be run first then a figure window will come. to draw an n-gonal object n-clicks at desired points should be made. After n clicks return keys needs to be be pressed which confirms the drawing of that object. After pressing return key new object can be drawn. In Obstacle_Draw.m the number of object can be changed by changing value of parameter Num_Object. value of Epsilon can also be specified there. These configuration will be saved in .mat file format in directory Obstacle_Files where the image of obstacle configuration corresponding to each .mat file is shown for reference. The filename can be changed by changing the string variable FILE_NAME. Now using Get_Voronoi.m any of the obstacle configuration can be loaded by specifying the filename stored in string variable LOAD_FILE_NAME. By running Get_Voronoi.m the Start and Goal points can be specified in the obstacle figure by two consecutive clicks after it Voronoi diagram for the obstacle configuration will be generated. After that, running Final_Path.m will give the path from Start to Goal.

VI. Results and Conclusion

Here in this work we are able to generate the final 'shortest-collision-free-path' using Voronoi Diagram for given obstacle configuration and start and goal points as seen from figure 8. This path is not always the shortest collision free path. In some configurations better shortest path which are collision free also exists. But in average sense our method gives results equivalent to shortest collision free paths.

References

[1] Steven Fortune, "A Sweepline Algorithm for Voronoi Diagrams"Algorithmica(1987) 2:153-154

[2] L. Paul Chew,Robert L., Dyrsdale, III"Voronoi diagrams based on convex distance functions"Proceeding SCG '85 Proceedings of the first annual symposium on Computational geometry Pages 235-244

[3] Alok Aggarwal, Leonidas J. Guibas, James Saxe, Peter W. Shor "A linear-time algorithm for computing the voronoi diagram of a convex polygon" 1989, Volume 4, Issue 1, pp 591-604

[4] "A retraction method for planning the motion of a disc", Colm 'Dnlaing1, Chee K Yap1, Journal of Algorithms Volume 6, Issue 1, March 1985, Pages 104111

[5] MILO EDA, VCLAV PICH "Robot Motion Planning Using Generalised Voronoi Diagrams" 8th WSEAS International Conference on SIGNAL PROCESSING, COMPUTATIONAL GEOMETRY and ARTIFICIAL VISION (ISCGAV08) Rhodes, Greece, August 20-22, 2008

[6] M.de Berg, M.van Kreveld, M.Overmars and O.Schwarzkopf, Computational Geometry: Algorithms and Applications, Springer-Verlag, Berlin, 2000.

[7] http://www.mathworks.com/matlabcentral/fileexchange/5550-dijkstra-shortest-path-routing

[8] http://en.wikipedia.org/wiki/Voronoi_diagram