Smoothing Voronoi-based Obstacle-avoiding Path by Length-minimizing Composite Bezier Curve

Yi-Ju Ho and Jing-Sin Liu, Member, IEEE

***Abstract*— In this paper, we present an obstacle avoiding path planning method based on Voronoi diagram and composite Bezier curve algorithm which obtains the shortest path length. In our algorithm, a Voronoi diagram is constructed according to the global environment. The piecewise linear rough path in the Voronoi diagram which keeps away from the obstacles is obtained by performing Dijkstra's shortest path algorithm. Dynamic programming is employed to subdivide the nodes on the piecewise linear path into control point subsequences to generate a collision free composite Bezier curve with shortest curve length. In the experimental results, the path length obtained by our algorithm is up to 8.83% shorter than the piecewise linear rough path along the Voronoi diagram.**

***Index Terms*— path planning, Voronoi diagram, shortest path, Bezier curve, dynamic programming**

I. INTRODUCTION

Path planning plays an important role in robotic and automation fields for both static and dynamic environments and many researchers have worked on it since 80's. Many techniques have been researched to utilize multiple path schemes for different applications [1], [2], [3], [4], [6], [7], [9], [10], [11], [13], [15]. These applicaions have been dealt within two strategies: one strategy is to use a pre-known global environment information and robot characteristics, while another builds up local environment with sensor information and using robot characteristics.

Voronoi diagram for partitioning a map is used in many researches to build up a collision free path in both global [1] and local environments [7], [10]. The resulting path is either a piecewise linear path or a path smoothed with splines. If the path is a piecewise linear path, the robots following the path have to stop and restart frequently. This causes extra waste of robot power and wear. In order to obtain a smooth path, many curves have been introduced as path primitives. The smooth path is constructed by connecting pieces of the primitive curves. [3], [4], [11] construct the path by connecting the way points with splines, and the work in [2], [7], [9], [15] construct the path by Bezier curves. In [4], the authors propose a path planning algorithm with path length by spline, but the method has large time complexity.

Among these different curves, Bezier curve is more intuitive due to its space property which we will express particularly in next section. [2] uses a set of reference points as control points for one Bezier curves instead of using the reference points as way points. We notice that for a set of reference points, to connect them by taking each point as a way point causes longer path length with comparing to connect them by taking each point as a control point of Bezier curves. If we take the reference points as control points instead of way points, the curve path does not need to pass all the reference points but only the end points of the Bezier curves. This greatly reduces the path length and retains the curve property. However, there may not exist such a single Bezier curve of the reference points in complicated environments filled with obstacles. In order to overcome this problem, we use a composite Bezier curve instead of a single Bezier curve.

In this paper, we propose an obstacle avoiding path planning algorithm for omnidirectional mobile robot. Our algorithm is a three step method. We first utilize the Voronoi diagram and the Dijkstra's shortest path algorithm to build up a collision free rough path. We then contruct a smooth curve path along the rough path. The Bezier curve is used as our curve primitive. We take the nodes on the rough path as control points of the composite Bezier curve such that the resulting curve path will closely follow the rough path. In the second step, the control points are divided into subsequences for each Bezier curve such that the convex hull of each subsequence does not collide with any obstacle. We use dynamic programming algorithm to generate these subsequences of control points ensuring that the corresponding composite Bezeir curve will have shortest curve length. The last step is to remove crowded control points to further reduce the curve length and to construct the Bezier curves for each control point subsequence meeting kinematic constraints.

Our experimental results show that the length of the composite Bezier curve constructed by our algorithm is 8.83% shorter than the path length of the piecewise linear rough path along the Voronoi diagram. With any given set of control points, our algorithm can obtain a composite Bezier curve as the collision free path with shortest curve length.

The rest of this paper is organized as below. Section II introduces the preliminary for Voronoi diagram and Bezier curve. Section III describes our algorithms: The Voronoi diagram construction for irregular environment, the dynamic programming algorithm to obtain the control point subsequences, the crowded control point removal and Bezier curve construction methods are described in the section. The experimental results are presented in section IV and section V concludes this paper.

The research was partially supported by National Science Council under contract NSC 96-2221-E-001-018-MY2.

Yi-Ju Ho and Jing-Sin Liu are with the Institute of Information Science, Academia Sinica, Nangang, Taipei, Taiwan 115, ROC. u882524@alumni.nthu.edu.tw, liu@iis.sinica.edu.tw

II. PRELIMINARY

In this section, we briefly introduce the definitions and properties of the Bezier curve [12] and the Voronoi diagram [14] .

A. Bezier Curve

A Bezier curve of degree n is represented by $n+1$ control points $P_0, ..., P_n$:

$$P(\lambda) = \sum_{i=0}^{n} B_i^n(\lambda) P_i, \quad \lambda \in [0,1], \qquad (1)$$

$$B_i^n(\lambda) = \begin{pmatrix} n \\ i \end{pmatrix} (1-\lambda)^{n-i} \lambda^i, \quad i \in \{0, 1, ..., n\}. \qquad (2)$$

The curve segment starts at P_0, ends at P_n and lies entirely within the convex hull of the control points. We can use these properties to construct composite Bezier curve to avoid obstacles. In fig.1, the convex hull of the control points P_0, P_1, P_2, P_3 and P_4 does not collide with obstacles. This guarantees that the Bezier curve constructed by the control points also does not collide with the obstacles. We will use this property in our dynamic programming algorithm in section III.

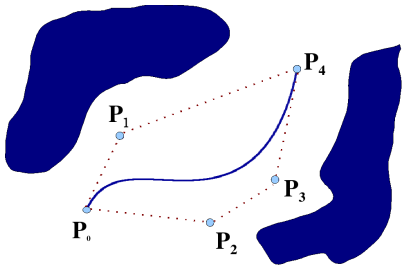

Fig. 1. convex hull of control points to avoid collision

Considering the Bezier curve of control points $\{P_0, P_1, P_2, ..., P_n\}$, the tangent at P_0 must be given by $P_1 - P_0$ and the tangent at P_n by $P_n - P_{n-1}$. The second derivative at P_0 must be determined by P_0, P_1 and P_2 and the one at P_n must be determined by P_{n-2}, P_{n-1} and P_n. This property can be generalized for higher order derivatives at the curve's endpoints. In general, the rth derivatives at the endpoint must be determined by its r neighboring control points. This property is used to ensure r degree continuity at the joins of the connections of Bezier curves.

Because the Bezier curve starts at the first control point and ends at the last control point, either one of the end points of two Bezier curves must have the same position to make these two curves connected. If we consider C1 continuity in connecting two Bezier curves, the last two control points of the first curve and the first two control points of the second curve must be collinear and the distances of the two control points in each curve are the same. Take fig.2 for example. The last control point of the Bezier curve B_i (P_n) and the first control point of the Bezier curve B_{i+1} (Q_0) have the same position to make the two Bezier curves connected. The last two control points P_{n-1}, P_n of B_i and the first two control points Q_0, Q_1 of B_{i+1} are collinear and $P_n - P_{n-1}$ equals $Q_1 - Q_0$ to make the composite Bezier curve C1 continuity.

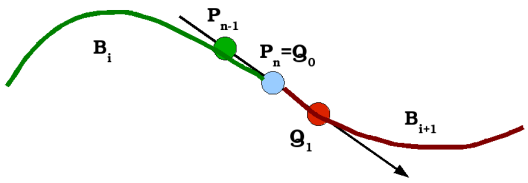

Fig. 2. C1 Continuity for Bezier Curves

B. Voronoi Diagram

Let $P = \{p_0, p_2, ..., p_n\}$ be a set of points in the two-dimensional Euclidean plane. These are called the sites. Partitioning the plane by assigning every point in the plane to its nearest site forms the Voronoi region $V(p_i)$. $V(p_i)$ consists of all the points at least as close to p_i as to any other site:

$$V(p_i) = \{x : |p_i - x| \leq |p_j - x| \forall\ j \neq i\}$$

The set of all points that have more than one nearest neighbor form the Voronoi diagram $V(P)$ for the set of sites. That any point in the Voronoi region is closest to the site is an useful property for obstacle avoiding path planning problem. Many algorithms have been proposed for constructing the Voronoi diagram and many researches study the application of it [14]. In this paper, we adopt Fortune's algorithm [8] to construct the Voronoi diagram because of its outstanding performance with worst case $O(nlog(n))$ time complexity.

III. PROPOSED ALGORITHM

Our algorithm is a three step method. The first step is to generate a continuous piecewise linear path by piecewise straight lines. The nodes on the piecewise linear path are used as control points for the composite Bezier curve. The second step is to subdivide the control points into subsequences such that each convex hull of the subsequences does not collide with the obstacles and the curve length of the composite Bezier curve will approaches the shortest. The last step is to further remove the crowded control points and to generate the composite Bezier curve by adding extra control points into the subsequences to meet the kinematic constraints.

A. Initialization: Piecewise Linear Path along Voronoi Diagram

For irregular obstacles, we divide the boundaries of each obstacle into segments and the end points of each segment are the sites of Voronoi diagram. Then we construct a Voronoi diagram basing on the sites, as shown in fig.3(a). Consequently, as shown in fig.3(b), all edges of the Voronoi diagram colliding with the obstacles are removed from the diagram. We connect the source and destination to the corners of the remaining Voronoi regions in which the source

and destination node are located and the newly created edges can not collide with the obstacles. We then use Dijkstra's shortest path algorithm to obtain the shortest path in the remaining diagram. The resulting path is the piecewise linear path that has better clearance to surrounding obstacles. The nodes on the path are the control points for piecewise Bezier curves.

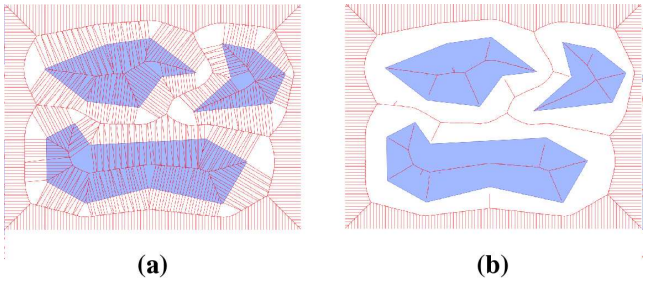

Fig. 3. (a). Voronoi diagram with sites on the boundaries of the obstacles (b). Voronoi diagram without edges colliding with the obstacles

B. Control Point Subdivision

We sample a number of nodes on the piecewise linear path as the control points for Bezier curves. The second step is to subdivide the control points into subsequences and each subsequence contains ordered control points for a Bezier curve. As the characteristic of Bezier curve, the curve segment will fall entirely in the convex hull of the control points. In order to obtain a collision free composite Beizer curve, we have to subdivide the control points into ordered subsequences such that the convex hull of each subsequence does not collide with the obstacles. There are many solutions to subdivide the control points to satisfy the collision free constraint. Take fig.4 for example, we want to design a path travelling from P_0 to P_6 with $P_0, P_1, ...P_6$ as control points. In order to avoid collision, we have to make sure the convex hull of the control points for each subsequence does not collide with the obstacles. In fig.4(a), we subdivide the control points into two ordered subsequences, $\{P_0, P_1, P_2\}$ and $\{P_2, P_3, P_4, P_5, P_6\}$, while in fig.4(b), we subdivide the control points into three ordered subsequences, $\{P_0, P_1, P_2\}$, $\{P_2, P_3, P_4\}$, and $\{P_4, P_5, P_6\}$. The composite Bezier curves formed by the two different subdivisions are both collision free. However, the path length of fig.4(a) is shorter than the one of fig.4(b). In this paper, we utilize dynamic programming to subdivide the control points into an ordered set of subsequences and the corresponding composite Bezier curve will have shortest curve length.

The problem we want to solve is then modeled as below: Let $P_0, P_1, ...P_n$ be an ordered control point sequence. We want to subdivide the control points into ordered subsequences $S_1, S_2, ..., S_m$, and the last node in S_{i-1} is the first node in S_i to make the resulting curves connected. The convex hull of each subsequence can not collide with the obstacles. We want to choose one subsequence division with shortest curve length.

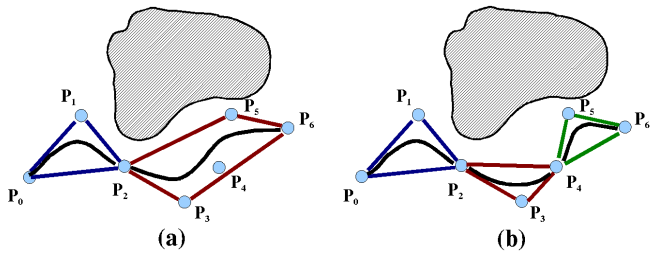

Fig. 4. (a). Subdivide the control points into $S_1 = \{P_0, P_1, P_2\}, S_2 = \{P_2, P_3, P_4, P_5, P_6\}$ for two Bezier curves (b). Subdivide the control points into $S_1 = \{P_0, P_1, P_2\}, S_2 = \{P_2, P_3, P_4\}, S_3 = \{P_4, P_5, P_6\}$ for three Bezier curves

Let $collide(i, j)$ denotes whether the convex hull of the control points $\{P_i, P_{i+1}, ...P_j\}$ collides with the obstacles.

$$collide(i,j) = \begin{cases} true, \ if \ collide(i, j-1) = true \ or \\ \qquad edge \ \overline{P_k P_j} \ collides \ with \ obstacles \\ \qquad \exists k, i \leq k < j \\ false, \ otherwise \end{cases} \tag{3}$$

If the convex hull collides with the obstacles, the value of $collide(i, j)$ is $true$, otherwise the value of $collide(i, j)$ is $false$. With the $collide(i, j)$ value, we can determine whether a solution is feasible. By observation, we know that the convex hull of $\{P_i, P_{i+1}, ..., P_{j-1}\}$ is entirely in the convex hull of $\{P_i, P_{i+1}, ..., P_{j-1}, P_j\}$. Thus, if the convex hull of $\{P_i, P_{i+1}, ..., P_{j-1}\}$ collides with the obstacles, the convex hull of $\{P_i, P_{i+1}, ..., P_j\}$ also collides with the obstacles.

For simplification, we use a $(n + 1) \times (n + 1)$ array to store the $collide(i, j)$ values for $n + 1$ control points. The algorithm $calcCollide$, as shown in fig.5, is used to calculate the $collide(i, j)$ values for all (i, j), $0 \leq i, j \leq n$.

Because the Bezier curve has no close form solution for the curve length, we have to divide the curve into small enough pieces and sum up their straight line distance as the curve length. The calculation of curve length takes a lot of time. In order to improve the performance, we use the straight line distance between two end points of a Bezier curve as its curve length.

Let P_{ix} denote the x position and P_{iy} denote the y position of the control point P_i. We define $F(i, j)$ as the shortest curve length among all subdivision solutions from point P_i to P_j,i.e.

$$F(i,j) = min\{|P_i - P_j|, F(i,k) + F(k,j) \ \forall i < k < j\} \tag{4}$$

where

$$|P_i - P_j| = \begin{cases} \sqrt{(P_{ix} - P_{jx})^2 + (P_{iy} - P_{jy})^2)} \\ \quad if \ collide(i,j) = false \\ inf, \ otherwise \end{cases} \tag{5}$$

We use a two dimensional array with size $n+1$ by $n+1$ to store the solutions for $n + 1$ control points. The control

Algorithm : calcCollide

Input : Control Point Sequence $\{P_0, P_1, ...P_n\}$

Output : $collide(i,j)\ array,\ \forall 0 \le i,j \le n$

// Calculate the $collide(i,j)$

1. Initialize $collide(i,j)\ equals\ false\ \forall 0 \le i,j \le n$
2. for $i = 0$ to n begin
3. for $j = i$ to n begin
4. if $(collide(i, j-1) = false)$ begin
5. for $k = i$ to $j-1$ begin
6. if ($\overline{P_k P_j}$ collides with an obstacle) begin
7. $collide(i,j) = true$;
8. break;
9. end if
10. end for
11. end if
12. else begin
13. $collide(i,j) = true$;
14. end
15. end for
16. end for

Fig. 5. collide(i,j) Calculation Algorithm

Algorithm : Control Point Subdivision

Input : Control Point Sequence $\{P_0, P_1, ...P_n\}$

Output : Control Point Subsequences $\{S_1, S_2, ...S_m\}$

1. calcCollide(); //calculate the $collide(i,j)$
2. for $k = 1$ to n begin
3. for $i = 0$ to $n-k$ begin
4. $j = i + k$
5. $e(i,j).val = inf$
6. if (!$collide(i,j)$) begin
7. $e(i,j).val = \sqrt{(P_{ix} - P_{jx})^2 + (P_{iy} - P_{jy})^2}$
8. $(e(i,j).r1, e(i,j).c1) = (i,j)$
9. end if
10. for $l = i+1$ to $j-1$ begin
11. if $(e(i,l).val + e(l,j).val < e(i,j).val)$ begin
12. $e(i,j).val = e(i,l).val + e(l,j).val$;
13. $(e(i,j).r1, e(i,j).c1) = (i,l)$;
14. $(e(i,j).r2, e(i,j).c2) = (l,j)$;
15. end if
16. end for
17. end for
18. end for
19. Backtrace the best solution from $e(0,n)$ to
20. obtain the subsequences $S_1, S_2..., S_m$

Fig. 6. Control Point Subdivision Algorithm

point subdivision algorithm is shown in fig.6. We use $e(i,j)$ to denote the entry in the array with index (i,j). Each $e(i,j)$ is a five tuple entry with value $(val, r1, c1, r2, c2)$. val is the $F(i,j)$ value of the entry, $(r1,c1)$ and $(r2,c2)$ denote the indexs of the entries by which we use to obtain the $F(i,j)$. If $F(i,j)$ is calculated by $|P_i - P_j|$, then $(r1,c1)$ equals to the current entry's index (i,j). We calculate the entry's value from $(0,1)$, $(1,2)$, ..., to (n,n) diagonally. After all entries' values are calculated, the entry $(0,n)$ represents the best solution how we subdivide the control points. We backtrace the solutions from $(0,n)$ to obtain the subsequences of control points. At backtrace step, we trace back recursively toward the entries $(e(i,j).r1, e(i,j).c1)$ and $(e(i,j).r2, e(i,j).c2)$. The backtrace step stops when $(e(i,j).r1, e(i,j).c1)$ equals (i,j) and then $P_i, P_{i+1}, ...P_j$ form a subsequence. Then the piecewise Bezier curves can be constructed according to the ordered control point subsequences.

C. Shortening and Smoothing by Control Point Removal and Addition and Bezier Curve Construction

Let $S_1, S_2, ...S_m$ denote the resulting ordered subsequences obtained by control point subdivision algorithm (see fig.6) and each S_i contains the control points used for a Bezier curve. The last step in our algorithm is to construct Bezier curves according to the control point subsequences. By observation, we notice that the crowded control points often result in longer curve length. As shown in fig.7(a), the crowded control points P_k, P_{k+1}, P_{k+2} attract the Bezier curve closer to them and result in longer curve length. If we remove the crowded control points carefully, we can reduce the curve length and retain the curve shape, as shown in fig.7(b). Before constructing the Bezier curves, we first reduce the crowded control points in each subsequence without significantly changing the curve shapes. We note that the convex hull of a subset of the control points in subsequence S_i is entirely in the convex hull of the control points in the subsequence S_i and therefore the Bezier curve of the new control point subsequence does not collide with the obstacles. For each subsequence S_i, the first control point and the last control point are non-removable since they're required for connection with other Bezier curves. For other control points in a subsequence, we determine whether there are nearby control points which can be removed from the subsequence. Take fig.7(a) for example, if P_k is the base control point being processed, we remove the consequent control points P_{k+1}, P_{k+2} which are all close to P_k within a threshold ϵ and fig.7(b) shows the resulting subsequence after removal. The crowded control point removal algorithm is shown in fig.8.

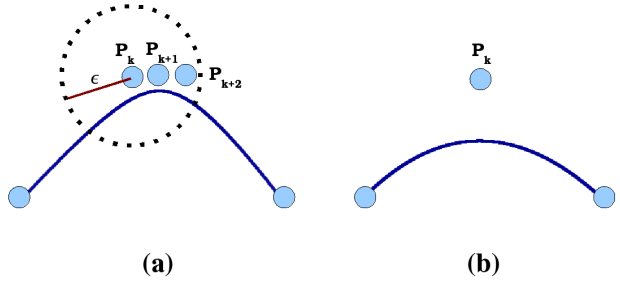

Fig. 7. (a). Bezier curve with crowded control points. (b). Shorter Bezier curve with crowded control points removed

To be generalized to high order continuity, $S_{i,extra} = \{P_{extra1}, P_{extra2}, ...\}$ depending on the connection continuity requirements are imposed on the composite Bezier curve.

Algorithm : Crowded Control Point Removal
Input : Control Point Subsequence $S_i = \{P_k, P_{k+1}, ...P_l\}$ Distance Threshold ϵ
1. $P_{base} = P_k$ 2. for $P_t = P_{base+1}$ to P_{l-1} begin 2. if ($\sqrt{(P_{tx} - P_{basex})^2 + (P_{ty} - P_{basey})^2} \leq \epsilon$) begin 3. remove P_t from S_i; 4. end if 5. else begin 6. $P_{base} = P_t$; 7. Goto line 2; 8. end 9. end for

Fig. 8. Crowded Control Point Removal Algorithm

In this paper, we only consider the C1 continuity even though the C2 and curvature continuity are easy to adapt. For the set of ordered subsequences $\{S_1, ...S_m\}$, suppose the Bezier curve of S_i is connected to the Bezier curves of S_{i-1} at P_{last}. In order to make the composite Bezier curves C1 continuous, it is required that an extra control point P_{extra1} to S_i is added so as to make $P_{extra1} - P_{last}$ equals the tangent of S_{i-1} at P_{last}, i.e. $P_{last} - P_{last-1}$. Fig.9 demonstrates the basic concept. After the extra control points are added in each subsequence, we construct the Bezier curve for each subsequence as the composite Bezier path. It has been proved that the curve length of a Bezier curve is upper bounded by the length of the piecewise linear path which connects the control points [8]. Our algorithm always obtains a shorter smoothing path than the piecewise linear path along the Voronoi diagram.

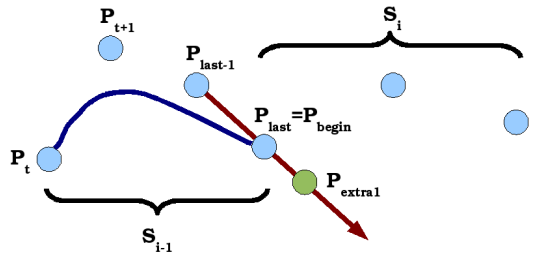

Fig. 9. Extra control point addition in sequence S_i

IV. EXPERIMENTAL RESULT

Instead of building our algorithm into real robots, we use software simulation to test our algorithm. We use the C++ programming language in Linux operating system with 2.4 GHz cpu and 1G Byte memory. We test several maps which contain different number of obstacles.

We test several maps for our algorithm and the maps are shown in fig.10,11,12 and 13. For each map, we construct four kind of paths. The first path is the globally shortest piecewise linear path in visibility graph[4], called VS-Path (shown as the straight line in Map1 to Map4). The second one is the piecewise linear path along the Voronoi diagram (R-Path), the third and fourth paths are the composite Bezier curve without/with crowded control point removal, called CB-Path/CBR-Path, shown as the blue and yellow Bezier curves in the maps respectively. We compare the path length and execution time of the VS-Path, R-Path, CB-Path and CBR-Path. The experimental results for the maps are listed in Table.I and Table.II. We note that in more complicated environments, the execution time for VS-Path is much longer than the CBR-Path while the CBR-Path length is over 10% longer than the VS-Path. This is due to that the time complexity of the VS-Path is $O(n^2)$ which is larger than the time complexity $O(nlog(n))$ of CBR-Path. Oppositely, our algorithm can obtain shorter length while comparing to R-Path. In map4, the CBR-Path obtained by our algorithm is 8.83% shorter than the R-Path.

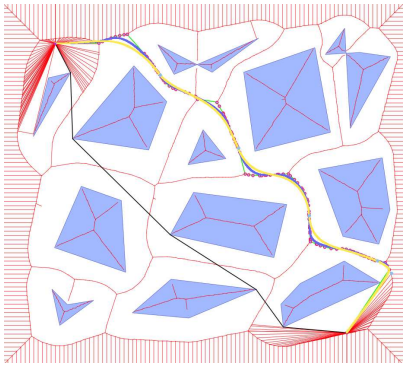

Fig. 10. Map1 : size 934x843, 12 obstacles

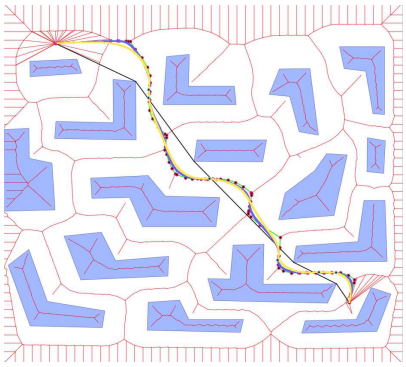

Fig. 11. Map2 : size 934x843, 16 obstacles

V. CONCLUSION AND FUTURE WORK

We develop an obstacle avoiding path planning algorithm based on Voronoi diagram and composite Bezier curve. Our algorithm can obtain a path with near shortest curve length while taking the Voronoi diagram as reference skeleton. In our experimental results, our algorithm can obtain a composite Bezier curve with curve length 8.83% shorter than the piecewise linear rough path on Voronoi diagram.

Our future work will focus on two topics. The first topic is to adopt our algorithm into real world applications, such as ball passing problem in Robot Soccer Game or grand

Map Name	VS-Path		CB-Path				CBR-Path ($\epsilon = 20$)			
	length	time (s)	length	inc.%	time (s)	red.(s)	length	inc.%	time (s)	red.(s)
map1	1036.5399	0.04	1221.8672	17.88%	2.85	-2.81	1188.4306	14.65%	2.85	-2.81
map2	931.4670	0.43	1089.3447	16.94%	1.45	-1.02	1059.6355	13.75%	1.42	-1.02
map3	1486.4561	21.73	1754.7962	18.05%	9.97	11.76	1699.6347	14.34%	9.97	11.76
map4	1328.3000	137.41	1471.6198	10.78%	18.01	119.4	1425.8437	7.36%	18.01	119.4

TABLE I

PATH COMPARISION : VS-PATH, CB-PATH AND CBR-PATH

Map Name	R-Path		CB-Path				CBR-Path ($\epsilon = 20$)			
	length	time (s)	length	red.%	time (s)	inc.(s)	length	red.%	time (s)	inc.(s)
map1	1272.9252	1.15	1221.8672	4.01%	2.85	1.70	1188.4306	6.63%	2.85	1.70
map2	1156.5316	0.5	1089.3447	5.80%	1.45	0.95	1059.6355	8.37%	1.42	0.95
map3	1826.9459	4.94	1754.7962	3.94%	9.97	5.03	1699.6347	6.96%	9.97	5.03
map4	1563.9850	11.72	1471.6198	5.90%	18.01	6.29	1425.8437	8.83%	18.01	6.29

TABLE II

PATH COMPARISION : R-PATH, CB-PATH AND CBR-PATH

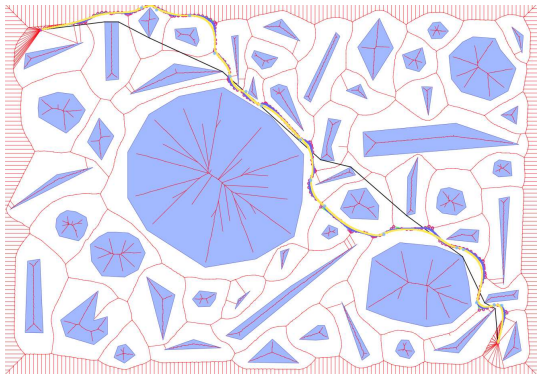

Fig. 12. Map3 : size 1329x928, 52 obstacles

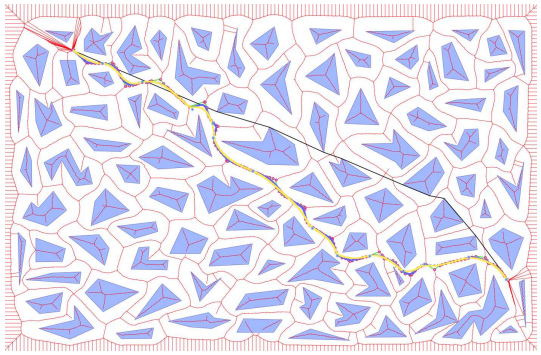

Fig. 13. Map4 : size 1408x917, 95 obstacles

challenge problem. The second one is to further improve our algorithm. Although our algorithm can achieve the short path length when the control points are determined, the control point generation method greatly affects the path planning result. Our next topic will focus on the control point refinement to obtain even shorter path length.

REFERENCES

[1] Priyadarshi Bhattacharya and Marina L. Grvrilova, "Voronoi diagram in optimal path planning", *in 4th IEEE International Symposium on Voronoi Diagrams in Science and Engineering*, 2007, pp.38-47.

[2] Ji-wung Choi, Renwick E. Curry and Gabriel Hugh Elkaim, "Obstacle Avoiding Real-Time Trajectory Generation and Control of Omnidirectional Vehicles", *in American Control Conference*, 2009.

[3] Trajano Alencar de Araujo Costa, Armando Morado Ferreira and Max Suell Dutra, "PARAMETRIC TRAJECTORY GENERATION FOR MOBILE ROBOTS", *ABCM Symposium Series in Mechatronics*, Vol.3, 2008, pp.300-307.

[4] Halit Eren, Chun Che Fung and Jeromy Evans, "Implementation of the Spline Method for Mobile Robot Path Control", *in 16th IEEE Instrumentation and Measurement Technology Conference*, Vol.2, 1999, pp.739-744.

[5] S. Fortune, "A sweepline algorithm for Voronoi diagrams", *Proceedings of the second annual symposium on Computational geometry*, 1986, pp.313-322

[6] Shilpa Gulati and Benjamin Kuipers, "High Performance Control for Graceful Motion of an Intelligent Wheelchair", *in IEEE International Conference on Robotics and Automation*, 2008, pp.3932-3938.

[7] El-Hadi Guechi, Jimmy Lauber and Michel Dambrine, "On-line moving-obstacle avoidance using piecewise Bezier curves with unknown obstacle trajectory", *in 16th Mediterranean Conference on Control and Automation*, 2008, pp.505-510.

[8] Ron Goldman, "PYRAMID ALGORITHMS: A Dynamic Programming Approach to Curves and Surfaces for Geometric Modeling", Morgan Kaufmann, 2003, p254.

[9] Jung-Hoon Hwang, Ronald C. Arkin and Dong-Soo Kwon, "Mobile robots at your fingertip: Bezier curve on-line trajectory generation for supervisory control", *in IEEE International Conference on Intelligent Robots and Systems*, Vol.2, 2003, pp.1444-1449.

[10] Shahin Mohammadi and Nima Hazar, "A Voronoi-Based Reactive Approach for Mobile Robot Navigation", *Advances in Computer Science and Engineering*, Springer Berlin Heidelberg, Vol.6, 2009, pp.901-904.

[11] Evgeni Magid, Daniel Keren, Ehud Rivlin and Irad Yavneh, "Spline-Based Robot Navigation", *in International Conference on Intelilgent Robots and Systems*, 2006, pp.2296-2301.

[12] M.E. Mortenson, "Geometric modeling", 2nd edition, John Wiley&Sons, 1997.

[13] K. Nagatani, Y. Iwai and Y. Tanaka, "Sensor Based Navigation for car-like mobile robots using Generalized Voronoi Graph", *in IEEE International Conference on Intelligent Robots and Systems*, Vol.2, 2001, pp.1017-1022.

[14] A. Okabe, B. Boots and K. Sugihara, "Spatial Tessellations: Concepts and Applications of Voronoi Diagrams", 2nd edition, John Wiley&Sons, 2000.

[15] Igor Škrjanc and Gregor Klančar, "Cooperative Collision Avoidance between Multiple Robots Based on Bézier Curves", *in 29th International Conference on Information Technology Interfaces*, 2007, pp.451-456.