

Implementation Roadmap using Voronoi Diagrams for Vision-based Robot Motion

Shahed Shojaeipour^{1,1}, Sallehuddin Mohamed Haris¹ and Ali Shojaeipour²
 Dept. of Mechanical & Material Engineering¹, Dept. of Computer Software Engineering²
 Universiti Kebangsaan Malaysia¹, Islamic Azad University of Shirvan²
 Bangi, Selangor, 43600, Malaysia¹, Shirvan University Ave, Iran²
 Malaysia¹, Iran²
 shojaei@vlsi.eng.ukm.my, salleh@eng.ukm.my, ali.shojaeipour@yahoo.com

Abstract: - In this paper, we present a method to navigate a mobile robot using a webcam. This method determines the shortest path for the robot to transverse to its target location, while avoiding obstacles along the way. The environment is first captured as an image using a webcam. Image processing methods are then performed to identify the existence of obstacles within the environment. Using the Voronoi Diagrams VD(s) method, locations with obstacles are identified and the corresponding Voronoi cells are eliminated. From the remaining Voronoi cells, the shortest path to the goal is identified. The program is written in MATLAB with the Image Processing toolbox. The proposed method does not make use of any other type of sensor other than the webcam.

Key-Words: - Mobile robot, Path planning, Voronoi Diagrams, Image processing, Visual servo.

1 Introduction

Image processing is a form of signal processing where the input signals are images such as photographs or video frames. The output could be a transformed version of the input image or a set of characteristics or parameters related to the image. The computer revolution that has taken place over the last 20 years has led to great advancements in the field of digital image processing. This has in turn, opened up a multitude of applications in various fields, in which the technology could be utilised.

The aim of this paper is to present a method for visual servo control using only visual images from a webcam. Visual servo is the use of image data in closed loop control of a robot. Without doubt, today, the use of vision in robotic applications is rapidly increasing. This is due to the fact that vision based sensors such as webcams are falling in price more rapidly than any other sensor. It is also a richer sensor than traditional ranging devices, particularly since a camera captures much more data simultaneously [1].

Images can be captured by camera, and subsequently, processed using some particular software. Among them, MATLAB, with its Image Processing toolbox, is well suited to perform such tasks. Information obtained from the image processing exercise can then be used to generate motion commands to be sent to the mobile robot. This sequence is depicted in Fig. 1. Consequently, the robot imitates human vision in stages as follows:

- Image acquisition
- Image processing
- Image analysis and assimilation

- Image intelligence
- Control signal reception
- Motion control of parts of the robot

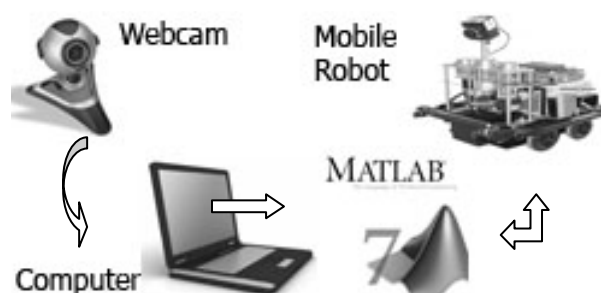


Fig. 1 Experimental Setup.

The Voronoi tessellation, originally proposed by Georgy Voronoi in 1907[2], is a special decomposition of a metric space based on the distances of points to a specified discrete set of sites within the space. For a set S containing i number of sites in Euclidean space, the tessellation is defined by associating a cell to each site. Cell i contains all points that are closest to s_i and the cell boundaries are hyper planes made up of points that are equidistant to two or more sites in S (Fig. 2). The Voronoi diagram (VD(S)) perfectly partitions the space, and it has found use in the sciences for solving problems that involve the assignment of space between groups of objects. In general, the interested reader should refer to the book written in [3]0.

Many methods have been developed and used by different researchers to compute Voronoi tessellations.

For example, in [4], the Voronoi C tree data structure was introduced to generate generalized 3D Voronoi diagrams. In [5], morphological operations were used to transform image data obtained from sensors into its corresponding VD(S) where the skeletal lines represent obstruction free paths. There also exist a number of software packages readily available for computing the Voronoi tessellation. These include, for example, the software package [6] which can compute VD(S) in arbitrary dimensions and the Voronoi function in MATLAB. The program in [7] which is well-known for mesh generation via Delaunay Triangulation also computes Voronoi tessellations.

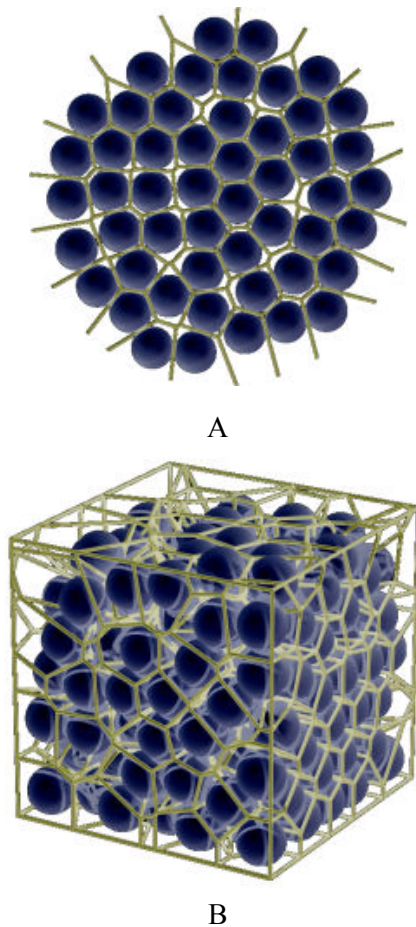


Fig. 2 (A) the 2D Voronoi tessellation (B) the 3D Voronoi tessellation.

Mesh computation codes can compute VD(S) of single objects which, given a set of points, will return the complete mesh dividing the space containing the set of sites into a mesh of cells as seen in figure 3. However, in practical applications, it is more natural to associate a single VD(S) with each particle, and compute them separately. This makes it easier to compute just a subset of Voronoi meshes, or to tailor the computations to handle special cases and complex boundary conditions.

This also makes it straightforward to compute mesh-based statistics, such as mesh volumes, or the number of faces per mesh [8], [9].

Assuming we have point-based obstacles and a point robot, we can use the Voronoi diagram to navigate. You can think of the Voronoi diagram as a Voronoi graph, made up of edges and vertices. To go from a $Start_{point}$ to a $Goal_{point}$, we simply find the nearest points on the Voronoi graph to $(Start_{point} ; Goal_{point})$: $(Start_{point} ; Goal_{point})$. We then use a standard graph search type algorithm (e.g Dijkstra's algorithm) to traverse the vertices and edges of the graph from $Start_{point}$ to $Goal_{point}$.

Suppose the robot isn't a point and the obstacles aren't a point? The two-dimensional region in which the robot moves will contain buildings and other types of barriers, each of which can be represented by a convex or concave polygonal obstacle. To find the generalized Voronoi diagram for this collection of polygons, we can use an approximation based on the simpler problem of computing the Voronoi diagram for a set of discrete points. We estimate the polygonal obstacle boundaries by discrete points.

1.1 Methodology:

1. Approached the boundaries of the polygonal obstacles with the large number of points that result from dividing each side of the original polygon into small segments.
2. The Voronoi diagram computes for this collection of approximating points.
3. Once this complicated Voronoi diagram is constructed, eliminate those Voronoi edges which have one or both endpoints lying inside any of the obstacles.
4. The Voronoi edges remaining form a good approximation of the generalized Voronoi diagram for the original obstacles in the map.

To take into account a robot which isn't point size, we need to find critical points and critical lines in the Voronoi diagram. These are places where the Voronoi path has a local minimum. At these points we can see if the robot's diameter will fit through the space at the critical point: is the diameter greater than the critical line length. Note that this assumes a fixed robot orientation, or we can use the maximum diameter of the robot over all rotations.

2 Visual Obstacles

The path to be transverse by the robot must be ensured to be free of obstacles. For this, their existence must be identified and their positions located. This section describes how this could be done.

The image is recorded by a webcam which is installed above the robot. The image is then sent via a USB cable to a PC, to be processed by MATLAB. The experiments were carried out using a computer with 3.60GB free space hard disk and 1GB RAM memory. The algorithm was developed using MATLAB (version 7.6 R2008a).

The image is divided into segments, which become the export databases; usually they are the raw pixels data abstracted from the captured image [10], [11] and [12]. The picture could be in JPG or BMP format, in which case, every pixel point uses three numerical values, representing intensity levels of the primary colours: red, green and blue (RGB) to depict its characteristics. Therefore, in such format, computing workload to perform image processing would be very high. Hence, it would be desirable to convert the coloured picture into a grayscale image [13], [14].

Image processing methods are firstly used to identify the existence of obstacles within the image frame. This is implemented in an eight step MATLAB (with the Image Processing Toolbox) program. The following describes the steps:

Step 1. Generate input video objects.

This can be implemented using the command

```
Obj = videoinput('adaptorname', device name, 'format');
```

where the adaptor name can be determined using the 'imaqhwinfo' command, device name is the name given to the device and format refers to the required image format.

Step 2. Preview the webcam video image.

```
Preview('object name');
```

where object name is Obj in the last command

Step 3. Set brightness level of image.

```
set(obj, 'property name' property value);
```

Step 4. Capture still image from webcam video.

```
getsnapshot(object name);
```

The captured image is stored as an array whose elements represent the light level.

Step 5. Remove the input device from memory.

```
delete( object name);
```

Step 6. Convert from RGB to grayscale mode.

```
I=rgb2ind(I,colorcube(150));
```

Step 7. Find edges of objects in the image.

```
I=edge(I,'sobel',(graythresh(I)*.1));
```

Step 8. Remove noise.

```
Se90=strel('line',3,90); Se0=strel ('line',3,0);
```

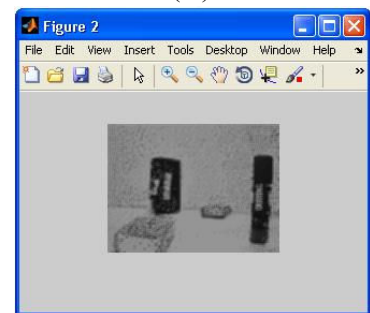
```
I=imdilate(I,[se90 se0]);
```

```
I=imfill(I,'holes');
```

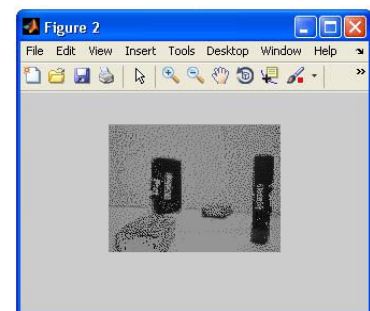
The results of running this program can be seen in Fig. 3, where (A) is the original image, (B)-(F) are the intermediate stages of the image and (G) is the final image.



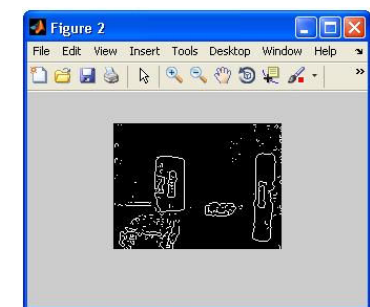
(A)



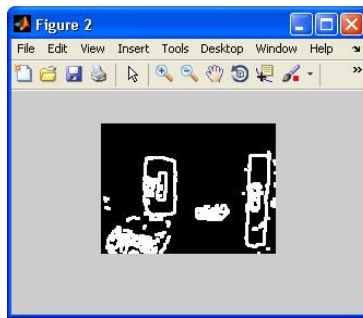
(B)



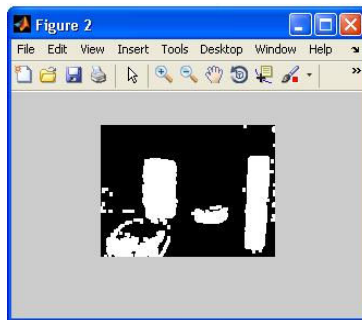
(C)



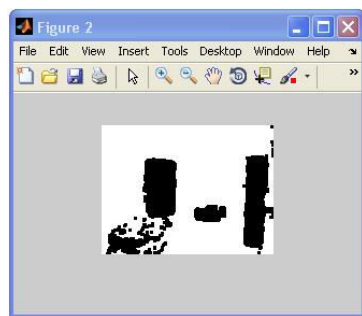
(D)



(E)



(F)



(G)

Fig. 3 (A) Original Image, from B to F- intermediate stages of filtering and (G) Final image.

3 Path Planning via Voronoi Diagrams

In order to select the shortest path for the robot to transverse, three actions need to be considered: the first is image capture using webcam, the second is to convert the image into a 3D scene, using the Spectral Fractal Dimension (SFD) technique [15] and [16]. Previous works have used just two images of the scene for this purpose [17], [18] and [19]. The third action is to use cell decomposition to identify and eliminate paths that are obstructed. The robot would then be able to choose the shortest of the remaining paths. Fig. 4 illustrates the path finding problem, where three paths are being considered. Then,

- Path(1) is not possible (encounter obstacle)
- Path(2) is the shortest distance and possible

Path(3) is possible but it isn't the shortest path. Obviously, the number of objects and the number of paths would vary, depending on situation. We will focus on using a general program to find the shortest path between the robot and the target [20].

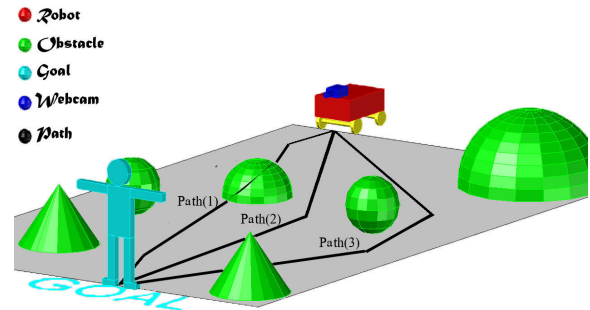


Fig. 4 Analyzed Paths.

In order to send the correct control commands to the robot such that the chosen path is followed, the precise position of objects must be measured. Feedback is the comparison of the target and actual positions, and is a natural step in implementing a motion control system. This comparison generates an error signal that may be used to correct the system, thus yielding repeatable and accurate results. [21]

For data transfer to and from the robot, the MATLAB Instrumentation Control toolbox is used. Data transfer may be performed via GPIB, VISA, Serial, TCP/IP or UDP interface.

3.1 Generation of Voronoi Diagrams:

VD(S) are constructed by first performing the Delaunay Tessellation which is regarded as the dual to Voronoi Tessellations. Firstly, any two sites p, q for which there exists a circle C that passes through p and q and doesn't contain any other site of S in its interior or boundary, are connected by a line segment. The set of such line segments form the edges of the Delaunay Tessellation $DT(s)$, called Delaunay edges[8], [23] and [24]. Now, bisection of the Delaunay edges by another set of line segments results in the formation of Voronoi cells where each cell contains one site enclosed by the line segments forming Voronoi edges.

An example is depicted in figure 3, where $VD(S)$ is depicted by solid lines and $DT(S)$ by dashed lines. Note that a Voronoi vertex need not be contained in its associated face of $DT(S)$. The sites p, q, r, s are co-circular, giving rise to a Voronoi vertex v . Consequently, its corresponding Delaunay face is boarded by four edges 0 [24].

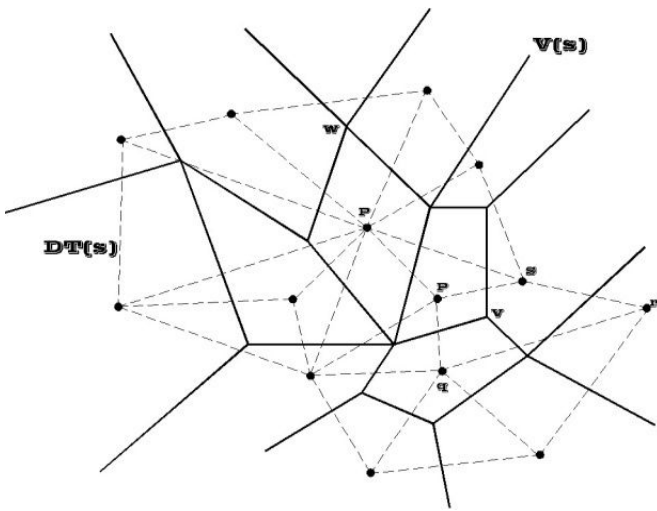


Fig. 5 Voronoi Diagram and Delaunay Tessellation.

3.2 Motion Planning using VD(S):

Application of Voronoi diagrams to mobile robot motion planning has been discussed in [24]. The concept is reproduced in the following excerpt.

Suppose that for a disc-shaped robot centered at some start point, s , a motion to some target point, t , must be planned in the presence of n line segments as obstacles.

We assume that the line segments are pair wise disjoint, and that there are four line segments enclosing the scene, as shown in figure 5. While the robot is navigating through a gap between two line segments, l_1 and l_2 , at each position x its "clearance", i.e. its distance

$$d(x, l_i) = \min\{d(x, y) : y \in l_i\} \quad (1)$$

to the obstacles, should be a maximum. This goal is achieved if the robot maintains the same distance to either segment. In other words, the robot should follow the bisector $B(l_1, l_2)$ of the line segments l_1 and l_2 until its distance to another obstacle gets smaller than $d(x, l_i)$. Roughly, this observation implies that the robot should walk along the edges of the Voronoi diagram $VD(S)$ of the line segments in $S = \{l_1, \dots, l_n\}$. This diagram is connected, due to the four surrounding line segments.

If start and target points are both lying on $VD(S)$, the motion planning task immediately reduces to a discrete graph problem: After labeling each edge of $VD(S)$ with its minimum distance to its two sites, and adding s and t as new vertices to $VD(S)$, a breadth first search from s will find, within $O(n)$ time, a path to t in $VD(S)$ whose minimum label is a maximum. If this value exceeds the robot's radius, a collision-free motion has been found.

If the target point, t , does not lie on $VD(S)$, we first determine the line segment $l(t)$ whose Voronoi region contains t . Next, we find the point $z(t)$ on $l(t)$ that is closest to t ; see figure 6. If its distance to t is less than the robot's radius then the robot cannot be placed at t and no motion from s to t exists. Otherwise, we consider the

ray from $z(t)$ through t . It hits a point t' on $VD(S)$ which serves as an intermediate target point.

Similarly a point s' can be defined if the original start point, s , does not lie on $VD(S)$ [24].

3.3 Shortest Path using Voronoi Diagrams:

A Voronoi Diagrams representation of free space is constructed in order to reduce the search space for finding a collision-free path for the mobile object [25]. In order to conduct an efficient search, the Voronoi Diagrams is first converted to an equivalent graph of nodes and arcs. A simple example of a Voronoi Diagrams graph is shown in Fig. 6 where the circles represent nodes, and the lines connecting them represent arcs.

There are three types of nodes which appear in the Voronoi Diagrams graph: junction nodes, terminal nodes, and pseudo nodes. A junction node is generated where three or more arcs of the Voronoi Diagrams intersect. A terminal node corresponds to a dead end of a Voronoi Diagrams arc. Terminal nodes arise when concave vertices are present. Since the workspace itself can be regarded as a hole inside of a large obstacle, the vertices of a rectangular workspace are terminal nodes as can be seen in Fig. 6, the third type of node is a pseudo node called a source node or a goal node. Source and goal nodes are artificial nodes inserted in the graph near the source and goal positions, respectively. These nodes represent entry and exit points at which the mobile object gets onto and off of the graph.

The arcs of the graph are lines connecting pair's nodes. Each arc is represented as a sequence of piecewise-linear segments between via-points. The data structure used to represent an arc is summarized in Table I.

TABLE I
Representation of a Voronoi Diagrams Graph

Field	Description	Type
1	node number of start node	integer
2	node number of end node	integer
3	length of arc	real
4	minimum radius of arc	real
5	pointers to via-point list	pointer

Each via-point is characterized by the parameters (x, y, R) where (x, y) represent the coordinates of via-point and R is the radius of the Voronoi Diagrams at via-point. Smooth parabolic curves can be approximated arbitrarily closely with piecewise linear arcs by controlling the spacing between via-points.

In order to search for the shortest path from source to goal, first a method must be developed to transfer the mobile object onto and off of the Voronoi Diagrams graph. Depending upon the relative positions of the workspace obstacles, this can be a subtle problem in its own right. For example, if the mobile object is a peg and the goal is to place the peg at the bottom of a narrow hole, then some careful planning is required to move the object from the Voronoi Diagrams graph to the goal. This type of task arises in fine motion planning for automated assembly [26]. The formulation presented here is not directed toward fine motion planning, but is instead restricted to gross motion planning. As such, no attempt is made to move an object that is in contact with an obstacle. Instead, the gross motion planner moves the object to a point somewhere near this destination a point at which a fine motion planner can then take over.

When the source and goal positions of the mobile object are not in contact with any workspace obstacles, the following simple heuristic method appears to work reasonably well for moving the object onto and off of the Voronoi Diagrams graph. First, a line is constructed through the centroid of the mobile object normal to the nearest obstacle edge. The mobile object then moves directly away from the nearest obstacle along this line until it reaches the Voronoi Diagrams graph. This point of intersection is the source node (s) or goal node (g), as appropriate. As the mobile object moves onto the graph from the source position to the source node, it rotates so that when it arrives on the graph its principal axis is aligned with the Voronoi Diagrams. Similarly, as the object moves off of the graph from the goal node to the goal position, it rotates so as to assume the proper orientation at the goal position.

A number of graph theory algorithms might be employed to search for a path from the source node to the goal node [14]. The technique used here starts by examining nodes adjacent to the source node. Nodes adjacent to these nodes are then investigated and the search continues to expand all branches as sub-graphs until each sub-graph reaches a dead end or the goal node. If the minimum radius of an arc is found to be less than half the width of the mobile object, a collision is inevitable.

Consequently, these narrow arcs are regarded as dead ends. When a node adjacent to sub-graph A is already reached by Sub-graph B, and the total length for sub-graph A exceeds the length for sub-graph B, sub-graph A is discarded. The shortest sub-graph which reaches the

goal node is then taken as the shortest collision-free Voronoi Diagrams path.

The criterion used to search the graph is simply the shortest path satisfying a minimum radius threshold. No effort is made to evaluate the difficulty of traveling that path. In some cases there may be a slightly longer path that does not require the mobile object to pass as close to the obstacles. One way to find such a path would be to increase the minimum radius to be something larger than half the width of the mobile object. This would tend to find safer paths. However, when the only path available is an extremely narrow path, it could fail to find any path at all. In workspaces with many obstacle vertices, a significant part of the total execution time is spent constructing the Voronoi Diagrams. Thus there appears to be no compelling reason not to use the shortest path unless there is difficulty in executing the motion along that path [27].

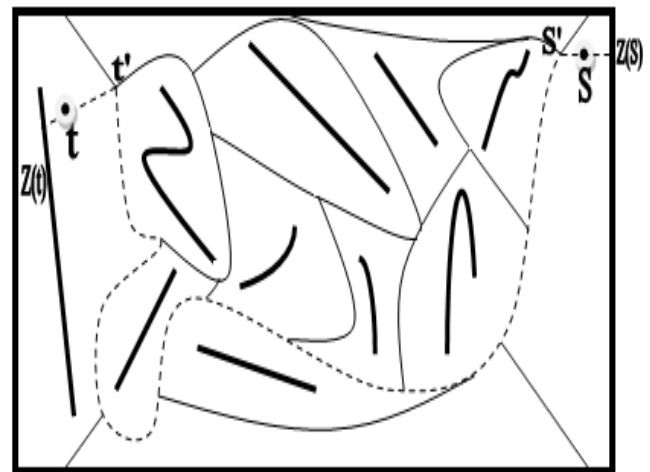


Fig. 6 The Motion Planning in Voronoi Diagrams Algorithm.

4 Trajectory Curve

The trajectory curve is very useful because its curvature varies linearly along the arc. Kanayama proposed to use this curve for motion trajectory design [28]. It is chosen for generating a flat path since it satisfies all the requirements for robot motion control tasks and modeling. The Trajectory curve is a real Spline and it cannot be expressed in a closed form. This is the biggest limitation and results in calculation difficulty. However, its curvature varies linearly along the arc, and the curve can be constructed from its curvature. On the other hand, since the parameter t is proportionate to the length of the arc, it can be used directly as a trajectory. The derivative of the curvature of trajectory curve is a constant which is similar with the new control theory in aimed at giving

solution to the optimal control problem. The trajectory curve is defined as following:

$$Cv(s) = k \times s + Cv_0 \quad (2)$$

Where s is the arc length, $Cv(s)$ is the curvature and k is a constant. The direction of the tangent vector is the integration of the curvature and is expressed by

$$\theta(s) = \int_0^s (k \times s + Cv_0) ds = \frac{1}{2} \times k \times s^2 + Cv_0 \times s + \theta_0 \quad (3)$$

In our work, two dynamically allocated points in a 2D space are identified at each state for each basic curve element configuration. Unlike a customary approach, we don't restrict the curve as a proportional double since it may encounter difficulties in a global configuration. The global trajectory is made up of a set of local curve pieces. Our approach is to use an antithetical trajectory curve element two dynamic control points $S2$ and $S4$ to offer the more flexible and powerful solution to the trajectory generation problem. In practice, a proportional pattern cannot always be guaranteed to be the optimal trajectory to follow, and the antithetical pattern is the general situation and offers more flexibility for the control process. In order to do a successful smooth connection at the joints between curves, keeping in mind that each goal state is also the new initial state for the next move step, the direction of the motion trajectory at the current goal location should be the same as at the next new initial robot location. The curvatures at the both locations should also be the same for smooth curvature transition. To meet these requirements, two transition points are recommended which are determined by the motion kinematical states of the robot to produce a smooth transition between curve elements.

As shown in Figure 7, the robot starts at $S1$. $S3$ and $S6$ are the points perceived, $S2$ and $S4$ are the two points added to control the robot movement to make a smooth transition between adjacent trajectory curve elements, which are derived from the robot motion requirements that satisfy the robot kinematical conditions and the equations (2) and (3). The first local curve element ends at the destination point $S4$, which is also the new initial position of the next trajectory curve element. At $S4$, the curvature is decreased to zero and the direction is from $S3$ to $S6$. The trajectory generation will keep going as long as subsequent path points are supplied [29].

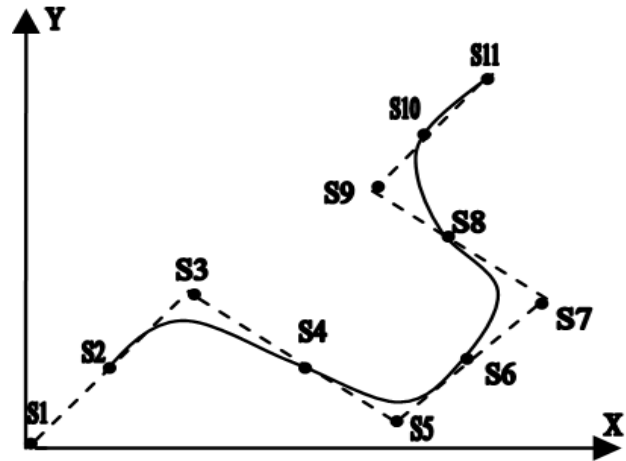


Fig.7. A Curve Configuration in 2D Environment

5 Experimental Results

5.1 Identify ink Workspace using Image Processing

Model shapes are made of black cardboard. Black colour is chosen because of the contrast between the light background and the model itself. There are five image processing shape models included in this experiment which are rectangle, cylinder, twice pyramid and cubic. Description Median filtering is a nonlinear operation often used in image processing to reduce "white and pepper" noise. Median filtering is more effective than convolution when the goal is to simultaneously reduce noise and preserve edges. By click "load figure" can be load an image from section image processing. The main window of MATLAB Software for path planning was shown in Figure 8 below.

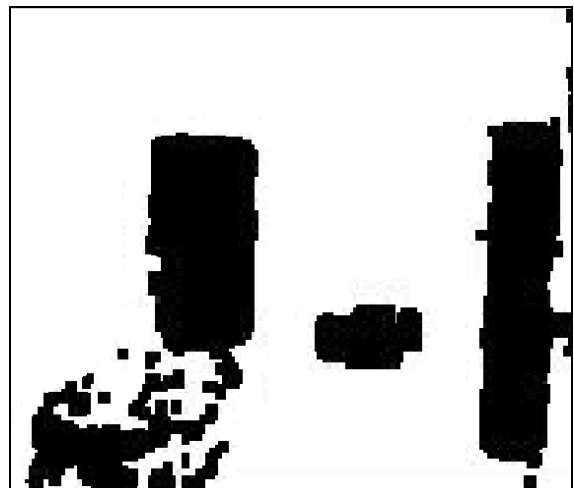


Fig. 7 Load Figure3 from Image Proceeding

5.2 Identify ink Workspace using Voronoi Diagrams

In this section we discussed about using Voronoi Diagrams method and in this part we intend to divide the workspace into multiple polygons using the Voronoi Diagrams method.

Firstly, the operator clicks "Load Figure" to load the processed image of the workspace (saw Fig. 8). Next the operator clicks "Voronoi Diagrams" to draw polygonal cells on the workspace (see Fig. 9). Finally, the operator clicks "Shortest Path" to draw the shortest path with continuous red circles. The total distance of the path and time taken to find the shortest path is displayed in the "Details" section of the program. The distance measured on the sample workspace is 13.46mm and the runtime was 1.14 seconds (see Fig. 10).

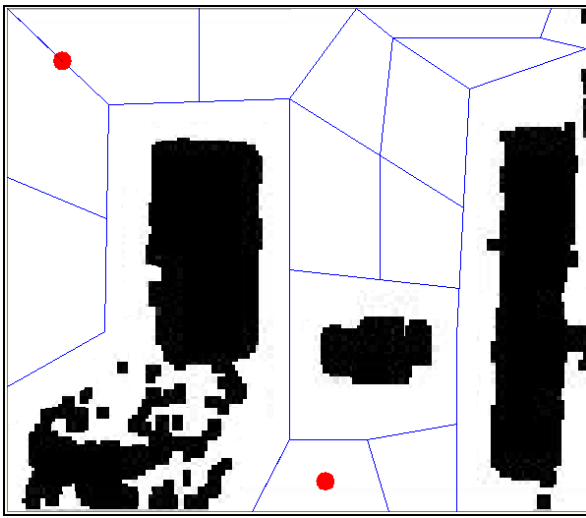


Fig.9. Changes from Actual Workspace to the Voronoi Diagrams Workspace

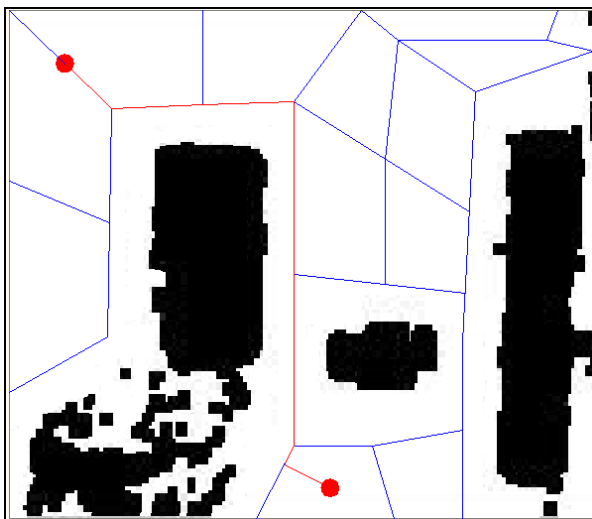


Fig.10. Find Shortest Path via the Voronoi VD(S) Method

5.3 Trajectory Curve Smoothing

A curved trajectory is very useful as it avoids sharp turns, leading to discontinuities in robot velocity, acceleration. To produce a curved trajectory, [28] proposed the concatenation of clothoids and cubic spirals through the use of cost functions such as the integral over the curvature's square and the integral over the square of curvature derivative. In [30], clothoids, approximated using Rational Bezier Curves, were used in trajectory generation for mobile robots.

In our work, we created a cubic spline from the optimal Voronoi Diagrams path to generate a smooth curved trajectory. In robot trajectory planning, cubic splines are widely used as they assure continuity of position, velocity and acceleration commands for each joint [31]. A fast compact algorithm for smoothing cubic splines implemented using MATLAB was introduced in [32]. The result of implementing the cubic spline smoothing algorithm on the generated quad tree map is shown in Figure 11.

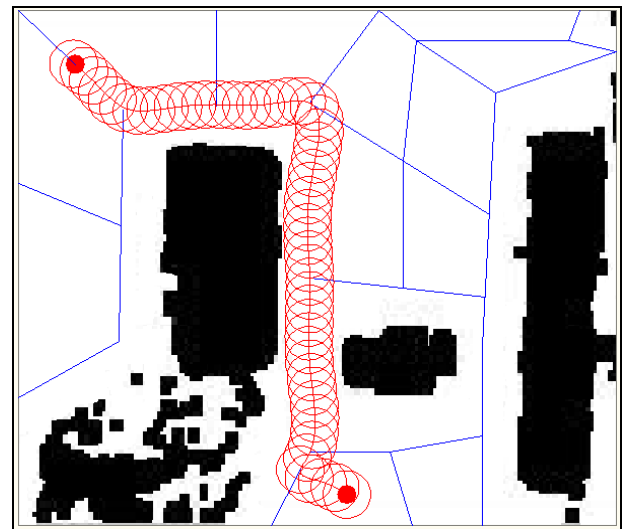


Fig.11. Fig. 3 Path after smoothing

6 Conclusion

This paper provides a framework for mobile robot navigation using a robot mounted webcam. Using images captured by the webcam, the location of obstacles are identified. Then, using the Voronoi Diagrams VD(s) technique, the shortest path to the target destination is determined. Future work would be to translate the generated optimal path into input commands to the actual robot. The system would also be further developed for situations with moving obstacles and moving targets.

Acknowledgments

This research was supported by the Ministry of Science Technology and Innovation, Malaysia, Grant No. 03-01-02-SF0459.

References:

- [1] J. Campbell, R. Sukthankar, Nourbakhsh, I. Sukthankar and A. Pahwa, A robust visual odometry and precipice detection system using consumer-grade monocular vision. Proc. ICRA2005, Barcelona, Spain. 2005.
- [2] G. Voronoi. Nouvelles applications des param`etres continus `a la theorie des forms quadratiques, Journal f`ur die Reine und Angewandte Mathematik 133, pp. 97–178, 1907.
- [3] A. Okabe, Barry Boots, K. Sugihara, and S. N. Chiu, "Spatial tessellations: concepts and applications of Voronoi diagrams," John Wiley & Sons, Inc., New York, NY, 2000.
- [4] I. Boada, N. Coll, N. Madern and J.A. Sellares, "Approximations of 3D Generalized Voronoi Diagrams," EWCG 2005, Eindhoven, March 9-11, 2005.
- [5] S. Garrido, L. Moreno, M. Abderrahim and F. Martin, "Path Planning for Mobile Robot Navigation using Voronoi Diagram and Fast Marching," Int. Conference on Intelligent Robots and Systems Oct, 2006.
- [6] Qhull code for convex hull, Delaunay triangulation, Voronoi diagram, and halfspace intersection about a point, <http://www.qhull.org/>.
- [7] Triangle: A two-dimensional quality mesh generator and Delaunay triangulator, <http://www.cs.cmu.edu/~quake/triangle.html>
- [8] E. Masehian and G. Habibi, "Robot Path Planning in 3D Space using Binary Integer Programming," International Journal of Mechanical System Science and Engineering ISSN 1307-7473, 2008.
- [9] G. Habibi, E. Masehian, M.T.H. Beheshti, "Binary Integer Programming Model of Point Robot Path Planning," The 33rd Annual Conference of the IEEE Industrial Electronics Society IECON ,Nov 5-8 , 2007.
- [10] R. Gonzalez, R. wood, "Digital Image Processing" 2nd edition. Prentice-Hall Inc, 2002.
- [11] A. Jain, "Fundamentals of Digital Image Processing". Prentice-Hall Inc, 1989.
- [12] R. Duda, E. Peter Hart and D. Stork. "Pattern Classification "2nd edition. John Wiley & Sons, Inc, 2000.
- [13] G. Blanchet, M. Charbit, "Digital Signal Image Processing Using MATLAB" ISTE Ltd, 2006.
- [14] L. Xingqiao, G. Jiao, J. Feng, Z . Dean, "Using MATLAB Image Processing to Monitor the ealth of Fish in Aquiculture". Proceeding of the 27th Chinese Control Conference July 16-18, Kunming, Yunnan China , 2008.
- [15] H. Akbar, A.S. Prabuwno, " Webcam Based System for Press Part Industrial Inspection" .IJCSNS International Journal of Computer Science and Network Security, VOL.8 NO.10, October, 2008.
- [16] G. Modesto, M. Medina, D. Baez-Lopez, " Focusing and Defocusing vision system (SIVEDI)". International Conference on Electronics, Communications and Computers (CONIECOMP 2005) IEEE, 2005.
- [17] K.T. Jenn, "analysis and application of Auto focusing and Three-Dimensional Shape Recovery Techniques based on Image Focus and Defocus". PhD Thesis SUNY in Stony Brook, 1997.
- [18] S.K. Nayar, M. Watanabe, M. Noguch, " Real-time focus range sensor". Intl. Conference on Computer Vision, PP.995-100, June, 1995.
- [19] M. Subbarao, " Spatial-Domain Convolution/Disconsolation Transform". Technique Report No.91.07.03, Computer Vision Levorotatory, State University of New York , Stony Brook, NY 11794-2350
- [20] S. Shojaeipour, S.M. Haris and M.I. Khairir, " Vision-based Mobile Robot Navigation using Image Processing and Cell Decomposition ." IVIC 2009 Springer Lecture Note in computer science LNCS-5857, pp.90-96 Nov, 2009.
- [21] Ch. Aung , H. Lwin , Y. K.T, M. Myint , " Modeling Motion Control System For Motorized Robot Arm using MATLAB". PWASET VOLUME ISSN 2070-3740 32 August, 2008.
- [22] E. Masehian and G. Habibi, " Robot Path Planning in 3D Space using Binary Ineger Programming," International Journal of Mechanical System Science and Engineering ISSN 1307-7473, 2008.
- [23] S.W. Bae and K.Y. Chwa, " Shortest Paths and Voronoi Diagrams with Transportation Networks Under General Distances," Springer – Verlag Brilin Heideberg, 2005.
- [24] F. Aurenhammer, R. klein, " Voronoi Diagrams," Research Concerning Voronoi diagrams, www.pi6.fernuni-hagen.de/publ/tr198.pdf
- [25] C. O'Dunlaing and C. K. Yap, "A retraction method for planning the motion of a disc," J. Algorithms, vol. 6, pp. 104-111, 1985.
- [26] T. Lozano-Perez, M. T. Mason, and R. H. Taylor, "Automatic synthesis of fine-motion strategies for robots," Znt. J. Robotics Res. vol. 3, no. 1, pp. 3-24, Spring 1984.
- [27] O. Takahashi and R.J. Schilling, "Motion Planning in a Plane using Generalized Voronoi Diagrams", IEEE Transaction on Robotic and Automation Vol. 5 No2. April, 1989.

- [28]Y. Kanayama, and B. I. Hartman, in: Smooth local path planning for autonomous vehicles. Robotics and Automation, vol. 3, pp. 1265-1270. 1989.
- [29]H.L. Weinert, in: A fast compact algorithm for cubic spline smoothing. Computational Statistics and Data Analysis, Vol. 53, pp. 932-940, 2009.
- [30]N. Montes, M.C. Mora and J. Tornero, in: Trajectory generation based on Rational Bezier Curves as clothoids. Proceedings of the 2007 IEEE Intelligent Vehicles Symposium, Istanbul, Turkey, pp. 505-510, 2007.
- [31]A. Visioli, “ Trajectory planning of robot manipulators by using algebraic and trigonometric splines. Robotica,” Vol. 18, pp. 611-631, 2000.
- [32]H.L. Weinert, “ A fast compact algorithm for cubic spline smoothing.” Computational Statistics and Data Analysis, Vol. 53, pp. 932-940, 2009.