Environment Detection and Path Planning Using the E-puck Robot

Muhammad Saleem Sumbal

Department of Electrical, Electronics and Automation Engineering
University of Girona, Spain
saleemsumbal@gmail.com

---***---

Abstract - ***Automatic path planning is one of the most challenging problems confronted by autonomous robots. Generating optimal paths for autonomous robots are some of the heavily studied subjects in mobile robotics applications. This paper documents the implementation of a path planning project using a mobile robot in a structured environment. The environment is detected through a camera and then a roadmap of the environment is built using some algorithms. Finally a graph search algorithm called A* is implemented that searches through the roadmap and finds an optimal path for robot to move from start position to goal position avoiding obstacles.***

Key Words: *Path Planning, E-puck robot, Environmental Detection.*

1. INTRODUCTION

Robot path planning can be categorized as a class of algorithms that accept high level description tasks and produce valid and efficient path combinations for the robot to follow. In simple words, path planning can be taken as a task in which the robot, whether it is a robotic arm or mobile robot, has to navigate from its start point to a specific (destination or goal) point by avoiding collisions with the obstacles in the way. Path planning has widespread usage in mobile robotics, manufacturing and automation etc. This paper aims at the implementation of a path planning project in a structured environment using a small mobile robot. The equipment used in this project is shown in Fig. 1. The robot used will be e-puck, which is a small mobile robot with simple mechanical structure and electronics software. It can be setup on a tabletop analysis of the results [2]. If the manuscript was written really have high originality, which proposed a new method or algorithm, the additional chapter after the "Introduction" chapter and before the "Research Method" chapter can be added to explain briefly the theory and/or the proposed method/algorithm [4]. next to a computer and connects with the computer through blue tooth, thus providing optimal working comfort.

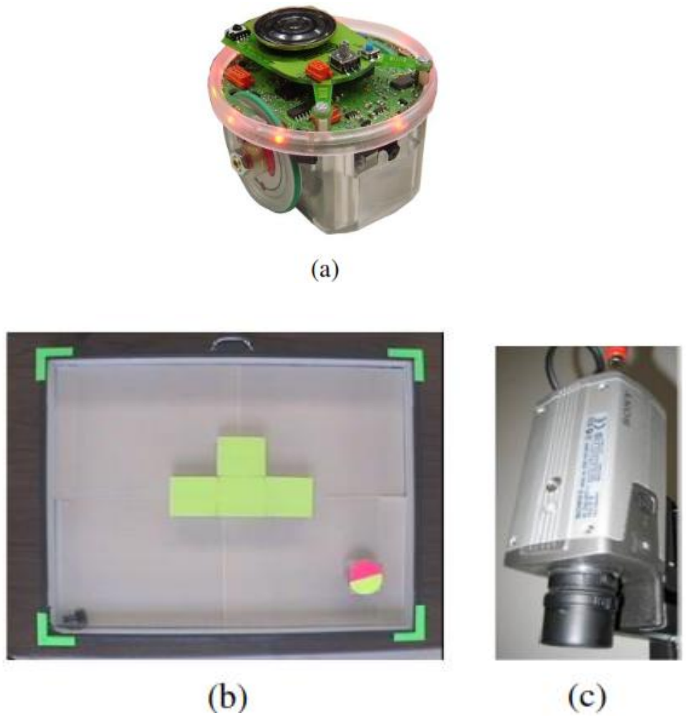


Fig- 1: (a) An e-puck robot (b) Environment containing the obstacle and robot (c) Camera for capturing images of environment

The environment consists of a wooden box containing some obstacles and e-puck. Colored papers have been used to identify the robot, obstacles and boundary of environment. Dark green color indicates the boundaries, light green color indicates the obstacles and two colors (magenta and light green) are used to indicate the robot. Two colors have been used for robot in order to determine its orientation. To detect the environment, a video surveillance camera (Sony SSCDC198P) is mounted on the top of this environment in the environment a goal point will be set by user and the epuck robot will have to reach this goal point from its current position (start point) by following the shortest collision free path in the environment. In order to achieve this, the project can be divided into following main tasks.

i) *Detection of the environment through camera and identification of the objects in the environment.*

First, an image obtained through camera will be segmented to get information about the boundaries, the obstacles and robot contained in the environment.

ii) *Detection of corners of the obstacles in the environment.*

The next step is to detect the true corners of the obstacles in the image which will be used along with the start and goal point to build a roadmap of the environment.

iii) *Obtaining a visibility graph of the environment.*

Visibility graph is one of the roadmap methods which will be used to represent environment by providing all the possible paths from start point to goal point.

iv) *Implementation of a graph search algorithm.*

Then, a graph search algorithm called A star (A*), will be implemented which will find optimal path by searching through all the paths provided by visibility graph.

v) *Programming the robot to follow the optimal path from start position to goal position.*

MathWorks MATLAB is used as programming language for implementing all these tasks.

2. METHODOLOGY

This section will describe the methodologies adapted for performing each of the tasks discussed in previous section.

2.1 Image Segmentation

In the field of robotics, the challenge is to do reliable segmentation of the scenes in order to plan the robot movement to perform a specific task. When the task is to identify few objects based on their color properties, threshold techniques have been used by researchers [1],[2]. In current case, as objects can be identified through their color, a thresholding technique has been used. Now, the image obtained from the

camera is an RGB image (see Fig. 2a). It is first converted from RGB space to HSV space. To select the threshold values for the three colors (light green, dark green and magenta), any homogeneous part of the image that only contains the required color is selected and the maximum and minimum values of the hue, intensity and saturation are obtained from this region. These values serve as the threshold values. After selecting the threshold values, all the image pixels are checked and if HSV values of a pixel fall within the threshold values of any of the three color classes, that pixel is assigned a specific value and thus pixel is represented by a unique color in the output image. For example, all the pixels that belong to class light green, will be shown in brown color in the segmented image. Similarly the pixels that belong to class dark green, will be shown in yellow color and the pixels belonging to class magenta will be shown in teal color. The pixels that belong to none of these classes, are assigned the blue color (see Fig. 2b). Only the hue and saturation value are taken into account for classifying the pixels, as it is expected that segmentation may become more robust to lighting variations if pixel luminance (intensity value) is discarded.

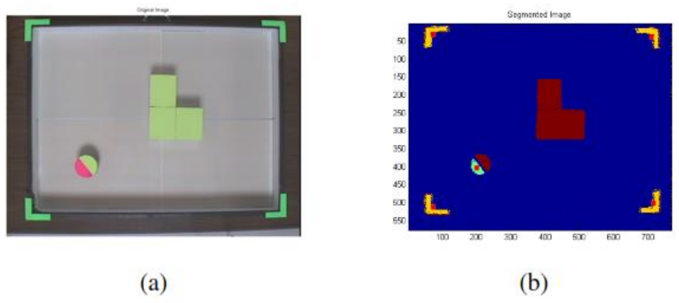


Fig-2: (a) Original Image and (b) Segmented Image

2.2 Identification of the Boundary Points and the Start Location

In Fig. 2b, there are red points in the corners and on the robot. The red points in corners are used to know the boundary of environment. These are obtained by calculating the individual centroids of the four boundary regions (in yellow color). The point on the robot is used to identify the start point. It is obtained by calculating the centroid of the segmented region with teal color. The pixels around the centroid point in a 5x5 window are also assigned the red color in order to make the point prominent.

2.3 Post Processing of the Segmented Image

In the segmented output of the image, the edges of the obstacles are not smooth. It is because the edges of the obstacles (in original image) are not sharp. The possible reasons may be because the camera used did not have a high picture quality and also because the illumination may not be uniform at time when corners of the obstacles which will serve as nodes in the visibility graph. First the image is converted to binary image and only obstacles are kept in image. Then the first step is to fill the holes in the image as there might be some missing pixels inside the segmented regions thus creating holes. After filling holes, two morphological operations opening and closing are performed on the image to smooth the boundaries. Fig. 3a shows the post processed image segmented region with teal color. Thus, post processing of the image is performed to smooth the edges. The smooth edges are required to detect the true corners of the obstacles which will serve as nodes in the visibility graph. First the image is converted to binary image and only obstacles are kept in image. Then the first step is to fill the holes in the image as there might be some missing pixels inside the segmented regions thus creating holes. After filling holes, two morphological operations opening and closing are performed on the image to smooth the boundaries. Fig. 3 shows the post processed image.

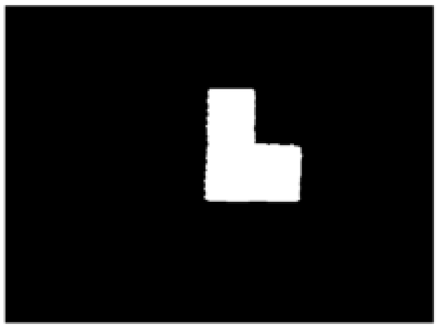
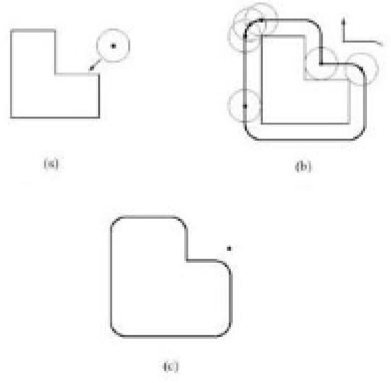


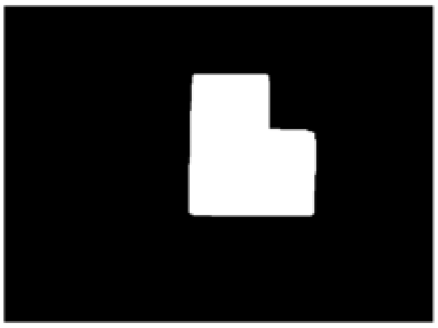
Fig-3: Post processed output of the image

2.4 Configuration Space Obstacles

Configuration space obstacle is the set of all configurations or positions of robot in which it can hit the obstacles. The robot is circular in current case and the center of robot is taken as a reference point. So sliding the robot around the obstacles and keeping track of the curve traced by reference point will give us the configuration space obstacle (C_{obs}) as shown in Fig. 4a. To get this C_{obs} , we dilate obstacles in image. As a circular curve is traced by robot around obstacles, a disk shaped structuring element should be used but for current project, the obstacles are dilated with a square shaped structuring element in order to preserve the corners that will be used in next step to build visibility graph. Now the radius of the robot is 3.65cm which is equivalent to 30 pixels. In order to keep the robot at safe distance from the obstacles, the number of pixels was multiplied with a factor 1.5. So in this way, the boundary of obstacles in the image was dilated to 45 pixels. Fig. 4b shows the configuration space obstacle for the robot.



(a)



(b)

Fig-4: (a) Configuration space obstacle for a circular robot taken from [10] (b) Configuration space obstacle obtained by dilation using a square structuring element.

2.5 Corner Detection

Two techniques were used for detecting the corners of the obstacles. The first technique used was the Harris corner detector [3] in which, first the image gradients I_x and I_y in x and y directions are calculated for each pixel. After that a neighborhood size as an area of interest is defined around each pixel. For each pixel in the image, the autocorrelation matrix is constructed from pixel and its neighborhood values.

$$M = \begin{bmatrix} \sum_W I_x^2 & \sum_W I_x I_y \\ \sum_W I_y I_x & \sum_W I_y^2 \end{bmatrix} \qquad (1)$$

Let λ_1 and λ_2 be the Eigen values of matrix M. Then an auto-correlation function R is defined as

$$R = \lambda_1 \lambda_2 - k(\lambda_1 + \lambda_2)^2 \qquad (2)$$

where k is an empirical constant. Sharply peaked values of R represent the corners in the image. A non maximum suppression is applied to get desired number of corner points for the obstacle. Now, the number of exact true corner points in an image can vary depending on the number of obstacles in it. So during non maximum suppression, this algorithm cannot determine by itself how many exact true corner points are there for the given obstacles in the image and thus these are given by user every time, a new image is taken. Secondly, in case of improper segmentation of the image, some of the corners of the obstacles do not remain sharp. Thus when Harris detector is applied, cornerness value of some of the true corner points is not that high as compared to cornerness value of other corner points (that are not true corners). As a result, Harris corner detector misses some corner points. Thus, another corner detection technique was tried to avoid these problems.

2.6 Corner Detector based on Local and Global Curvature Properties

This corner detector [4] is proposed by Xia Chen He and Nelson H.C Young and detects both fine and coarse features accurately at low computational cost. The main steps of this corner detector are

i) Applying canny edge detector to detect edges in image.

To detect the edges of obstacles using canny edge detector, the default matlab command for edge detection is used. The output of the canny edge detector is a binary edge map.

ii) Extracting the contours from the edge map.

If there is only one obstacle in the image then there will be only one contour and the points obtained through edge detection will represent this contour. For more than one obstacle, it is required to extract all the contours and to know which edge points belong to which contour.

iii) Computing the curvature for each contour and obtaining the local maxima.

After the contours have been extracted, the next step is to calculate the curvature value of the pixels of each contour as shown in (3). The curvature value for a corner point will be higher as compared to that of an edge point.

$$K_i^j = \frac{\Delta x_i^j \Delta^2 y_i^j - \Delta^2 x_i^j \Delta y_i^j}{[(\Delta x_i^j)^2 + (\Delta y_i^j)^2]^{1.5}} \quad \text{for i=1,2,..,N} \qquad (3)$$

From (3), all the local maxima of the curvature function will provide us the initial list of corner candidates.

iv) Round corner removal

From the initial corner list, round corners are removed first. Now, region of support (ROS) of a corner is defined as the segment of the contour bounded by corner's two nearest curvature minima. The ROS of each corner is used to calculate a local threshold adaptively given by (4) where u is the position of the corner candidate on the contour, L1 + L2 is the size of the region of support centered at u and R is a coefficient with value equal to 1.5.

$$T(u) = R \times \overline{K} = R \times \frac{1}{L_1 + L_2 + 1} \times \sum_{i=u-L_2}^{u+L_1} |K(i)| \qquad (4)$$

where $\overline{K}$ is the mean curvature of the ROS. This T(u) calculated for each corner will be compared with corner's absolute curvature value. Corners with absolute curvature value less than T(u) will be round corners and will be eliminated.

v) False Corner Removal

The next step is to remove the false corners due to trivial details and noise. Generally, a true corner will have a relatively sharp angle. So, the idea is to calculate the angle of each corner and compare it with a preset angle value to decide if it is a true or false corner. A three point method is used to do this which calculates angle of corner using tangents. Here ROS of a corner is defined as the segment of the contour bounded by the two neighboring corners of the current corner. In Fig. 5, point C is the corner in question and E and F are its two neighboring corners. In three point method, first on one arm of ROS (from C to E), three points(C, the midpoint M and E) are selected. If these 3 points are collinear, then the tangent direction is simply from C to E else the center of a suppositional circle is taken. The distance of the center point (C) of this circle is same from the 3 points. Let C = (x_1,y_1), M = (x_2,y_2) and E = (x_3,y_3). Using the coordinates of C,M and E, coordinates of C_0 are obtained . Then a line is drawn from C to C_0 and θ represents the direction of this line. Similarly the direction of line from C to M is given by θ. Then the tangent of C at this side of ROS will be given as:

$$\gamma_1 = \theta + sign(sin(\phi - \theta)) \cdot \frac{\pi}{2} \qquad (5)$$

where sign is a signum function. Similarly the tangent of ROS from C to F will be γ_2 and is determined by the same method.

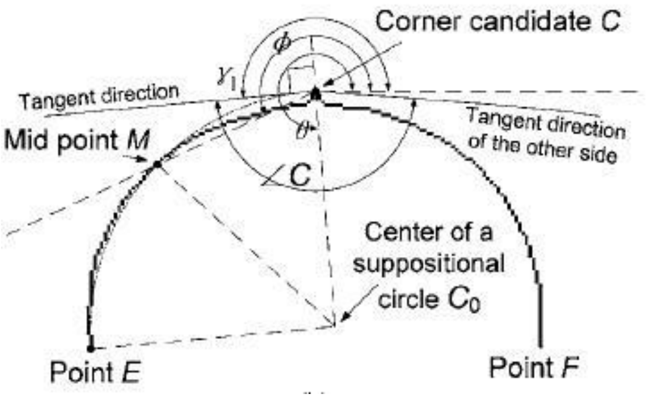


Fig-5: Explanation of angle calculation for corner C using tangents

Finally the angle of the corner C is given as:

$$\angle C = \begin{cases} |\gamma_1 - \gamma_2| & if\, |\gamma_1 - \gamma_2| < \pi, \\ 2\pi - |\gamma_1 - \gamma_2| & otherwise \end{cases} \qquad (6)$$

Then, the corner checking criterion is given as follows:

C is true corner if <Ci ≤ θobtuse

C is false corner if <Ci > θobtuse

The parameter θ_{obtuse} is a threshold value which is set to 162 degrees. Fig. 6 shows the corners detected for the obstacle in current image, using curvature based corner detector.

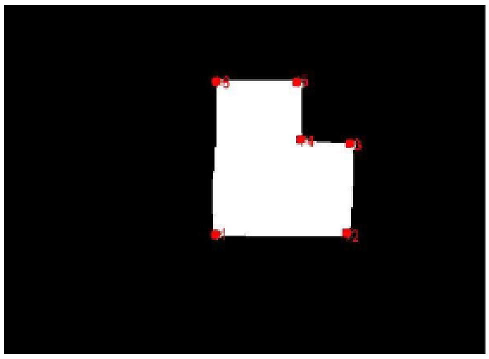


Fig-6: Output of Curvature based corner detector for one obstacle

2.8. Visibility Graph Construction

A visibility graph is constructed using start point , goal point and the corner points. Before constructing the visibility graph, the convex hull of the obstacles is calculated. The convex hull of a geometric object (such as a point set or a polygon) is the smallest set of points containing that object. The reason for obtaining convex hull is that the robot will always follow a shortest path to reach the goal. So in Fig.7, the robot will move from corner C to E and will never move from corner C to D and then from D to E as this path is not optimal. So the inner edges should not be taken into account and to do this, the convex hull is computed using the default matlab command 'convhull'.

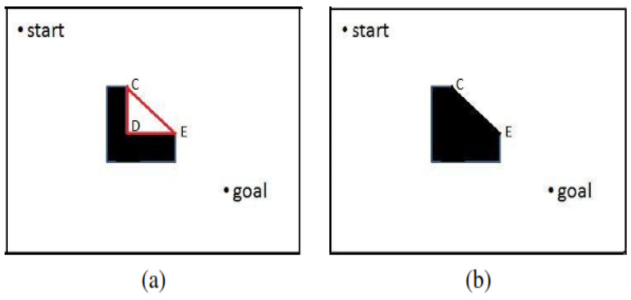


Fig-7: a) Original obstacle. b) Obstacle after computing convex hull

Now, first a brute force approach was implemented for visibility graph. According to this, if V is set of all nodes (vertices of obstacles, start point and end point) in the graph, then for each element, v ϵ V, the idea is to check whether all the line segments $\overline{vv_i}$, $v \neq v_i$ intersect completely an edge of the polygon. Finally, all the segments that do not intersect the edges of polygons constitute the visibility graph. The computational complexity of this approach is $O(n^3)$. To reduce the computational time, another approach called rotational plane sweep algorithm [5],[6] was implemented. Fig. 8 shows an example with some polygons and a start point P and a goal point named as Goal. Let S = {w_1, w_2, ..., w_{13}} be the set of nodes consisting of vertices of polygons, start point and goal point. For computing the set of vertices visible from a node (say from P), we will sweep a line l emanating from P and rotate the line from 0 to 2π in anticlockwise direction. If a vertex w is visible to P, then it is added to visibility graph otherwise not. In Fig. 8, solid black lines are the paths from P to corresponding vertices which are visible whereas the red dotted lines indicate the paths to vertices which are not visible from P. In this way, the process is repeated for all the nodes in the graph. The complete algorithm of rotational sweep is shown in Fig.9 is taken from Mark et al [6].

The sub routine 'VISIBLE' in algorithm 1 decides whether a vertex is visible from current point or not. First it is checked if $\overline{pw_i}$ intersects any interior of obstacle. If yes, w_i is not visible, otherwise, the search is only made in search tree T and edge closest to the point P is checked. If this edge intersects the segment pwi, then w is not visible else visible. In case of multiple vertices (vertices which are at same angle from the observation point) on the scan line l, the algorithm sorts the points in the increasing order of distance from the observation point (here P) and then performs visibility test through subroutine visible.

3. IMPLEMENTATION OF A* ALGORITHM

A*[7] is used to find the shortest path among all these possible paths provided by visibility graph. A* algorithm is noted for its accuracy and prominence. It enjoys a wide spread use in the field of computer science and robotics. A* uses an evaluation function f (n) to determine the order in which the nodes in the graph are to be searched. This function is expressed as sum of two functions.

1) The path cost function, g (n), which is basically the cost from starting node to the current node.

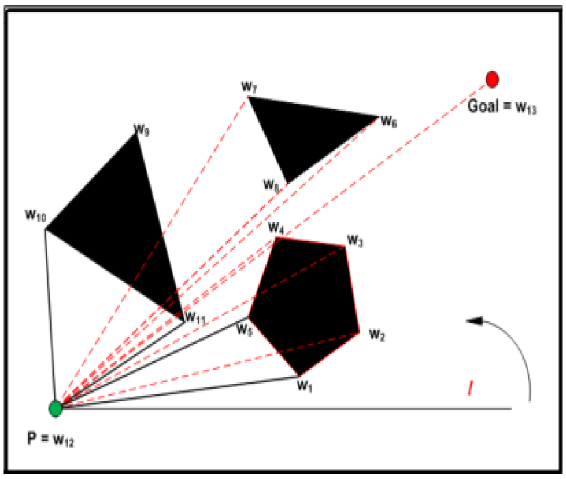


Fig- 8: Explanation of rotational plane sweep algorithm

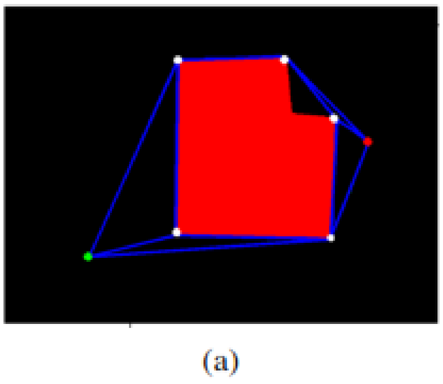
Algorithm 1 Rotational plane sweep algorithm

```
1:  Input: A set S of polygonal obstacles, start point and goal
    point. A point p which can be either any of the vertices
    of the polygons or can be the start or goal point.
2:  Output: The set of all nodes visible from p.
3:  Sort the nodes according to the anti-clockwise angle that
    the half-line from p to each node makes with the positive
    x-axis. In case of ties, nodes closer to p should come
    before nodes farther from p. Let $w_1, ...., w_n$ be the sorted
    list.
4:  Let $l$ be the half-line parallel to the positive x-axis
    starting at p. Find the obstacle edges that are properly
    intersected by $l$, and store them in a balanced search tree
    T in the order in which they are intersected by $l$.
5:  $lim_{W \to \infty}$
6:  for $i = 1, ...., n$ do
7:    if VISIBLE($w_i$) then
8:      Add $w_i$ to $W$
9:    end if
10:   Insert into T the obstacle edges incident to $w_i$ that lie
      on the anti-clockwise side of the half-line from p to
      $w_i$
11:   Delete from T the obstacle edges incident to $w_i$ that
      lie on the clockwise side of the half-line from p to $w_i$.
12: end for
13: return $W$
```

Fig-9: Rotational Plane Sweep Algorithm

2) A heuristic estimate of the distance from the current node to the goal node. It is given as h(n).

This evaluation function, f(n) = h(n) + g(n), maintains a balance of the two function as it moves from the start point to the goal point. A* starts with the start node and maintains a priority queue ('open list') of the nodes to be visited along with their costs (value of f(n). Another list called 'closed list' is also used which contains the nodes that have been visited. This list also contains the back pointer to the visited node. Back pointer points to the node from which the visited node originated. Fig. 10 shows the outputs for visibility graph and shortest path calculated by A*.



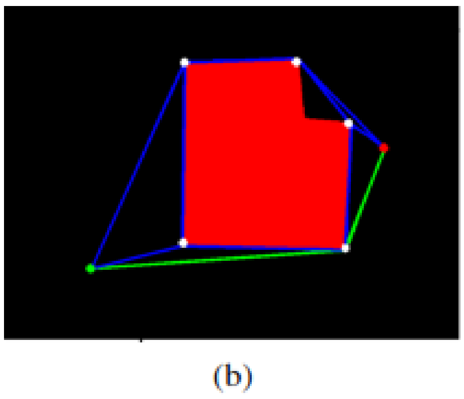


Fig-10: (a) Visibility Graph with green node as start point and red node as goal point (b) Shortest path (in green color) found by A*

4. PROGRAMMING THE E-PUCK ROBOT TO FOLLOW THE PATH

This part comprises of several steps. First, a toolbox ePic2 is used to control e-puck within matlab. This toolbox allows the user to develop an interface between e-puck and matlab using a set of commands. Now, the shortest path by A* basically comprises of the nodes whose x,y coordinates are known with respect to the image coordinate system. To move the robot in actual environment, we transform these coordinates from image coordinate system to world coordinate system. For this, we use a transformation matrix that comprises of intrinsic and extrinsic parameters of camera which are obtained through camera calibration using the well known Bouguet's toolbox. The images of a checkerboard pattern taken at different angles and different positions were used for computing the intrinsic and extrinsic parameters. Next, we determine the orientation of robot, required to make the robot move in the proper direction. For this purpose, the robot has been assigned two colors. The idea is to obtain the centroids of the 2 segmented regions on robot. The centroids are calculated using the matlab function regionprops. Now the light green color indicates front side of robot whereas magenta color indicates the back side of the robot.

Let (xm; ym) be the centroid of magenta region and (x_{lg}; y_{lg}) be centroid of light green region. Then, the orientation of the robot is calculated in matlab using atan2 ($y_{lg} - y_m$, $x_{lg} - x_m$). Fig. 11a shows how the orientation is calculated. After determining the orientation, the angle of the target point with respect to the robot is calculated. Now, if there are three nodes in the path, the robot will go from start node to second node and then to goal node. So the first target for the robot will be the second node and then the final target will be the goal node. Hence the term target is used in this sense.

Let (x_{target} , y_{target}) be the target point. Then, the angle of target from the center of robot is calculated as atan2($y_{target} - y_{center}$, $x_{target} - x_{center}$) (see Fig. 11b). In order to move towards the target point, the robot orientation should be towards target. That means that theta robot and theta target should be the same. So depending on the angle of the robot at start, the robot moves clockwise or counter clockwise to align itself with the target and to make the theta robot equal to theta target. When the difference between theta robot and theta target is less than some threshold, the robot starts moving towards the target at a constant speed. As the robot moves, the orientation of robot will not remain aligned with the target point due to odometry errors. So to handle this, the images of the environment are continuously captured and theta robot and theta target are calculated again and again as the robot is moving. Depending on these angles the movement of the robot is adjusted to make sure that robot moves in proper direction. Also a check is kept on distance between the robot center and the target point. When the distance is less than 1 cm, the robot stops meaning that it has reached the target. If the robot has not reached the goal node (final target), it then moves towards the next node by following the procedure explained above and thus keeps on going until it reaches the goal node. Fig. 11c shows how the robot has reached the goal position by following the shortest path.

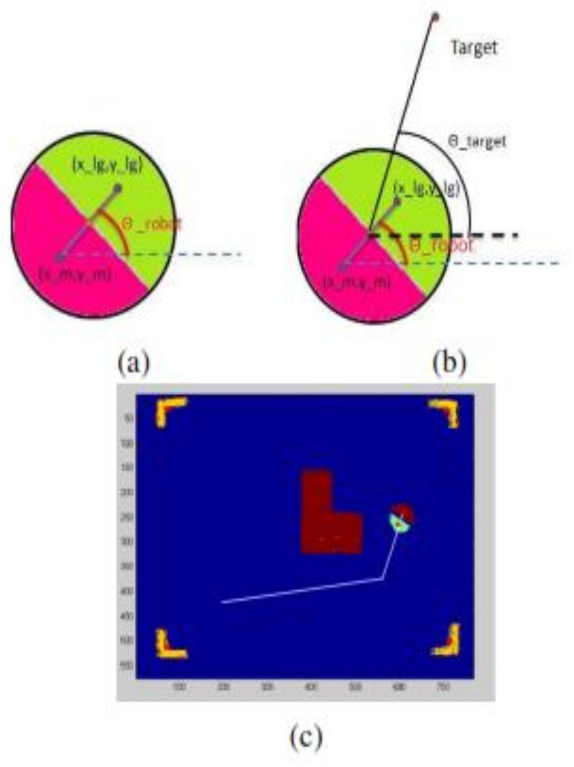


Fig-11: (a) Orientation of robot (b) Explanation of target angle with respect to robot (c) Robot reaches the goal position by following the shortest path

5. RESULTS

Results obtained for different experiments will be described now. Fig. 12 gives an example how the outputs will be in case of more than one obstacle. Table-I explains the overall efficiency of the algorithms in case of varying number of obstacles and Table-II compares the performance of two approaches used for constructing visibility graph.

No of Obstacles	Time(sec)				
	Segmentation	Corner detection	Visibility Graph	Shortest path by A*	Total time
1	0.2694	0.5306	0.435	0.0598	1.2948
2	0.271	0.5655	0.584	0.0604	1.4809
3	0.2691	0.5909	0.704	0.0619	1.6259
4	0.2684	0.611	0.936	0.0641	1.8795
5	0.2685	0.8476	1.19	0.06397	2.3701

TABLE I: Table explaining the performance of the algorithms for different number of obstacles considering obstacles of same size and shape to avoid bias

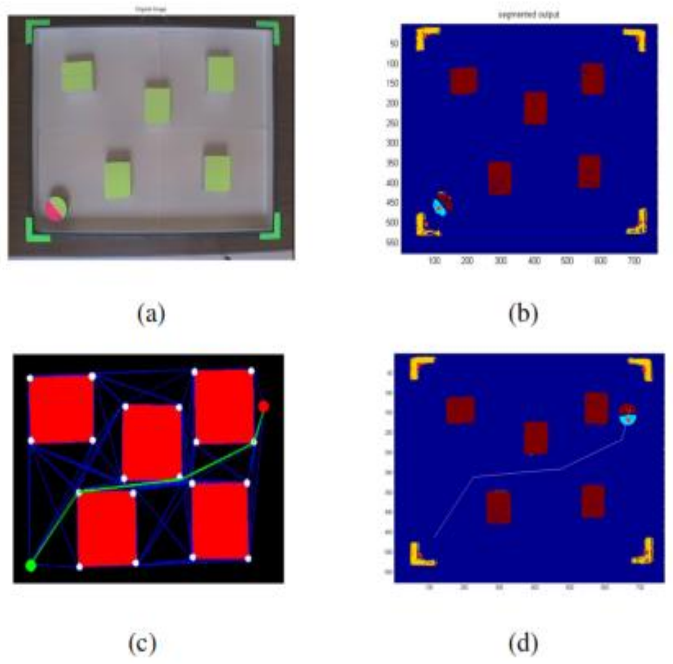
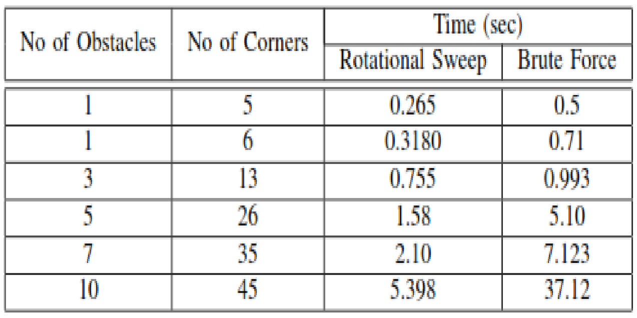


Fig-12: Outputs of the algorithms in case of 5 obstacles (a) Original image (b)Segmented image (c) Visibility graph along with the shortest path by A* shown in green color (d) Path followed by robot to reach goal



No of Obstacles	No of Corners	Time (sec)	
		Rotational Sweep	Brute Force
1	5	0.265	0.5
1	6	0.3180	0.71
3	13	0.755	0.993
5	26	1.58	5.10
7	35	2.10	7.123
10	45	5.398	37.12

TABLE II: Table explaining the performance of two approaches for visibility graph

6. CONCLUSION AND FUTURE WORK

6.1 Conclusion

From previous section, some important inferences can be made regarding efficiency of algorithms. With the increase in number of obstacles (of same size and shape), the number of corner points to be detected also increase. So, the corner detection algorithm will take more time. This trend can change if obstacles of varying shapes and sizes are considered as number of corner points of each obstacle can vary in that case. Then, time taken for the visibility graph increases with increase in number of nodes as more paths will be generated among nodes. Also, rotational plane sweep approach for constructing visibility graph outperforms the brute force approach. Time taken by A* depends on where the goal is located in the environment and how many obstacles are there in between goal and start point. If the goal point is located close to the start point, algorithm will take less time as it will search a few paths to reach goal and vice versa. Also, proper lightening conditions should be maintained for image segmentation otherwise the results can vary.

6.2 Future Work

In this project, static environment is assumed i.e. the obstacles are fixed. However, when there are moving obstacles in the environment, it is required to re plan the path after specific intervals in order to know if any change occurred in the environment. D* [8] algorithm which is an extension of A*, can be implemented in this case. Then, in this project, post processing is done after image segmentation to smooth the boundaries of obstacles. An efficient technique was found later on called simplifying polygons which can be applied to get the smooth boundaries of obstacles. Improvements can also be made regarding visibility graph construction. More efficient algorithms for example Ghosh and Mount [9], can be implemented to further reduce the computation time for visibility graph.

REFERENCES

[1] James Bruce and Tucker Balch and Manuela Veloso, ”Fast and Inexpensive Color Image Segmentation for Interactive Robots”, In Proceedings of IROS-2000,2061–2066, 2000.

[2] Robert T. McKeon; Mohan Krishnan; Mark Paulik., ”Obstacle recognition using region- based color segmentation techniques for mobile robot navigation.”, Proc. SPIE,,6384, 63840R 2006.

[3] C. Harris and M. Stephens, ”A Combined Corner and Edge Detector”, 4th ALVEY Vision Conference, pp 147–151,1988.

[4] He, Xiao C. and Yung, Nelson H. C., ”Corner detector based on global and local curvature properties”, Optical Engineering, 47(5),2008.

[5] Lee,D.T, Ph. D. thesis and Tech. Report,ACT-12, Coordinated Science Laboratory, University of Illinois at Urbana-Champaign, Urbana, IL(1978).

[6] M. de Berg, M. van Kreveld, and M. Overmars, ”Computational Geometry: Algorithms and Applications”, Springer,Berlin,1997.

[7] Hart, Peter E. and Nilsson, Nils J. and Raphael, Bertram, ”A Formal Basis for the Heuristic Determination of Minimum Cost Paths”, SIGART Bull.,(37): pp 28–29,1972.

[8] Anthony Stentz, ”Optimal and Efficient Path Planning for Partially Known Environments”, IJCAI’95: Proceedings of the 14th international joint conference on Artificial intelligence,1652–1659, 1995.

[9] Ghosh, Subir Kumar and Mount, David M., ”An output-sensitive algorithm for computing visibility”, SIAM J. Comput.,20(5), pp 888– 910,1991.

[10]http://www.cs.cmu.edu/~motionplanning/lecture/Chap3-Config-Space_howie.pdf